MagicDraw 19.0 LTR SP4
User Manual

No Magic, Inc., a Dassault Systèmes company, 2020

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 1998-2020 by No Magic, Incorporated, a Dassault Systèmes company. All Rights Reserved.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Contents

Plugin documentation 19

User Guide 19

MagicDraw getting started 20

User and product registration 20

Understanding the user interface 24

Most common shortcut keys 61

Assigning shortcut keys 63

Customizing environment options 65

Customizing and Selecting Perspective 67

Configuration files 73

Disabling inbound network activity 76

Improving performance 77

Support 84

News Reader 97

Creating path variables 99

Launchers 101

Special characters 103

Working with projects 104

Basic concepts 105

Creating projects 105

Saving projects 113

Autosave 114

Opening projects 115

Importing data from other UML tools and formats 116

Exporting UML models 201

Creating and using project templates 202

Projects with file attachments 208

Setting project options 214

Searching 216

Filtering 233

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Project partitioning 236
MOF support 269
Ecore support 271
Working with standard profiles 288
Forward compatibility 289
CATIA Systems Synthesis 291
Diagrams 294
UML diagrams 294
Analysis diagrams 333
Extension diagrams 376
Tables 405
Diagramming 482
Creating diagrams 483
Opening diagrams 485
Dragging objects 487
Displaying elements 490
Copying and pasting objects 517
Using diagram tabs 519
Specifying diagram properties 520
Specifying diagram style properties 522
Diagram name and its context name synchronization 523
Diagram frame 523
Drawing diagram shapes 525
Overviewing other diagrams 526
Legends 530
Table with diagram information 553
Changing diagram type 553
Working with symbols of model elements 555
Working with paths and relationships 557
Smart manipulation 567
Compartments 569

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Smart shape sizing 571
Selection and multiple selections 572
Copying text or images to diagrams 573
Nesting image shapes 574
Zooming 574
Using the grid 576
Pusher and magnet 577
Layout 579
Saving as image 584
Layout templates 589
Diagram aspects 608
Complete diagrams 622
Printing 626
UML elements 628
Model elements and symbols. 629
Action 629
Activity 643
Activity Final Node 646
Actor 646
Anchor 648
Artifact 649
Association 651
Behavior 662
Class 666
Collaboration 680
Comment 683
Component 685
Component realization 686
Connector 686
Constraint 689
Containment 692
Control Flow 695

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Data Type 696
Decision and Merge 698
Dependency 699
Device 708
Event 708
Exception Handler 711
Execution Environment 712
Extend 713
Extension 714
Flow Final Node 714
Fragment 715
Function Behavior 718
Gate 719
Generalization 722
Include 730
Information Flow 732
Information Item 733
Initial Node 735
Instance 735
Interaction 739
Interface 741
Interface realization 744
Internal Transition 744
Interruptible activity region 745
Lifeline 746
Link 750
Manifestations 755
Message 756
Message Occurrence Specification 780
Metaclass 780
Model 780
Node 782

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Note 791
Object Flow 794
Object Node 798
Opaque Behavior 799
Package 800
Part 804
Pin 805
Port 809
Profile 816
Profile Application 818
Realization 818
Reception 820
Report Template 821
Slot 824
Smart Package 824
State Machine 842
Stereotype 856
Substitution 860
Subsystem 860
Swimlanes 861
Tag 864
Template 866
Transition 872
Trigger 875
Use Case 876
Value Specification 899

Auxiliary diagram symbols 900

Representation of the Common category in the Diagram pallet 901
Image Shape 901
Diagram Overview 902
Separator 904
Rectangular shape 906

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Text Box 907

Working with model elements 909

Model elements and symbols. 910

Creating elements from other resources 911

Specification window 913

Default property values 919

Editing property values 921

Sorting elements 934

Refactoring 936

Extracting 947

Formatting symbols 975

Customizing lists of properties and commands 1001

Manage Navigation 1005

Selecting elements 1031

Selecting port and part types 1036

Generic numbering mechanism 1040

Favorites 1050

HTML editor 1055

Parameters synchronization 1064

Copying and opening element URLs 1069

Setting date and time 1070

Owner of model elements 1073

Selecting element types 1075

Changing the tooltips style 1079

Marking elements as auxiliary 1080

Reordering elements 1081

Tools 1083

Model Merge 1083

Metric Suites 1114

Model Transformation Wizard 1131

Executable concepts 1147

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Resource Manager 1150
Creating Setters and Getters 1154
Implementing or Overriding Operations 1155
Automatic Instantiation Wizard 1156
Pattern Wizard 1162
Spelling Checker 1164
Integration with external Evaluators 1173
Project Statistics 1189
Model analysis 1192
Analyzing usages and dependencies 1192
Traceability 1201
Specifying criteria for querying model 1227
Element usage in diagrams 1278
Identifying Package Dependencies 1280
Comparing projects 1291
Validation 1306
Model Visualizer 1346
Suspect Links 1354
Collaborative modeling 1360
Basic concepts 1361
Scenario to get started 1363
Starting a collaboration session 1363
Using Teamwork Cloud 1367
Using Teamwork Server 1479
Report Wizard 1547
Report Wizard dialog 1547
MRZIP file automatic deployment 1589
Generating reports from the Containment tree 1590
Report Wizard Quick Print 1597
Report Wizard template language 1599
Template variables 1607

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Helper tools 1621
Report Wizard template editor 1727
Generating Reports from Report Wizard 1734
Generating reports from the command line 1793
Configuring Report Wizard 1811
Debug report template 1820
Use Case driven 1824
Javadoc syntax tool 1833
Import tool 1840
JavaScript tool 1844
Groovy script tool 1850
Ruby Script tool 1853
Dialog tool 1857
Text tool 1866
Model validation tool 1871
Metrics tool 1875
Generic table tool 1884
Dependency Matrix tool 1898
Query tool 1904
Template Creation Tutorial 1916
Appendix A: report extensions 1951
Appendix B: office open XML format template 1959
Appendix C: OpenDocument format template 1975
Appendix D: HTML tag support 1985
Appendix E: DocBook Support 2021

UML Profiling and DSL Guide 2034

Basic Concepts 2035
Working with Profiles 2036
Creating Profiles 2036
Applying Stereotypes 2043
Modifying Profile 2044

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Customizing diagrams 2045
Creating new diagram type 2046
Creating new dependency matrix type 2052
Customizing diagram palette 2057
Creating new relation map type 2060
Creating new generic table type 2064
Domain Specific Language Customization 2067
Creating Customization Data 2068
Using Customization Data 2093
Distributing Profiles 2156
Exchanging Profiles via XMI 2156
Developer Guide 2156
Plugins 2158
How plugins work 2158
Writing plugins 2159
Compilation classpath 2161
Testing plugin 2162
Detail information 2163
Resource dependent plugin 2169
Creating test cases 2171
Developing plugins using IDE 2179
Distributing Resources 2185
How to distribute resources 2186
Creating required files and folders structure 2188
Resource manager descriptor file 2192
Building a resource distribution file 2196
Adding new functionality 2197
Invoking actions 2197
Creating new actions 2198
Actions' hierarchy 2205
Working with icons 2206

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Running action with progress 2209
Working with project 2210
Project concept 2210
Project descriptor 2211
Projects management 2211
Working with projects from Teamwork Server 2219
Working with projects from Teamwork Cloud server 2220
Project structure decomposition 2221
Project options 2221
Project life-cycle events 2224
UML model 2225
Working with UML model 2226
Event support 2249
Working with stereotypes and tagged values 2257
UML Model Implementation Using EMF 2267
Working with diagrams 2269
Diagram types 2270
Creating a diagram 2276
Presentation elements (symbols) 2277
Selection in diagrams 2300
Diagram events 2300
Custom diagram painters 2303
Creating diagram images 2305
Creating custom Drag&Drop handlers on diagram 2305
Managing navigation in model 2306
Customizing double-click behavior 2306
Managing hyperlinks 2306
Properties 2308
Information logging 2308
Environment Options 2310
Example of adding custom environment options 2310

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example of accessing environment options 2310
Browser 2310
Adding custom panel or tree into browser 2312
Configuring node text and icon 2312
Using and extending other UI components 2313
Configuring element Specification window 2313
Showing Element seletion dialog 2314
Showing question, error, warning dialogs 2316
Showing notifications, adding text into Message Window 2316
Creating PopupMenu or other menu 2317
Adding custom project window 2318
Generic tables 2319
Creating generic table 2319
Setting generic table filter 2319
Getting generic table element types 2320
Adding/getting columns 2320
Getting cell value 2321
Getting column names 2321
Getting row elements 2322
Adding row element 2323
Removing row element 2323
Working with Dependency Matrix 2324
Creating a custom dependency matrix type 2325
Customizing a dependency matrix 2327
Accessing dependency matrix data 2334
Annotating the elements 2335
Creating validation rules 2335
Validation rule developer's roadmap 2336
Create OCL2.0 validation rule 2337
Binary validation rule 2339
Implementing a binary validation rule 2340

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Binary validation rule in the plugin 2345
Script validation rule 2346
How to provide a solution for a problem found during validation? 2346
Custom Legend Item Implementation 2347
Custom elements numbering 2347
IJavaNumberPart interface 2348
INumberingAction interface 2350
Creating Use Case Scenario 2355
Merging and differencing 2357
Code engineering using API 2357
Code engineering set 2358
Managing code engineering sets 2359
Samples of the forward and reverse engineering 2361
Oracle DDL generation and customization 2363
Understanding the Oracle DDL template structure 2364
Customizing templates 2365
Utility class 2365
Example 2374
Running programs in batch mode 2375
Specifying batch mode program classpath and required system properties 2376
Starting MagicDraw or other modeling tool as part of another application 2382
MagicDraw file format 2382
File format in MagicDraw 3.6-17.0 2382
File format in MagicDraw 17.0.1 and up 2382
Jython Scripting 2385
Creating script 2385
Plugins migration 2388
Plugins migration to MagicDraw 18.2 and later Open API 2388
Plugins migration to MagicDraw 18.1 and later Open API 2388

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Plugins migration to MagicDraw 18.0 and later Open API 2388
Plugins migration to MagicDraw 17.0.1 and later Open API 2388
Plugins migration to MagicDraw 15.0 and later Open API 2393
Getting licensing information 2395
Multi-threading 2395
Idle job service 2396
Application environment 2397
Attached Files 2397
Element Referencing in Texts 2397
Creating text that refers an element 2397
Setting text with element references 2398
Code Engineering 2399
Abbreviations 2399
Code Engineering Sets 2400
Round Trip Set dialog box. Add files tab 2402
Round Trip Set dialog box. Add data from model tab. 2403
Generating Code 2404
Code Generation for a Set 2404
Reverse Options 2407
Importing JAVA .JAR packages 2426
Rules of association or attribute creation on reverse 2428
Global options for Code Engineering 2431
Code Engineering Options for all Sets in your Project 2431
Java Documentation Properties dialog box 2439
Round Trip 2440
Type Mapping Table 2441
Files of Properties 2442
Java Code Engineering 2443
References 2443
Java support in modeling tools developed by No Magic, Inc. 2443

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Java Mapping to UML 2444
Java CE Options 2465
Method Implementation Reverse 2470
C++ Code Engineering 2470
References 2471
Demo 2471
Supported dialects 2471
C++ ANSI Profile 2472
C++ Mapping to UML 2487
Conversion from an Earlier Project Version 2527
DSL Customization 2557
ANSI Profile Constraints 2562
Project Constraints 2563
Microsoft C++ Profiles 2567
Symbian macro 2619
@see on documentation 2619
Doxygen-after-member documentation 2620
CG Properties Editor 2620
Support C++ dialects 2622
C# Code Engineering 2622
C# 2.0 features 2622
C# 3.0 features 2648
Migration from old C# project version 2650
C# mapping to UML rules 2655
C# Project Constraints 2676
CORBA IDL Mapping To UML 2680
CORBA interface implementation 2682
WSDL 2683
WSDL Mapping to UML Elements 2685
Macro Engine Documentation 2700
Introducing Macro Engine 2700

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Contents

Selecting a default macro language 2701
Creating a Macro 2703
Adding a Macro and editing Macro information 2703
Opening Macro Information Dialog 2704
Adding a Macro and Its Information 2705
Editing Macro Information 2710
Macro Information Dialog Mnemonic Keys 2711
Deleting and executing Macros 2713
Deleting a Macro 2713
Executing a Macro 2713
Organizing Macros Dialog Mnemonic Keys 2716
Macro keyboard shortcuts 2718
Assigning a keyboard shortcut to a Macro 2720
Removing keyboard shortcuts from Macro 2720
Opaque objects 2721
Getting an Opaque Object 2722
Getting Element Property Values 2723
Setting Element Property Values 2724
Getting the Child of an Element 2728
Getting the Owner of an Element 2729
Creating a New Element 2729
Creating a Relationship Between Elements 2729
Removing an Element 2730
Adding a Stereotype to an Element 2730
Removing a Stereotype from an Element 2730
Printing Element Details 2730
Recording Macros 2731
Supported scripting languages 2733
Using code completion to develop BeanShell scripts 2734
Using NetBeans IDE to develop Groovy Scripts 2735
Using Eclipse to develop Groovy scripts 2736

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Installing Gems for JRuby 2736
Adding a scripting language to MagicDraw 2737
UML 2.5.1 Meta Model 2739
UML 2.5.1 support 2739
Migration to UML 2 user guide 2740
Basic concepts and instructions 2740
Migration of projects using MagicDraw Project Converter 2741
Migration of used projects and profiles 2742
Converting Used Projects 2742
Solving Cyclic Usages 2743
Converting Stereotypes/Profiles 2743
Migration of Teamwork Projects 2744
Preparing old Teamwork projects 2744
Converting Teamwork Projects 2744
Special Cases Of Teamwork Used Projects 2745
Converting Local Versions of Teamwork Projects 2746
Converting Teamwork Projects on IDE Integration 2746
Problems and solutions of data maping 2746
Tracing migration problems 2750
To trace marked elements 2750
Autoupdate issues 2750
Legal Notices 2751
Trademarks 2751
Third-Party Copyrights Notices 2751
Restricted Rights 2785

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Plugin documentation

SysML Plugin[1]

Cameo Simulation Toolkit(see page 19)

Cameo Requirements Modeler Plugin[2]

UAF Plugin[3]

UPDM 2 Plugin[4]

Excel Import Plugin(see page 19)

Cameo DataHub Documentation(see page 19)

CSV Import Plugin[5]

Docs of other versions

- MagicDraw 19.0 LTR SP3[6]
- MagicDraw 19.0 LTR SP2[7]
- MagicDraw 19.0 LTR SP1[8]
- MagicDraw 19.0 LTR[9]
- MagicDraw 18.4[10]
- MagicDraw 18.3[11]
- MagicDraw 18.2[12]

User Guide

Welcome! This user guide will walk you through the basics of using MagicDraw, including working with projects, setting up the modeling environment, collaborating with the help of Teamwork Server (see page 1479)or Teamwork Cloud(see page 1367), generating reports, and more.

To meet the customer needs, special editions of the modeling tool are developed:

- MagicDraw editions[13]. Click the **Editions** tab on the page.

1 https://docs.nomagic.com/display/SYSMLP190SP4/SysML+Plugin+Documentation
2 https://docs.nomagic.com/display/CRMP190SP4/Cameo+Requirements+Modeler+Plugin+Documentation
3 https://docs.nomagic.com/display/UAFP190SP4/UAF+Plugin+Documentation
4 https://docs.nomagic.com/display/UPDM2P/UPDM+2+Plugin+Documentation
5 https://docs.nomagic.com/display/CSVIP184SP4/CSV+Import+Plugin+Documentation
6 https://docs.nomagic.com/display/MD190SP3/MagicDraw+Documentation
7 https://docs.nomagic.com/display/MD190SP2/MagicDraw+Documentation
8 https://docs.nomagic.com/display/MD190SP1/MagicDraw+Documentation
9 https://docs.nomagic.com/display/MD190/MagicDraw+Documentation
10 https://docs.nomagic.com/display/MD184/MagicDraw+Documentation
11 https://docs.nomagic.com/display/MD183/MagicDraw+Documentation
12 https://docs.nomagic.com/display/MD182/MagicDraw+Documentation
13 https://www.nomagic.com/products/magicdraw

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Cameo Systems Modeler[14]. Click the **Editions** tab on the page.
- Cameo Enterprise Architecture[15]. Click the **Editions** tab on the page.

Use the search box to find a specific topic or select one from the list below:

- MagicDraw getting started(see page 20)
- Working with projects(see page 104)
- Diagrams(see page 294)
- Diagramming(see page 482)
- UML elements(see page 628)
- Auxiliary diagram symbols(see page 900)
- Working with model elements(see page 909)
- Tools(see page 1083)
- Model analysis(see page 1192)
- Collaborative modeling(see page 1360)
- Report Wizard(see page 1547)

MagicDraw getting started

This section will help you get started if you are new to MagicDraw. You can get acquainted with the user interface, learn how to register your product, customize the modeling environment and perform other pre-modeling tasks.

Use the search box to find a specific topic or select one from the list below:

- User and product registration(see page 20)
- Understanding the user interface(see page 24)
- Most common shortcut keys(see page 61)
- Assigning shortcut keys(see page 63)
- Customizing environment options(see page 65)
- Customizing and Selecting Perspective(see page 67)
- Configuration files(see page 73)
- Disabling inbound network activity(see page 76)
- Improving performance(see page 77)
- Support(see page 84)
- News Reader(see page 97)
- Creating path variables(see page 99)
- Launchers(see page 101)
- Special characters(see page 103)

User and product registration

14 https://www.nomagic.com/products/cameo-systems-modeler
15 https://www.nomagic.com/products/cameo-enterprise-architecture

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

When you register your product, you create a personal profile at www.nomagic.com[16], where you can access the following professional services:

- Dedicated customer support for all problems and questions.
- Access to early releases and evaluations.
- Access to new versions, patches, and updates.

A modeling tool always checks the registration status at startup after a license has been provided, an unlock key has been added, or a floating server has been selected. If you are not a registered user, the **Registration** dialog will open at startup, prompting you to register.

Do I have to register?

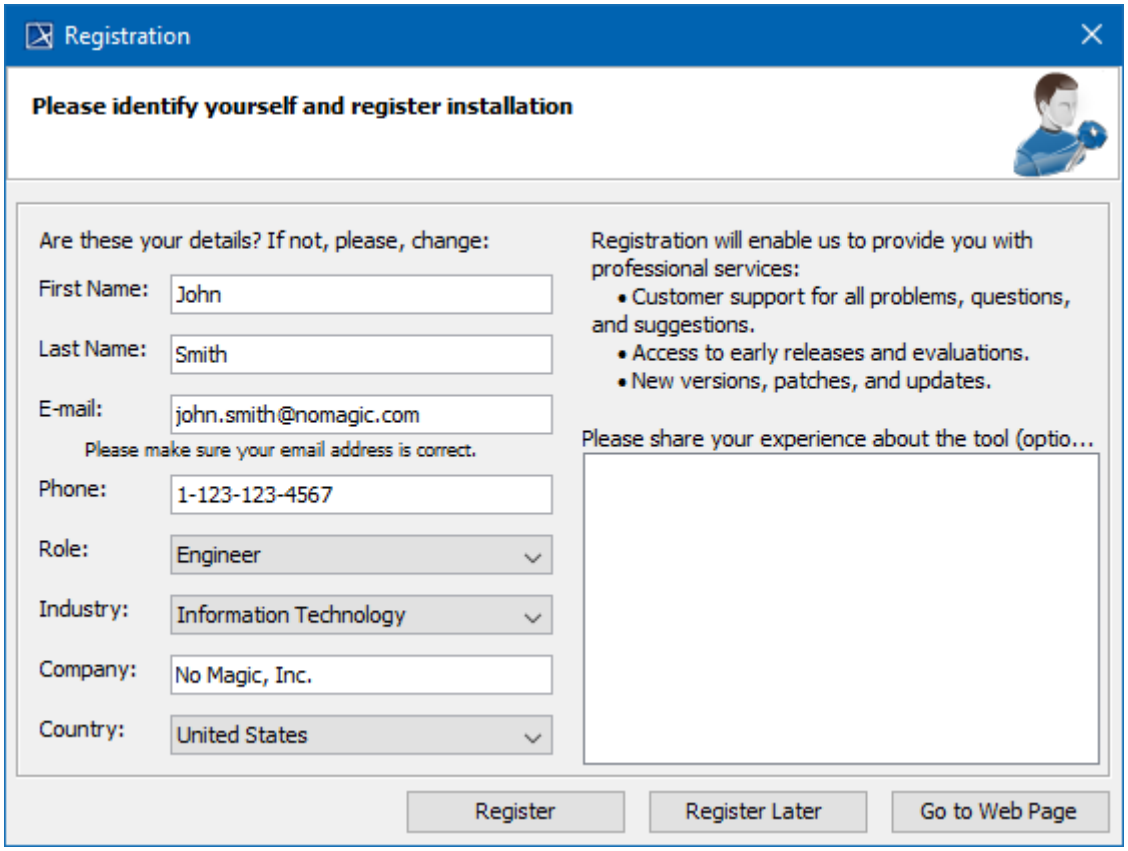
- You are not required to complete the online registration to use your modeling tool. However, registration is recommended in order to receive the benefits available for a registered user.
- Registration for owners of commercial licenses is mandatory!

To register a product

1. Start your modeling tool and click **Help** > **Register** in the main menu.
2. In the **Registration** dialog, provide the requested information. Some information may be prefilled from the key owner profile.
3. Share your experience with the tool (optional).
4. Click **Register** to send data to the server, or click **Register Later** to complete the registration later .

16 https://www.nomagic.com/

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The Registration dialog with sample user information.

If you do not complete the registration process, a message reminding you to register will appear the next time you start your modeling tool. After successfully submitting the online registration form, an e-mail with registration data and confirmation link will be sent to the e-mail address you provided. Click the confirmation link to confirm the registration and create your dedicated area at www.nomagic.com[17].

Privacy information
No Magic, Inc. respects your privacy. We will only use your personal information for communications, management of your online account, and the products you register with your account.

Use the login name and password you received upon confirming your registration to log in to your profile at www.nomagic.com[18], and get dedicated online support, evaluate new products and access other professional services.

After 30 days from registration, or when a new license from the same licensed owner is applied, you will be requested to confirm that you are the one using the software installation at your modeling tool startup. Ensure all information is correct and click the **Confirm** button.

17 http://www.nomagic.com/
18 http://www.nomagic.com

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Registration issues
if you encounter any registration related issues, see Troubleshooting registration issues(see page 23).

Related pages

- MagicDraw getting started(see page 20)
- Understanding the user interface(see page 24)
- Customizing environment options(see page 65)
- Customizing and Selecting Perspective(see page 67)
- Troubleshooting registration issues(see page 23)

Troubleshooting registration issues

Use the following table to solve any registration related issues.

Issue	Solution
Registration is requested at each startup.	The **Registration** dialog opens each time your modeling tool starts until you register the product installation. Click the **Register Later** button to close the **Registration** dialog. We recommended that you register your installation. Registration(see page 20) is simple and allows you to access dedicated online support, evaluate new products and more.
Registration confirmation is requested at each startup.	If you do not click the registration confirmation link, the installation will not be confirmed. You may request a new confirmation email from the registration confirmation message.
No email with a confirmation link is received.	Your mailbox spam or anti-virus filter may filter out the e-mail with a confirmation link. Check your spam folder, or request a new confirmation e-mail by clicking **Help** > **Register** in the main menu.
A message appears saying there is no connection to the registration server.	Check your internet connection. Note that registration is not necessary to use a modeling tool.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Issue	Solution
The **Registration** dialog does not open at modeling tool startup.	This may happen for a few reasons: • You registered less than 30 days ago. • There are network limitations preventing a modeling tool from checking your registration status. Note that registration is not necessary to use a modeling tool.

Related pages

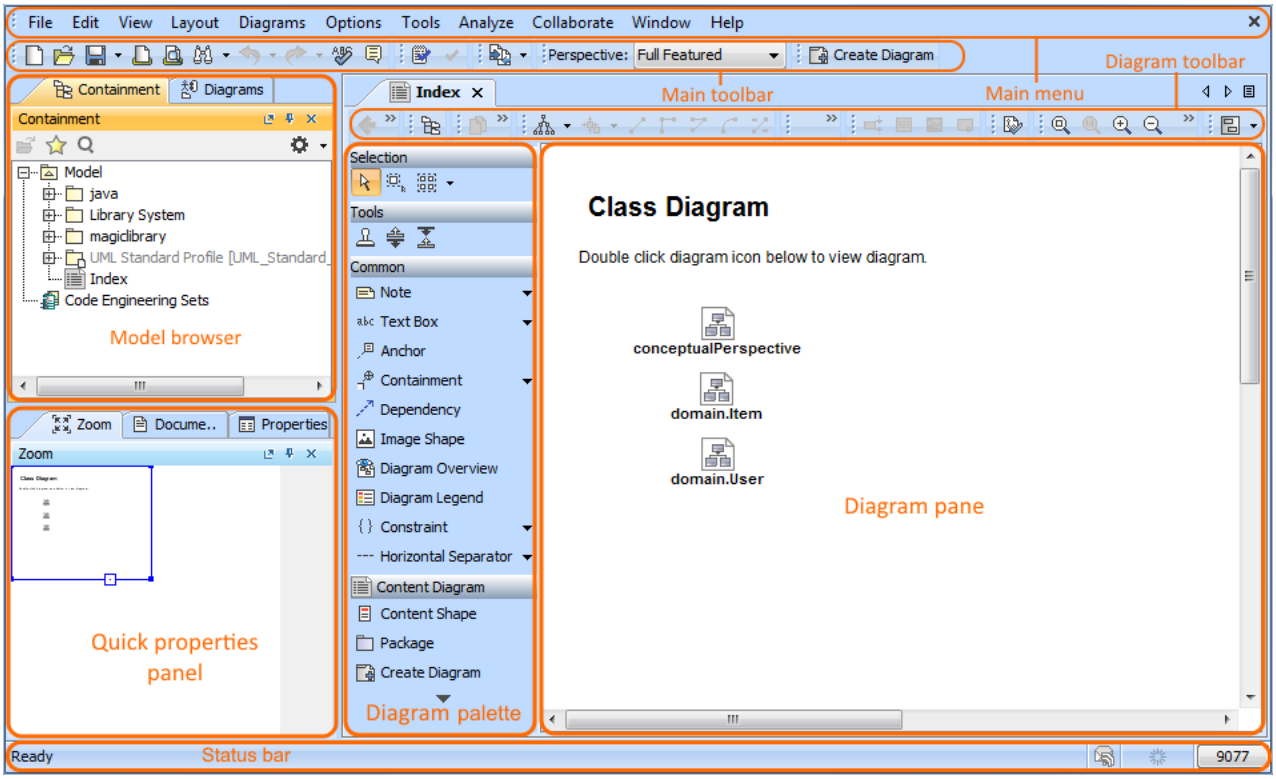
- User and product registration(see page 20)

Understanding the user interface

The user interface is where you will control your software installation. The MagicDraw window contains the following:

- Main menu(see page 25)
- Main toolbars(see page 27)
- Model Browser(see page 29)
- Diagram toolbars(see page 27)
- Diagram palette(see page 2057)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Watch this demo to get familiar with MagicDraw working environment, main concepts and functions.

Sorry, the widget is not supported in this export.
But you can reach it using the following URL:

https://www.youtube.com/watch?v=0B76lH-QtrM&

Related pages

- MagicDraw getting started(see page 20)
  - User and product registration(see page 20)
  - Most common shortcut keys(see page 61)
  - Customizing environment options(see page 65)
  - Customizing and Selecting Perspective(see page 67)
  - Configuration files(see page 73)

Main menu

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

The main menu located at the top of the modeling tool window contains the following elements:

Menu element	Description
File	Contains basic project management related commands, e.g., New Project(see page 106), Open Project(see page 115), and Save Project(see page 113). Among other actions, this menu also allows you to import data from external sources, export a project, or print diagrams(see page 626).
Edit	Use this menu when working with diagrams on the Diagram pane. The **Edit** menu contains the commands for selecting, cutting, copying and pasting(see page 517) symbols or entire diagrams, and finding elements in the current project.
View	Contains commands allowing you to zoom in or zoom out(see page 574) of a diagram, use and customize a diagram grid(see page 576), manage diagram tabs (if you have several open diagrams) and customize the toolbars(see page 27) of your modeling tool.
Layout	Use this menu for managing the layout(see page 579) of the symbols on an active diagram. To enable certain **Layout** menu commands, you must select more than one symbol.
Diagrams	Allows you to create a new diagram(see page 483), access diagram wizards and history, and create a custom diagram type(see page 2045).
Options	Use this menu to specify project options(see page 214) and environment options(see page 65), select or customize perspectives(see page 68), and choose the style for your modeling tool.
Tools	Contains the set of commands allowing you to execute model transformations(see page 1131) for converting models between different modeling domains, add hyperlinks to model elements, generate reports(see page 1734), evaluate expressions(see page 1273) on the actual testing model, generate code for the selected model data, create and organize macros(see page 2700), access the DataHub plugin[19] and more.
Analyze	Use this menu to access the wizards of different types of diagrams(see page 1346), create implementation matrices, analyze and show model dependencies, compare two project versions(see page 1291), validate a project(see page 1306), display paths(see page 513) or related elements(see page 508), and perform other model analysis related actions.
Collaborate	Contains the commands for working with server projects(see page 1360).
Window	Allows you to manage the layout of different windows in your modeling tool.
Help	Among other actions, this menu allows opening online documentation, reporting an issue, updating a modeling tool[20] and plugins[21], managing resources, accessing modeling tool related information.

19 https://docs.nomagic.com/display/CDH190SP4
20 https://docs.nomagic.com/display/NMDOC/Updating+modeling+tool
21 https://docs.nomagic.com/display/NMDOC/Updating+plugins

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Customizable menus
You can choose which of these menus are shown/hidden by customizing modeling tool perspectives(see page 68).

Related pages

- Understanding the user interface(see page 24)
- Toolbars(see page 27)
- Model Browser(see page 29)
- Image Library tab(see page 57)

Toolbars

Toolbars help to speed up your work when performing commonly used tasks. Your modeling tool has two types of toolbars:

- Main toolbars - contain commands for basic project related tasks and are located at the top of the modeling tool window, right under the main menu.
- Diagram toolbars - contain commands for working with diagram symbols and are located at the top of the diagram pane.

To show or hide different toolbars

- In the main menu, click **View**, select either **Main Toolbars** or **Diagram Toolbars**, and in the showed menu select/clear the check boxes of the toolbars that you want to show/hide.
- Right-click the desired toolbar area and in the shortcut menu select/clear the check boxes of the toolbars that you want to show/hide.

Customizing toolbars

You can customize existing toolbars by adding or removing buttons, and also create new custom toolbars.

To add a new toolbar

---

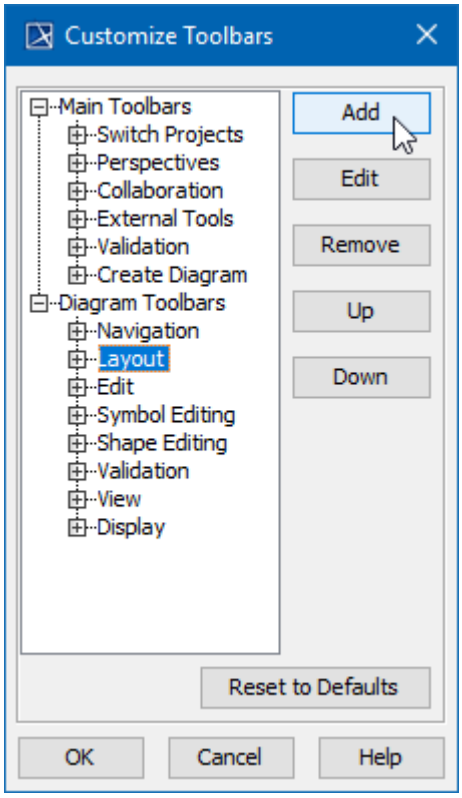
1. Do one of the following:
    - In the main menu, select **View** > **Main Toolbars**/**Diagram Toolbars** > **Customize**.
    - Right-click the toolbar area, and in the shortcut menu select **Customize**.
2. In the **Customize Toolbars** dialog, select the desired toolbar group and click the **Add** > **New Toolbar**.
3. Enter the name of the new toolbar and click **OK**.

After completing the above steps, a new toolbar is created in the selected toolbar group, and you can choose to show/hide it in the shortcut menu of the respective toolbar area. Note that the newly created toolbar does not have any buttons, so you need to add them after creating it.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To add a new button to a toolbar

1. Do one of the following:
    - In the main menu, select **View** > **Main Toolbars**/**Diagram Toolbars** > **Customize**.
    - Right-click the toolbar area, and in the shortcut menu select **Customize**.
2. In the **Customize Toolbars** dialog, select the desired toolbar and click the **Add** > **Button**.



3. In the **Add Button** dialog, select the desired button or command and click **OK**.
4. Click **OK** one more time to close the **Customize Toolbars** dialog.

A new button is added to the selected toolbar. To add more than one button, repeat steps 2 and 3 as many times as you need.

To remove a toolbar or button

1. Do one of the following:
    - In the main menu, select **View** > **Main Toolbars**/**Diagram Toolbars** > **Customize**.
    - Right-click the toolbar area, and in the shortcut menu select **Customize**.
2. In the **Customize Toolbars** dialog, select the toolbar or button you want to remove and click the **Remove** button.

After completing the above steps, the selected toolbar or button is removed from the respective

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

toolbar area or toolbar. Toolbars that have been removed do not appear in the menu for selecting which toolbars you want to show.

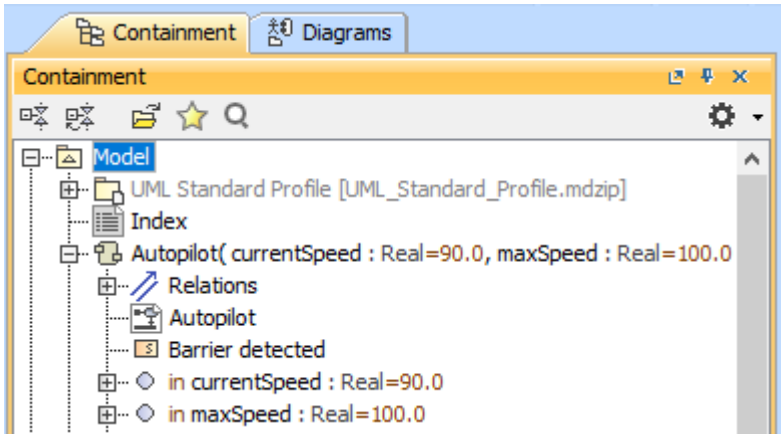
> **Restoring default toolbars**
> To undo all the actions for customizing toolbars, open the **Customize Toolbars** dialog, as described in step 1 above, and click the **Reset to Defaults** button.

Related pages

- Understanding the user interface(see page 24)
- Main menu(see page 25)
- Model Browser(see page 29)
- Image Library tab(see page 57)

Model Browser

Model Browser provides a visual representation of the hierarchy of your model elements.



The items in the Model Browser hierarchy can be:

- Compressed – a plus sign next to an icon indicates that the icon is compressed. This is the default setting. It means that the element contains other model elements. Click the plus sign to expand the icon and view its subordinate items.
- Expanded – a minus sign next to an icon indicates that the icon is fully expanded. Click the minus sign to collapse the item.

If there is no plus or minus sign next to an icon, it does not contain other model elements.

The Model Browser is a hierarchical navigation tool that allows you to manage your model data, including packages, components, classes, all UML diagrams, extension mechanisms, and other data.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

The Model Browser may be used as an alternative tool to the MagicDraw menus and toolbars. It is easier to work with project diagrams and data elements using the Model Browser.

You can perform the following operations in the Model Browser:

- Create and specify model elements.
- Copy, cut and paste model elements.
- Delete model elements.
- Drag model elements to the Diagram pane and inside the Model Browser.
- Drag data in the Code engineering sets (you may create data in the root package *Model* branch, drag it to the Code Engineering sets, and then the round trip object is created automatically).
- View the hierarchy of all model elements.
- Create a symbol for the selected model element in the current diagram.
- Manage diagrams.
- Manage extension mechanisms, such as constraints, stereotypes, and tagged values.
- Java reverse of a class from the classpath.
- Adjust the code engineering sets.
- Generate code for particular sets.
- Filter visible items by any model type (for example, class, package, operation, component, state, and others - for both views and dates).
- Sort the visible items of the selected model element.
- Sort all model elements.
- Display search results.
- Display locked elements in the server project.

The Model Browser window has the following tabs**:**

- **Containment** tab groups data into the logical sets.
- **Inheritance** tab represents the model hierarchy of the project.
- **Diagrams** tab groups diagrams according to the diagrams type or shows them as a list.
- **Model Extensions** tab represents all predefined and created constraints and stereotypes.
- **Search Results** tab displays search results.

> The **Search Results** tab is not displayed by default. It appears after you perform searching.

- **Lock View** tab represents locked elements of the server project.

> The **Lock View** tab is not displayed by default. It can appear if you are working on a server project. For more information, see Representation of locked elements(see page 1403).

Related pages

- Using the Model Browser(see page 31)
- Containment tab(see page 32)
- Structure tab(see page 37)
- Inheritance tab(see page 41)
- Diagrams tab(see page 44)
- Model Extensions tab(see page 46)
- Search Results tab(see page 47)
- Lock View tab(see page 49)
- Working with model elements in the Model Browser(see page 51)

Using the Model Browser

Model Browser is displayed when at least one project is open. By default, Model Browser is placed at the left side of the main window. It is possible to move the Model Browser to any place on the MagicDraw window. Also, all tabs can be viewed separately and you can set up the Model Browser according to your needs by hiding the desired tabs.

To change the size of either part of the Model Browser

- Drag the bar that separates the two parts.

To change the Model Browser position

- Using the **Options** menu
  a. From the **Options** menu, select **Environment**. The **Environment Options** dialog opens.
  b. Click on the **Browser** option group.
  c. In the Browser environment options pane, change the **Browser Position** property to **Right** or **Left**.
- Using Toggle
  In the upper-right corner of the selected tab in the Model Browser, click the Toggle auto-hide or Toggle floating button.

  This action changes the layout of the MagicDraw interface.

To close or reopen the desired tab of the Model Browser

- Using the **Window** menu, choose the Model Browser tab you want to close or open.

To reset all Model Browser tabs to the default position

- From the **Window** menu, choose the **Reset Windows Configuration** command.

To sort items in the Model Browser alphabetically

---

1. From the **Options** menu, select **Environment.** The **Environment Options** dialog opens.
2. Click the **Browser** option group.
3. In the **Browser** options pane, set the **Sort Always** value to *true* (the default value is *true).*

Related pages

- Model Browser(see page 29)

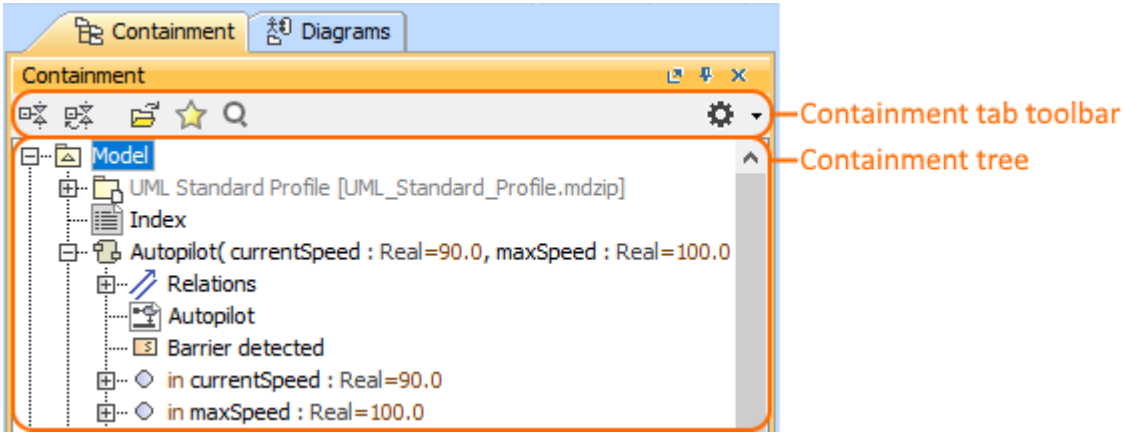
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Containment tab

On this page

- The root package Model (see page 33)
- Code engineering sets(see page 34)
- Containment tab toolbar(see page 34)
- Working with model elements in the Containment tree(see page 35)

The Containment tab displays model data grouped in logical sets. Using the Containment tab, you can quickly access each model element, change a model element specification, or create a new model element.



To open the Containment tab, do one of the following

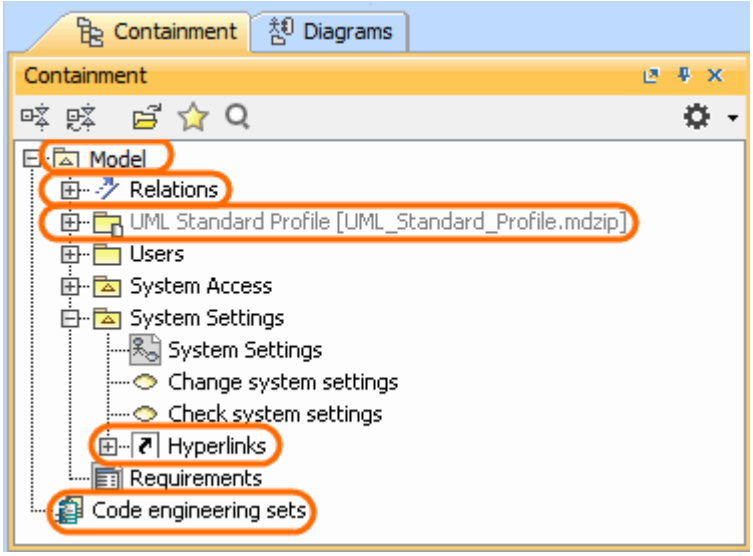
- At the top of the Model Browser, click the **Containment** tab.
- If the Containment tab is hidden, from the **Window** menu, select **Containment**.

> You can change some Containment tab properties in the **Environment Options** dialog(see page 65) the **Browser** options group.

The Containment tab consists of the following two components:

- Containment tab toolbar(see page 0)
- Containment tree(see page 0)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The Containment tree includes

- The root package *Model*(see page 0):

  The Relations branch
  The UML Standard Profile
  The Hyperlinks branch

- Code engineering sets(see page 0)

The root package *Model*

The root package *Model* in the Containment tree is the top model element wherein the entire model structure is created. The root package *Model* represents the model element that is named *Model*. All model elements are stored in the root package *Model*. You can create the structure of your project by creating packages in the root package *Model*. This helps to distribute model elements into the logical groups.

Along with model elements that are created by the user, the root package *Model* may contain the following default items:

- The **Relations** branch. This branch appears when at least one relationship is created in the project. The *Relations* branch collects all relationships from the particular owner. For example, if the *Relations* branch is in the root package *Model*, that means, all relationships from the root package *Model* are listed in the *Relations* branch owned by the root package *Model*. The *Relations* branch can exist in other elements which may have inner elements and relations among these elements. For example, in packages, classes, and others.
- The **UML Standard Profile** package. This is a profile which is included to every MagicDraw project. It contains a list of stereotypes, data types, and other elements from the UML 2 metamodel. By default, the *UML Standard Profile* is hidden. To display the *UML Standard Profile*, click the **Show Auxiliary Resources** button. There can be more standard profiles (such as, SysML, UPDM, Java, and others).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

> We highly recommend not to modify our provided standard profiles and libraries as it could cause problems on version updates, plugins, core MagicDraw tool malfunctions, and model corruptions.

- The **File View** package. This package appears when at least one code engineering set is created. The *File View* package contains components that are created on code reverse and represents source files.
- The **Hyperlinks** branch. This branch appears when at least one hyperlink is added to the particular element. The Hyperlinks branch contains the list of hyperlinks to the file, element/symbol, or web page.

Code engineering sets

> Code engineering is available in Professional, Architect, and Enterprise editions.

Code engineering sets can be considered a gateway between your source code and model data. Using these sets, you can perform Java, C++, C#, IDL, CIL, CORBA IDL, EJB 2.0, and WSDL round-trip code engineering, that is, code generation and reverse engineering.

To display Code engineering sets

- In the Containment tab toolbar, click the **Options** button, then click to select the **Show Code Engineering Sets** check box.

In Code engineering sets branch, you can create a new code engineering set, generate code, and perform other actions.

Containment tab toolbar

The Containment tab toolbar contains the following buttons:

Button	Tooltip text	Description
	Collapse All	Collapse all of the currently elements in the Containment tree.
	Collapse Selected Recursively	Collapses only selected node and all subnodes in that node.
	Open in New Tab	Opens the contents of the selected element in a new tab of the Model Browser.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

	Favorites	Opens the Favorites menu for choosing to do one of the following: • Add a selected element to favorites • Open the dialog for managing favorites • Navigate to a desired favorite in the Containment tree
	Quick Find	Opens the **Quick Find** dialog.
	Options	Click to open the Options menu and then select: • **Filter** - opens the **Items Filter** dialog wherein you can select element types to be visible in the Containment tree. • **Show Auxiliary Resources** - shows or hides modules and profiles that are used in your project. • **Show Applied Stereotypes** - shows or hides the applied stereotypes next to the element name in the Containment tree. • **Show Full Types** - shows or hides the full types of typed elements (for example, operations, attributes, relationships, and other) in the Containment tree. • **Show Code Engineering Sets** - shows or hides the code engineering sets.

The same actions you may perform using the shortcut menu of the Containment tree.

Working with model elements in the Containment tree

You can work with elements in the Containment tree using the commands available from the selected element's shortcut menu.

> Particular actions are available for the particular elements.

The shortcut menu contains the following commands:

Command	Function
Create Element	Creates a new element inside the selected element. This command is available if the selected element can be edited.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Create Diagram	Creates a new diagram inside the selected element. This command is available if the selected element can be edited.
Create Relation > **Outgoing / Incoming**	Creates a new relation to or from the selected element.
Open in New Tree	In the Model Browser, opens a new tab for the selected element.
Select in Structure Tree	In the Model Browser, the Structure tab opens and the structured classifier is selected. This command is available for structured classifiers (SysML).
Open in New Structure Tree	In the Model Browser, opens a new Structure tab for the selected structured classifier. This command is available for structured classifiers (SysML).
Specification	Opens the element Specification window wherein you can specify various element properties.
Go To	Opens or selects the item associated with the selected element. The item may be diagram, model element, or hyperlink.
Refactor	Converts the selected element to the element indicated by a user.
Related Elements	Opens a list of functions available for the selected element.
Tools	Opens a list with tools that are available for the selected element. Depends on the selected element.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Stereotype	Opens the drop down list, wherein you can apply an existing stereotype, or create a new one.
Find	Opens the **Find** dialog.
Apply Profiles	Opens the list of the available to apply profiles. • This command is available in the package, model, and system boundary shortcut menu. • To see this command you have to expand the shortcut menu, that is, click on the little arrow at the bottom of the shortcut menu.
Modules	Opens the list with the commands for working with modules. • This command is available in the package, model, and system boundary shortcut menu. • To display this command in the shortcut menu you have to expand the shortcut menu, that is, click on the little arrow at the bottom of the shortcut menu.
Generate Code Framework	Generates code for the selected item. Opens the Message Window with the information.
Check Syntax	Checks syntax in the model according to the default code engineering language. Opens the Message Window with the information.
Generate Report	Opens the list of the available report templates.
Reverse from Classpath	This command is available in the class shortcut menu.

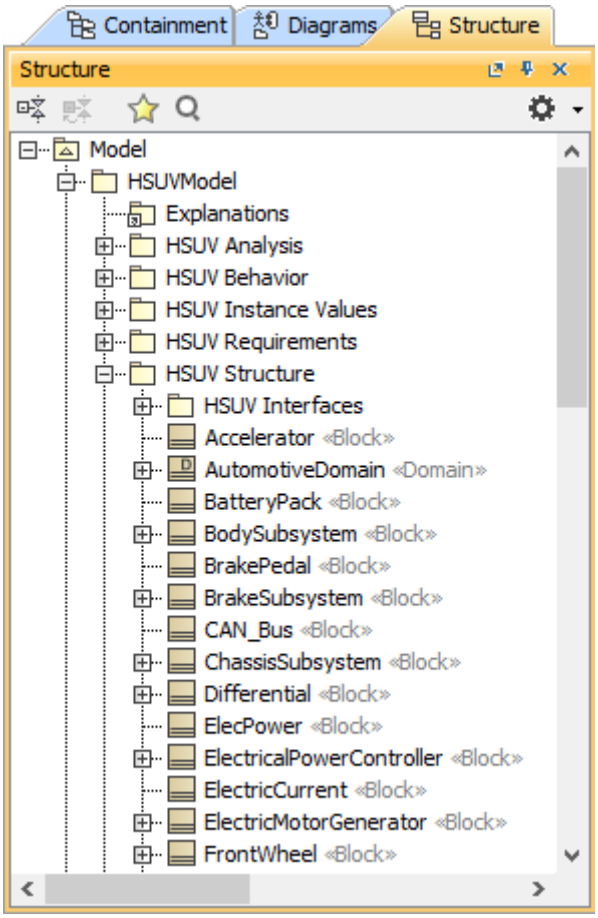
Related pages

- Model Browser(see page 29)

Structure tab

This tab will appear in the Window menu only if the SysML plugin is installed.

The Structure browser allows you to browse for deep nested structures of the structured classifier in your model. For example see the image below, the node: **diameter : mm** represents the property: **diameter : mm** of the classifier: **Wheel** and also the property: **w : Wheel** is the property of the classifier: **WheelAssembly**.



The Structure tab in the Model Browser.

Opening the Structure tab

To open the Structure tab

---

1. From the main menu, select **Window** > **Structure**.

The property nodes, which are shown inside the property node (the parent property node), are the properties of the classifier that type the parent property node.

In the Structure tab, you can open any selected element in a new tree.

To open in a new tree

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Right-click the selected element, and in the shortcut menu, click **Open in New Tree**.

There are two display options in the Structure tab:

- Display as Plain List(see page 39)
- Display Inherited Structure(see page 39)

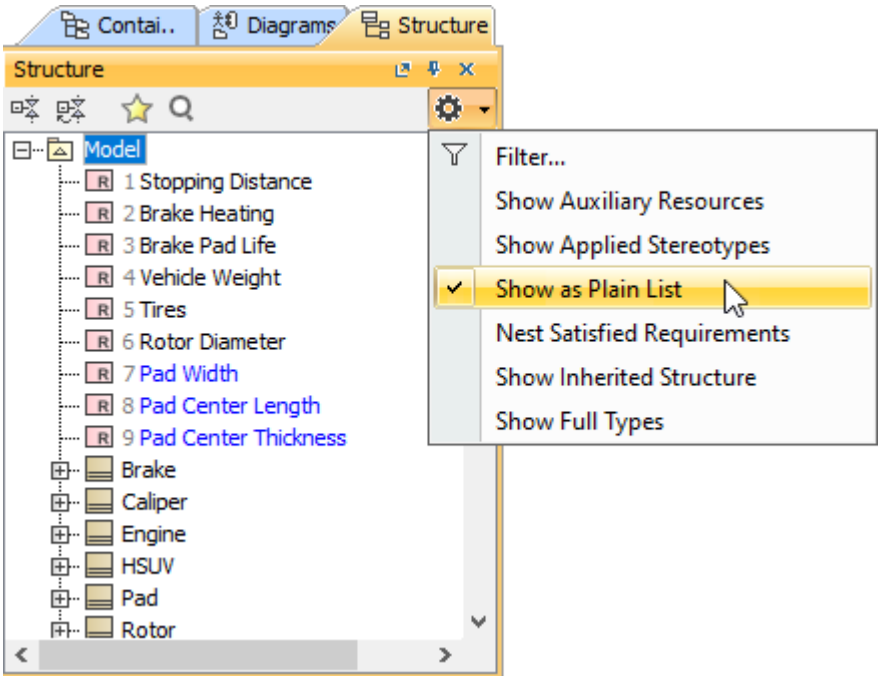
Display as Plain List option

The classifiers of structure in your model is normally displayed in a Package, Model, or Profile hierarchy. Use the **Display as Plain List** option to show all classifiers of the structure in the model in the same level without consideration of their owner. When you select the **Display in Plain List** option, the classifiers are sorted by their name.

To display elements in the Structure tree as plain list

---

1. Click on the right of the Structure Browser.
2. Select the **Display as Plain List** option.



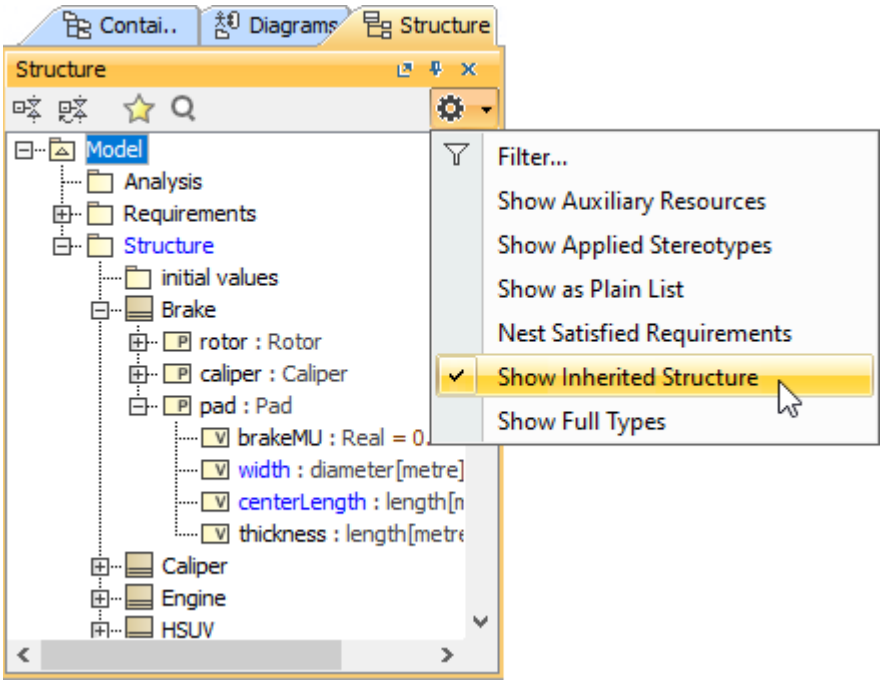
Displaying Inherited Structure option

The Structure tab can show the properties that are inherited from the generalization classifier.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To display inherited structures from the generalization classifier in the Structure tree

1. Click on the right of the Structure Browser.
2. Select the **Show Inherited Structure** option.



Nesting Satisfied Requirements option

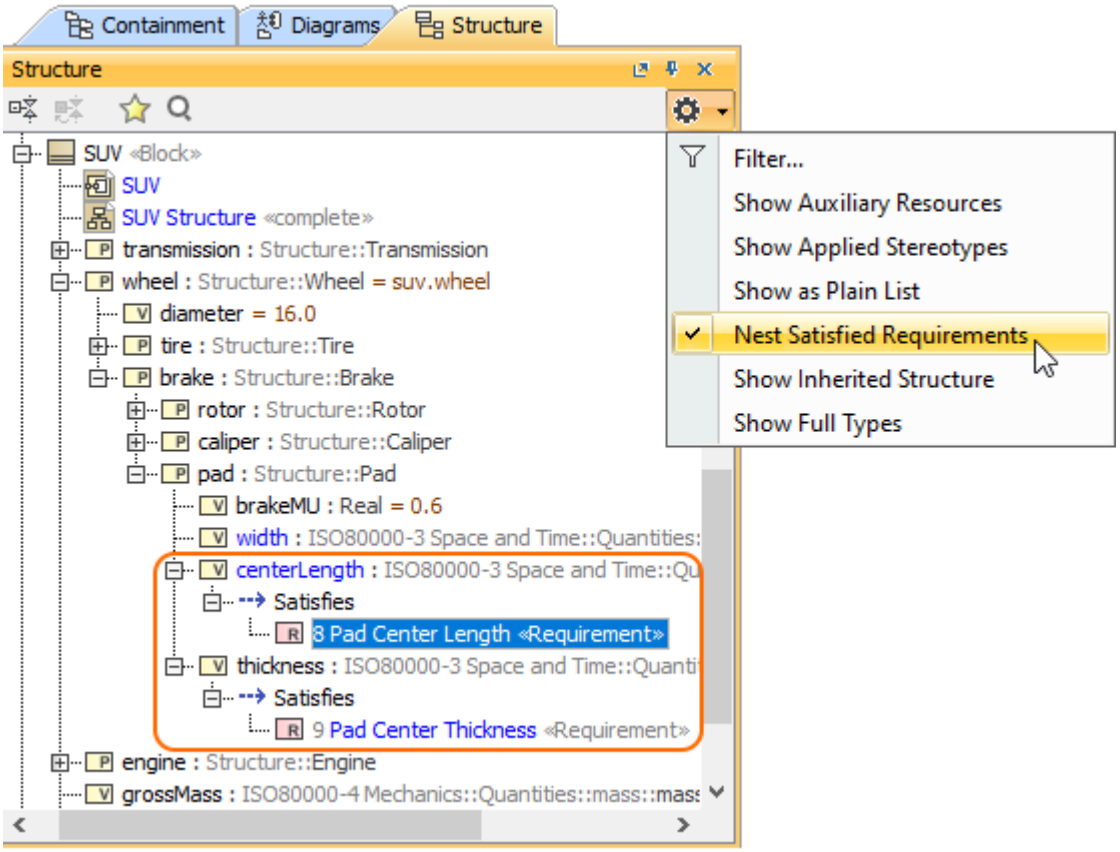
The Structure Tab can also show requirements nested within elements that satisfy them.

> If the relationship is contextualized, its context is considered as well. Learn more about contextual relationships >>[22]

To nest requirements under the satisfying elements

1. Click on the right of the Structure Browser.
2. Select the **Nest Satisfied Requirements** option.

22 https://docs.nomagic.com/display/SYSMLP190SP4/Contextual+Relationships



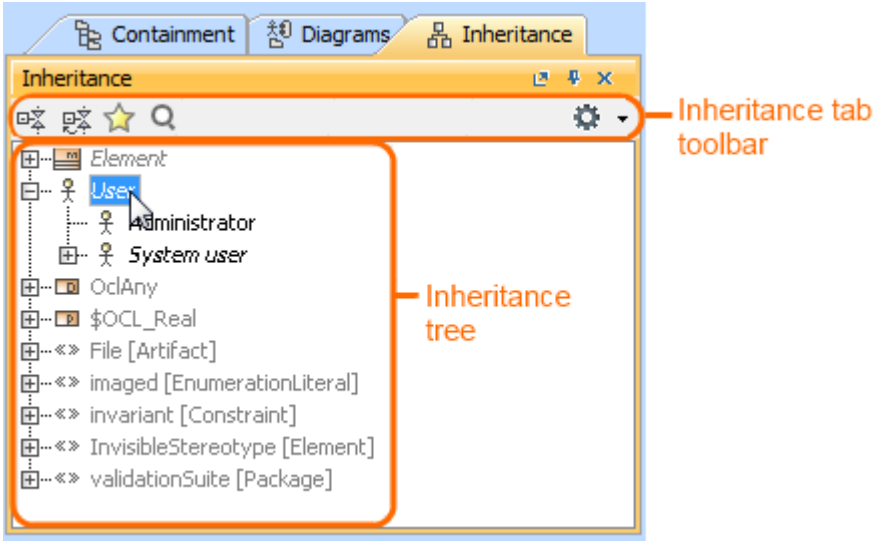
Related pages

- Model Browser(see page 29)

Inheritance tab

The Inheritance tab represents classifiers, packages, data types, and stereotypes hierarchy within your project. Inheritance according to the UML Specification is shown using the generalization relationship.

> ⓘ If a model element has no generalization relationship, it will not be represented in the Inheritance tab.



To open the Inheritance tab, do one of the following

- At the top of the Model Browser, click the **Inheritance** tab.
- If the **Inheritance** tab is hidden, from the **Window** menu, select **Inheritance**.

> You can change some Inheritance tab properties in the **Environment Options** dialog(see page 65) the **Browser** options group.

The Inheritance tab consists of the following two components:

- Inheritance tab toolbar
- Inheritance tree

Inheritance tab toolbar

The Inheritance tab toolbar contains the following buttons

Button	Tooltip text	Description
	Collapse All	Collapse all of the currently elements in the Containment tree.
	Collapse Selected Recursively	Collapses only selected node and all subnodes in that node.

	Favorites	Opens the Favorites menu for choosing to do one of the following: • Add a selected element to favorites • Open the dialog for managing favorites • Navigate to a desired favorite in the Containment tree
	Quick Find	Opens the **Quick Find** dialog.
	Options	Click to open the Options menu and then select: • **Show Applied Stereotypes** - shows or hides the applied stereotypes next to the element name in the Containment tree. • **Show Only Hierarchies** - at the top of the hierarchy shows the parent element. • **Invert Tree** - Inverts the tree hierarchy - shows the child element at the top of the hierarchy

The same actions you may perform using the shortcut menu of the Inheritance tree.

Working with model elements in the Inheritance tree

You can work with elements in the Inheritance tree using the commands available from the selected element's shortcut menu. The shortcut menu contains the following commands:

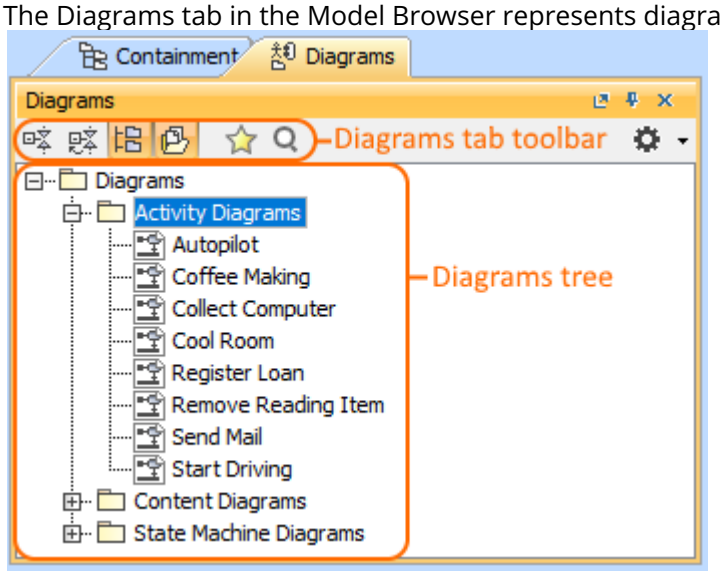
Command	Function
New Specific Classifier	Creates a new element that is connected with the current element with the generalization relationship.
Select in Containment Tree	Opens the Containment tab and selects the current element in the Containment tree.

Related pages

- Model Browser(see page 29)

Diagrams tab

The Diagrams tab in the Model Browser represents diagrams created in your project.



Just diagrams grouped by types that are used in your project are listed in the **Diagrams** tab in the Model
Browser, not all diagram types as it was in earlier versions.

To open the Diagrams tab, do one of the following

- At the top of the Model Browser, click the **Diagrams** tab.
- If the Diagrams tab is hidden, from the **Window** menu, select **Diagrams**.

> You can change some Diagrams tab properties in the **Environment Options** dialog(see page 65) the **Browser** options group.

The Diagrams tab consists of the following two components:

- Diagrams tab toolbar
- Diagrams tree

Diagrams tab toolbar

The Diagrams tab toolbar contains the following button:

Button	Tooltip text	Description
	Collapse All	Collapse all of the currently elements in the Containment tree.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

	Collapse Selected Recursively	Collapses only selected node and all subnodes in that node.
	Group by Diagram Type	Groups diagrams into packages by diagram types.
	Group by Diagram Name	Groups diagrams into packages by the first symbol of their name, when the model contains more than 99 diagrams.
	Favorites	Opens the Favorites menu for choosing to do one of the following: • Add a selected element to favorites • Open the dialog for managing favorites • Navigate to a desired favorite in the Containment tree
	Quick Find	Opens the **Quick Find** dialog.

The same action you may perform using the shortcut menu of the Diagrams tree.

Working with model elements in the Diagrams tree

You can work with diagrams in the Diagrams tree using the commands available from the selected diagram's shortcut menu. The shortcut menu contains the following commands:

Command	**Function**
Open	Opens the selected diagram.
Print	Opens the **Print** dialog.
Group by Diagram Type	Groups diagrams into the packages according to the diagram type.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

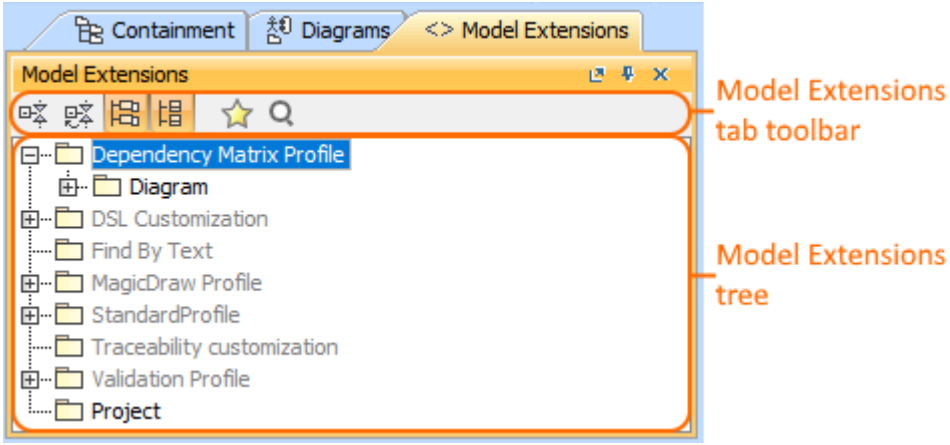
Select in Containment Tree	Opens the Containment tab and selects the current element in the Containment tree.

Related pages

- Model Browser(see page 29)

Model Extensions tab

The Model Extensions tab contains all Stereotypes that are predefined and created manually in the project. In this tree, you can create, review, copy/paste, and delete extension mechanisms.



To open the Model Extensions tab, do one of the following

- At the top of the Model Browser, click the **Model Extensions** tab.
- If the Model Extensions tab is hidden, from the **Window** menu, select **Model Extensions**

> You can change some Model Extensions tab properties in the **Environment Options** dialog(see page 65) the **Browser** options group.

Model Extensions tab consists of the following two components:

- Model Extensions tab toolbar
- Model Extensions tree

Model Extensions tab toolbar

Model Extensions tab toolbar contains the following buttons:

Button	Tooltip text	Description

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

	Collapse All	Collapse all of the currently elements in the Containment tree.
	Collapse Selected Recursively	Collapses only selected node and all subnodes in that node.
	Group by Profiles	Groups extensions by the profiles.
	Group by Metaclasses	Groups extensions by the metaclasses.
	Favorites	Opens the Favorites menu for choosing to do one of the following: • Add a selected element to favorites • Open the dialog for managing favorites • Navigate to a desired favorite in the Containment tree
	Quick Find	Opens the **Quick Find** dialog.

The same actions you may perform using the shortcut menu of the Model Extensions tree.

Working with model elements in the Model Extensions tree

You can work with elements in the Model Extension tree using the commands available from the selected element's shortcut menu.

> For more information about commands available from the shortcut menu in the Model Browser, see Working with model elements in the Model Browser(see page 51).

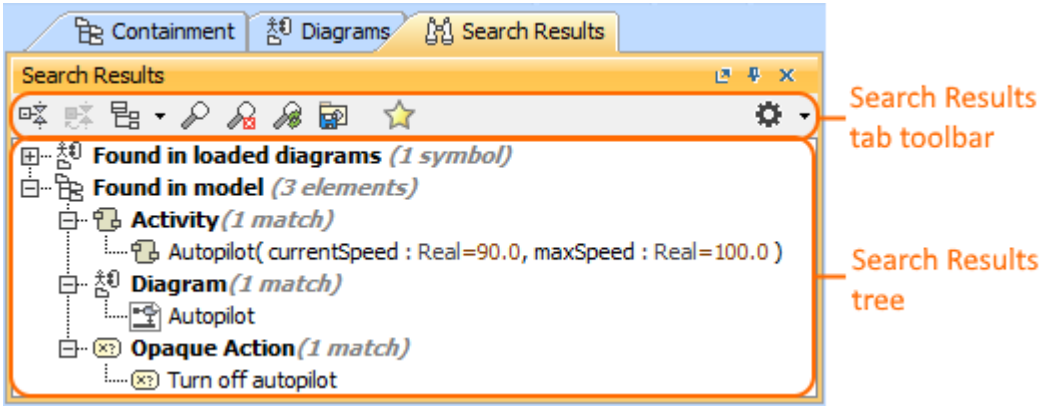
Related pages

- Model Browser(see page 29)

Search Results tab

The Search Results tab is used to display search results. This tab opens only after performing the search(see page 217) by pressing Ctrl+Shift+F, or by clicking **Edit** > **Find**.

The tab is not displayed when performing a Quick Find(see page 223).



The **Search Results** tab consists of two components:

- Search Results tab toolbar (see page 0)
- Search Results tree(see page 0)

Search Results tab toolbar

The Search Results tab toolbars contains the following buttons:

Button	Tool tip text	Description
	Collapse All	Collapse all of the currently elements in the Containment tree.
	Collapse Selected Recursively	Collapses only selected node and all subnodes in that node.
	Find...	Opens the **Find** dialog.
	Clear Results	Clears the results of the previous search.
	Rerun Search	Reruns a search with the already specified search criteria. Search results are updated according to changes done in the model.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

	Save Search Results	Saves as a smart package either the search options or the search results.
	Favorites	Opens the Favorites menu for choosing to do one of the following: • Add a selected element to favorites • Open the dialog for managing favorites • Navigate to a desired favorite in the Containment tree
	Options	Click to open the Options menu and then select: • **Show Applied Stereotypes** - shows or hides the applied stereotypes next to the element name in the Containment tree. • **Show Full Types** - shows or hides the full types of typed elements (for example, operations, attributes, relationships, and other) in the Containment tree.

> The same actions you may perform using the shortcut menu of the Search Results tree. For more information about commands available from the shortcut menu in the Model Browser, see Working with model elements in the Model Browser(see page 51).

Working with model elements in the Search Results tree

The search results in the **Search Results** tree are displayed in the following two packages:

- **Found in loaded diagrams** – symbols that have been found in the diagram(s) are displayed in this package.
- **Found in model** – elements that have been found in the model data are displayed in this package.

Related pages

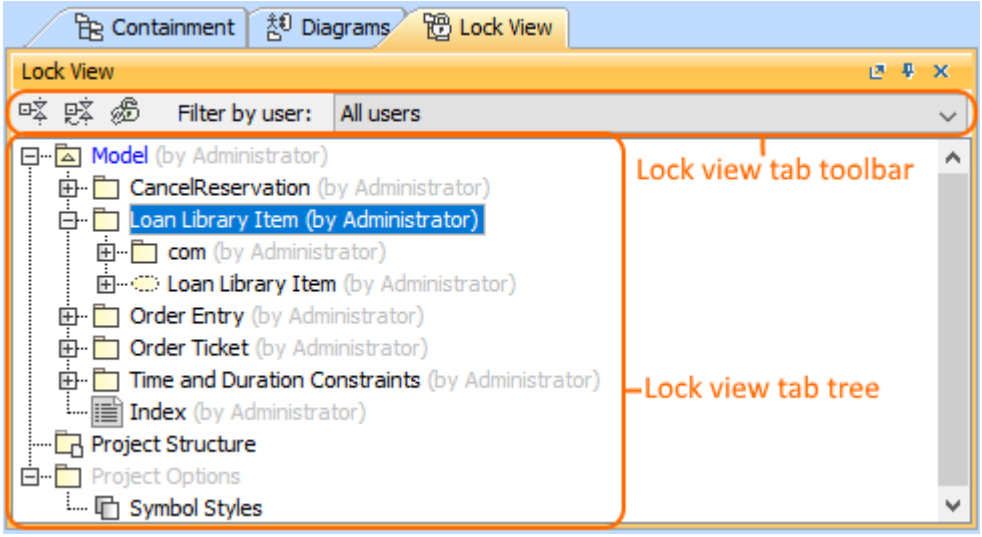
- Model Browser(see page 29)

Lock View tab

> The Lock View tab is displayed only when working with server projects(see page 1360).

The Lock View tab is used to display what was locked in the project (locked elements, project structure, symbol styles) and by which users. Next to the locked element, you can see the name of the user who has locked it. Using the **Filter by user** list in the Lock View tab toolbar, you can see the elements, locked by the particular user.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



To open the Lock View tab, do one of the following

- At the top of the Model Browser, click the **Lock View** tab.
- If the Containment tab is hidden, from the **Window** menu, select **Lock View**.

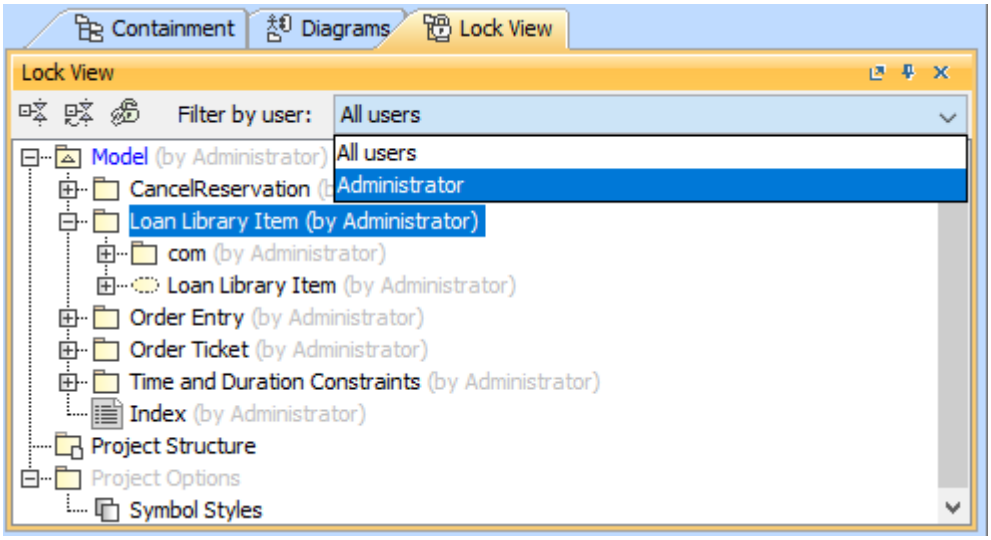
The Locked Elements tab consists of the following two components:

- Lock View tab toolbar
- Lock View tree

Lock View tab toolbar

The Lock View tab toolbar contains the following:

- **Update Lock Information** button
- **Filter by user** list
- **Collapse All** button
- **Collapse Selected Recursively** button



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Using the **Filter by user** list you can select to display only the elements locked by the particular user.

Working with model elements in the Lock View tree

You can work with elements in the Lock View tree using the commands available from the selected element's shortcut menu. The shortcut menu contains the following commands:

Command	Function
Lock	Locks or unlocks the particular element for edit.
Select in Containment Tree	Opens the Containment tab and selects the current element in the Containment tree.
Select in Inheritance Tree	Opens the Inheritance tab and selects the current element in the Inheritance tree.

Related pages

- Model Browser(see page 29)

Working with model elements in the Model Browser

In the Model Browser you can do the various actions with model elements, such as, rename, cut, copy, delete, and others. These actions are available from the particular model element shortcut menu. See the list of the concrete actions in the following table.

Command	Function
Rename	Switches the element name into the edit mode. You can also press F2 for this.
Copy	Copies the selected element.
Copy URL	Copies a model element URL to a clipboard.
Cut	Cuts the selected element.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Paste	Pastes the copied element.
Delete	Deletes the selected element.

To copy/cut and paste the selected model element between different projects

1. Select the model element in the Model Browser.
2. From the model element shortcut menu, select **Copy** or **Cut**.
3. Open the project to which you want to paste the selected element(s).
4. Select the container, where you wish to paste the element(s).
5. From the container's shortcut menu, select **Paste**.

To change the location of an element in the Model Browser

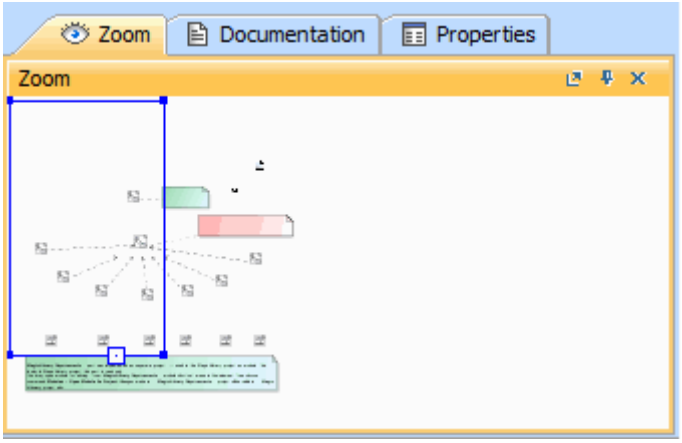
1. Select the element in the Model Browser.
2. Drag it to the new location (possible owner).

Related pages

- Model Browser(see page 29)

Quick properties panel

The **Quick properties panel** has three tabs, as shown in the figure below:



Quick properties panel

- The **Zoom** tab zooms in on the current diagram.

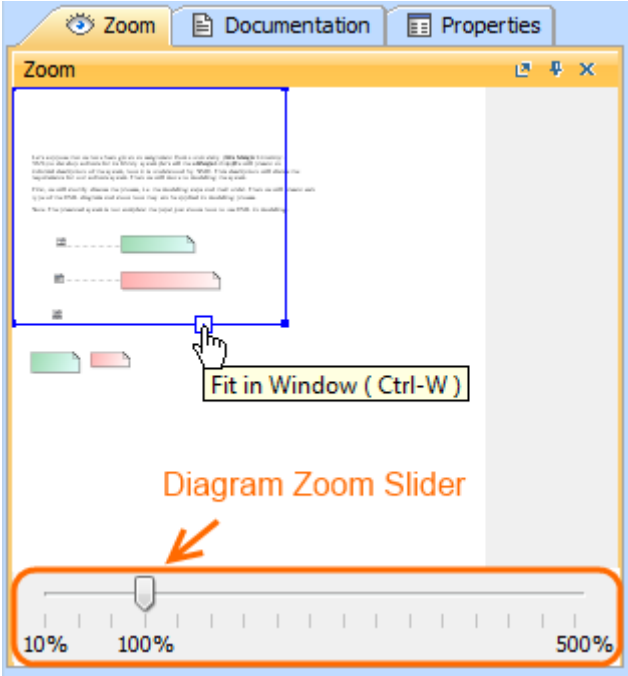
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- The **Documentation** tab shows documentation associated with the selected item.
- The **Properties** tab shows the specification and symbol properties of the selected element or diagram.

Related pages

Zoom tab

The **Zoom** tab enables you to preview a selected diagram. To select a diagram, go to the **Containment** or **Diagram** tab. You can also use the **Zoom** tab to zoom in or out of the active diagram.



The Zoom tab in the quick properties panel.

To open the **Zoom** tab, do one of the following

- In the Model Browser, click the **Zoom** tab.
- On the **Window** menu, click **Zoom**.

To zoom in or out of the active diagram

- In the Zoom tab, drag corners of the blue square.

To quickly access any part of the diagram

- In the **Zoom** tab, drag the blue square to the desired part of the diagram.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

The selected diagram part will be displayed on the diagram pane.

To fit the diagram to the window

- In the **Zoom** tab, on the blue square , click the **Fit in Window** button or press **CTRL+W**.

To show the **Diagram Zoom Slider**

1. From the **Options** menu, select **Environment**. The **Environment Options** window opens (see the figure above).
2. Select the **Browser** options group.
3. In the **Browser** category, select the **Show Diagram Zoom Slider** check box.

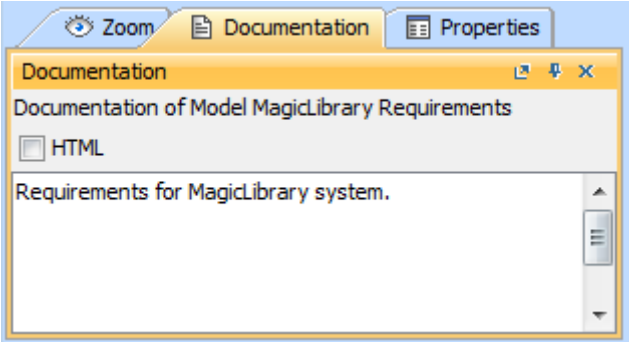
The **Diagram Zoom Slider** is displayed in the **Zoom** tab (see the figure above).

Related pages

- Model Browser(see page 29)

Documentation tab

The **Documentation** tab shows information associated with a selected model element in the Model Browser or on a diagram pane.



The Documentation tab in the quick properties panel.

To open the **Documentation** tab, do one of the following

- In the Model Browser, click the **Documentation** tab.
- On the **Window** menu, click **Documentation**.

If there is no documentation for the selected element, the text area is empty.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To write documentation for the selected model element

- In the **Documentation** tab, click the text area and type the text.

To turn on the HTML text

- In the **Documentation** tab, click the **HTML** check box.

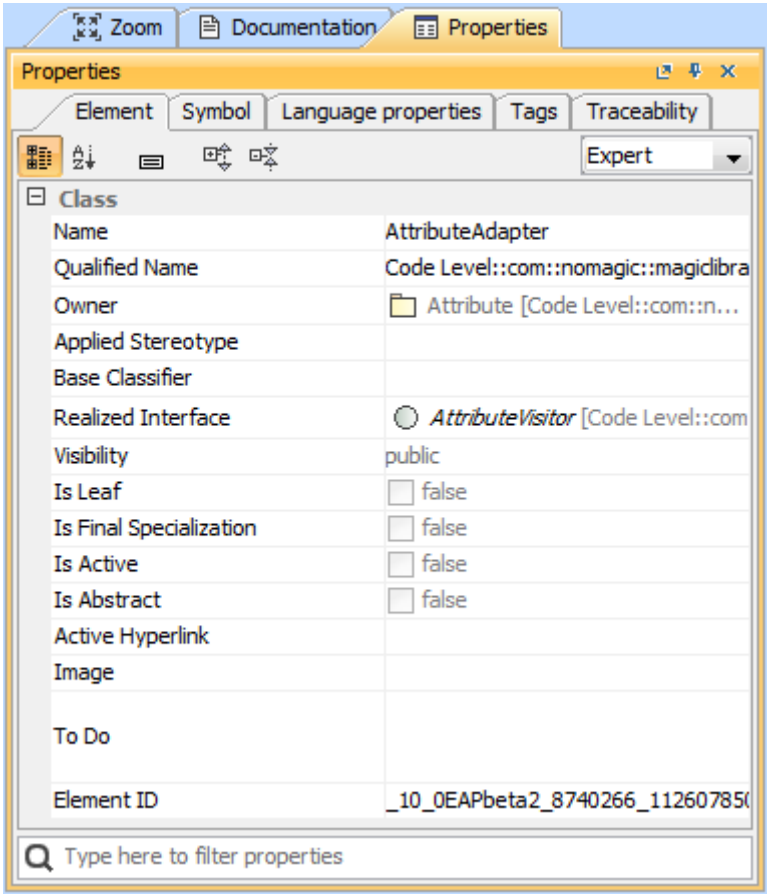
The HTML toolbar opens and you can type the HTML text.

Related pages

- Model Browser(see page 29)

Properties tab

The **Properties** tab at the bottom of the Model Browser allows you to quickly access basic information about a selected element or diagram.



An example of the Properties tab in the Quick properties panel

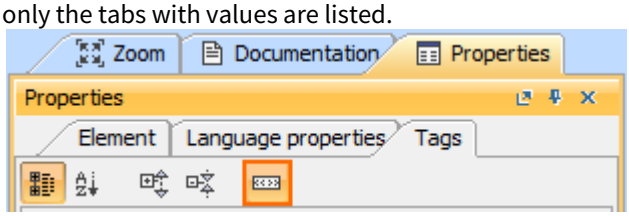
Multiple selection

- If multiple elements or symbols of different kinds are selected, only their common properties are displayed in the **Properties** tab.
- If more than 10 elements in the Containment tree or symbols on the diagram pane are selected, properties are displayed only by request. You need to click the link **Show common properties for the selected items**.

To open the **Properties** tab, do one of the following

- In the Model Browser, click the **Properties** tab.
- On the **Window** menu, click **Properties**.

The **Properties** tab includes the following tabs:

Tab name	Description
Element	Contains the main properties of the element's or diagram's specification.
Symbol	Contains the element symbol's properties. This tab is available for symbols only.
Tags	Contains the tagged values of the selected element. When you click the **Show Only Applied Stereotypes Tags** button, only the tabs with values are listed. 

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Language properties	Contains the same data as the **Language Properties** property group in the element's **Specification** window. This tab is available only for the elements used for code generation, that is, class, attribute, operation.
Traceability	Contains the same data as the **Traceability** property group in the element's **Specification** window.

Related pages

- Model Browser(see page 29)

Image Library tab

The **Image Library** tab displays images grouped in logical sets. You can use a bundled image collection or add your own set of images.

To open the **Image Library** tab

- From the **Window** menu, select **Image Library.**
  The **Image Library** tab opens on the right side of the window.

To add your own set of images into **Image Library**, do either

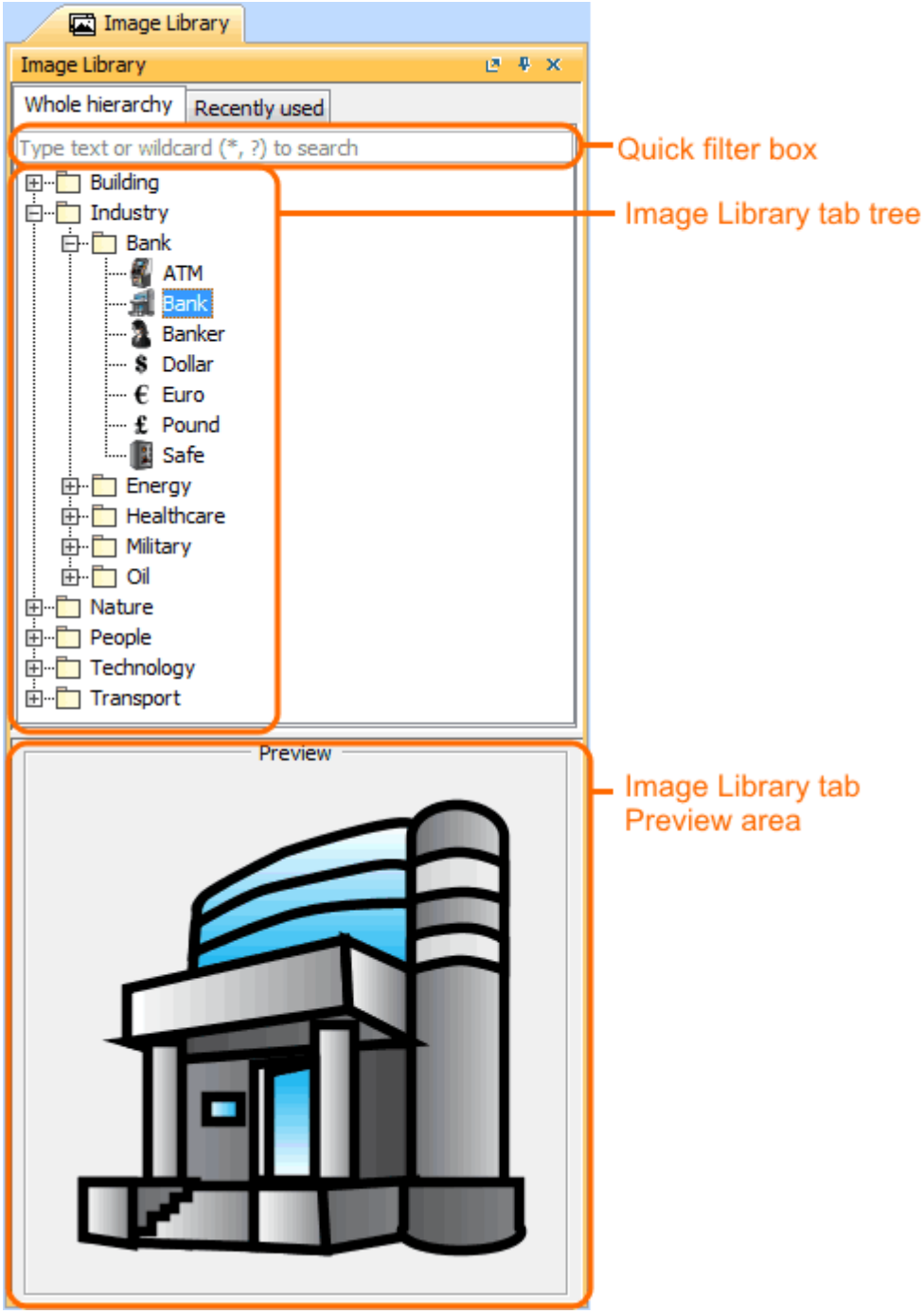
- Compress a folder (.zip) with a set of desired images and place it in the MagicDraw installation directory: *\data\imagelibraries\commonsymbols*.

  Image formats
  You can add images of the following formats: .gif, .jpg, .jpeg, .svg, .png, .wmf.

- Create an **Image Library** plugin. Follow up an example Writing plugins(see page 2159).

Using the **Image Library** tab, you can apply an image on symbol(see page 977).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The **Image Library** tab consist of three components:

- Quick filter(see page 59)
- Image Library tree(see page 59)
- Image Library Preview area(see page 59)

Quick filter

You can search for images in the **Whole hierarchy** or **Recently used** tabs using one of the following way:

- Simple Search: type what you want to find in the hierarchy.
- Pattern Search: type * to find any images.

> For example, type "*Bank" if you want to find "Bank" somewhere in the hierarchy.

Image Library tree

You can drag an image from the **Image Library** tree directly:

- On the symbol (image is applied for symbol).
- To the diagram pane (image shape is created).

Image Library Preview area

You can preview the image before applying it on the symbol in the **Image Library** Preview area.

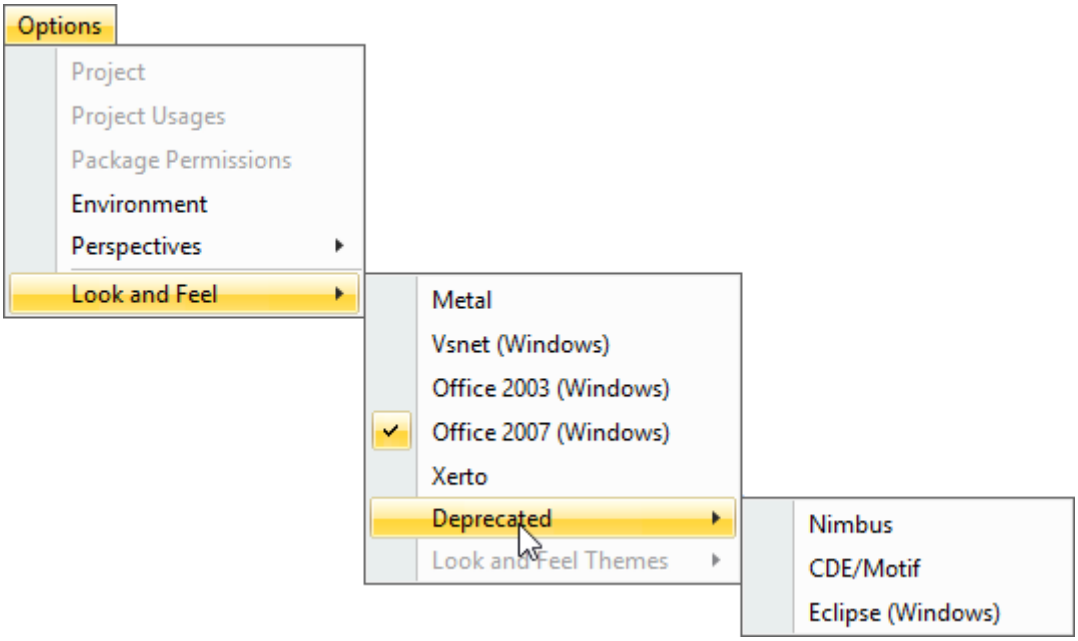
Changing interface style

You can change the appearance of your modeling tool windows, dialogs and menus. The **Look and Feel** menu allows you to personalize the user interface using a number of predefined styles and themes. You can also create a custom theme with your favorite colors and fonts.

To change the interface style

1. In the main menu of a modeling tool, select **Options** > **Look and Feel**.
2. Select the desired style from the list of available styles.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The Look and Feel menu options.

Deprecated styles
The use of styles listed under **Look and Feel** > **Deprecated** is discouraged as they are outdated and only kept for compatibility purposes.

Important
- The selected style may not look exactly like the style in related operating systems.
- Depending on your operating system, some style options may be unavailable.

Creating a custom theme

Define your own interface style by creating a custom theme. The **Custom** theme option is located in the **Look and Feel Themes** submenu which also offers 8 predefined themes. The **Custom** theme allows you to specify text fonts and sizes of the desired interface elements, as well as their colors.

To create a custom theme

1. In the main menu of a modeling tool, select **Options** > **Look and Feel** > **Metal**.
2. In the same menu, select **Options** > **Look and Feel** > **Look and Feel Themes** > **Custom**.
3. In the open dialog, change the values of the desired style properties.
4. Click **OK**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Look and Feel Themes
The themes listed in the **Look and Feel Themes** submenu are only available if you select the **Metal** style.

Related pages

- Understanding the user interface(see page 24)
- Main menu(see page 25)
- Toolbars(see page 27)
- Model Browser(see page 29)
- Image Library tab(see page 57)

Most common shortcut keys

The following table outlines the most commonly used shortcut keys:

What do you want to do?	Shortcut keys	
	Windows	**OS X**
Open the **Find** dialog	Ctrl+Shift+F	Cmd+Shift+F
Open the **Quick Find** dialog	Ctrl+Alt+F	Cmd+Alt+F
Open the **Find and Replace** dialog	Ctrl+R	Cmd+R
Remove both symbols from an active diagram and element from the model	Ctrl+D	Cmd+D
Remove only a symbol from an active diagram, but not the model element	Delete	Del
Select all shapes on an active diagram	Ctrl+A	Cmd+A
Select an element in the Containment tree	Alt+B	Alt+B

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Add more symbols to selection	Shift+Left mouse button	Shift+Left mouse button
Select elements of the same type	Alt+Left mouse button	Alt+Left mouse button
Open a list of possible elements to assign when typing on a symbol	Ctrl+Spacebar	Ctrl+Spacebar
Add a new line in the compartment of a shape, for example, create an attribute for a class	Shift+Enter	Shift+Enter
Open element's Specification window	Enter	Enter
Open element's **Symbol Properties** dialog	Alt+Enter	Alt+Enter
Open the **Report Wizard**	Ctrl+Shift+G	Cmd+Shift+G
See the list of recently opened diagrams	F12	–
Cut a selected item	Ctrl+X	Cmd+X
Copy a shape or text	Ctrl+C	Cmd+C
Copy a shape	Drag the shape to the empty place on the diagram, while holding down Ctrl	–
Paste	Ctrl+V	Cmd+V
Paste with new data	Ctrl+E	Cmd+E

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Save	Ctrl+S	Cmd+S
Commit changes to the server	Ctrl+K	Cmd+K
Undo	Ctrl+Z	Cmd+Z
Redo	Ctrl+Y	Cmd+Y
Print	Ctrl+P	Cmd+P
Assign a shortcut key to print a report	Alt+ number 1 to 9	Alt+ number 1 to 9
Turn on the Full Screen mode	F11	–
Open a recently closed diagram	Ctrl+Shift+T	–
Paste a symbol style	Ctrl+Shift+V	Cmd+Shift+V
Reverse path direction	Draw the path while holding down Alt	Draw the path while holding down Alt
Reorder elements/properties in a list, tree, and Specification window	Ctrl+Up Ctrl+Down	Alt+Up Alt+Down

Related pages

- Assigning shortcut keys(see page 63)

Assigning shortcut keys

Most commonly used commands in a modeling tool have shortcut keys(see page 61), allowing you to access the commands quicker. However, you can assign custom shortcut keys to modeling tool commands and change or remove already-existing shortcut keys.

To assign a new shortcut key to a command

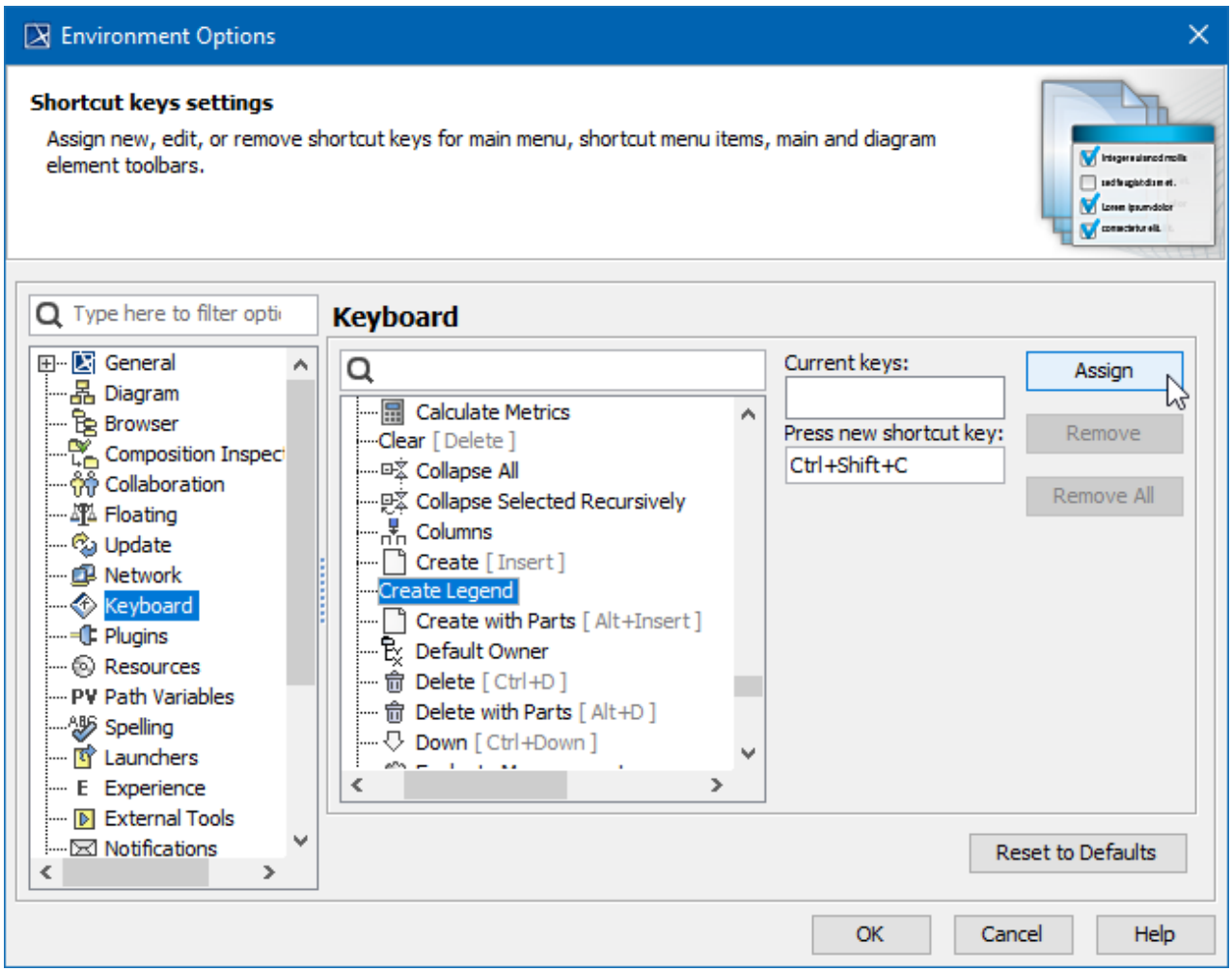
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

1. From the main menu of a modeling tool, select **Options** > **Environment**. The **Environment Options** dialog opens.
2. From the option group list on the left side of the dialog, select **Keyboard**.
3. Search for the command you need to assign a shortcut key to.

   > You can also search by the shortcut keys.

4. In the **Press new shortcut key** box, press the shortcut you want to assign. Do not type the shortcut, but press the keys as you would when executing the command.
5. Click the **Assign** button. The assigned shortcut appears in the **Current keys** box.
6. Click **OK**.

The shortcut key is then assigned to the selected command and you can start using it.



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Assigning a shortcut key.

Usage information

- You can assign multiple shortcut keys to one command.
- If you enter a shortcut key that is already assigned to another command, a notification appears under the **Press new shortcut key** box.
- If you assign a shortcut key that is already assigned to another command, the shortcut key is automatically unassigned from that command and assigned to the new one.

Removing shortcut keys

You can remove both default and custom shortcut keys.

To remove a shortcut key

1. From the main menu of the modeling tool, select **Options** > **Environment**. The **Environment Options** dialog opens.
2. From the option group list on the left side of the dialog, select **Keyboard**.
3. Search for the command you need to assign a shortcut key to.

   You can also search by the shortcut keys.

4. In the **Current** keys box, select the shortcut key you want to remove.
5. Click the **Remove** button. To remove all shortcut keys assigned to the selected command, click the **Remove All** button.
6. Click **OK**.

Restoring shortcut keys
To restore all shortcut keys to their default values, open the **Environment Options** dialog, select **Keyboard**, and click **Reset to Defaults**.

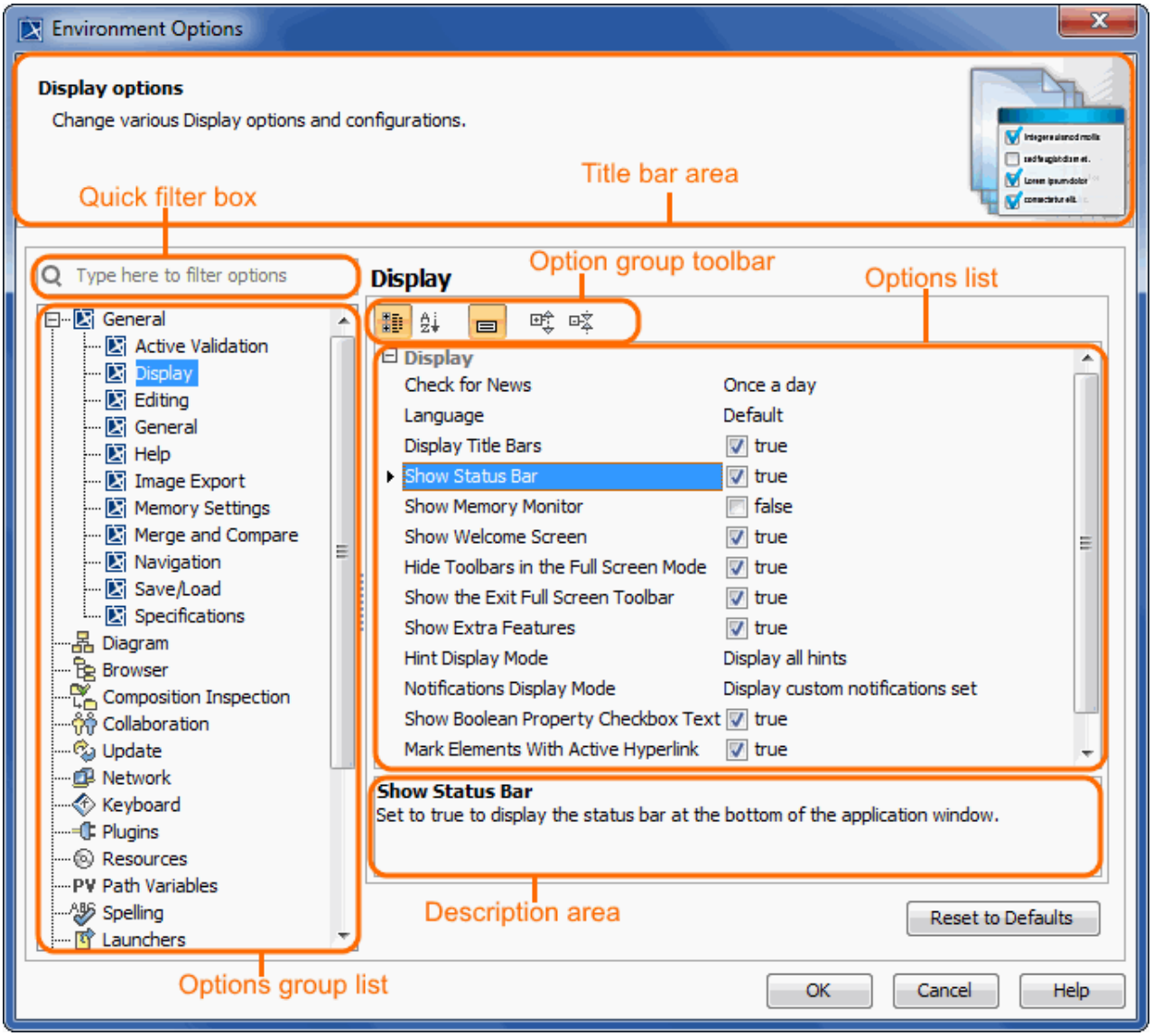
Customizing environment options

You can customize the application environment according to your preferences via the **Environment Options** dialog.

To open the **Environment Options** dialog

- From the **Options** menu, select **Environment**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The **Environment Options** dialog contains various project-independent options grouped by different features (for example, diagrams, Model Browser, code engineering). Each option group is available in a different tab. Tabs are displayed in the tab tree.

An option value can be simply changed by typing a new value, setting a value to true / false, or selecting a value from the list. Click the **Reset to Defaults** button to restore default option values.

Changing options

In order to change a desired environment option, first of all you may need to find it. In this case, use the quick filter(see page 918) in the dialog. Filtering is performed in the all available options, not only in the selected option group.

For better understanding an option you can read its description that tells what is the effect of changing the option value.

To read the option description

1. Make sure that the Show Description mode is turned on in the **Environment Options** dialog. To turn the Show Description mode on or off, click the Show Description button on the tab toolbar.

2. Click an option, whose value you want to change.
3. Read the option description in the area below the tab options list. You are ready now to change the option value.

Customizing and Selecting Perspective

The Startup dialog appears when you launch a modeling tool (MagicDraw, Cameo Systems Modeler, or Cameo Enterprise Architecture) for the first time. Select your work perspective in this dialog.

Due to their growing numbers, many features may be configured for the standard or expert user. The tool better satisfies the needs of different software development process roles by using Perspective.

Use perspectives for:

- Selecting a predefined configuration and features according to your software development process role.
- Finding features faster, because there are fewer.
- Choosing a suitable experience mode with a single click.
- Customizing a set of predefined features and configuration based on user needs.

There are four perspectives:

- **Full Featured** perspective provides all features available in the modeling tool and installed plugins.
- **Quick Start** perspective provides basic features dedicated to modelling with an interface that is not overcrowded, making it quicker to learn. Code engineering, transformations, and other adva nced features are hidden yet easily reachable in expert mode.
- **Software Architect** perspective provides features primarily involved in designing and implementing projects. It is a set of roles consisting of Software Architect, Designer, Interface designer, and Database designer. This is the default modeling tool configuration. All fun ctionalities are available for expert mode.
- **System Analyst** perspective provides features primarily dedicated to obtaining requirements and modeling the system. Analysis features are highlighted. Configuration is modeling oriented. Code engineering, transformations and other features are hidden.

To set the perspective for the modeling tool environment

- Launch he program for the first time. The **Startup** dialog will appear with the ability to switch between perspectives. Select the desired perspective from the list and click **OK**.
- From the **Options** menu, choose **Perspectives** > **Perspectives**. The **Select Perspective** dialog opens. Select the desired perspective and click the **Apply** button.
- On the **Perspectives** toolbar, select the desired perspective from the list.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related pages

Customizing Perspectives

Perspective customization allows grouping of functional features into standard/expert modes. Customi zation also allows the user to hide unnecessary commands, making a modeling tool more convenient.

To open the **Customize Perspectives** dialog

- From the **Options** menu, choose **Perspectives** and then **Customize**.
- In the **Perspectives** dialog, click the **Customize** button.

There are seven customizable areas in different perspectives:

- Main menu
- Main toolbars
- Diagram toolbars
- Diagram modeling elements toolbar
- Smart manipulators
- Shortcut menu actions
- Create Diagram dialog

Each of these areas has a set of commands, which can be shown in the standard / expert mode or hidden.

To customize the selected area in a predefined perspective

1. In the **Customize Perspectives** dialog, select the perspective and click the **Edit** button near the selected area. The appropriate **Customize** dialog opens.
2. Expand tree sections and select radio buttons beside the items in the tree depending on your ch oice for **Standard and Expert**, **Expert** only or **Hidden** modes.
3. Click **OK** when you are done.

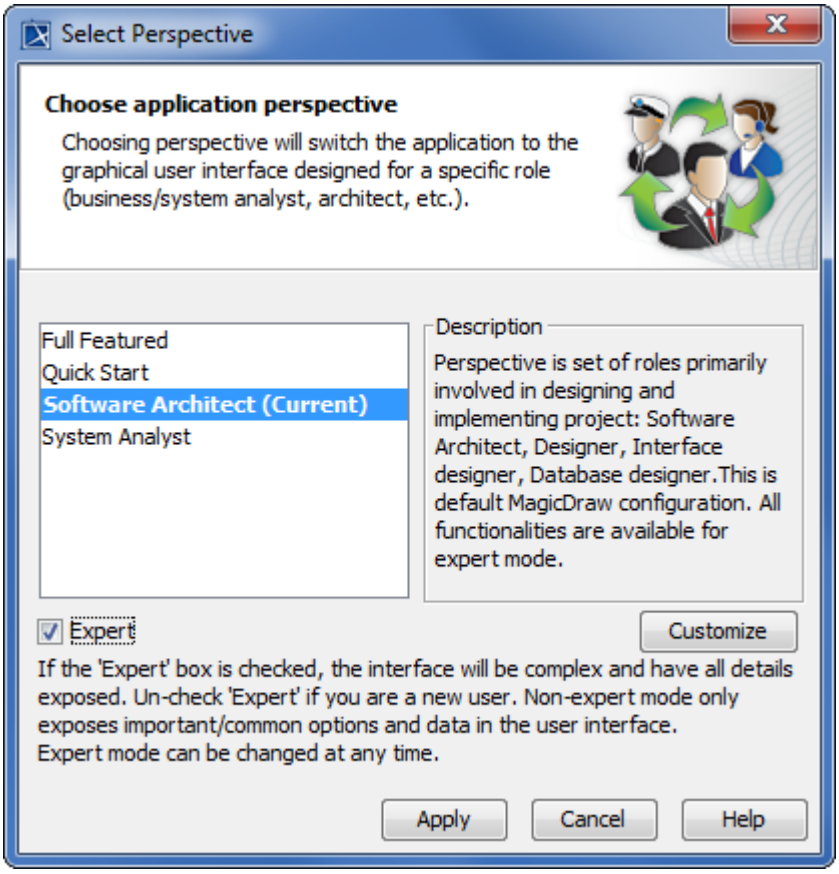
To switch between Standard/Expert modes, do one of the following

- From the toolbar shortcut menu, select/clear the **Expert Menu Mode** or **Expert Toolbar Mode** check box.
- From the diagram palette shortcut menu, select/clear **Expert Mode** check box.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Select Perspective dialog

When you launch a modeling tool for the first time, the **Select Perspective** dialog appears with the ability to switch between perspectives.



Element Name	Description
Full Featured **Quick Start** **Software Architect** **System Analyst**	List of the available perspectives with the predefined configuration.
Expert	Choose Expert mode only if you are an advanced user. All features and commands will be loaded. If you are a new user, you should not select Expert mode.The non-expert mode only loads the basic and most important commands and features.

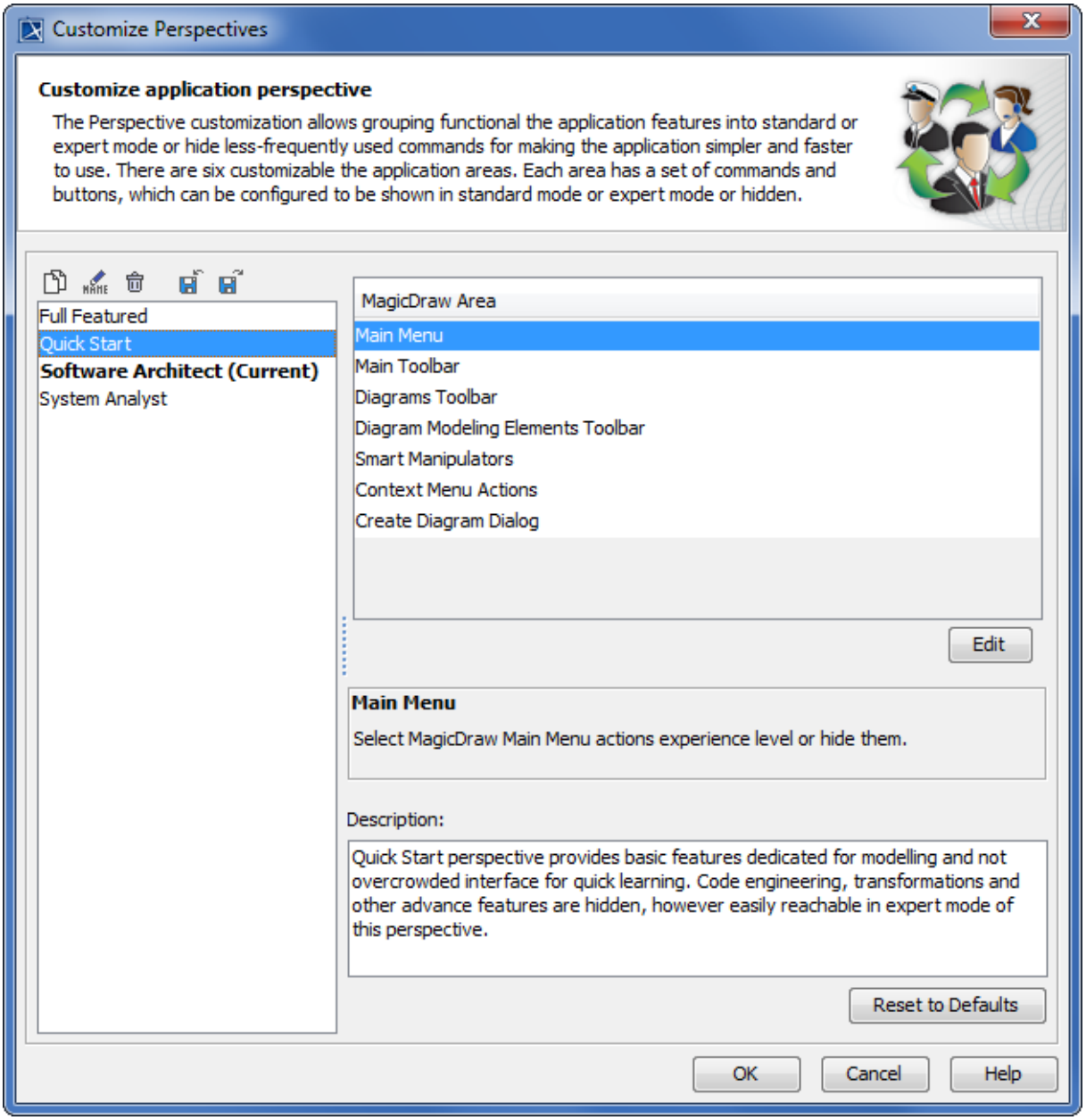
Element Name	Description
Customize	Open the **Customize Perspectives** dialog to customize the selected perspective. You can make your tool less complicated by hiding irrelevant features or commands. For example, you can group functional features, show, or hide selected commands.

Customize Perspectives dialog

To open the Customize Perspectives dialog

- From the **Options** menu, choose **Perspectives** and then **Customize**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Element Name	Description
Clone Selected Perspective	Copy the selected perspective to a new one.
Rename Selected Perspective	The **Enter Perspective Name** dialog opens. Change name of the perspective and click **OK**. You can also rename perspectives by pressing F2.
Remove Selected Perspective	Delete the selected perspective from the list.

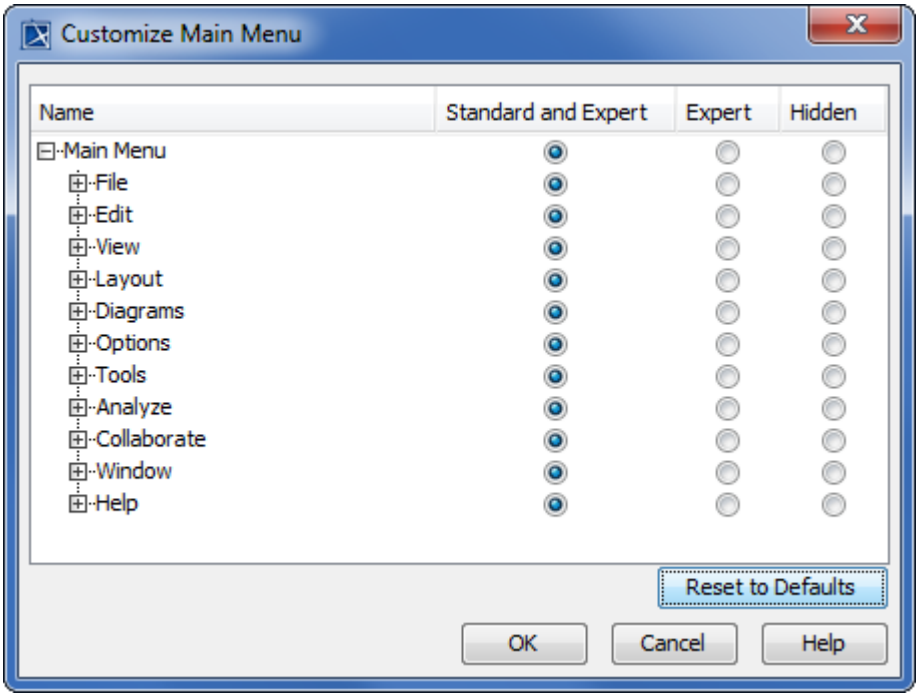
Element Name	Description
Import New Perspective	The **Open** dialog appears. Select **.umd* extension file and click **Open** to import the perspective into the modeling tool environment.
Export Selected Perspective	The **Save** dialog opens. Type a name for the created perspective and click **Save** to store it as **.umd* extension file.
Full Featured **Quick Start** **Software Architect** **System Analyst**	List of the available perspectives with the predefined configuration.
MagicDraw Area	List of customizable toolbars and command sections.
Edit	Click the **Edit** button to open the **Customize Main Menu** dialog. Change he commands mode by selecting radio buttons.
Description	Learn the short description about each selected area.
Reset to Defaults	Reset all changes back to the default configuration.

Dialog for customizing selected modeling tool area

The **Customize <Area>** dialog enables you to customize your own environment, from simple to complex or vice versa. You choose which menus and commands should be displayed and which should be hidden. Customization allows you to make your own working environment.

The name of this dialog depends on the area you select. For example, if you select Main Menu, the dialog's name is **Customize Main Menu**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



In the open dialog, select each environment element individually and choose the mode this element should be available in. There are three modes:

- **Standard and Expert**. The selected element will be visible in the **Standard** as well as in the **Expert** mode. The Standard mode is less complicated and is meant for standard users.
- **Expert**. The selected elements will be visible only in the Expert mode.
- **Hidden**. The selected element will not be shown in any mode.

To assign an element to the mode

---

1. Select an element.
2. Click the radio button under the desired mode.

Configuration files

On this page

- Getting the path to configuration files(see page 74)
- Storing configuration files in the installation directory or Common Application Data location on Windows(see page 75)
- Storing configuration files in a custom location(see page 76)

A configuration file is where the initial parameters and settings of a program are defined. All default parameter values come with a program installation but you can edit configuration files and define your own values, for example, define the Java home directory. This page describes the locations where you can find the configuration files of your modeling tool, how you can modify them and change their default storage location. Configuration files can be used to:

- Define application arguments
- Set the Java Home
- Change Java parameters (such as memory allocation)
- Update default Java classpath (advanced: for plugin developers)

The configuration files can be opened and modified using any standard text or code editor. Once modified, changing the default location of configuration files can allow you to store multiple configuration files for different versions of the tool.

This page covers all modeling tools that are developed by No Magic, Inc., that is

- MagicDraw(see page 19)
- Cameo Enterprise Architecture[23]
- Cameo Systems Modeler[24]

By default the configuration and auxiliary files of a modeling tool are stored in the following location:

OS	Configuration files storage
Windows Vista/7/8/10	C:\Users\<USERNAME>\AppData\Local\.<modeling tool name>\<modeling tool version number>
Other OS	<user.home>/.<modeling tool name>/< modeling tool version number>

Modeling tool file naming
In a file system, modeling tools' folders are named as follows:

Modeling tool	File name
MagicDraw	*magicdraw*
Cameo Enterprise Architecture	*cameoea*
Cameo Systems Modeler	*cameo.systems.modeler*

Getting the path to configuration files

If you do not know the exact path to your modeling tool configuration files, follow the steps below to get it.

To get the path to configuration files

23 https://docs.nomagic.com/display/CEA190/Cameo+Enterprise+Architecture+Documentation
24 https://docs.nomagic.com/display/CSM190/Cameo+Systems+Modeler+Documentation

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

1. From the **Help** menu, select **About** <modeling tool name>. The **About** dialog opens.
2. Click the **Environment** tab.
3. Click the hyperlink next to **Configuration Files**. The folder containing modeling tool configuration files opens.

You can also save configuration files in any of the following locations:

- <modeling tool installation directory>
- Common Application Data location (on Windows OS)
- Other directory

Storing configuration files in the installation directory or Common Application Data location on Windows

Note
The Common Application Data location on Windows Vista/7/8/10 is *C:\ProgramData*.

To store configuration files in the modeling tool installation directory or Common Application Data location on Windows

1. Open the *<modeling tool name>.properties* file, which is located in *<modeling tool installation directory>\bin*.
2. In the JAVA_ARGS line, change the **-DLOCALCONFIG** property value according to the following table:

-DLOCALCONFIG	=true	=false
Windows OSx	Common Application Data	Modeling tool installation directory
Other	Modeling tool installation directory	Modeling tool installation directory

Important!
It is important to add the **-DWINCONFIG=false** parameter on Window OS, otherwise the configuration files will not be stored at the modeling tool installation directory.

You may also store configuration files in your custom location.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Storing configuration files in a custom location

> **Note**
> If you have defined to store files in the modeling tool installation directory, files will not be stored in the defined location.

To store modeling tool configuration files in a custom location

---

Do either:

1. Open the *<modeling tool name>.properties* file, which is located in *<modeling tool installation directory>\bin*.
2. In the JAVA_ARGS line, add the following property:
   `-Dlocalconfig.location=<absolute path to a custom location>`

> **Sample property value**
> -Dlocalconfig.location\=E\:\\temp\\innerPack

Or:

1. Go to the folder, where the configuration files are stored(see page 74).(see page 73)
2. Create a new file named *magicdrawredirect*.
3. In the file, type the absolute path to the folder, wherein modeling configuration files will be saved, e.g., C:/<directory name>.

Related pages

- Customizing environment options(see page 65)
- Customizing and Selecting Perspective(see page 67)

Disabling inbound network activity

Note that after disabling inbound network activity, you will not be able to open the modeling tool Help or submit an issue.

> **Useful information**
> If you disable inbound network activity, double-clicking a project file (.mdzip) opens the project in a new modeling tool window.

To disable inbound network activity

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

1. Go to *<modeling tool installation directory>\bin* and open the properties file, e.g., *magicdraw.properties*.
2. In the JAVA_ARGS line, add the following argument:

```
-DDISABLEINBOUNDCON=true
```

3. Save and close the file.

Related pages

- MagicDraw getting started(see page 20)
- Understanding the user interface(see page 24)
- Most common shortcut keys(see page 61)
- Customizing environment options(see page 65)
- Customizing and Selecting Perspective(see page 67)
- Configuration files(see page 73)
- Support(see page 84)

Improving performance

When working with very large models or a number of diagrams at a time, the modeling tool performance may decrease. To increase efficiency, we suggest the following solutions:

- Increase memory allocation value(see page 81) (java heap size). If you want to make sure you have a sufficient amount of memory allocated to your modeling tool, use the memory monitor(see page 78) to track memory usage.
- Increase an active validation period(see page 83) and/or do not keep unused diagrams open.
- Split a project into several separate project files, and load only the files that you work with. This may help only if your project contains several parts with minimal dependencies between them. For more information, see Project partitioning(see page 236).
- Use Garbage Collector(see page 78) to free unused memory.
- Cancel the background tasks(see page 77) that run too long.

Related pages

- MagicDraw getting started(see page 20)
- Customizing environment options(see page 65)
- Customizing and Selecting Perspective(see page 67)
- Configuration files(see page 73)
- Disabling inbound network activity(see page 76)
- Support(see page 84)

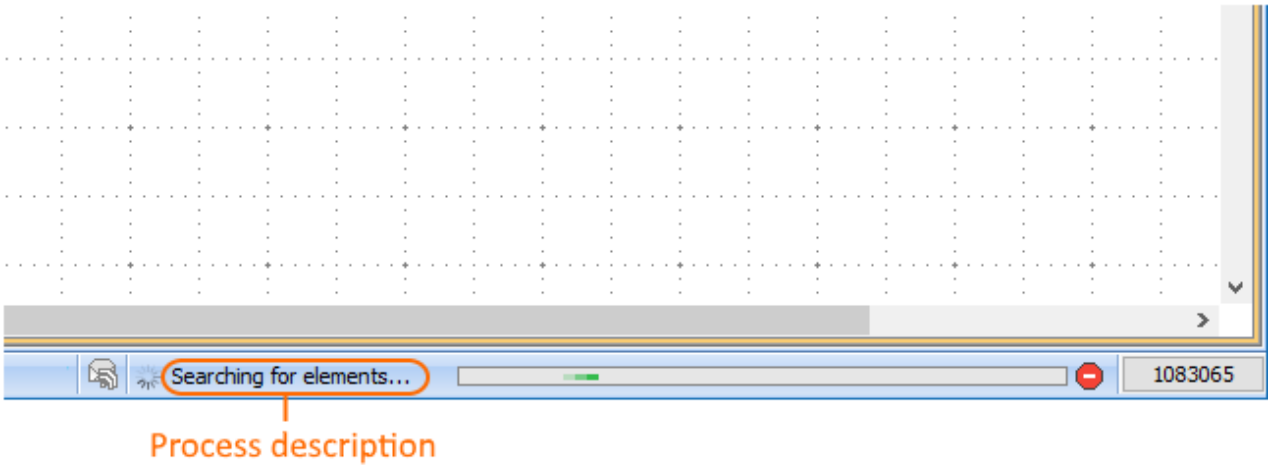
Background Task Manager

Low-priority or routine tasks are moved to separate threads and are performed as background tasks. These tasks do not stop a modeling tool from running, and you can work with your model as usual. Background tasks are displayed in the Background Task Manager at the bottom right corner of the modeling tool window. For example, when a search process is running, the progress bar is displayed in the Background Task Manager, shown in the following figure.

If a background task runs for too long, you can cancel it as described below.

To end an ongoing process

- Click in the Background Task Manager status bar to cancel the selected process. If you cancel the last of several tasks (or only one existing process), the Background Task Manager is turned off.



The Background Task Manager showing the progress of a searching task.

Related pages

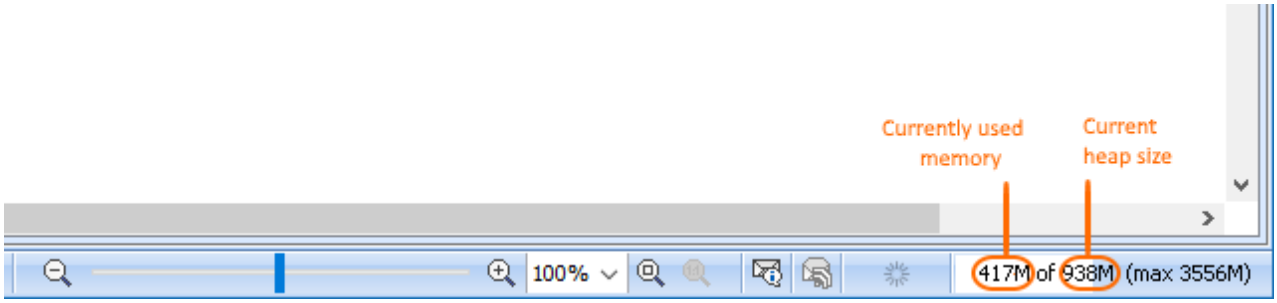
- Improving performance(see page 77)
- Memory monitor(see page 78)
- Memory allocation(see page 81)
- Active validation period(see page 83)

Memory monitor

When working with larger models, turn on the Memory Monitor to monitor memory used by a modeling tool. The Memory Monitor shows currently used memory and the current Java heap size.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Typically, when you start working with a model, used memory increases because actions are stored in the undo list, and opened diagrams are loaded.



The memory monitor bar showing currently used memory and the current Java heap size.

To turn on the Memory Monitor

- In the main menu of a modeling tool, select **View** > **Status Line** > **Show Memory Monitor**.

The Memory Monitor bar appears in the bottom-right corner of the modeling tool window.

> **Running out of memory**
> When used memory takes more than 85% of the total heap size, the Memory Monitor bar turns red, indicating that a modeling tool may run out of memory soon.

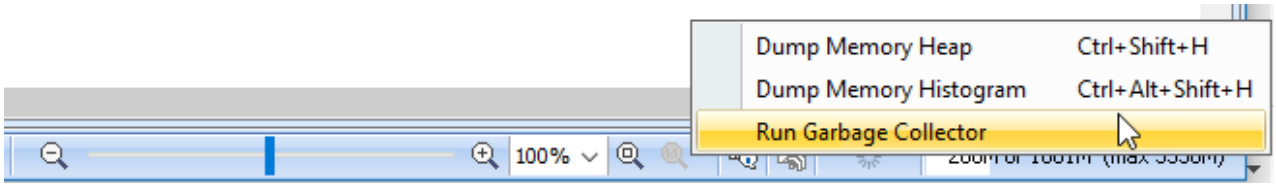
If a modeling tool uses all the Java heap size, you can try to recover enough memory for saving a project with the Garbage Collector, but this may take some time.

To free unused memory with the Garbage Collector

Do one of the following:

- Click anywhere on the Memory Monitor bar and select **Run Garbage Collector**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The Memory Monitor pop-up window

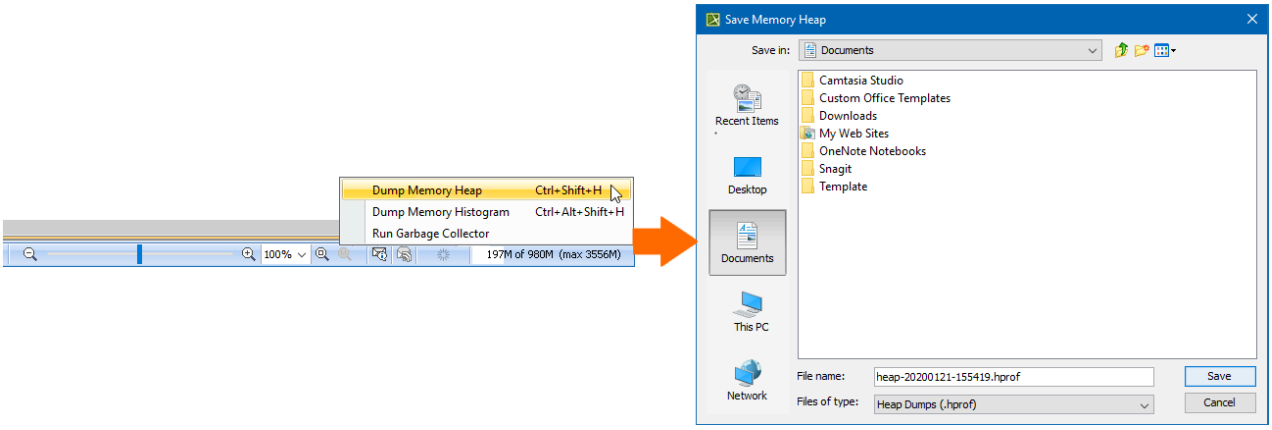
If you find it impossible to use the modeling tool and/or see the Out of Memory error, you should dump memory heap to the **.hprof** file that is created in the defined location on the computer and then submit this file to customer support(see page 84) for further assistance regarding the memory consumption issue.

To dump memory heap

Do one of the following:

- Click anywhere on the Memory Monitor bar, select **Dump Memory Heap**, and then set the desired name and location of the memory heap file. By default, the name of the file is generated according to the current timestamp.
- Press Ctrl+Shift+H on your keyboard and then define the name and location of the generated memory heap file.

You must create a new file for the heap dump since the memory heap cannot be dumped to an existing file.



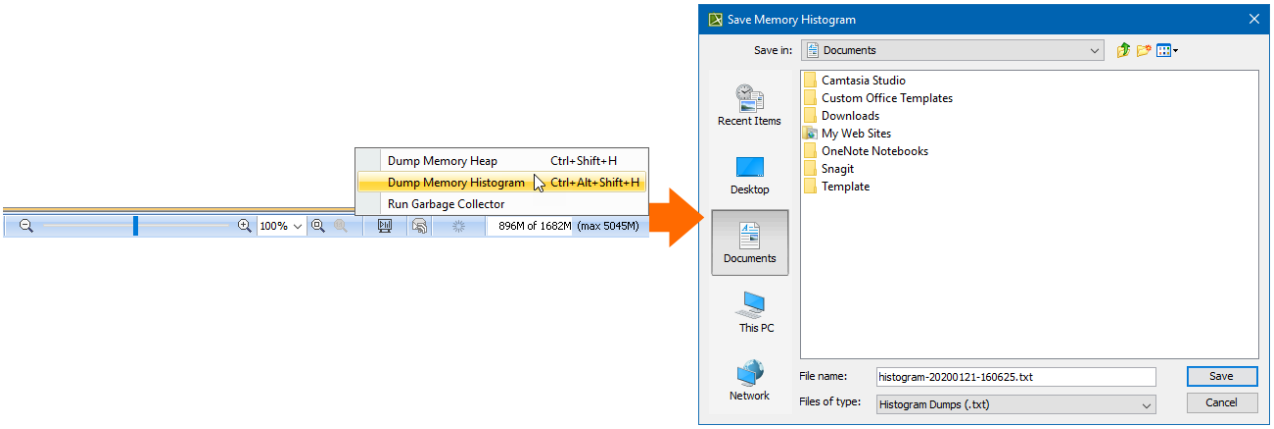
Dumping memory heap

If the Memory Monitor bar is still red, save the model, restart the modeling tool, and reload the model. If the Memory Monitor bar stays red after reloading the model, you should increase the maximum Java heap size(see page 81).

To dump memory histogram

Do one of the following:

- Click anywhere on the Memory Monitor bar, select **Dump Memory Histogram**, and then set the desired name and location of the memory histogram file. By default, the name of the file is generated according to the current timestamp.
- Press Ctrl+Alt+Shift+H on your keyboard and then define the name and location of the generated memory histogram file.



Dumping memory histogram

Related pages

- Improving performance(see page 77)
- Background Task Manager(see page 77)
- Memory allocation(see page 81)
- Active validation period(see page 83)

Memory allocation

If you receive an error message saying your modeling tool is out of memory, the application does not have enough memory to operate. You can change the amount of allocated memory directly in the error message dialog or by changing modeling tool properties.

Memory allocation concept
Memory allocation for the product is the same as Java heap size. To increase the amount of memory allocated for the product, simply increase the Java heap size.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Recommended amount of allocated memory

The amount of allocated memory (or Java heap size) should be set to less than the available physical RAM on your computer. To calculate the approximate amount of memory you should allocate, subtract the amount of memory taken by any other processes that will run concurrently with your modeling tool from the total amount of RAM in your computer. Otherwise your JVM process will likely swap, and that will slow down the application.

Do not allocate too much memory
If you increase the Java heap size too much, you will get an error message that the Java virtual machine cannot be created, and a modeling tool will not start.

The maximum theoretical heap limit for the 32-bit JVM is 4G. In practice, the limit can be much lower, due to various additional constraints such as available swap, kernel address space usage, memory fragmentation, and VM overhead.

- On most 32-bit Windows systems the maximum heap size ranges from 1.4G to 1.6G.
- On 32-bit Solaris kernels the address space is limited to 2G.
- On 64-bit operating systems running the 32-bit VM, the max heap size can be higher, approaching 4G on many Solaris systems.
- On 64-bit VMs, the maximum Java heap size is limited only by the amount of physical memory and swap space your system provides.

Changing the amount of allocated memory

You can set the amount of memory allocated for your modeling tool either in the **Environment Options** dialog(see page 65) or in the modeling tool properties file.

To change the amount of allocated memory in the **Environment Options** dialog

1. In the main menu of your modeling tool, go to **Options** > **Environment**. The **Environment Options** dialog opens.
2. Expand the **General** options group on the left side of the dialog, and select the **Memory Settings** category.
3. In the option specification area on the right side of the dialog, change the value of the **Maximum Heap Size** or **Thread Stack Size** option.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Low memory notification
While working on the project, you can get *Low memory* notification. This notification is triggered by mechanism which tracks whether allocated amount of memory is enough to work with your currently opened projects. Click *Open Environment Options* to open **Environment options** dialog where you can adjust memory settings.

To enable/disable notifications about low memory

1. Go to **Environment Options** > **Display and find Notify about low memory option**
   - Set to *true* to show notifications about low memory.
   - Set to *false* to disable notifications about low memory. Also, this notification can be disabled, by clicking *Do not show again* option in the **Low memory** notification.

To change the amount of allocated memory in the modeling tool properties file

1. Go to *<modeling tool installation directory>\bin* and open the modeling tool properties file.
2. In the **JAVA_ARGS** line, change the value of the **-Xmx** or **-Xss** property. For example, change the **-Xmx800M** to **-Xmx1066M**.
3. Save and close the file.
4. Restart your modeling tool.

Related pages

- Improving performance(see page 77)
- Background Task Manager(see page 77)
- Memory monitor(see page 78)
- Active validation period(see page 83)

Active validation period

By default, an open model is periodically validated. An active validation period is the period of time between subsequent validation occurrences. When an active validation period is short, it may have a negative effect on the performance of your modeling tool, as it needs to constantly process model data.

If you notice that modeling tool performance has decreased, try setting a longer active validation period or opening your model without loading its diagrams.

To change an active validation period

1. In the main menu of your modeling tool, go to **Options** > **Environment**. The **Environment Options** dialog(see page 65) opens.
2. Expand the **General** options group on the left side of the dialog, and select the **Active Validation** category.
3. In the option specification area on the right side of the dialog, change the value of the **Active Validation Period**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

4. Restart your modeling tool.

To open a model without loading diagrams

---

1. In the main menu of your modeling tool, go to **Options** > **Environment**. The **Environment Options** dialog opens.
2. Expand the **General** options group on the left side of the dialog, and select the **Save/ Load** category.
3. In the option specification area on the right side of the dialog, set the **Diagrams Loading Mode** to **Do not load diagrams**.
4. Restart your modeling tool.

Related pages

- Improving performance(see page 77)
- Background Task Manager(see page 77)
- Memory monitor(see page 78)
- Memory allocation(see page 81)

Support

FAQ

Before calling or registering an issue, please have a look in our FAQ section[25]. It is constantly updated and may contain an answer to your question, or issue. If your questions are unanswered in this section, then you have the option of the posting your question, or issue, in our Forum[26] page. If you are interested in reporting an issue, then please refer to our Reporting Issues(see page 88) page. Nevertheless, please continue perusing this page to find out about all the support we provide for you.

Forum

Discuss and get answers about our modeling tools in the No Magic Community Forum[27]. It already has thousands of publicly available posts. You are welcome to post your comments and questions there.

> **Previous Discussions**
> If you are interested in previous discussions from the newsgroups, you can find them in our Forum[28] as well as in the newsgroups with a read-only mode.

---

25 https://www.nomagic.com/support/faq
26 https://community.nomagic.com
27 https://community.nomagic.com
28 https://community.nomagic.com

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

New User
If you are a new user, ask questions and get started on learning about UML and our modeling tools in the New User[29] section.

These are the available options you have on our forum:

- Discuss issues related to technologies and processes in the Standard/ Methodologies[30] section.
- Find new product versions, updates, and major events in the Announcement[31] [32]section.
- Share your professional experience in terms of using **the modeling tool**, including installation and running, OpenAPI, scripting, integrations with other 3rd party tools, etc. in the **Magic Draw**[33] section.
- Post your suggestions for improvements and new features that you would like to see added to our modeling tool, in the Suggestions subsection of the Other[34] section.

Customer Support Levels

We provide free professional support for:

- Registered users with a valid No Magic Software Assurance contract[35] (SA). SA provides you with technical support, software upgrades and maintenance releases at no additional cost for a contract period.
- Pre-sales users during the evaluation period.
- New customers for 30 days.
- Inquiries about registration, licensing and product updates.

More support includes:

- Knowledge base and New issues registration - https://support.nomagic.com.
    - You will need to log in to browse the knowledge base.
- sales@magicdraw.com[36] - e-mail address for questions regarding academic or site license discou nts and quotes.
- contact@magicdraw.com[37] - e-mail address for the other contacts.

If you are a registered user and have already connected to No Magic support system, you can create an issue, or comment on your support issue(s), through email as well.

Reporting Issues
Please refer to Reporting Issues(see page 88) to find instructions on the different ways to inform us of issues you may encounter.

The support system https://support.nomagic.com provides:

- Submitted issues status tracking.
- Ability to submit private and public questions, suggestions, improvements, and problems.

29 https://community.nomagic.com/new-to-magicdraw-f18.html?sid=503891be74c4870fc7bea68e4e600338
30 https://community.nomagic.com/standards-methodologies-f19.html?sid=503891be74c4870fc7bea68e4e600338
31 https://community.nomagic.com/suggestions-f49.html
32 https://community.nomagic.com/viewforum.php
33 https://community.nomagic.com/magicdraw-f21.html?sid=503891be74c4870fc7bea68e4e600338
34 https://community.nomagic.com/other-f48.html
35 https://www.nomagic.com/support/sales-and-licensing/software-assurance-maintenance-contracts
36 mailto:sales@magicdraw.com
37 mailto:contact@magicdraw.com

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Ability to search through the existing public issues, view status of your issues, provide your vote for suggestions.

A customer support level determines customer access rights and the response time for support requests. To find out yours customer support level, please, visit the knowledge base website - https://support.nomagic.com .

Support Level	Applies For	Ensured Support
Limited Support	Customer with no Software Assurance (SA) [38]	• Self-help: FAQ, Knowledge base, documentation, online demos, No Magic Community Forum. • Inquiries about registration, licensing and product updates.
Full Support	• Customer with SA[39] • Customer with valid evaluation key.	• Professional online support with trouble ticket status tracking. • Inquiries about No Magic software problems and usage. • Response within business hours, with 24 - 48 hours response time based on severity*. • Reasonable No Magic efforts to provide a usable work-around solution or to correct the issue in an upcoming maintenance release or update.
Premium Support	• Customer with SA • VIP customer** (marked as VIP)	• Response within business hours, with 24 hours response time.
Dedicated Support	• Block hours are purchased[40]	• Includes integration, customization, migration, plugin development, remote or onsite support.

38 https://www.nomagic.com/support/sales-and-licensing/software-assurance-maintenance-contracts
39 https://www.nomagic.com/support/sales-and-licensing/software-assurance-maintenance-contracts
40 https://www.nomagic.com/support/dedicated-support

Attention
All requests for support must be submitted by accessing the JIRA request form directly from the modeling tool (from the main menu, choose **Help** > **Report an Issue**)*** or directly through No Magic Customer Support System[41].
* Severity let us know the impact of the problem on business.
- Severity 1: Critical business impact or system down, this condition requires a solution.
- Severity 2: Significant business impact, this indicates the program is usable, but is severely limited.
- Severity 3: Some business impact, this indicates the program is usable with less significant features.
- Severity 4: Minimal business impact.

**VIP customers are usually our partner.
*** If your modeling tool stops responding, the executable tools can be used for analyzing and submitting the status of the process. To report an issue using an executable file, start the *reportissue.exe* file, located in *<modeling tool installation directory>\bin*.

Support Period
During the support period covered by our SA[42], you can also report any software problems or errors. If reported, and reproducible, issue in the software exists, and the issue significantly impacts the usability of the software, No Magic agrees to make reasonable efforts to provide a usable workaround solution or to correct the problem in an upcoming maintenance release or update.

End-of-Life Policy

The end of life policy depends on the release type in use: Long-Term Releases (LTR) or Future Releases (FR).

The Long-Term Releases (LTR) are extremely reliable and rock-solid. No Magic provides guaranteed support (including service packs and private patches) until the next Long-Term Release is released, approximately every two years. Longer-term support, of up to five years, for a Long-Term Release, is available for an additional charge. For more information about the Long-Term Releases, see http://www.nomagic.com/support/long-term-releases.html[43]

Support for Feature Releases (FR) discontinues when the next Feature Release is released. Therefore, bug fixes for old Feature Releases (FR) are unavailable.

Providing Patches
We support and provide patches only for the users who have Software Assurance contracts (SA)[44] on their purchased products, so please make sure that you are covered.

41 https://www.magicdraw.com/main.php?ts=customer_support&cmd_show=1&menu=customer_support_management
42 https://www.nomagic.com/support/sales-and-licensing/software-assurance-maintenance-contracts
43 https://www.nomagic.com/support/long-term-releases
44 https://www.nomagic.com/support/sales-and-licensing/software-assurance-maintenance-contracts

Public service packs are released primarily for the newest versions. As we understand that switching from one version to another can take some time, in the case of significant issues, we provide service packs, or private patches, for up to one-year-old versions, when applicable.

We always strongly recommend using the newest version because it contains all newest fixes, usability features, new capabilities, and support for standards and technologies.

Related pages

- Reporting Issues(see page 88)
- Configuration files(see page 73)

Reporting Issues

On this page

- Reporting issues directly from the modeling tool(see page 89)
- Report an issue when the program is unresponsive(see page 91)
- Report an issue in the offline mode(see page 92)
- Reporting Issues Through Email(see page 93)
- Commenting on a Support Issue(see page 94)
- View and Submit Internal Errors(see page 95)

Need help? Send notifications of a problem, suggest improvements for our modeling tools, or ask about new features directly from the modeling tool. Simply choose **Help** > **Report an Issue** from the main menu. If the modeling tool stops responding, you can use the executable tools for analyzing and submitting the status of the process. To report an issue using an executable file, start the *reportissue.exe* file, located in *<modeling tool installation directory>\bin*.

These reports help us address issues in a timely manner, as well as speeding up maintenance releases with fixes that are free of known defects.

Automatic Report Build
When reporting an issue directly from the modeling tool, information is automatically incorporated into a report.

File unable to load
If the modeling tool is unable to load your file, and the file is non-confidential, please attach it to your submission. It will assist us when we analyze your issue.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

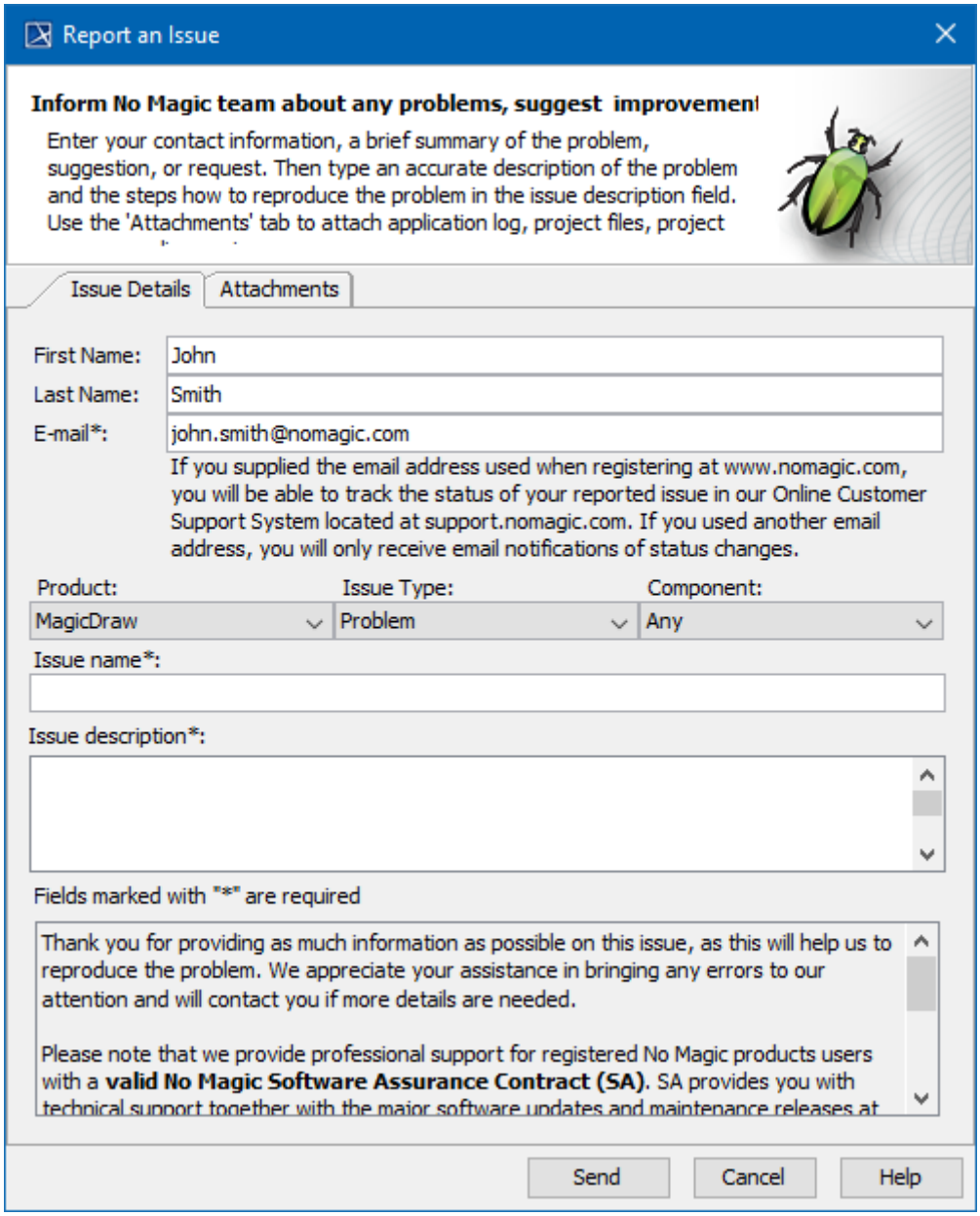
Reporting issues directly from the modeling tool

We strongly recommend submitting a report for any problem, suggestion, or question about new features directly from the modeling tool. Your report will create an issue in our Online Customer Support System[45].

To report an issue directly from the modeling tool

1. On the **Help** menu, click **Report an Issue**. The **Report an Issue** dialog opens.

45 https://support.nomagic.com/secure/CreateIssue!default.jspa



Report an Issue dialog in the modeling tool

2. Fill in your first and last name together with your email address.

> **Email**
> If you supply the email address you used when registering at www.magicdraw.com[46], you will be able to track the status of your issue in our Online Customer Support System[47]. If you used another email address, you will only receive email notifications of status changes.

3. Choose the **product**, **issue type**, and **component**.
4. Describe your issue. Please provide as much information as possible.

46 http://www.magicdraw.com
47 https://support.nomagic.com/secure/CreateIssue!default.jspa

> **Registered MagicDraw Users**
> Please note that we provide professional support for registered users with a valid No Magic Software Assurance Contract[48](SA). SA provides you with technical support, together with major software updates and maintenance releases, at no cost throughout the contract period. Issues are normally handled within one or two business days during regular business hours.

5. In the **Attachments** tab, select files you would like to send together with your issue report:
   - **Attach log file** - the log file will be sent with your issue report.

     > Each modeling tool instance (opened version) has a separate .log file. While the first opened instance has a dedicated *<modeling tool>.log* file in the main directory, all other opened instances create separate folders, e.g., *instance_1* and *instance_2* with .log files. To access the file used by the specific instance, go to **Help** > **About** > **Environment** > **Log File**.

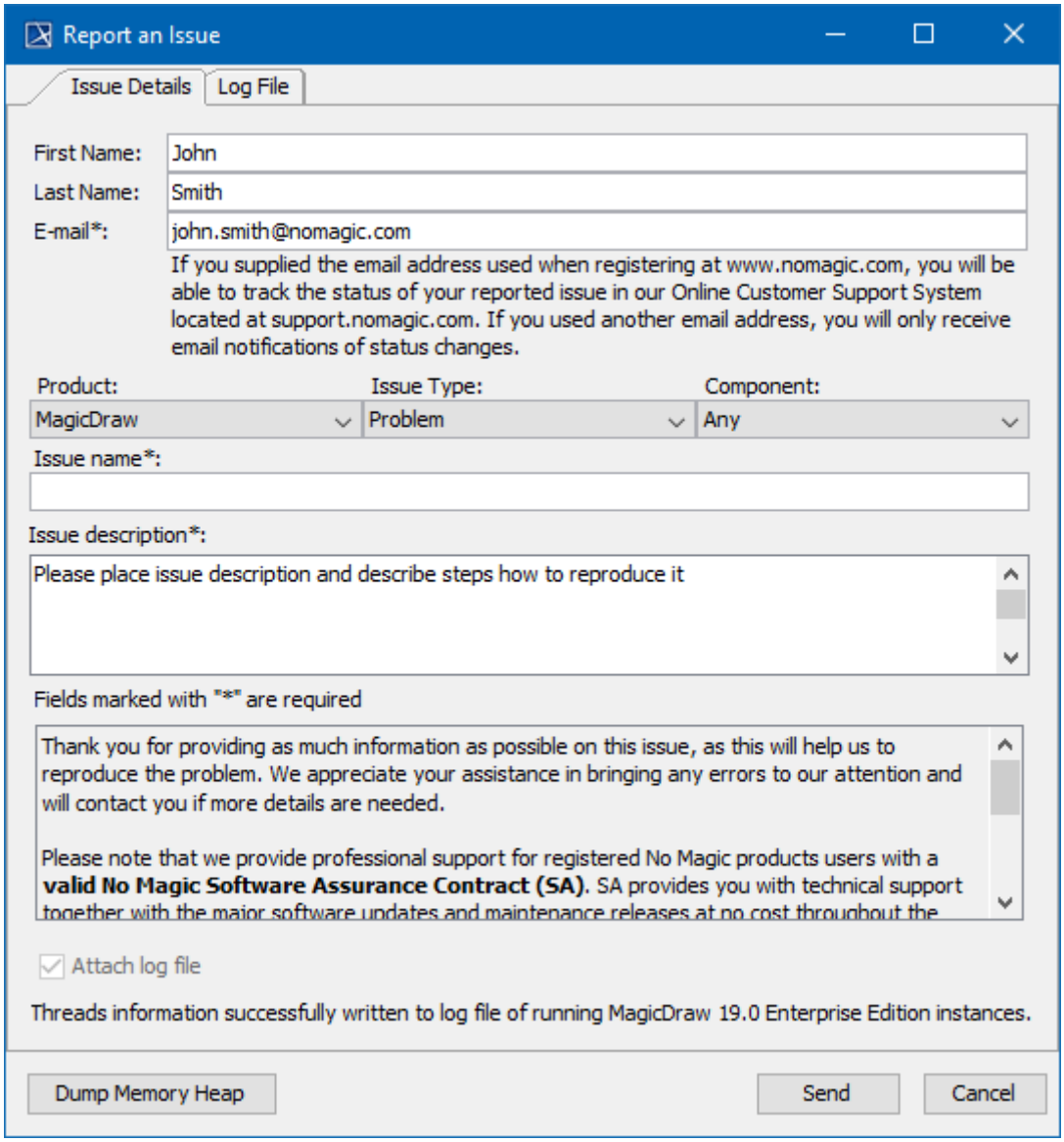
   - **Attach project file** - the opened and last saved project file will be sent with your issue report.
   - **Attach used projects** - modules that have been used in the opened project file will be sent with your issue report.
   - **Attach diagrams images** - choose diagrams, whose images you would like to send with your issue report. Also, specify the format of the images.
6. If you are a registered user, track your issue at https://support.nomagic.com[49].

> **Autofill**
> If you are already registered, your personal information will be automatically filled in on the **Report an Issue** dialog.

Report an issue when the program is unresponsive

- If the modeling tool becomes unresponsive, a separately executable tool is provided for analyzing the status of the process to aid in bug submission. In these situations, manually start the *submit_issue.exe* file (located in the *<modeling tool installation directory>\bin folder*) and follow directions. After you start *submit_issue.exe*, the **Report an Issue** dialog opens.

48 https://www.nomagic.com/support/sales-and-licensing/software-assurance-maintenance-contracts
49 https://support.nomagic.com/

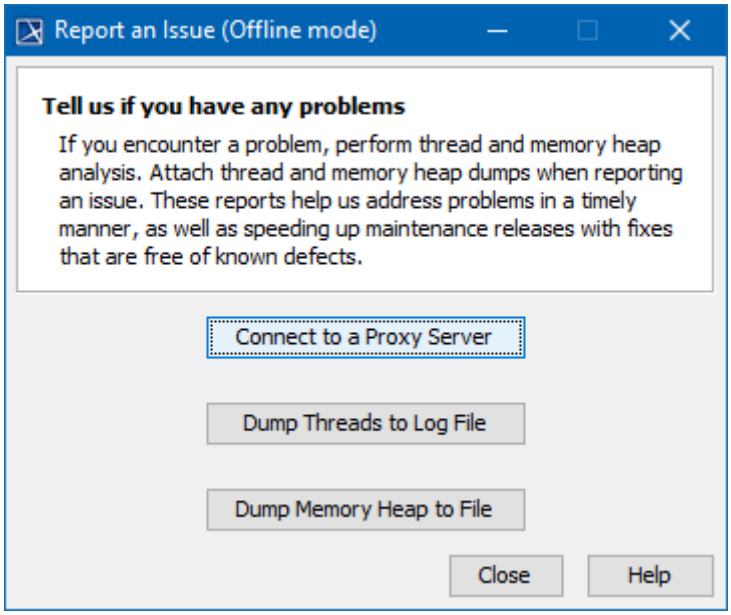


The Report an Issue dialog in an online mode

- If the Out Of Memory error occurs when running the program, start *submit_issue.exe* and then click the **Dump Memory Heap** button in the bottom-left corner of the **Report an Issue** dialog to generate a memory heap file. You need to submit this file to customer support to get further assistance regarding the encountered memory issue.

Report an issue in the offline mode

If there is no Internet connection (or the version of the product you use does not allow sending data out), the Report an Issue window does not appear. Instead, a new offline mode dialog is displayed:



The Report an Issue dialog in the offline mode

- **Connect to a Proxy Server** - try to reconnect to the Internet by entering the Proxy server details.
- **Dump Threads to Log File** - threads will be dumped to a Log file.
- **Dump Memory Heap to File** - memory heap will be dumped to a new .hprof file.

Reporting Issues Through Email

The e-mail address for registered users to access our customer support system is support@nomagic.com[50].

The system allows you to include keywords in the email's subject line. These keywords will enable us to route the issue to the proper support organization more quickly and resolve the issue in a more timely manner. They also allow the definition of many fields for the newly created issue (reporter, project, priority, affected components, affected version).

> **Support Issues**
> Support Issues can only be reported or commented on from the email address registered in your www.nomagic.com[51] account.

The parameters are listed below:

- #PROJECT
- #SECURITY_LEVEL

50 mailto:support@nomagic.com
51 http://www.nomagic.com/

Format of the Email
For example, if you have a question about documentation in UPDM, you might submit the following email:

> **Subject:** Where could I find information about requirements management #QUESTION #PROJECT =UPDM #COMPONENT=Documentation,Requirements__Management #SEVERITY4
>
> **Body:** Hello, I would like to get information ....

This email sent from the registered user will create a support issue in the UPDM project. The Issue's component will be Documentation and Requirements Management and its severity will be low.

Parameters

1. Parameters should be listed in the email subject line with a **single** space between them.
2. **No commas** should be used between parameters.
3. **No spaces** should be used in the body of the parameter.
4. If a parameter value contains spaces, you **must** replace each space with **two** underscore characters. For instance, *#COMPONENT=Test__Component* associates the issue to Test Component. If you specify an invalid component or none at all, the issue is associated to No Component by default.

Emailing support requests can be faster compared to the web interface http://knowledgebase.nomagic.com[52]; however, when reporting issues through email, additional important information might be lost. This information loss can be mitigated by including parameters from the list above.

If an email has email addresses in either the **CC:** or the **BCC:** fields, they will be added to the support issue as External Watchers. All watchers receive email notifications about any activity on the issue.

Support Issues
Support issues that are marked private are not viewable within the support system, even if you are listed as an external watcher. Non-registered users also will be unable to see issues within the support system.

Commenting on a Support Issue

To comment on an existing support issue, please reply to the notification email. The **body** of the email will become your comment on the issue. If you send attachments with your email, they will become attachments on the issue.

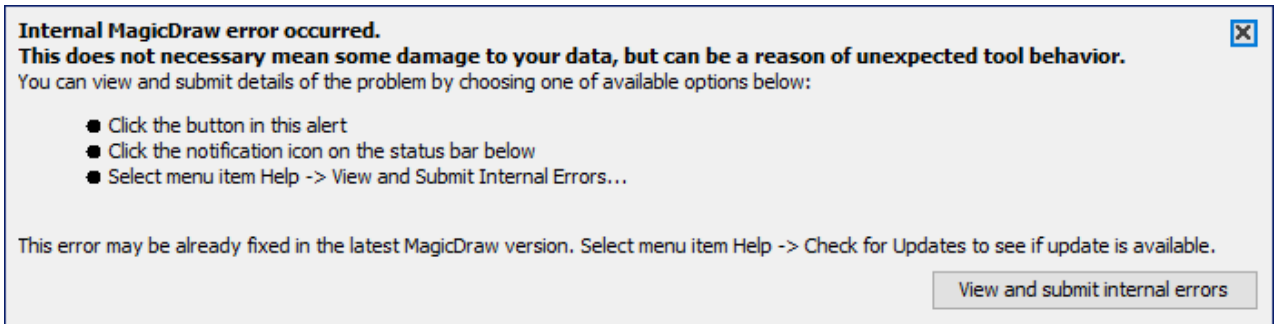
Issue ID
It is important that the issue id (e.g. [MDUMLCS-2058]) exists in the subject line.

52 http://knowledgebase.nomagic.com/

Not a registered user
If a commenter is not registered in www.nomagic.com[53], his or her comment is not recorded in the support issue.

View and Submit Internal Errors

Internal errors are displayed as notifications by default. The internal error does not necessarily mean damage to your data, but can be a reason for unexpected tool behavior. We highly recommend submitting internal errors to No Magic for further investigation. You can submit either manually or automatically. In both cases, we strongly recommend attaching the log file. An internal error message appears at the bottom of the modeling tool.



Example of Internal Error message in MagicDraw modeling tool

To view internal errors

To view internal errors you must open the Internal Errors dialog, using one of the three methods outlined below:

- Click the **View and Submit Internal Errors** button in the Notification Window.
- From the **Help** main menu, choose the **View and Submit Internal Errors** command.
- Click the **notification** icon on the status bar.

Button Availability
The **View and Submit Internal Errors** buttons in the **Help** main menu and the red button at the bottom of the status bar only exist if the **Submit Errors** dialog actually contains errors.

To submit an error

---

1. Open the **Internal Errors** dialog.
2. Click the **Submit** button in the **Internal Errors** dialog box. The **Submit Error** dialog appears.
3. Fill in **Your name**, **Your e-mail** and **Bug description** fields and click the **Send** button.

The error will be sent to the support team. Submitted internal errors will not be shown for 24 hours (or until you restart your modeling tool) after clicking the Submit or Clear And Close buttons.

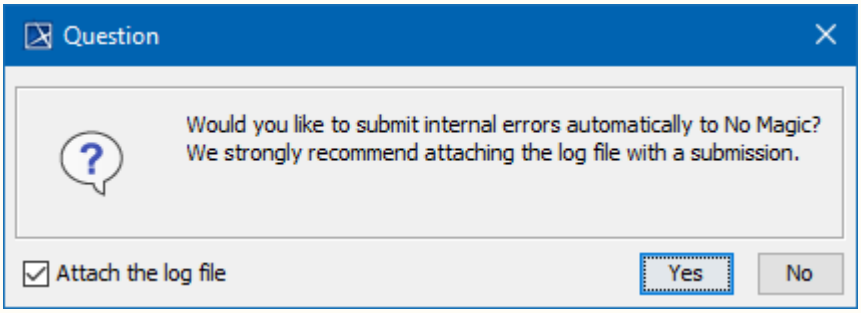
53 http://www.nomagic.com

To allow automatic internal errors submission

1. From **Options** menu, select **Environment**. The Environment Options(see page 2310) dialog will open.
2. Click on the **General** options group **Internal Errors.**
3. To allow automatic internal errors submission: in the Internal Errors options list, change the Automatically Submit Internal Errors value to *true.*

Question dialog
If you set the **Automatically Submit Internal Errors** value to *false* in the **Environment Options** dialog, the **Question** dialog shown below appears when closing the error notification or before the **Internal Errors** dialog opens.

- Select **Attach the log file** check box to attach the log file for Submitting Internal Errors,
- To automatically submit internal errors to No Magic, click **Yes.**



To attach the log file when automatically submitting Internal Errors to No Magic

1. From the **Options** menu, select **Environment**. The **Environment options** dialog will open. In the **General** options group, click **Internal Errors**.
2. To attach the log file, in the **Internal Errors** options list, change the **Attach Log File Submitting Internal Errors** option value to *true.*

To Display Internal Errors

1. From the **Options** menu, select **Environment**. The **Environment options** dialog will open.
2. In the **General** options group**,** click **Internal Errors.**
3. In the **Internal Errors** options list change the **Display Internal Errors** option value to *true* to display internal errors.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Background Task Manager
You can see the submitted internal errors status in the Background Task Manager(see page 77) at the bottom right corner of the modeling tool window. Click the **Background tasks** icon to see the status.

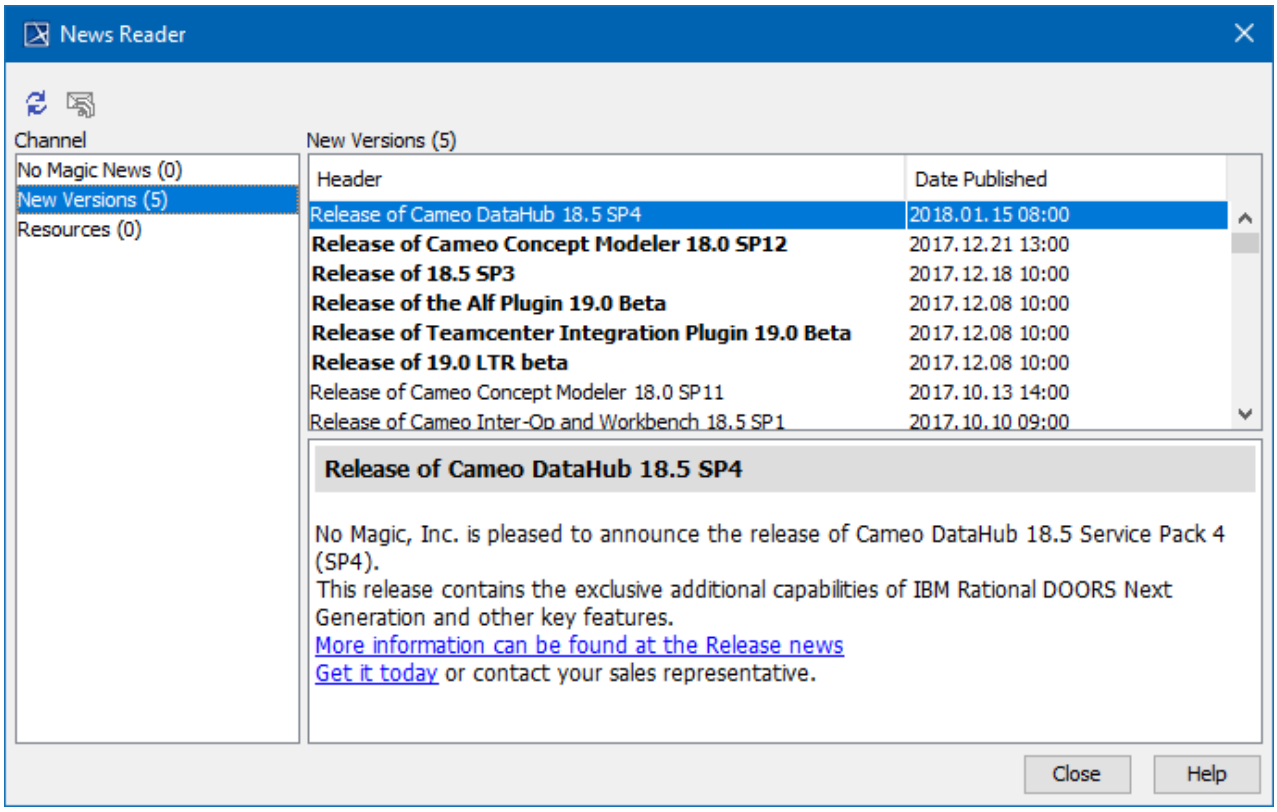
- During the submission process, the status will be *submitting.*
- If the submission process was successful, the status will be *submitted automatically.*
- If the submission process failed, the status will be *automatic submission failed,* and the automatic submission will resubmit errors a maximum of 5 times (every 10 minutes). Otherwise, you can submit internal errors manually using the dialog Report an Issue(see page 89).

Related pages

- Support(see page 84)
- Configuration files(see page 73)

News Reader

The **News Reader** dialog provides information about the latest events.

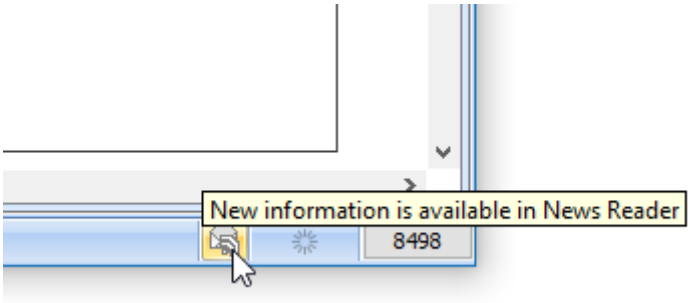


The News Reader dialog.

To open the **News Reader** dialog

1. On the main menu, click **Help** > **News Reader**.
2. Select a channel:
    - **No Magic News** - to read company news, products news, new services provided, etc.
    - **New Versions** - to read messages about new releases, betas, service packs, plugins.
    - **Resources** - to read messages about updates of program resources.
3. Select one of the channels in the list. The description appears below the title. Unread messages are displayed in **bold.**

A small envelope icon appears on the status bar at the bottom of the program window when new information becomes available.. Click that icon to open the **News Reader** dialog.



Envelope icon informs new information is available in News Reader.

To refresh news

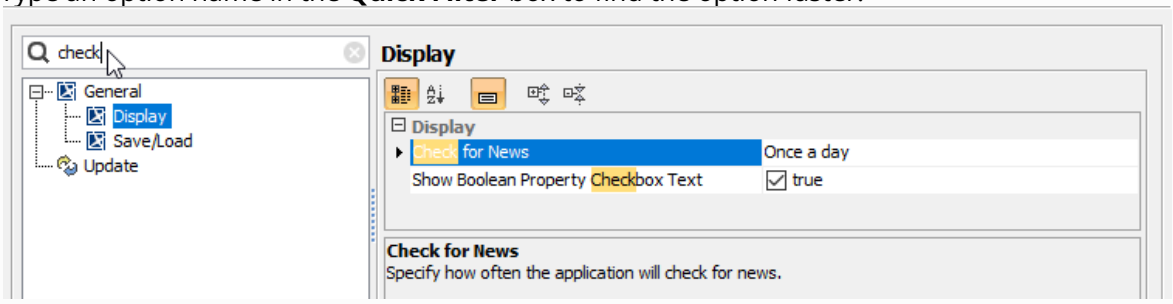
- At the top left corner of the **News Reader** dialog, click .

To change the news checking period for the News Reader

1. On the main menu, click **Options** > **Environment.** The **Environment Options** dialog(see page 65) opens.
2. In the options group list, select **General** > **Display**. The list of **Display** options opens.
3. Set the **Check for News** option value to check for news:
    - Once a day
    - Once a week
    - Once a month
    - Do not check
4. Click **OK** when you are done.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Tip
Type an option name in the **Quick Filter** box to find the option faster.



Related pages

- Getting[54] S[55]tarted[56]
- MagicDraw Editions[57]

Creating path variables

The main purpose for the path variables feature is to give additional sharing flexibility. It enables you to store paths in your project as named variables that can be easily adjusted on different machines.

Path Variables may be used in:

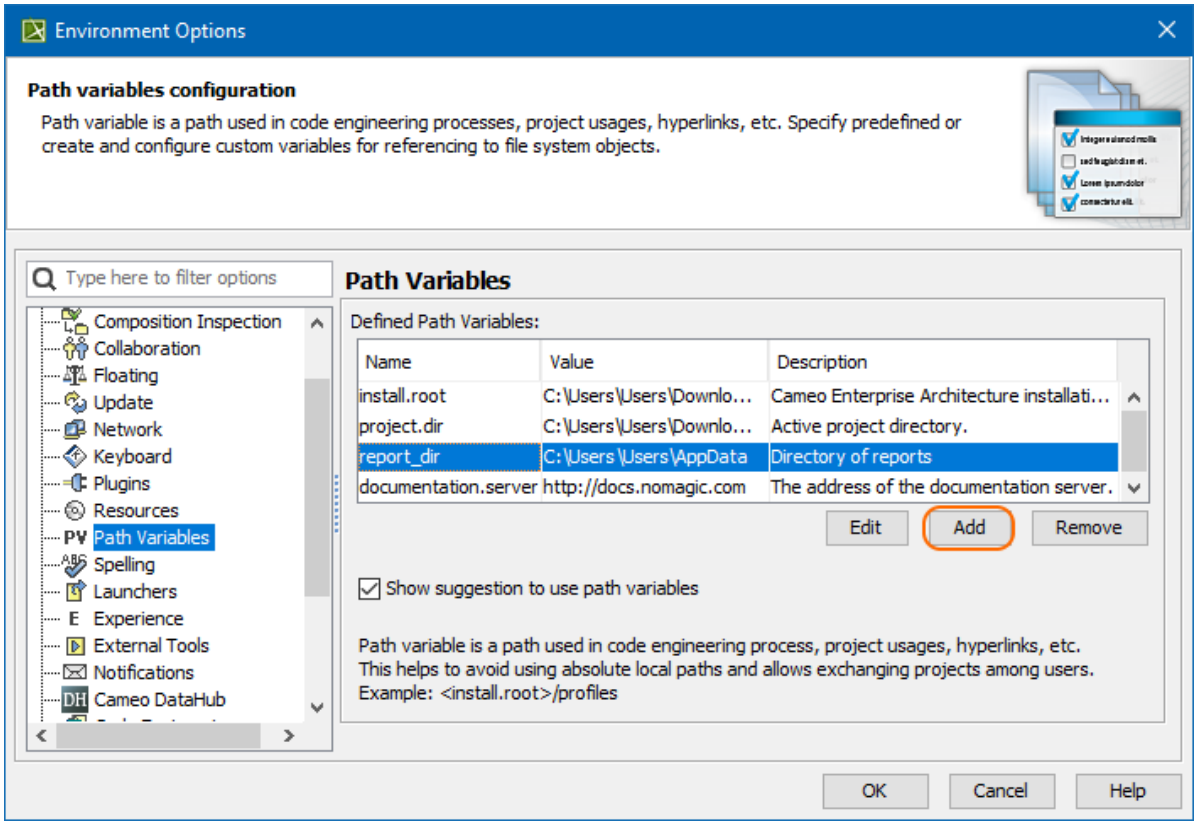
- Code engineering classpath
- Working directories
- Hyperlinks
- Modules paths
- Reports

To create a path variable

1. In the main menu, go to **Options** > **Environment**. The **Environment Options** dialog(see page 65) opens.
2. In the left part of the dialog expand the **General** element and select **Path Variables**.
3. In the **Path Variables** area click the **Add** button. The **Path Variable** dialog opens.

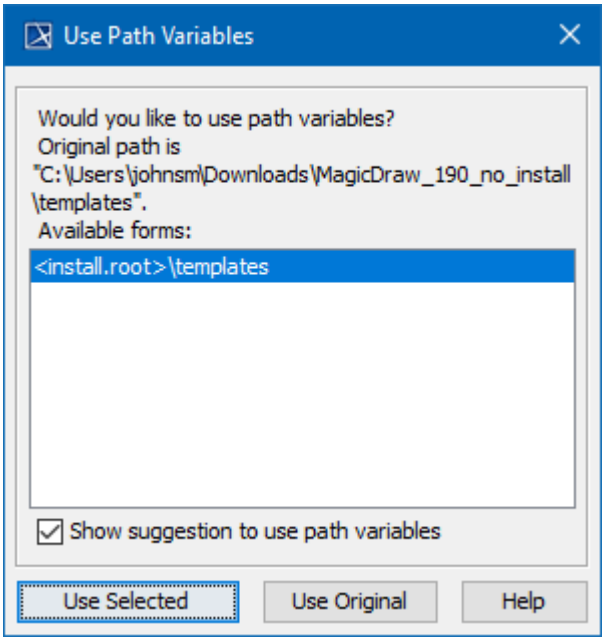
54 file:///Users/ancaandrei/Desktop/getting_started.html
55 file:///Users/ancaandrei/Desktop/getting_started.html
56 file:///Users/ancaandrei/Desktop/getting_started.html
57 file:///Users/ancaandrei/Desktop/editions.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



4. In the appropriate boxes enter the path variable name and description.
5. Next to the **Value** box, click ..., navigate to the desired location and click **Open**.
6. In the **Path Variable** dialog, click **OK**. Your custom path variable is created and shown in the **Defined Path Variables** list.
7. In the **Environment Options** dialog, click **OK**.

Every user can specify different values of the same path variables. However path variables should be resolved only on demand, when they are used. If a modeling tool determines you can use path variables, once you specify your absolute path, the application automatically opens the **Use Path Variables** dialog box. For example, this dialog may open when in your project, you try to use another project **From predefined location** and add a new path to the **Paths to used projects** list.



If you do not want your modeling tool to automatically display the **Use Path Variables** dialog box, clear the **Show suggestion to use path variables** check box. To use the original path and cancel this dialog, click the **Use Original** button.

To use a suggested path variable

- In the **Available forms** list, select a path and click the **Use Selected** button.

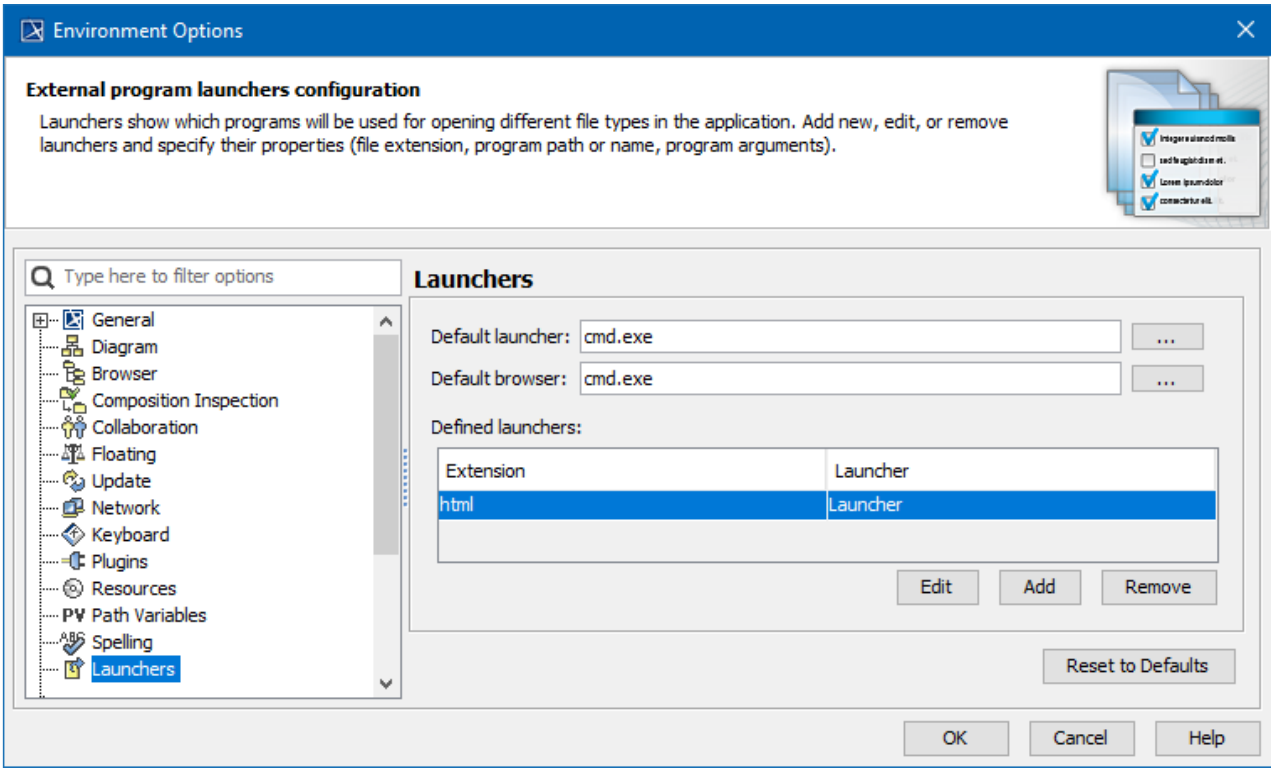
If you have your own documentation running on the local server, you can define the path in the *documentation.server* option.

Launchers

Launchers show which programs will be used for opening different file types in the application. Add new, edit, or remove launchers and specify their properties (file extension, program path or name, program arguments).

To open the **Launcher Properties** dialog

1. From the **Options** menu, choose **Environment**.
2. In the Environment Options(see page 2310) dialog, from options group list select **Launchers**.

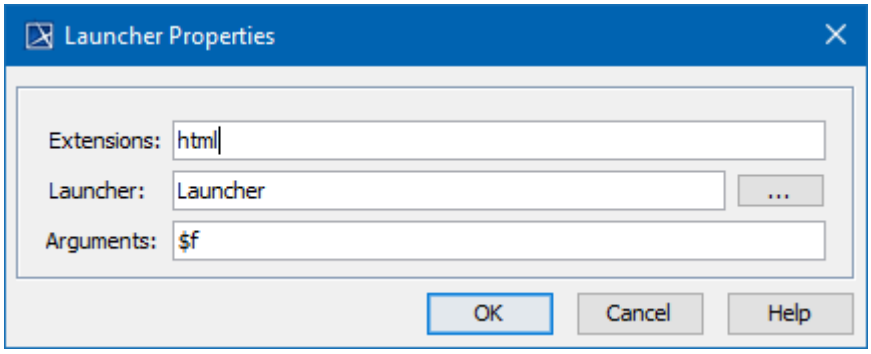


To specify properties of the default launchers

1. To edit Default launcher, click the ... button next to the **Default Launcher** or **Default Browser** text boxes.
2. The **Launcher Properties** dialog appears.
3. You can change Launcher`s executable file and arguments.

To create the defined launcher

1. Click the **Add** button, found below the **Defined Launchers** list.
2. The **Launcher Properties** dialog appears.
3. Specify the file extension, which will be opened with the specified launcher.
4. Type the name of the launcher in the text box, or browse to the location of the launcher by clicking the ... button.
5. Specify external launcher arguments.



Properties are described in the following table.

Property Name	Function
Extensions	Type the extensions of the files which will be opened with the specified launcher. You can define more than one extension. Separate them using commas (i.e., html, xls, doc, cpp, h).
Launcher	An executable file with which the extension file will be opened.
Arguments	Arguments are required by the external launcher, so the modeling tool provides the ability to pass some common arguments to the launcher: • *$f* - a file name • *$md_dir* - the modeling tool installation directory • *$prj_dir* - the project directory (if the project is saved) • *$r* - a row (used when opening Code Engineering object source code file with "Edit Source" action) • *$c* - a column (used when opening Code Engineering object source code file with "Edit Source" action).

Related pages

- Customizing environment options(see page 65)

Special characters

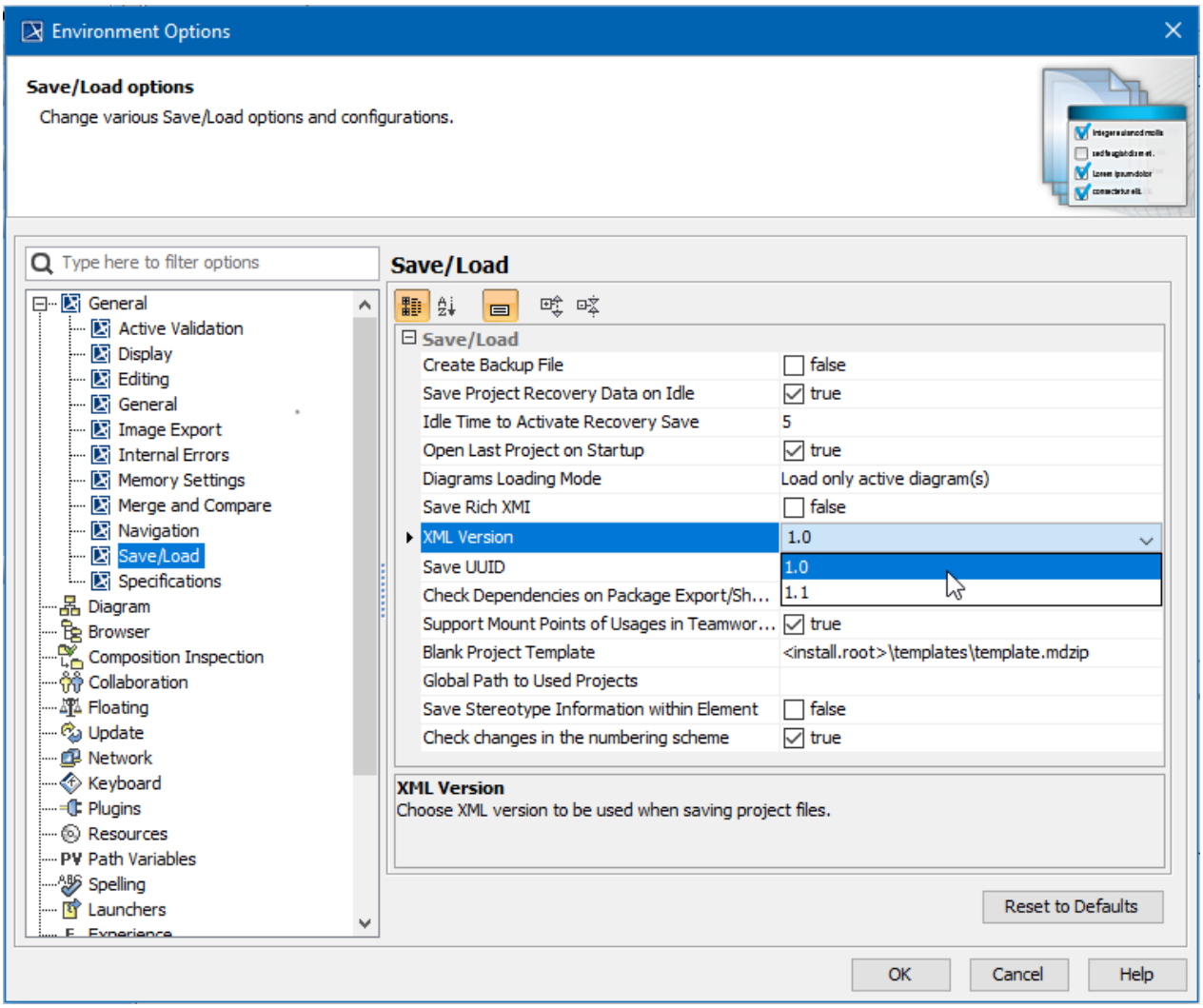
Our modeling tool can use either XML version 1.0 or XML version 1.1. Because XML version 1.0 does not support some special characters, our modeling tool contains a mechanism that replaces certain characters in element names with the underscore symbol (_). However, only a small part of them are replaced, including profile, stereotype and stereotype property names. Please try to avoid special characters and use ASCII letters or numbers. In comparison, XML 1.1 supports all characters, except code point 3000 from “Japanese-style punctuation," which means the tool will replace it with an underscore symbol (_).

The default XML version in the modeling tool is 1.0, to use XML 1.1, you need to manually change it.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To change the XML version

1. In the main menu of a modeling tool, select **Options** > **Environment**. The **Environment Options** dialog opens.
2. In the Environment Options(see page 2310) dialog, select **General** > **Save/Load** from the options group list
3. Click the **XML Version** drop-down list in the option specification area on the right side of the dialog, and select the desired XML version.



Working with projects

Basic concepts

The following table presents more in-depth descriptions of further material.

Concept	Description
Project	A physical working unit that consists of model, model visualizations (diagrams, tables, matrices, etc.), configuration data and references to other elements residing in used projects.
Used project **(up to 18.0 – Module)**	A project having one or more shared packages. Used projects are created for the purpose of decomposing projects into parts in order to manage each part separately and/or reuse those parts in other projects. Used project is a role that one project plays in the composition of another project. It is not a type of project. Each used project can be opened and modified separately.

Related pages

- Creating projects(see page 105)
- Saving projects(see page 113)
- Autosave(see page 114)
- Opening projects(see page 115)
- Importing data from other UML tools and formats(see page 116)
- Exporting UML models(see page 201)
- Creating and using project templates(see page 202)
- Projects with file attachments(see page 208)
- Setting project options(see page 214)
- Searching(see page 216)
- Filtering(see page 233)
- Project partitioning(see page 236)
- MOF support(see page 269)
- Ecore support(see page 271)
- Working with standard profiles(see page 288)
- Forward compatibility(see page 289)
- CATIA Systems Synthesis(see page 291)

Creating projects

Watch this demo to learn the basic procedures regarding creating and managing projects.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Sorry, the widget is not supported in this export.
But you can reach it using the following URL:

http://youtube.com/watch?v=0B76lH-QtrM

Related pages

- Creating a new project(see page 106)
- Project Properties dialog(see page 108)
- Working with multiple projects(see page 111)

Creating a new project

All project information is stored in a single file that has the same name as the project itself.

A newly created project consists of the following packages:

- The empty root package *Model,* storing all model elements.
- *File View* package, containing components that are created during code engineering and representing source files. Adding a resident element to a particular component causes that element to be generated within the source file.
- *UML Standard Profile,* containing stereotypes that are necessary in order to work with MagicDraw, primitive data types and constraints (which are UML standard), and UML 2 metamodel elements. The following data types are specified in MagicDraw: boolean, byte, char, date, double, float, int, Integer, real long, short, void, and string.

You can create your own packages for keeping model elements as well. By default, packages cannot be deleted or renamed in a project (except for the *File View* package).

To start a new project, first you must create a new workspace for it. You can select a project type from the following domains:

- **General-Purpose Modeling** (UML, Use Case, Guide to UML Diagrams projects)
- **System Engineering** (SySML project)
- **Enterprise Modeling** (DoDAF, DoDAF 2.0, MODAF projects)
- **Software Engineering** (Scrum project, Project from Existing Source Code)
- **Business Process Modeling** (BPMN 2.0 project)
- **Service-Oriented Modeling** (Cameo SOA+ project)
- **Other** (Project from Template, Process Guide project)

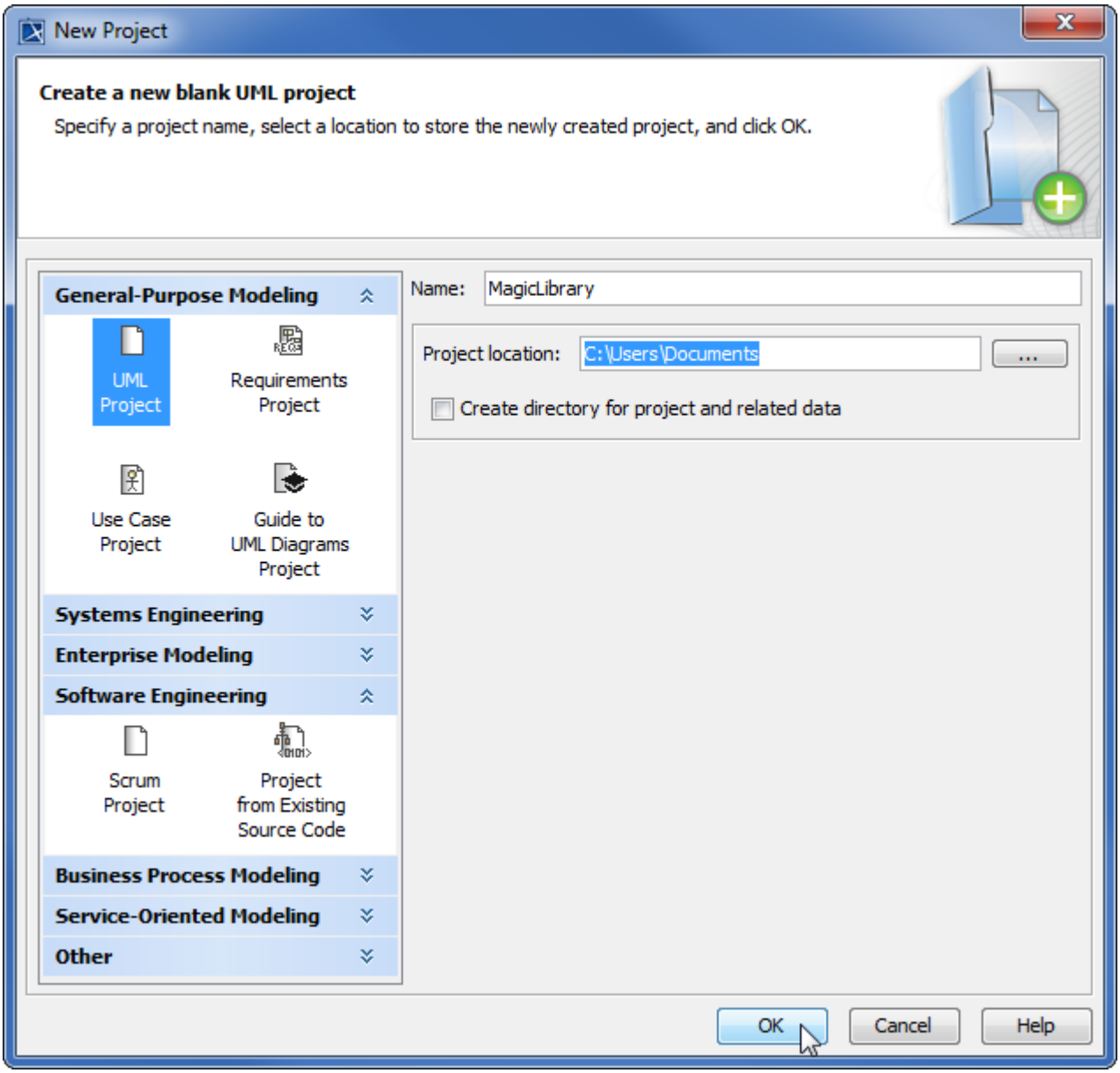
Upon creation of a new project, the **General-Purpose Modeling** domain opens by default.

To create a new workspace for a blank project

1. Do one of the following:

- From the **File** menu, select **New Project**.
- On the main toolbar, click the **New Project** button.
- Press Ctrl + Shift + N.

Any of the above actions opens the **New Project** dialog.



2. Select the **UML Project** in the **General-Purpose Modeling** domain.

3. Specify the file name in the **Name** box.

4. Click the ... button to select the location to store the created project.

5. Click **OK**.

Related pages

- Creating projects(see page 105)
- Project Properties dialog(see page 108)
- Working with multiple projects(see page 111)

Project Properties dialog

On this page

- Compatibility range(see page 110)
- Advanced options(see page 110)

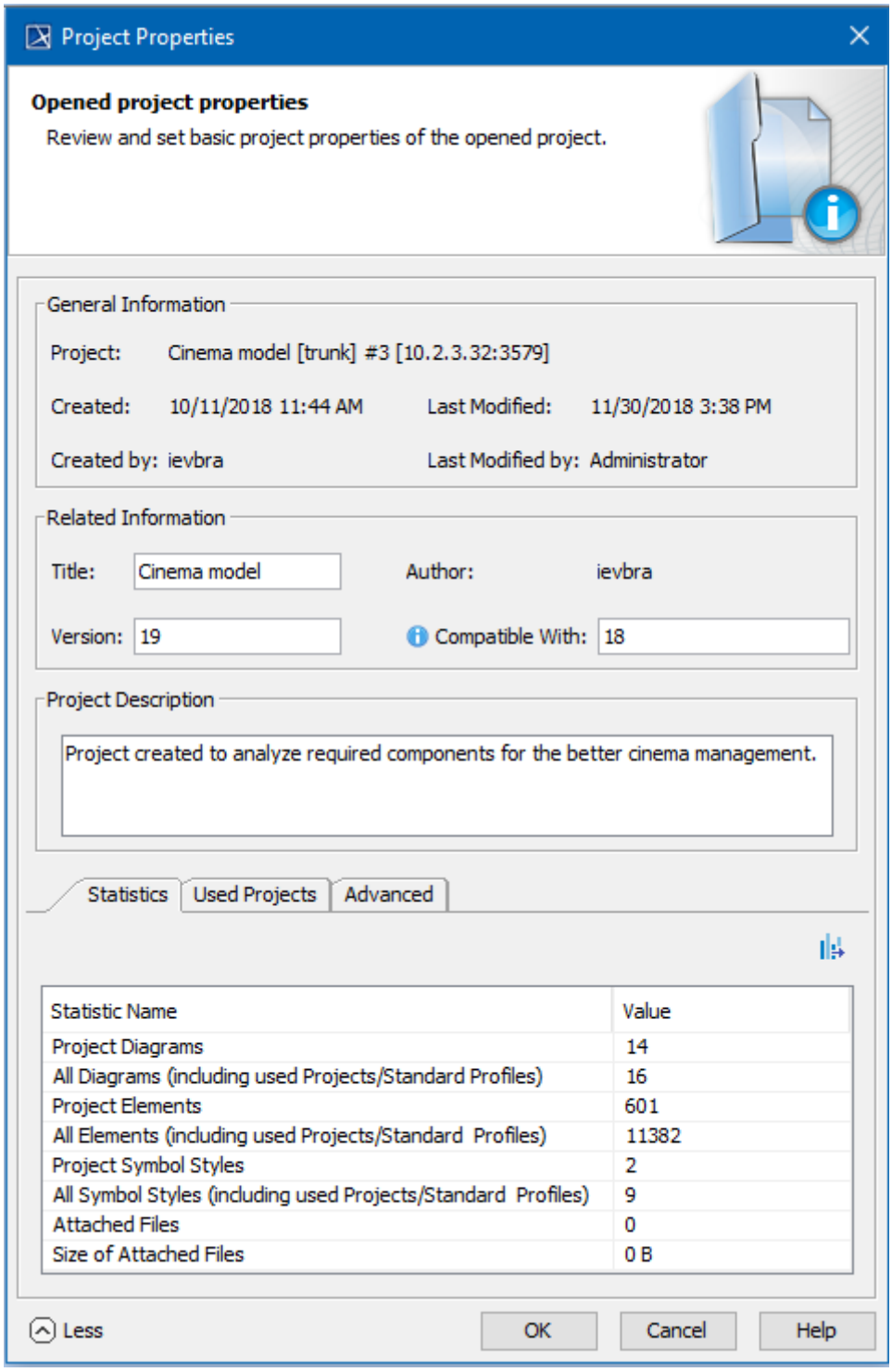
The **Project Properties** dialog provides general information about a project. You can add a title, author, version, and project description using this dialog.

To open the **Project Properties** dialog

---

1. Open the project.
2. Select the **Project Properties** command from the **File** menu.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Project properties dialog.

Title
The project title is assigned as a value of the project Documentation property(see page 54).

Compatibility range

To specify compatibility range

1. Enter the **Version** value.
2. Enter the **Compatible with.** The value must be less or equal to the **Version** value.
3. Click **OK** after you are done.

> **Compatible With**
> **Compatible With** defines the compatibility range between two Standard/System profile versions. Once specified, it helps to avoid read-only constraints in TWCloud projects when you use earlier profile versions locally rather than those used in the server project.
> When the **Version** of the Standard/System profile on your computer is earlier (e.g. 19.0 SP1) than the **Version** of this profile on TWCloud (e.g. 19.0 SP3), but higher or equal to the value of **Compatible with** (e.g. 19.0), you will still be able to edit TWCloud projects using the profile.

Advanced options

To use advanced options

1. Click **More** to see more options:

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Review project statistics in the **Statistics** tab.
- See the list of shared packages in the **Shared Packages** tab. In this tab you can:
  - Define a preferred path to the particular package. Click the **...** button to select the package(see page 1031) in the **Preferred Path** value box.
  - Set the project as a standard/system profile by selecting the **Standard/System Profile** check box.

    > **Note**
    > The **Shared Packages** tab is available for local projects only.

- The **Used Projects** tab shows the projects and standard/system profiles used in the open project. Click the **Open Project Usages** button to open the **Used Projects** dialog(see page 245), where you can modify project usages.
- The **Advanced** tab shows project meta information. In this tab you can:
  - Reset all project element IDs. All project elements have their own ID (a unique identifier). While working with several related projects, you may receive a message about duplicated element IDs. After clicking the **Reset IDs** button, new IDs will be created for all project elements.

    > **Note**
    > Make sure the project you are resetting IDs for is not used in other projects.

  - Reset the project ID. After clicking the **Reset Project ID** button, new IDs are created for the project.

    > You cannot reset the project ID for server projects.

Related pages

- Creating projects(see page 105)
- Working with multiple projects(see page 111)

Working with multiple projects

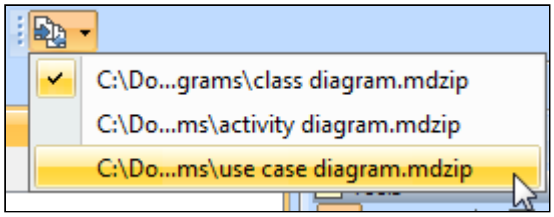
You may need to manage several projects at the same time, and MagicDraw allows you to do that easily. All open projects are held in separate workspaces. Different active projects may exchange data. Entities from one project can be copied or moved to another.

To switch between open projects, do one of the following

- From **File** menu, select **Switch Projects** and then click the name of the project you want to switch to.

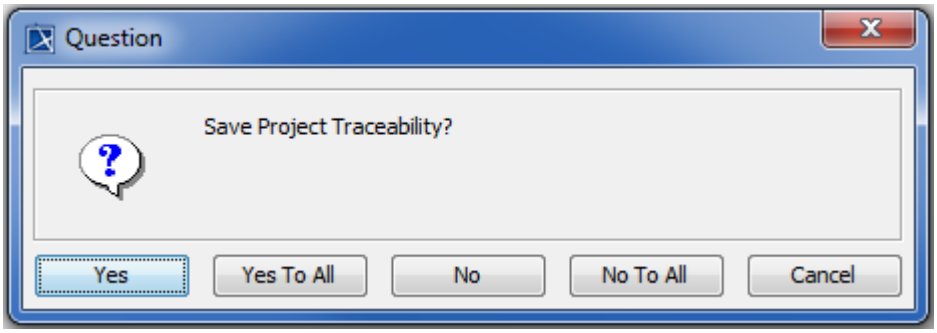
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- On the main toolbar, click and then select the name of the project you want to switch to.



To close all open projects

- Select **Close All Projects** from the **File** menu. The **Question** message appears.



Choose the way your projects will be closed:

Yes	The project you are currently closing will be saved (its name appears in the question). The dialog is displayed again when the next project closes.
Yes To All	Save all projects without prompting. The **Save** dialog will not appear for each open project.
No	Project you are currently closing will not be saved. The dialog is displayed again when the next project closes.
No To All	All the projects will be closed without saving or further prompting.
Cancel	Cancel saving projects.

To exchange model elements between open projects

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Use the **Cut**, **Copy**, and **Paste** commands from the **Edit** menu or the selected element's shortcut menu, or the appropriate shortcut keys: Ctrl + X, Ctrl + C, Ctrl + V.

Rrelated pages

- Creating projects(see page 105)
- Project Properties dialog(see page 108)

Saving projects

The native modeling tool format is **.mdzip* and **.mdxml*. You can also save projects as **.xml* and **.xml.zip*.

To save changes for later sessions, projects must be saved. While saving, you can edit the name of the project and its file format.

To save the project

---

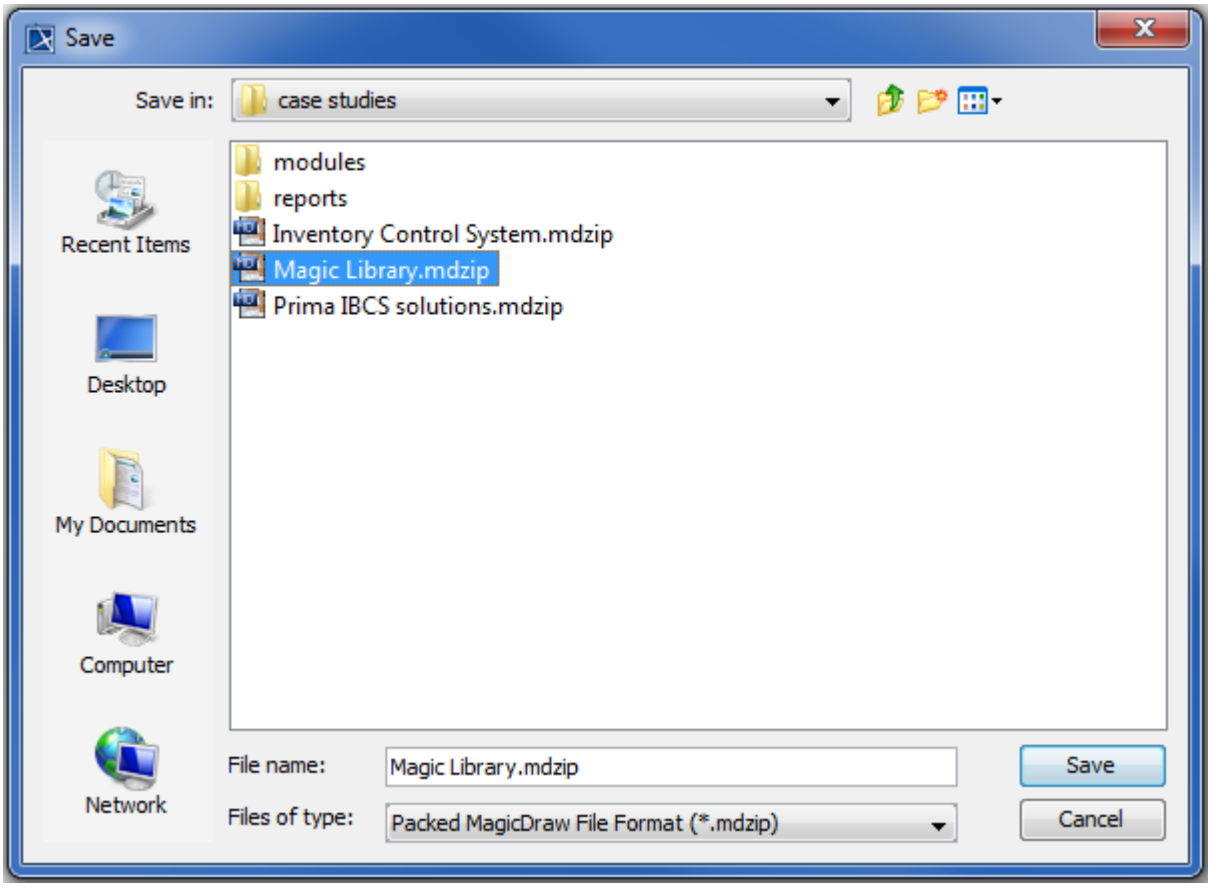
1. From the **File** menu, select **Save Project** or **Save Project As**. Alternatively, you can click the **Save** button on the main toolbar or press the shortcut keys Ctrl + S. The **Save** dialog opens.
2. Select the destination directory (where you want the project to be saved) and type the chosen file name.

   The default project saving location is set according to your operating system:
   - **Windows**: My Documents/
   - **Linux**: /home/<username>/
   - **OS X**: Users/Documents/

3. Select the format for saving a project:

   - Packed MagicDraw File Format (**.mdzip*) (default). **.mdzip* is a MagicDraw native format. It contains various parts of the project compressed. You should select it if you intend to work with the project only in the MagicDraw environment.
   - MagicDraw File Format (**.mdxml*). You should choose it if you intend to analyze the project's structure, xml, or for other purposes. You can open *.mdxml projects with other UML tools as well.
   - XML (**.xml*). It is the same file format as **.mdxml*, only that **.mdxml* files can be opened with MagicDraw or other UML tools, whereas **.xml* files can only be opened with an XML editor.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



- If the **Create Backup File** check box is selected in the **Environment Options** dialog, MagicDraw automatically creates a backup file of the previously saved project. The backup is held in a file with a name identical to that of the project.
- If you want to save maximum additional information to an **.xmi* file, select the **Rich XMI** check box in the **Environment Options** dialog.

Autosave

After you stop working with MagicDraw, an idle time passes and the current project is saved to a special file called the AutoRecovery file.
If the application is terminated normally, the AutoRecovery file is removed. If the application crashes, the AutoRecovery file is kept. On startup, MagicDraw checks for an AutoRecovery file. If it exists, MagicDraw suggests loading the project from this file.

To save an AutoRecovery file of the open project(s) when a system is not in use

1. Open the **Environment Options** dialog.
2. In the **General** pane, select the **Save Project Recovery Data on Idle.** Enter the system idle time (in minutes) in the **Idle Time to Activate Recovery Save** box. This is the necessary idle time in order to activate the AutoRecovery save.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Opening projects

The native modeling tool format is *.*mdzip* and *.*mdxml*.

Opening previously created projects

To edit or review previously created projects, do one of the following

- From the **File** menu, select **Open Project**. In the **Open** dialog, select the project and click **Open**.
- On the main toolbar, click the **Open Project** button.
- Drag the project from the open window of your file system to modeling tool. The project starts immediately.
- Double-click the project file with the *.*mdxml* or *.*mdzip* extension. A new modeling tool window opens.

To start modeling tool with the last project you worked on, select **Options** > **Environment** and set **Open Last Project on Startup** to *true*.

XMI 2.4 / UML 2.4 is the main file format, used by MagicDraw for storing models. This format does not specify how to store diagrams, so MagicDraw stores and opens diagram data in XMI extension sections. Thus if you use MagicDraw to open an XMI file exported from another UML tool, only the model will be opened, but not diagrams or views.
Accordingly, if you open a MagicDraw file in another UML tool, diagrams or views will not be opened in that tool (unless the tool supports MagicDraw-specific file extensions).
You may open more than one project within the same MagicDraw window. A separate workspace will be created for each opened project.

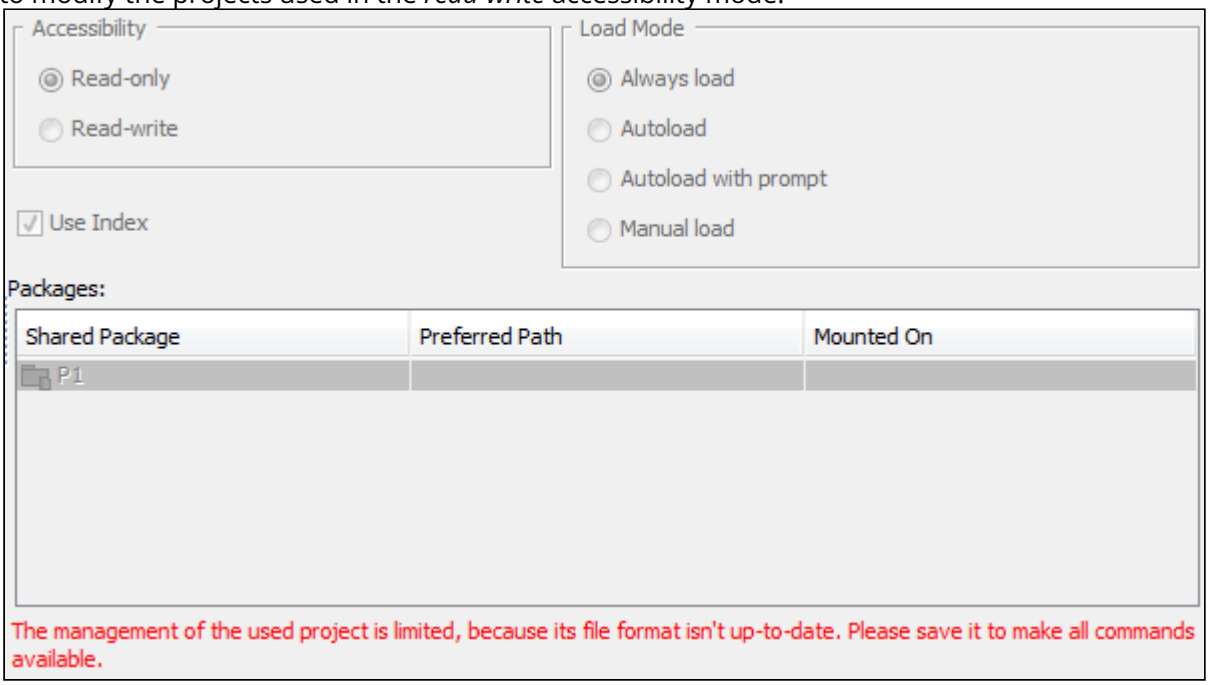
Opening projects saved with modeling tool 17.0 or earlier

Projects saved with modeling tool 17.0 or earlier can be as well opened with modeling tool 17.0.1 or later.
To update older models, simply open them with modeling tool 17.0.1 or later and then re-save them. Models will be automatically converted to the latest format.
In case your model is stored in a project that uses other projects, you should open and re-save the used projects first and only then do the same with the main project. Be advised that having only the used projects successfully converted to the latest format, you may expect the smooth conversion of the project that uses them.

If you convert only the project and leave the used projects saved in an old format, you will not be able

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

to modify the projects used in the *read-write* accessibility mode.



The appropriate procedure should be applied in case your model is stored in a server project with composite structure: firstly, the used projects should be migrated to Teamwork Server 17.0.1 or later, and only then the main project should be migrated.

Importing data from other UML tools and formats

If you need to open a model of a different format or created with another UML tool, you have to import it to MagicDraw instead of simply opening it. MagicDraw allows importing projects from the following file formats:

- Eclipse UML2 (v2.x, v3.x, v4.x, v5.x) XMI file. Select **File** > **Import From** > **Eclipse UML2 (v2.x, v3.x, v4.x, or v5.x)** for this.
- Most of the model data from an Enterprise Architect 7.1, 7.5 or 8.0 file.

  ⚠ Enterprise Architect does not export 100% standard UML 2.1 XMI, and this causes some data loss during the import.

- Model from an IBM® Rational Rose® Modeler file. You can use Cameo Inter-Op[58] for this.
- UML diagrams from Microsoft Visio using XMI files.

  ⓘ Microsoft Visio 2007 and earlier requires extra add-ons for exporting UML model to XMI files.

58 https://www.nomagic.com/products/cameo-inter-op

Enterprise Architect Import

The Enterprise Architect Import has the functionality to import UML models conforming to various XMI versions (including XMI 2.1) from Sparx Systems Enterprise Architect (EA), a modeling and visualization tool based on the UML 2.3 standard. EA has the ability to import and export XMI compliant models, so you can use EA to import UML2.1(XMI2.1). However, XMI models exported from EA contain some XMI conflicts and EA-specific data that do not conform to UML standards.

Enterprise Architect Import allows you to migrate XMI models from EA to a modeling tool flawlessly by using an additional transformation process with a set of mapping rules.

The main purpose of Enterprise Architect Import is to help users who need to import models from EA by managing conflicts that can cause problems while loading the XMI models in the modeling tool, as well as transforming some EA-specific data into UML elements with stereotypes(see page 856).

Enterprise Architect Import supports Enterprise Architect Versions 7.1, 7.5, and 8.0 (most of the testing procedures performed on EA 7.1.833 and EA 7.5.847). The feature helps you import and transform an EA exported XMI using the UML2.1 (XMI2.1) option into a model file (*.mdxml).

In addition to the ability to import model elements, the feature also allows for the import of diagrams. The feature supports:

- Class diagrams
- Package diagrams
- Object diagrams
- Component diagrams
- Deployment diagrams
- Use Case diagrams
- Activity diagrams
- Sequence diagrams
- Communication diagrams
- StateMachine diagrams
- CompositeStructure diagrams
- InteractionOverview diagrams

Related pages

- Diagramming(see page 482)
- Stereotype(see page 856)

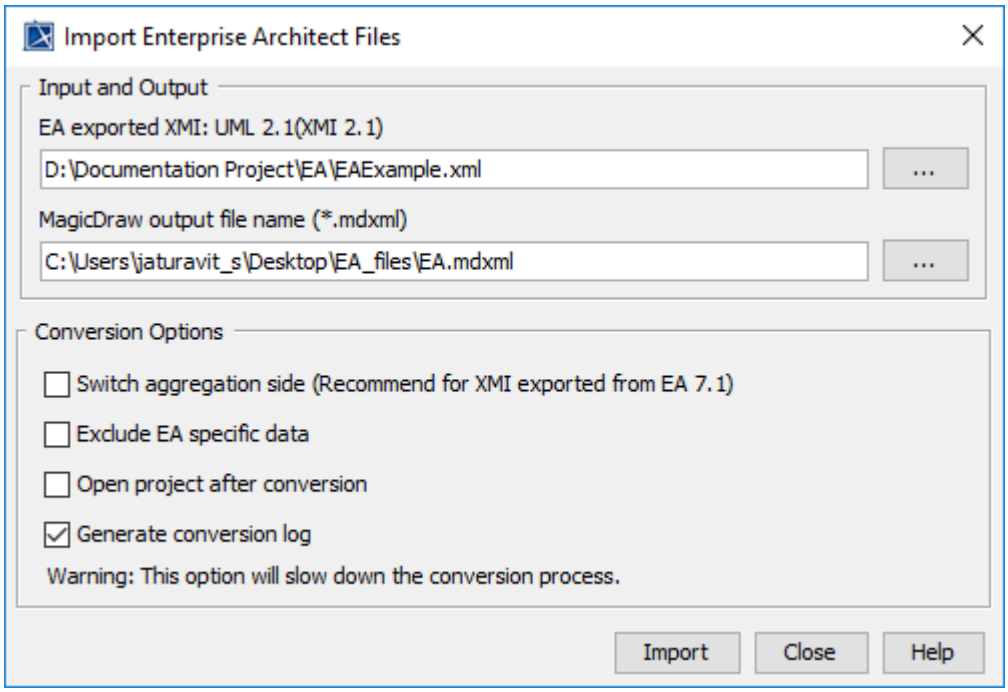
Working with Enterprise Architect Import

Enterprise Architect Import (EA) will automatically load when you start a program. You can use it to import UML 2.1 XMI 2.1 to your modeling tool.

To import Enterprise Architect UML XMI file

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

1. On the main menu of the modeling tool, click **File** > **Import From** > **Enterprise Architect UML2.1 XMI 2.1 File**. The **Import Enterprise Architect Files** dialog opens.



2. In the dialog, you must specify:
   - The input file - the XMI file exported from Enterprise Architecture

     > **Note**
     > Enterprise Architect Import supports the EA XMI files exported with the option specified as: XMI Type = **UML2.1(XMI 2.1).**

   - The output file name
   - A conversion option
3. Click **Import** to import the file.

The following table describes conversion options:

Conversion Option	Description
Switch aggregation side	To configure the **aggregation switch-side**. This option is recommended for XMI files exported from EA 7.1.
Exclude EA specific data	To exclude all EA-specific data from being imported (see Transforming EA Specific Data(see page 121) to see a list of EA specific-data that can be transformed into UML elements with stereotypes).
Open project after conversion	To load the output project file once the conversion process has been completed.

Conversion Option	Description
Generate conversion log	To generate a conversion log and save it in the same folder as the output file. The same conversion information will also be displayed on the **Notification Window**. **Note** You can open the window by pressing **Ctrl** + **M**.

Related pages

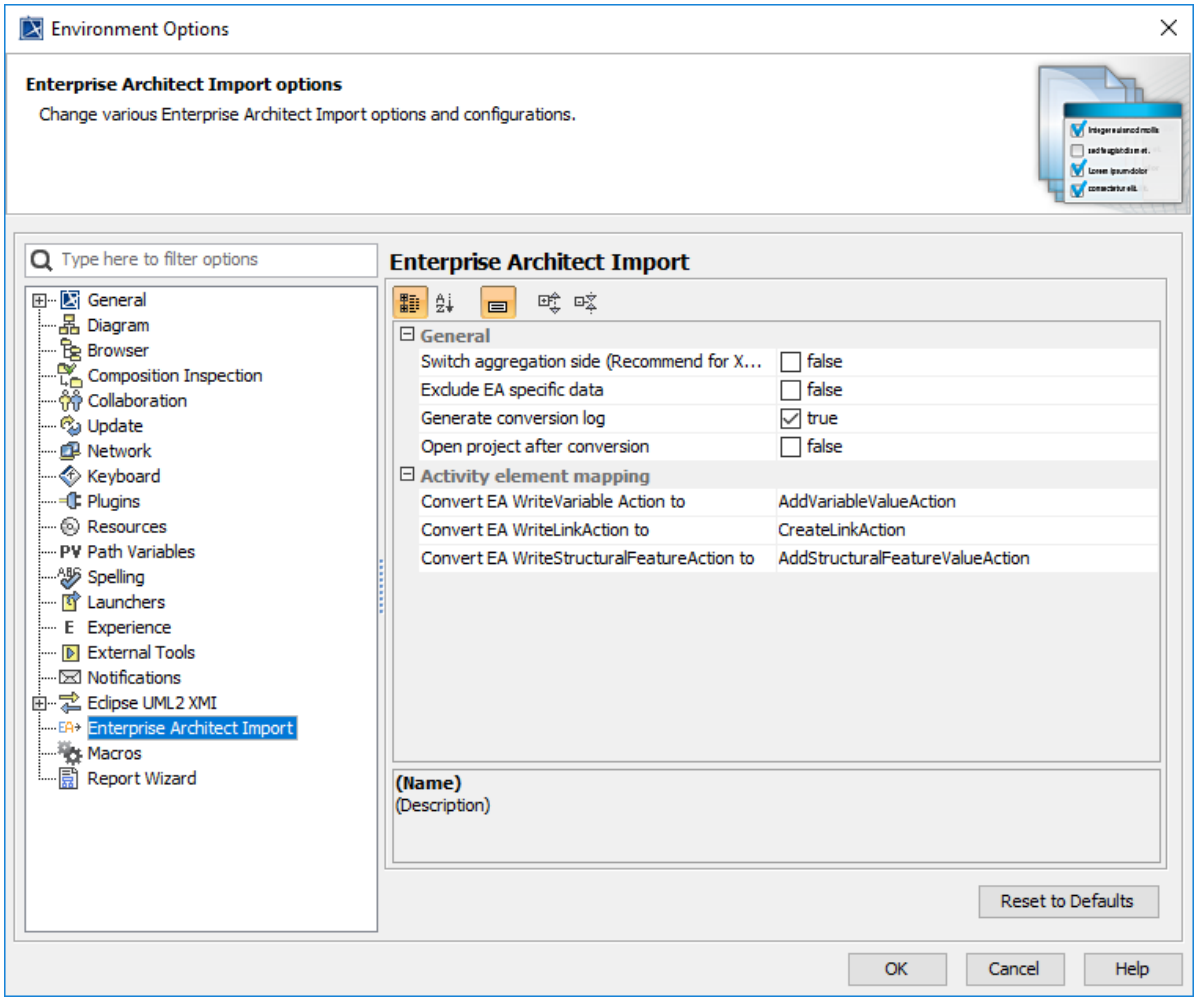
- Customizing the conversion options(see page 119)

Customizing the conversion options

In addition to the options that you have in the **Import Enterprise Architect File** dialog, several other conversion options exist in the modeling tool, and you can specify it in the **Environment Options** dialog(see page 65).

To customize Enterprise Architect Import options in the **Environment Options** dialog

1. On the main menu, click **Options** > **Environment** to open the **Environment Options** dialog.
2. In the options group list, select **Enterprise Architect Import**.



3. The transformation options are classified into two groups: **General** and **Activity element mapping**:
    - Options in the **General** group are the same as those in the **Import Enterprise Architect Files** dialog.
    - The **Activity element mapping** group provides options to convert the EA elements to other element types as it is described in the following table:

Option	Description
Convert EA WriteVariableAction to	To convert EA WriteVariableAction to either AddVariableValueAction or RemoveVariableValueAction.
Convert EA WriteLinkAction to	To convert EA WriteLinkAction to either CreateLinkAction or DestroyLinkAction.

Option	Description
Convert EA WriteStructuralFeatureAction to	To convert EA WriteStructuralFeatureAction to either AddStructuralFeatureValueAction or RemoveStructuralFeatureValueAction.

Transforming EA Specific Data

On this page

- Constraints(see page 122)
- Requirements(see page 123)
- Scenarios(see page 123)
- Files(see page 124)
- Requirements (external)(see page 124)
- Changes(see page 124)
- Issues(see page 125)

In addition to UML data, each EA-exported XMI contains EA-specific information. The Enterprise Architect Import can transform this information into UML elements with the stereotypes applied if you include EA-specific data before importing the XMI file. This data includes:

- Constraints: name, description, type, weight, and status.
- Requirements: name, description, type, status, difficulty, priority, and last update.
- Scenarios: name, description, type, and weight.
- Files: file path type.
- Requirements (External): type, status, difficulty, priority, last update, created, and note.
- Changes: type, status, difficulty, priority, last update, created, and note.
- Issues: type, status, difficulty, priority, last update, created, and note.

Note
You can access and specify the EA information in the **Property** dialog in EA.

To include EA-specific data in the transformation process, a set of stereotypes and tag definitions are created as the EA Profile.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- EA Profile [EA_Profile.mdzip]
  - Relations
  - EAActor [Class]
    - -base_Class : UML Standard Profile::UML2 Metamodel::Class [1]
  - «» EAChange [Class]
    - -status : String [1]
    - -type : String [1]
    - -difficulty : String [1]
    - -priority : String [1]
    - -last update : String [1]
    - -base_Class : UML Standard Profile::UML2 Metamodel::Class [1]
    - -created : String [1]
  - «» EAConstraint [Constraint]
    - -type : String [1]
    - -weight : String [1]
    - -status : String [1]
    - -base_Constraint : UML Standard Profile::UML2 Metamodel::Constraint [1]
  - «» EADiagram [Diagram]
    - Relations
      - Generalization[EA Profile::EADiagram -> UML Standard Profile::MagicDraw Profile::InvisibleStereotype]
    - -base_Diagram : UML Standard Profile::UML2 Metamodel::Diagram [1]
    - -Version : String [1]
  - «» EAIssue [Class]
    - -base_Class : UML Standard Profile::UML2 Metamodel::Class [1]
    - -status : String [1]
    - -type : String [1]
    - -difficulty : String [1]
    - -priority : String [1]
    - -last update : String [1]
    - -created : String [1]
  - «» EARequirement [Class]
    - -base_Class : UML Standard Profile::UML2 Metamodel::Class [1]
    - -name : String [1]
    - -description : String [1]
    - -type : String [1]
    - -status : String [1]
    - -difficulty : String [1]
    - -priority : String [1]
    - -last update : String [1]
    - -created : String [1]
  - «» EAScenario [Element]
    - -base_Element : UML Standard Profile::UML2 Metamodel::Element [1]
    - -name : String [1]
    - -type : String [1]
    - -weight : String [1]
  - «» entity [Lifeline]
    - -base_Lifeline : UML Standard Profile::UML2 Metamodel::Lifeline [1]

The EA Profile in a treeview.

Constraints

Each EA constraint will be transformed into a UML constraint (see page 689)and «EAConstraint» will be applied to the constraint. The properties of an EA constraint will be mapped either to the properties of

a UML constraint or to the tag values of «EAConstraint». The following table shows the constraint mapping details.

EA	Modeling tool by NoMagic
name	The name property of a UML constraint.
description	EAConstraint::type tag value.
type	EAConstraint::weight tag value.
weight	EAConstraint::status tag value.
constraint owner	Constrained Element property point to the constraint owner.

Requirements

Each EA requirement will be transformed into a UML Class(see page 666). Because a requirement cannot be created in an element that is the owner of a Class in EA, the transformed requirement will be kept in a separate Package(see page 800), named **EA Requirement**. A Realization (see page 818)will then be created from the owner of the requirement into a transformed requirement. See the following table for details.

EA	Modeling tool by NoMagic
name	EARequirement :: name tag value
description	EARequirement :: description tag value
type	EARequirement :: type tag value
status	EARequirement :: status tag value
difficulty	EARequirement :: difficulty tag value
priority	EARequirement :: priority tag value
last update	EARequirement :: name update value

Scenarios

Each EA scenario will be transformed into a UML Comment(see page 683) and «EAScenario» will be applied to the comment. The properties of a scenario will be mapped either to the properties of each UML Comment or to the tag values of «EAScenario». See the following table for details.

EA	Modeling tool by NoMagic
name	EAScenario::name tag value
description	The Body property of a UML Comment.
type	EAScenario :: type tag value
weight	EARequirement :: weight tag value

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

EA	Modeling tool by NoMagic
subject	An annotated Element property pointing to an EA subject element.

Files

EA can add files to a UML element. The information will be transformed into a Hyperlink (see page 750)in a UML model.

EA	Modeling tool by NoMagic
Local file	File
Web address	Webpage.

Requirements (external)

An EA-created Requirement differs from the one you create as an internal element for each element. EA requirements will appear in the Model Browser(see page 29) and can be pasted on a diagram. Each EA Requirement will be transformed into a Class (see page 666)and «EARequirement» will be applied to the requirement.

EA	Modeling tool by NoMagic
type	EARequirement :: type tag value
status	EARequirement :: status tag value
difficulty	EARequirement :: difficulty tag value
priority	EARequirement :: priority tag value
last update	EARequirement :: last update value
created	EARequirement :: created tag value
note	Documentation

Changes

EA can create a Change and will export it as a Class(see page 666). The Class information will be transformed into the «EAChange» tag values. See the following table for details.

EA	Modeling tool by NoMagic
type	EAChange :: type tag value
status	EAChange :: status tag value
difficulty	EAChange :: difficulty tag value
priority	EAChange :: priority tag value

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

EA	Modeling tool by NoMagic
last update	EAChange :: last update value
created	EAChange :: created tag value
note	Documentation

Issues

EA can create an Issue and will export it as a Class(see page 666). The Issue information will be transformed into the «EAIssue» tag values. See the following table for details.

EA	Modeling tool by NoMagic
type	EAIssue :: type tag value
status	EAIssue :: status tag value
difficulty	EAIssue :: difficulty tag value
priority	EAIssue :: priority tag value
last update	EAIssue :: last update value
created	EAIssue :: created tag value
note	Documentation

Importing Diagrams

Enterprise Architect Import allows you to import diagrams. The imported diagram information includes:

- Geometry Properties(see page 125)
- Color Properties(see page 126)
- Display Properties(see page 126)

Geometry Properties

The geometry properties imported to a UML model are:

- Positions on a diagram (for shape elements)
- Width and height (for shape elements)
- Path break points (for link elements)

> **Note**
> Other display properties can override geometry information. For example, if an imported element width is shorter than the required width to display text on the element, the width will be adjusted automatically.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Color Properties

Color properties will be imported along with the diagrams to a modeling tool developed by NoMagic. Each color property has a different name in a modeling tool. The following table shows color properties mapping.

EA	Modeling tool by NoMagic
Background	Fill color
Border color (for shape element)	Pen color
Font color	Text color
Line color (for link element)	Pen color

Display Properties

Display properties in EA can be categorized into three groups: shape, link, and diagram. Only those properties corresponding to MagicDraw will be imported. For example, the **Show Diagram Details** property in EA will be imported as the **Show Diagram Info** property(see page 553) in a modeling tool developed by NoMagic.

Special Transformation

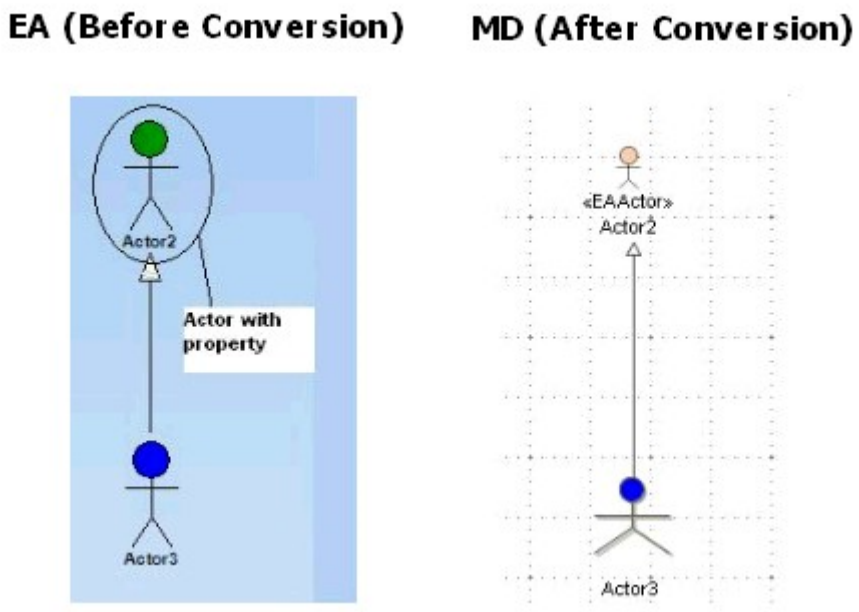
An EA-exported XMI contains both non-standard UML elements and elements that can break the XMI schema. To retain standard UML elements and keep the XMI schema intact, the Enterprise Architect Import applies specific transformation rules. The following sections describe how each model element is transformed, enabling you to import a complete XMI model conforming to UML standards.

Use Case diagram elements

This page describes all Use Case diagram(see page 300) elements.

Actor with Properties
An Actor (see page 646)with properties will be transformed into a Class (see page 666)with the «EAActor» stereotype.

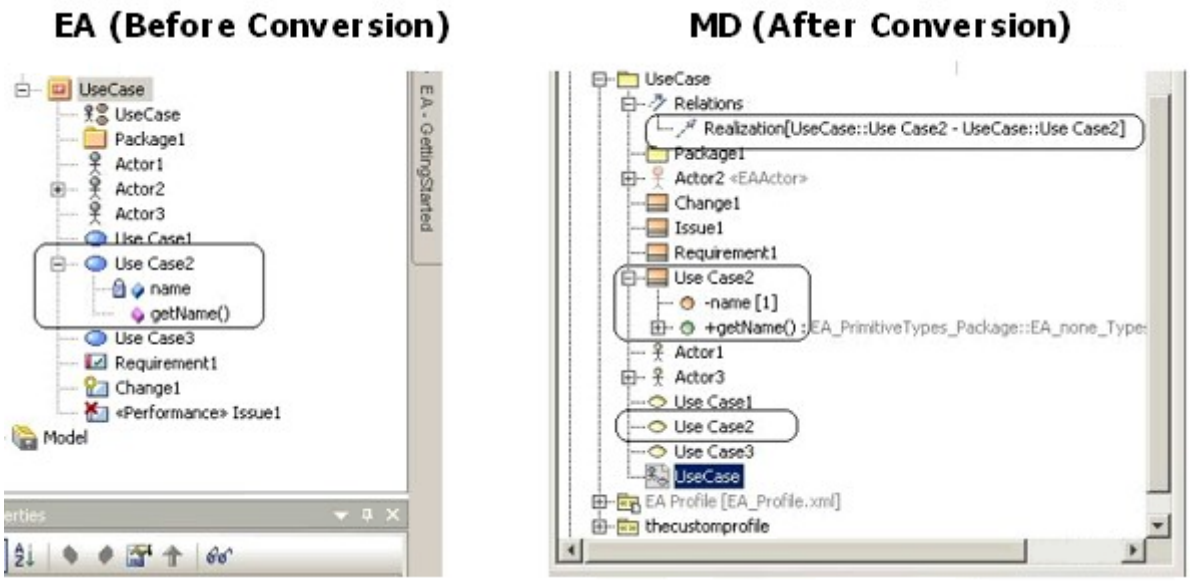
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



An actor with properties.

> **Note**
> An Actor that has been converted to a Class with «EAActor» will not display some properties (such as Fill Color) because the stereotype image will be shown instead.

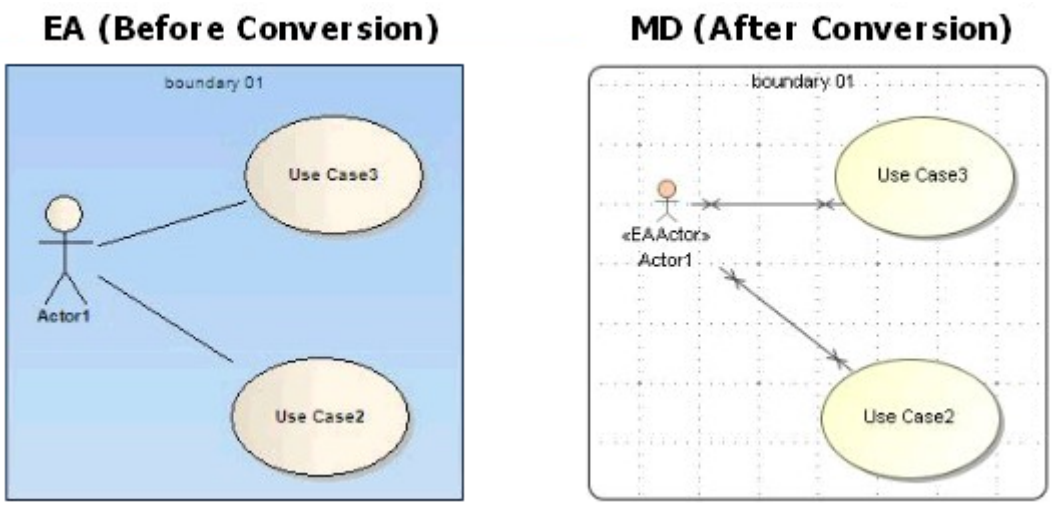
Use Case with Invalid Inner Elements
A NestedClassifier, ownedComment, ownedRule, ownedAttribute, or ownedOperation cannot be an inner element of a uml:UseCase. It will be moved to a new created realized Class(see page 666).



A Use Case with invalid inner elements.

Boundary

A boundary in EA will be converted into a rectangle with rounded corners. The boundary can contain inner elements. Unlike the rectangular boundary, the boundary in EA will take all inner elements with it whenever it is moved.



Boundaries.

Related pages

- Use Case diagram(see page 300)

Activity diagram elements

On this page

- Activity(see page 128)
- Activity diagram(see page 130)
- Object as the inner element of an Activity(see page 130)
- Synch Node(see page 132)
- Activity Parameter(see page 132)
- Exception Handler(see page 133)
- ObjectFlow(see page 134)
- ExpansionRegion(see page 135)
- InterruptibleActivityRegion(see page 135)
- Swimlane(see page 136)
- StructuredActivity(see page 137)
- InterruptFlow(see page 138)
- ExpansionNode(see page 139)

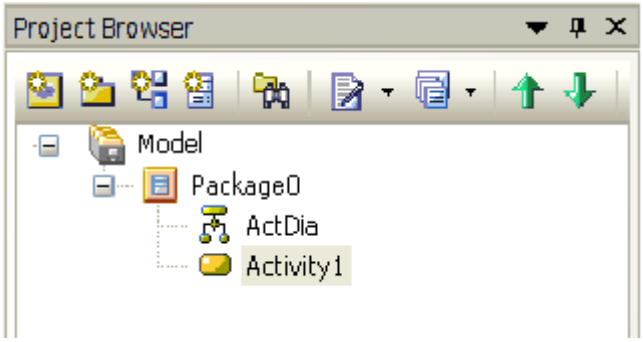
Activity

You can directly place an Activity element in EA as an element view on an Activity diagram(see page 311). However, this behavior conflicts with MagicDraw (CEA and CSM) and UML notation. In MagicDraw, CEA, or CSM, if you drag an Activity from the containment tree to an Activity diagram, a new CallBehaviorAction (see page 633)view will be created and the **Behavior** property of the CallBehaviorAction will be set to the Activity. This same behavior will be used in the import process.

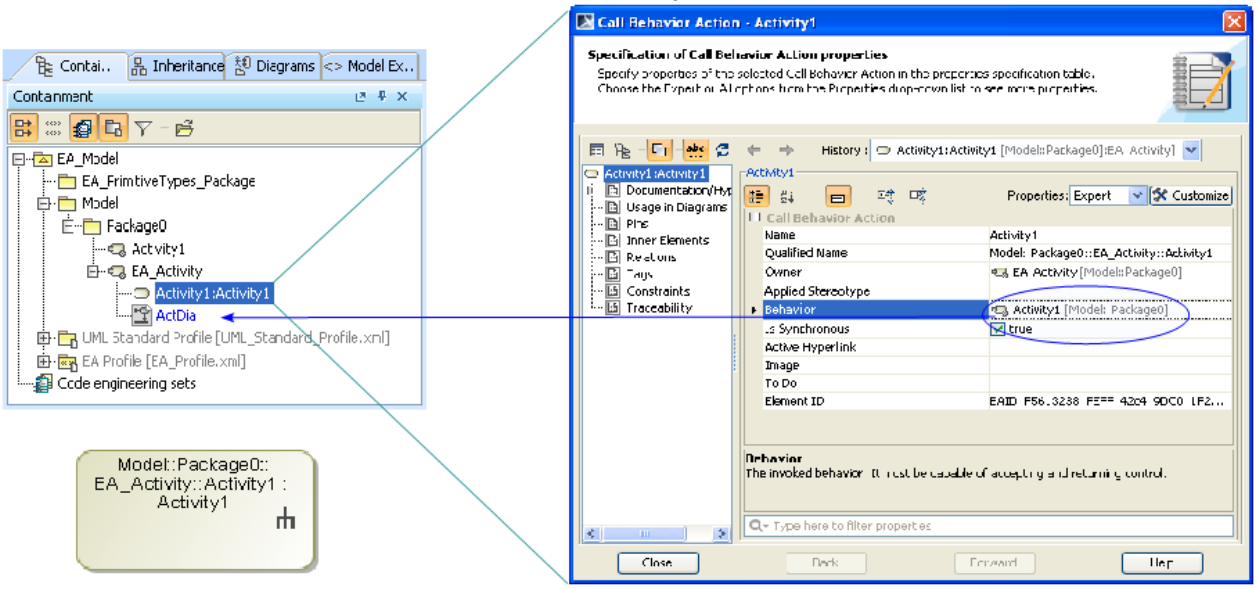
An Activity element created in EA and placed on an Activity diagram will be transformed into two elements: Activity, and CallBehaviorAction elements. Both elements will have the same name and will be linked through the property of a CallBehaviorAction element called *Behavior*.

After transforming the element, the following transformation message opens:

```
Updated element <xmi:id>: A new CallBehaviorAction was created and its
Behavior was set to the element.
```



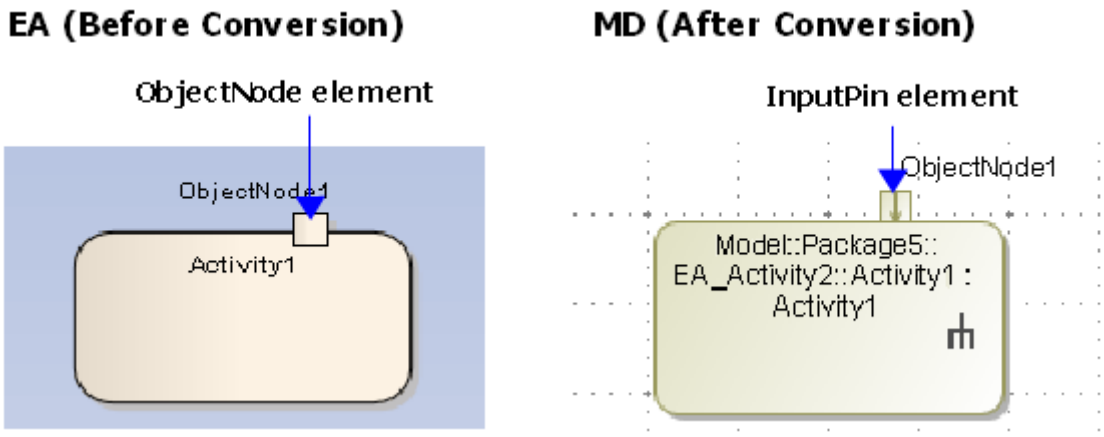
EA activity.



Activity with a New CallBehavior in MagicDraw.

> **Note**
> The EA Activity and CallBehaviorAction elements have similar characteristics in that you can attach a control flow to it and others. EA has its own CallBehaviorAction element.

Additionally, any ObjectNode elements attached to the Activity element will be transformed into InputPin elements and attached to the newly created CallBehaviorAction element.



An ObjectNode converted into an InputPin.

Activity diagram

Every Activity Diagram element from EA will be placed inside an Activity element that has the same name.

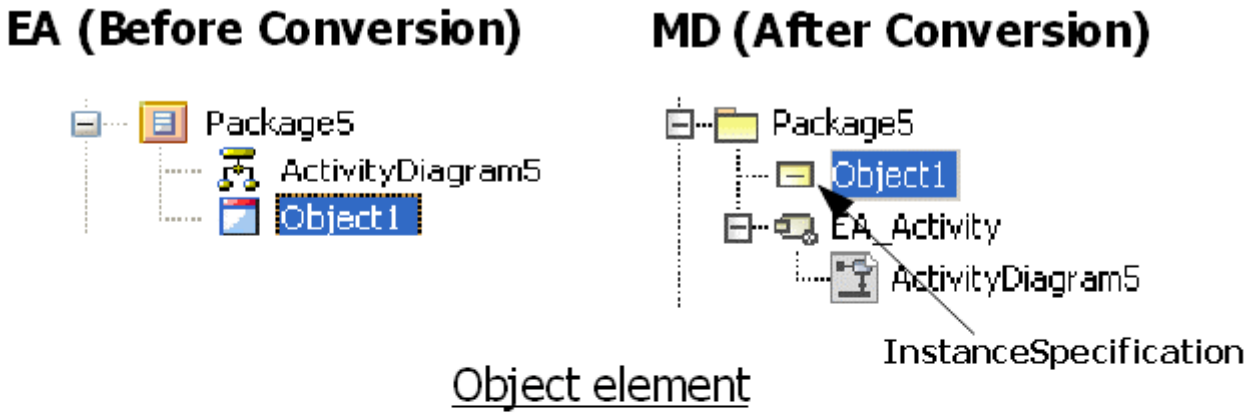
Activity diagram in the MagicDraw Containment tree.

> **Note**
> In MagicDraw, CEA, or CSM, every Activity diagram element must be placed inside an Activity element that has the same name. However, this is not the case in EA.

Object as the inner element of an Activity

Object elements inside an Activity element in EA will be removed.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



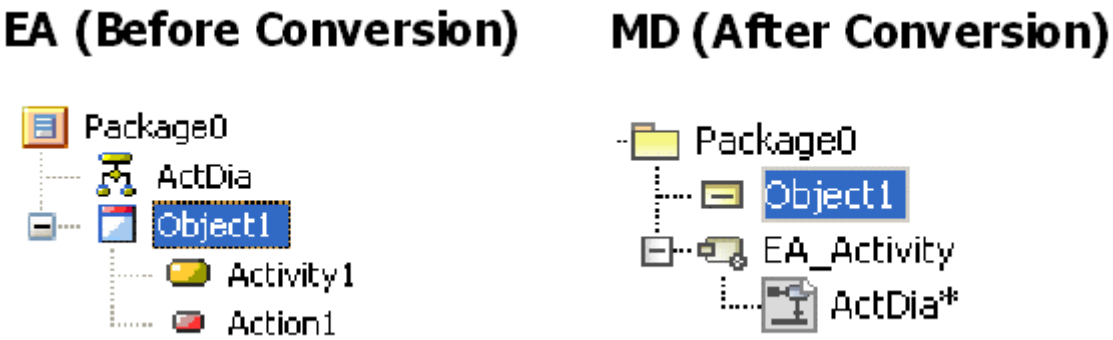
Object element

Object element in Activity element

Object elements transformation.

> **Note**
> Object elements in MagicDraw, CEA, or CSM have their XMI types defined as *uml:CentralBufferNode*. However, those in EA have their XMI types defined as *uml:InstanceSpecification*, which do not belong to an Activity diagram.

An Object element containing any ActivityDiagram-related elements will be removed.



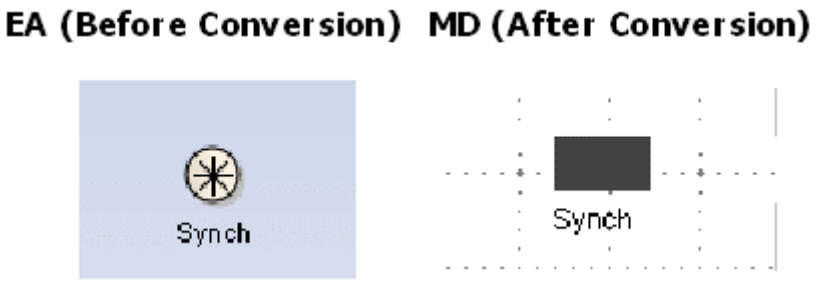
Object Containing Activity-related Elements.

> **Note**
> In MagicDraw, CEA, or CSM, an Object element (CentralBufferNode) is not allowed to contain elements other than comments and hyperlinks.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Synch Node

A Synch element in EA will be transformed into a Join (see page 847)element in MagicDraw, CEA, or CSM. It will look exactly like a Fork/Join element.

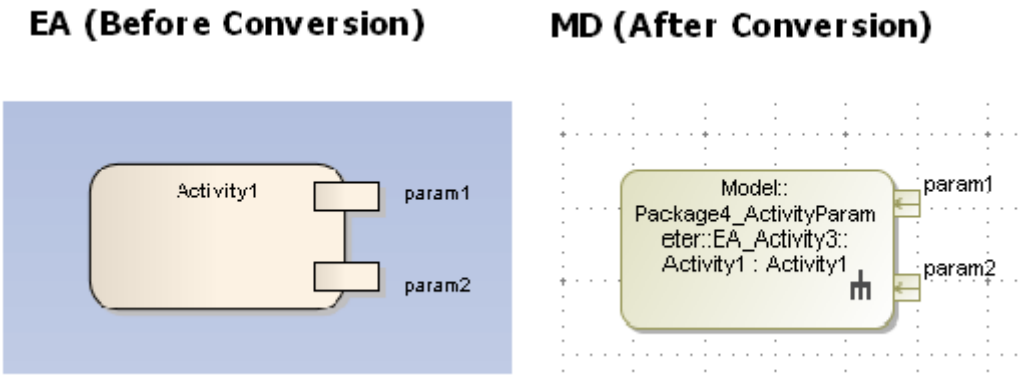


Synch element transformation.

> **Note**
> A MagicDraw (CAE or CSM) Fork/Join element (whose type is *uml:ForkNode*) can be used to construct either a Fork and Join node in an Activity diagram. The JoinNode element (whose type is *uml:JoinNode*) is allowed to be placed in the Activity diagram, but the element's image will be displayed as the Fork/Join element's default image.

Activity Parameter

If you create an ActivityParameter element, MagicDraw (CEA or CSM) will automatically create an ActivityParameterNode (see page 783)element to represent it. Every ActivityParameterNode element in EA will be transformed into a Pin (see page 805)element.

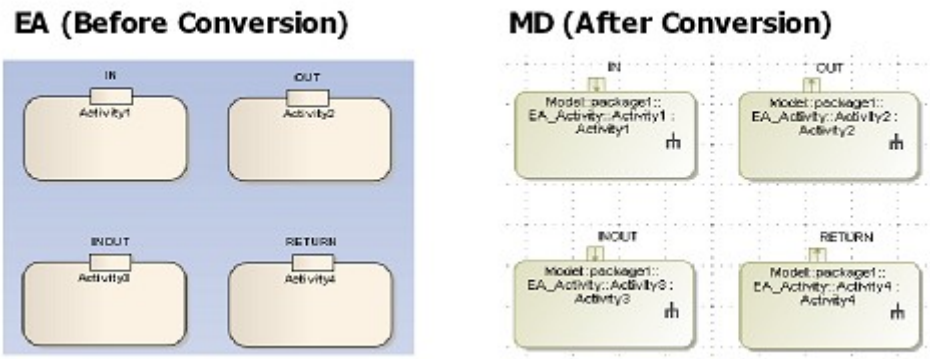


Activity parameter node.

You can specify four parameter types for each Activity Parameter element: in, out, inout, and return.

The ActivityParameterNode element of an ActivityParameter element whose parameter type is either 'in' or 'inout' will be transformed into an InputPin element. The ActivityParameterNode element of an ActivityParameter element whose parameter type is either *out* or *return* will be transformed into an OutputPin element.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Activity parameter type.

> **Note**
> Usually, if you specify the parameter type of an ActivityParameter element as *inout*, two Pin elements (InputPin and OutputPin elements) will be created for the element. Since EA will only create one ActivityParameterNode element, this element will be transformed into an InputPin element.

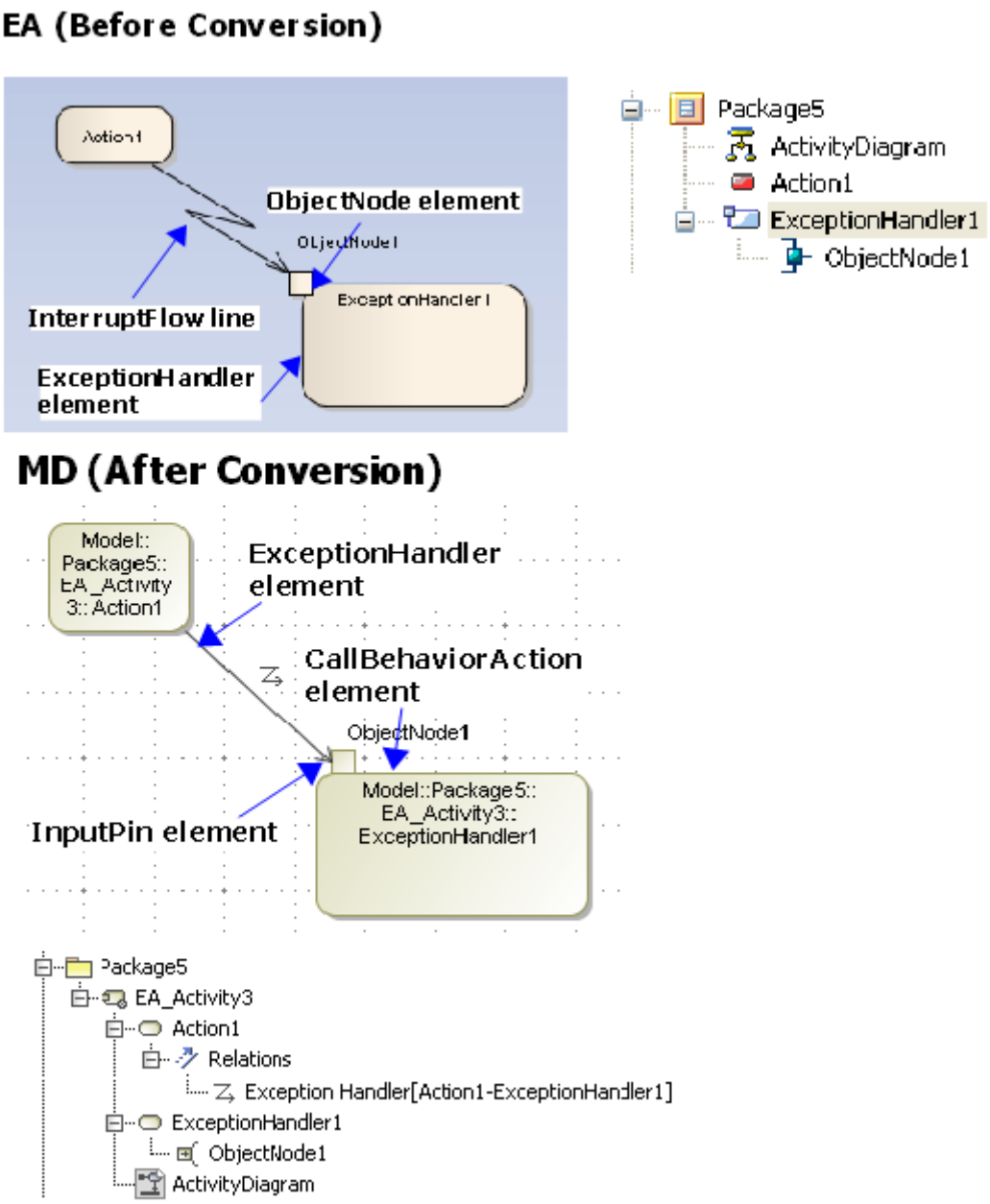
Exception Handler

The Exception Handler element in EA differs from the UML's ExceptionHandler(see page 711). This EA element will be transformed into a CallBehaviorAction element. Any ObjectNode element attached to it will be transformed into an InputPin element and any InterruptFlow line will be transformed into an ExceptionHandler line in MagicDraw, CEA, or CSM.

After completing the transformation, the following transformation messages open:

```
Updated element <xmi:id>: EA ExceptionHandler is transformed to an
CallBehaviorAction with and input pin.

Updated element <xmi:id>: EA InterruptFlow was transformed to an
ExceptionHandler.
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



ExceptionHandler.

ObjectFlow

An ObjectFlow (see page 794)line whose ends are not attached to any of the following elements will be transformed into a ControlFlow.

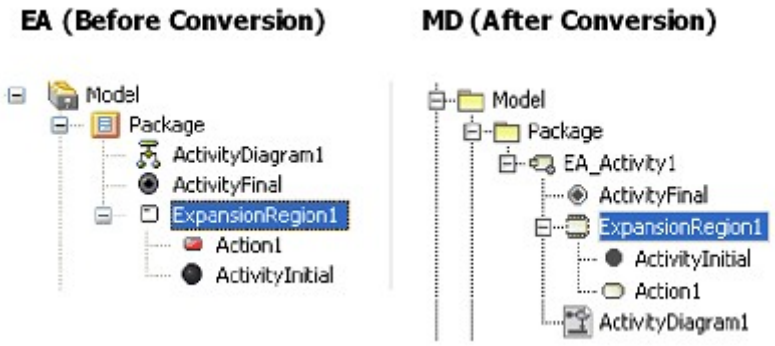
- InputPin
- OutputPin
- ObjectNode
- CentralBufferNode
- DataStoreNode

After completing the transformation, the following transformation message opens:

```
Updated element <xmi:id>: uml:ObjectFlow updated to uml:ControlFlow.
```

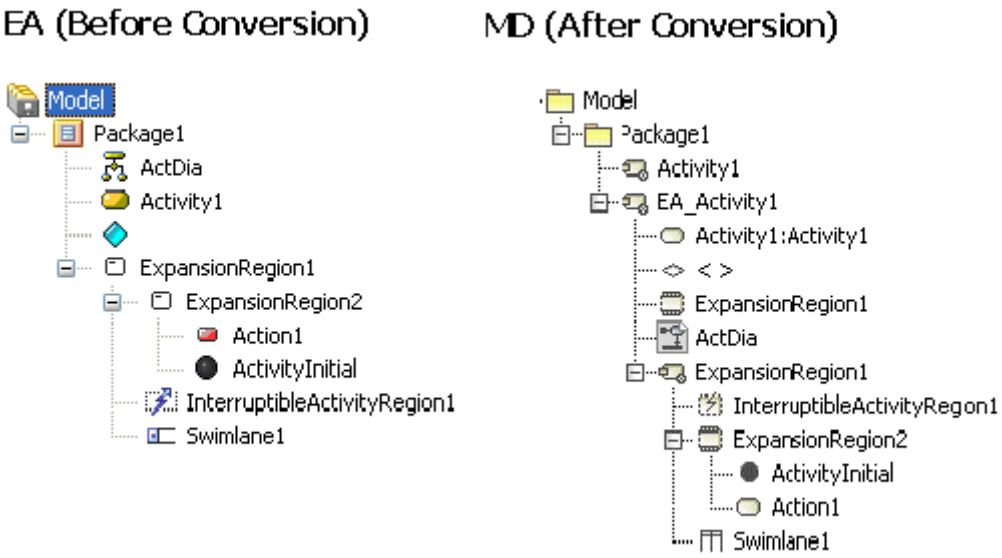
ExpansionRegion

Most of the elements placed inside any ExpansionRegion elements in EA will stay in their original place.



ExpansionRegion tree view.

However, if there is any Activity, Swimlane, InterruptibleActivityRegion, StructuredActivityNode, LooNode, SequenceNode, ConditionalNode, or other ExpansionRegion contained within an ExpansionRegion, it will be placed within a dummy Activity element. The created dummy will have the same name and will be placed at the same level as the ExpansionRegion (see page 788)element.



Nested ExpansionRegion tree view.

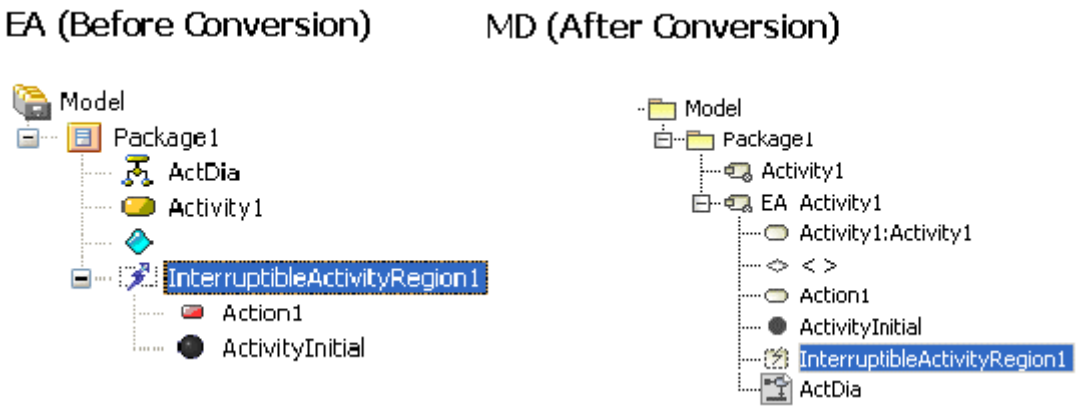
After completing the transformation, the following transformation message opens:

```
Updated element <xmi:id>: ExpansionRegion cannot contain some inner elements.

An Activity with the same name as the ExpansionRegion was created to contain
inner elements.
```

InterruptibleActivityRegion

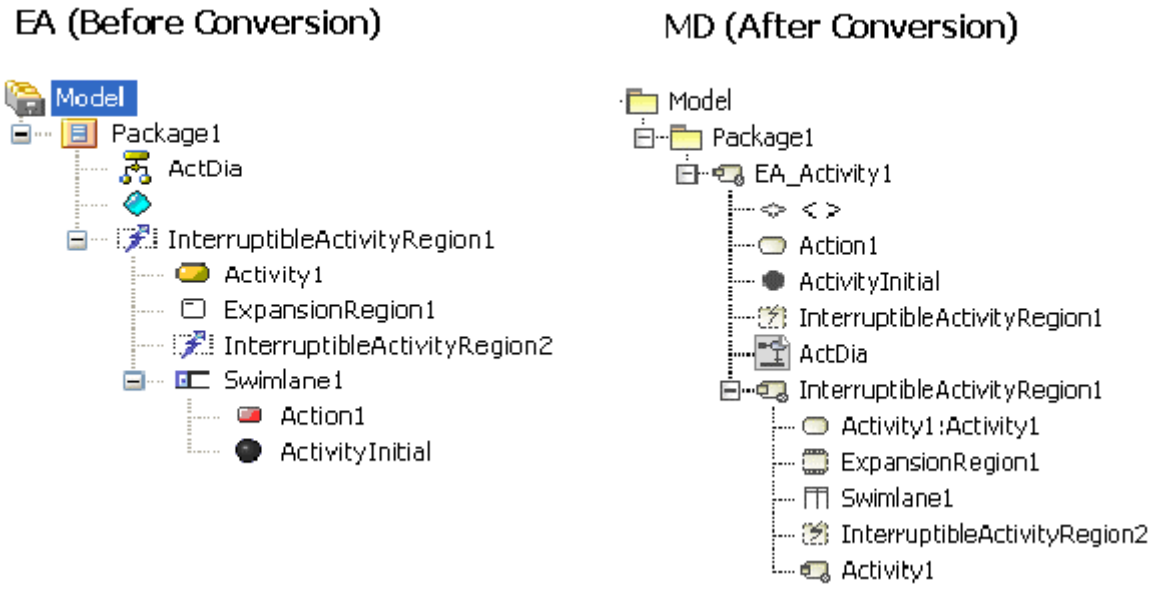
Most of the elements placed inside an InterruptibleActivityRegion element in EA will be placed at the same level as the InterruptibleActivityRegion element.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



InterruptibleActivityRegion tree view.

However, if there is any Activity, Swimlane, ExpansionRegion, StructuredActivityNode, LoopNode, SequenceNode, ConditionalNode, or other InterruptibleActivityRegion contained within an InterruptibleActivityRegion, it will be placed within a dummy Activity. The created dummy will have the same name and will be placed at the same level as the InterruptibleActivityRegion.



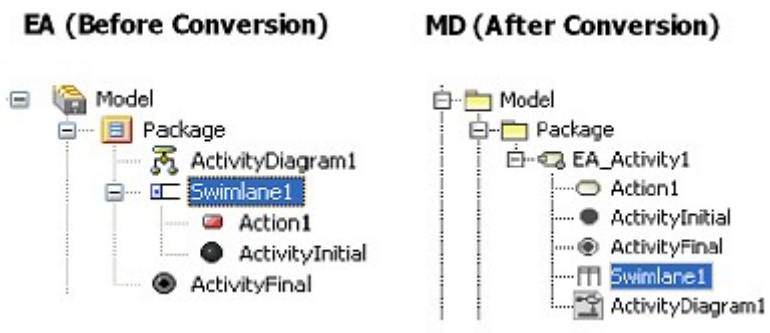
Nested InterruptibleActivityRegion tree view.

After completing the process, the following transformation message opens:

```
Updated element <xmi:id>: InterruptibleActivityRegion cannot contain some
inner elements.

An Activity with the same name as the InterruptibleActivityRegion was created
to contain inner elements.
```

Swimlane

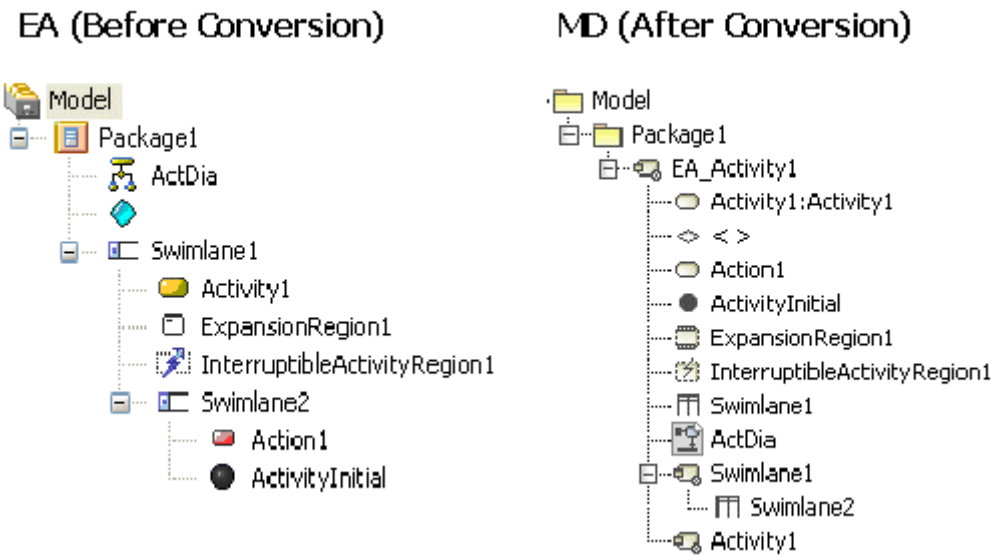
Most of the elements placed inside any Swimlane element in EA will be placed at the same level as the Swimlane (see page 861)element.



Swimlane tree view.

A dummy Activity will also be created to hold any other Swimlanes that it may contain. The dummy activity will have the same name and will be placed at the same level as the Swimlane.

If two or more Swimlanes are nested together, then every element (except Swimlane element) contained within either of them will be placed at the same level as the Swimlane topping the nested-Swimlane-elements hierarchy.



Nested Swimlane tree view.

After completing the process, the following transformation message opens:

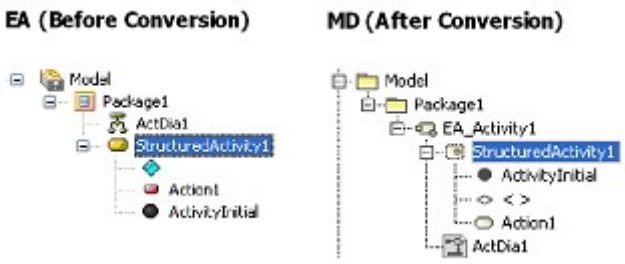
```
Updated element <xmi:id>: Swimlane cannot contain some inner elements. The XMI
structure was fixed.
```

StructuredActivity

Four elements are classified as Structured Activity elements in EA:

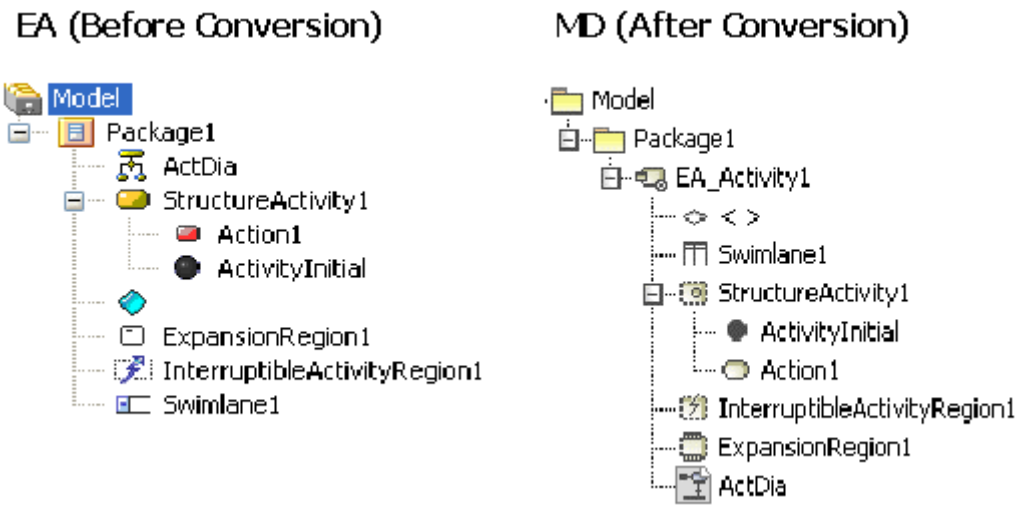
- StructuredActivityNode element
- LoopNode element
- SequenceNode element
- ConditionalNode element

Most of the elements placed inside any StructuredActivityNode, LoopNode, SequenceNode, or ConditionalNode elements in EA will stay in their original place.



StructuredActivityNode tree view.

However, if there is any Activity, Swimlane, InterruptibleActivityRegion, ExpansionRegion, or another StructuredActivity contained within a StructuredActivity, it will be placed within a dummy Activity. The created dummy will have the same name, and will be placed at the same level as the StructuredActivity.



Nested StructuredActivityNode tree view.

After completing the process, the following transformation messages will open, depending on the Structured Activity elements involved:

- Updated element <xmi:id>: StructuredActivityNode cannot contain some inner elements. An Activity with the same name as the StructuredActivityNode was created to contain inner elements.
- Updated element <xmi:id>: ConditionalNode cannot contain some inner elements. An Activity with the same name as the ConditionalNode was created to contain inner elements.
- Updated element <xmi:id>: LoopNode cannot contain some inner elements. An Activity with the same name as the LoopNode was created to contain inner elements.
- Updated element <xmi:id>: SequenceNode cannot contain some inner elements. An Activity with the same name as the SequenceNode was created to contain inner elements.

InterruptFlow

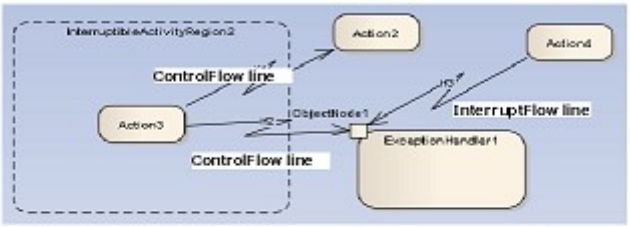
In some cases, EA InterruptFlows are ControlFlow lines. Their image will be displayed as the InterruptFlow line in the Activity diagram. An InterruptFlow is not a ControlFlow line if the InterruptFlow line is drawn from one element in an InterruptibleActivityRegion to another outside the InterruptibleActivityRegion. In an XMI file, this line will be imported as a ControlFlow line, and its image will be changed to that of the ControlFlow line.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

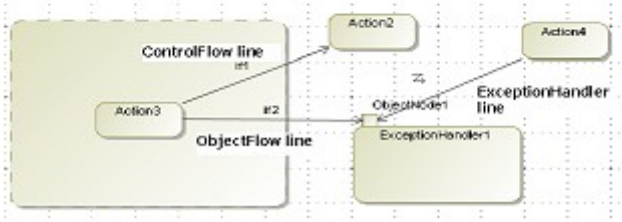
However, if either end of the line is any of the following elements, it will be transformed into an ObjectFlow line.

- InputPin element
- OutputPin element
- ObjectNode element
- CentralBufferNode element
- DataStoreNode element

EA (Before Conversion)



MD (After Conversion)



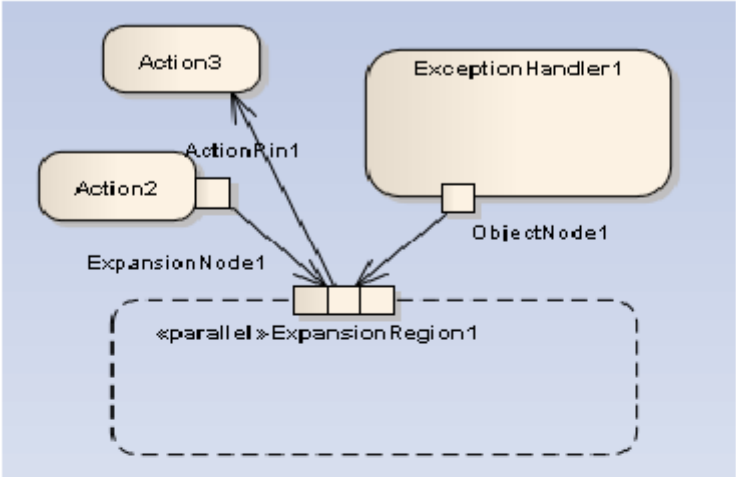
InterruptFlow.

ExpansionNode

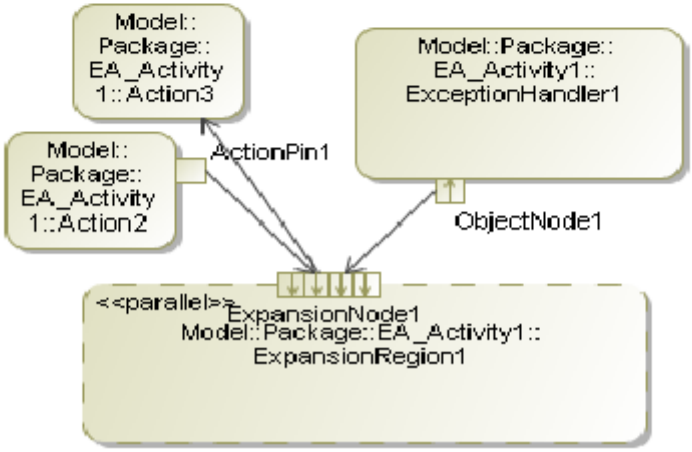
An ExpansionNode is a Pin which can only be contained within an ExpansionRegion and will be imported like any other Pin elements. However, if an ExpansionNode in EA is created inside another element rather than an ExpansionRegion, that particular ExpansionNode will not be imported.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

EA (Before Conversion)



MD (After Conversion)



ExpansionNode.

Related pages

- Activity diagram(see page 311)

Sequence diagram elements

On this page

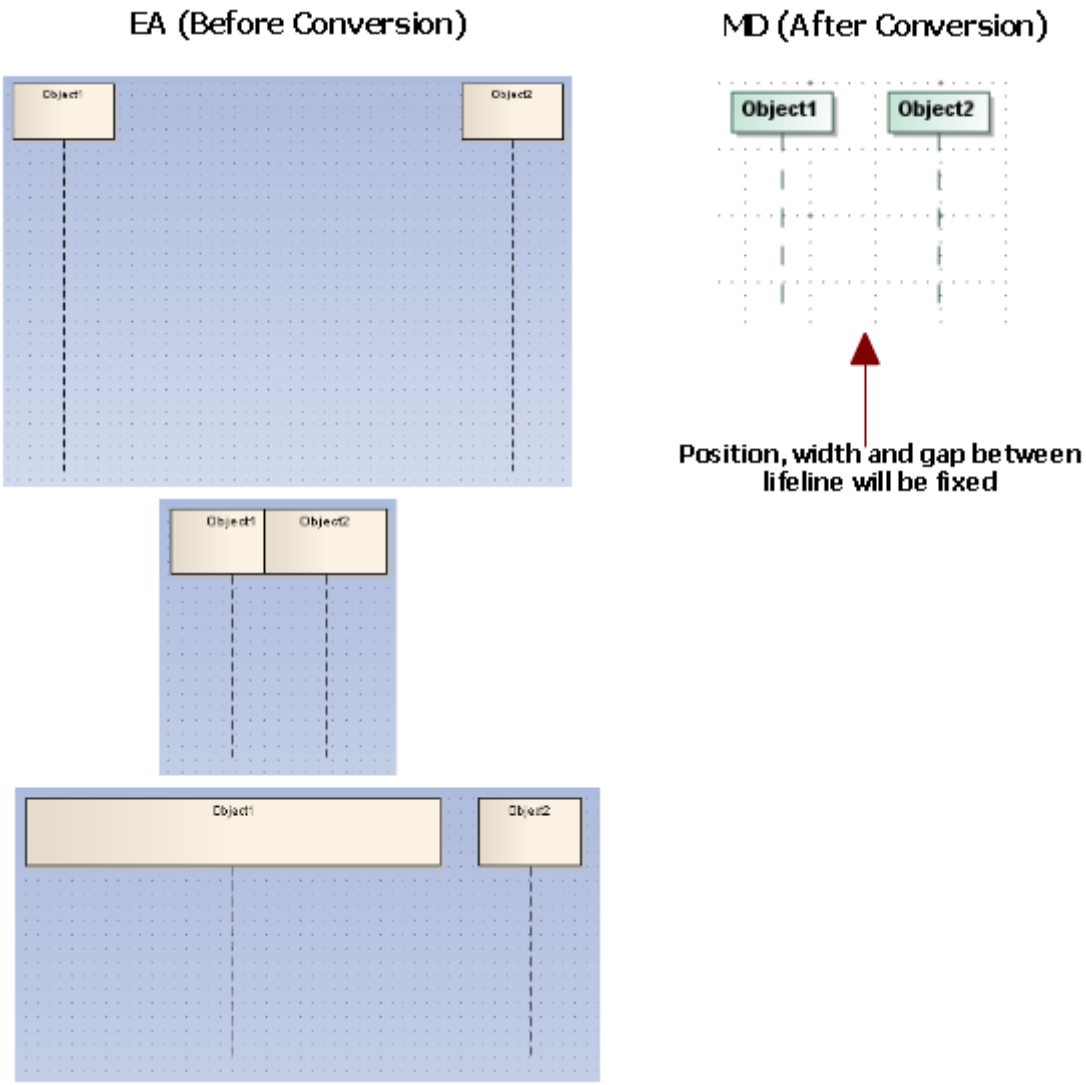
- Lifelines(see page 141)
- Gate(see page 143)
- Endpoints(see page 147)
- Delete Messages(see page 147)
- Branch Messages(see page 149)
- Boundaries, Controls, and Entities(see page 150)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Actors(see page 151)
- General Ordering(see page 152)
- State Invariant(see page 153)
- Continuation(see page 154)
- Diagonal Message(see page 154)
- Synchronous Message behavior(see page 156)
- Asynchronous Message behavior(see page 159)
- Return Message behavior(see page 160)
- Activation options(see page 161)

Lifelines
All of the EA Lifelines will be imported, but a part or port within a Lifeline will be transformed into a new separate Lifeline(see page 746).

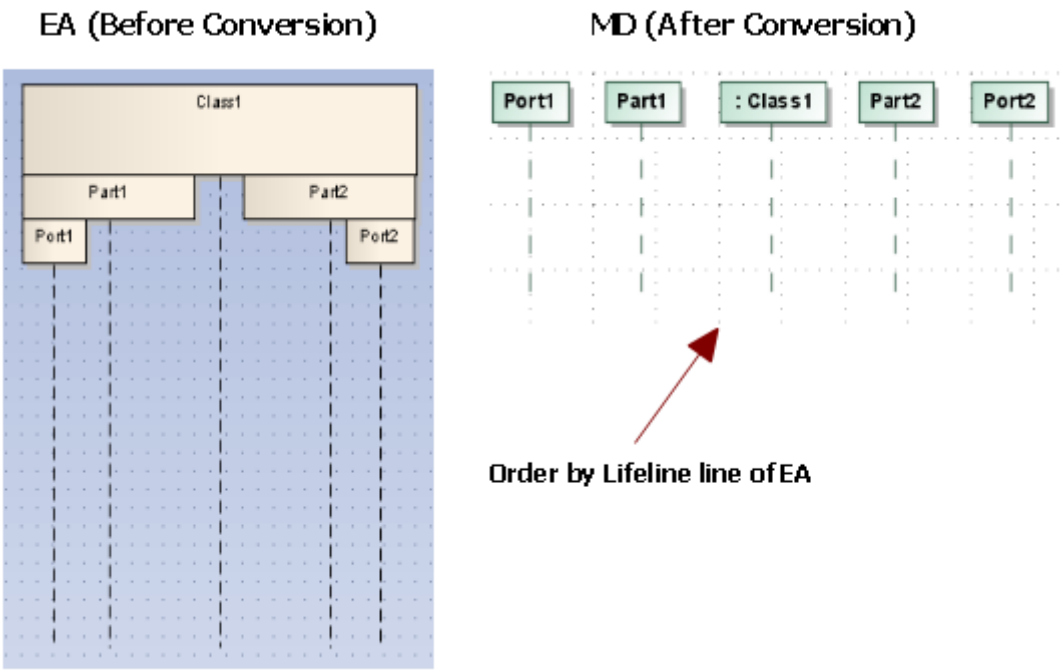
Gaps between Lifelines
The position and width of any Lifeline created in EA will not be imported. Every Lifeline will be given a fixed value and position in MagicDraw, CEA, or CSM. MagicDraw, CEA, or CSM will place the first Lifeline on the left-hand side of the diagram and the second Lifeline on the right-hand side next to the first one. The length of the gap between the Lifelines will be fixed.



Gaps between Lifelines.

Lifelines Arrangement

A Lifeline can be nested within another component, such as a Part (see page 804)or Port(see page 809). If this is the case, every component nested within the Lifeline and the Lifeline itself will be drawn separately. They will be arranged in order depending on the position of their Lifeline lines.

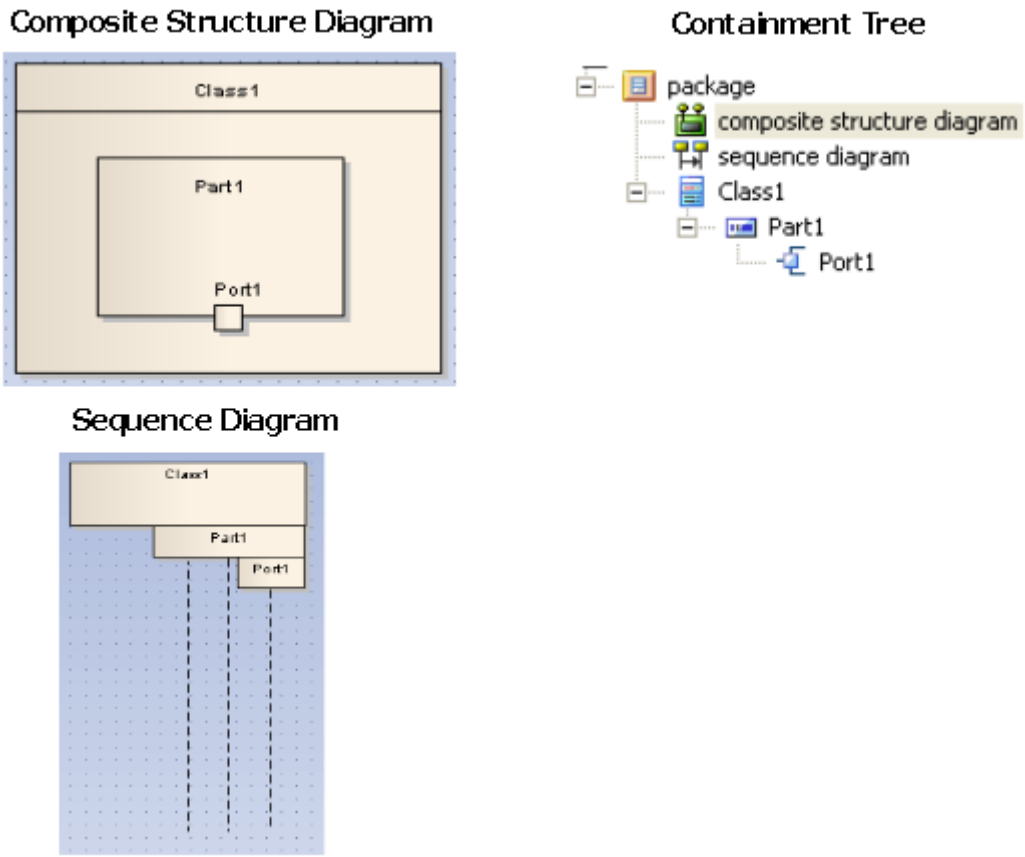


Lifelines arrangement.

Class, Part, and Port

When represented as Lifelines, Classes, Parts, and Ports have different characteristics from the others. They will be bundled according to their relationships. A Composite Structure diagram(see page 325) provides one convenient way to create a Class(see page 666), Part, or Port.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

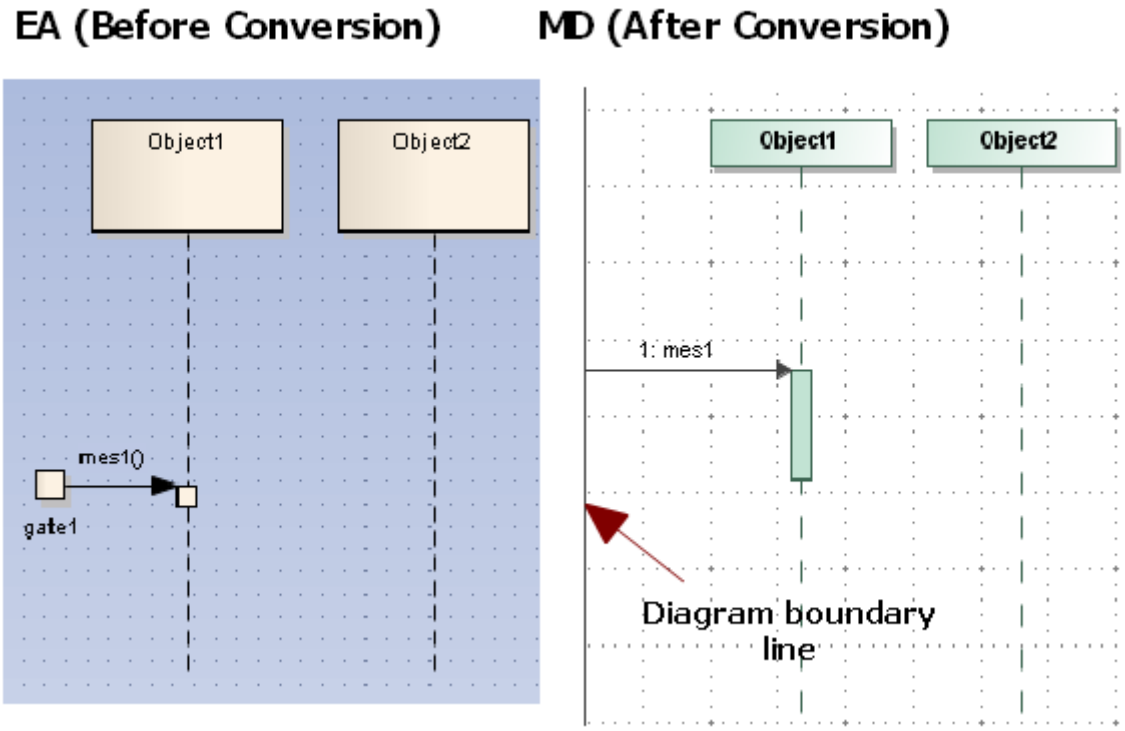


Class, Part, and Port.

Gate

A Sequence Message whose tail is connected to a Gate and head connected to a Lifeline in EA will be transformed into a Sequence Message(see page 756) with its tail connected to one of the boundary lines of the diagram in which it is contained. The Gate (see page 719)itself will be removed.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

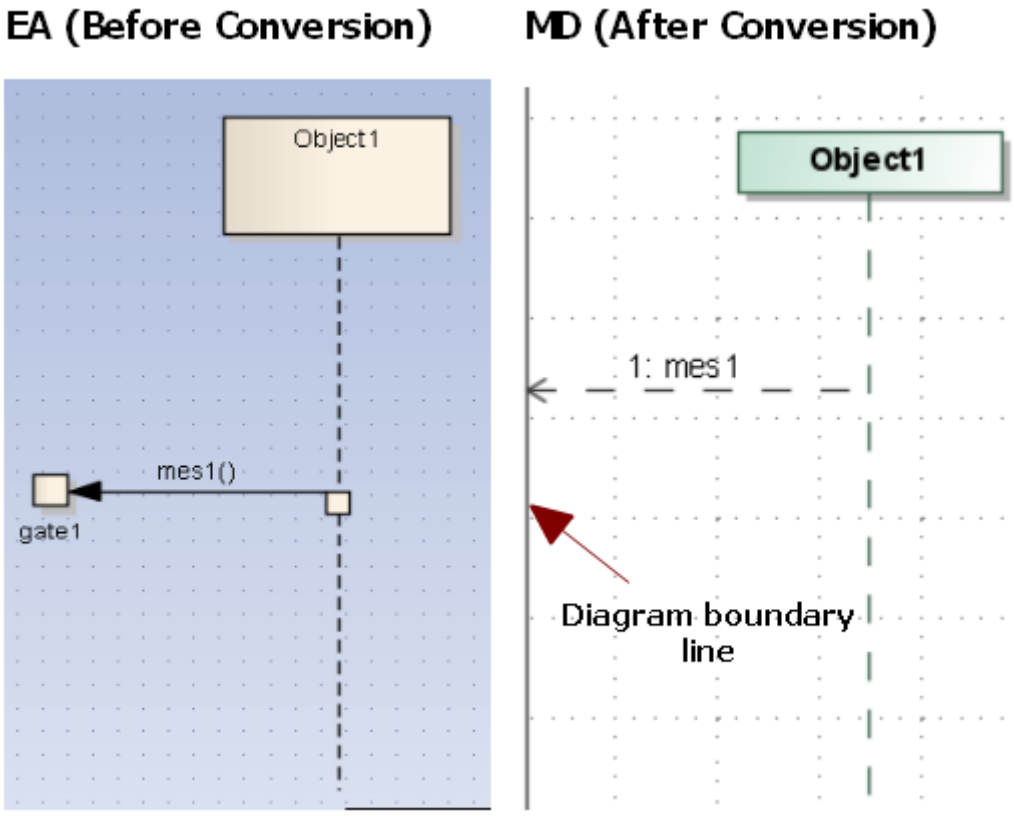


Gate.

Once the Gate has been removed and the transformation process has been completed, the following transformation message will open:

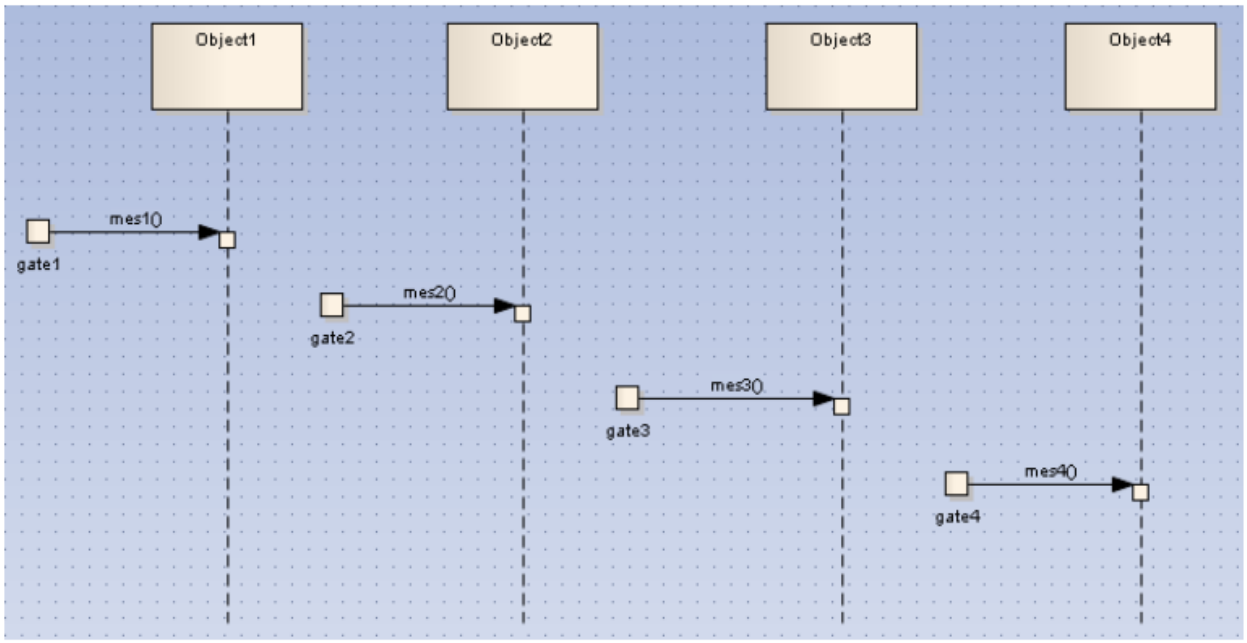
```
Removed element <xmi:id>: uml:Gate.
```

A Sequence Message whose head is connected to a Gate and tail connected to a Lifeline line in EA will be transformed into a Reply Message with its tail connected to one of the boundary lines of the diagram in which it is contained. The Gate itself will be removed

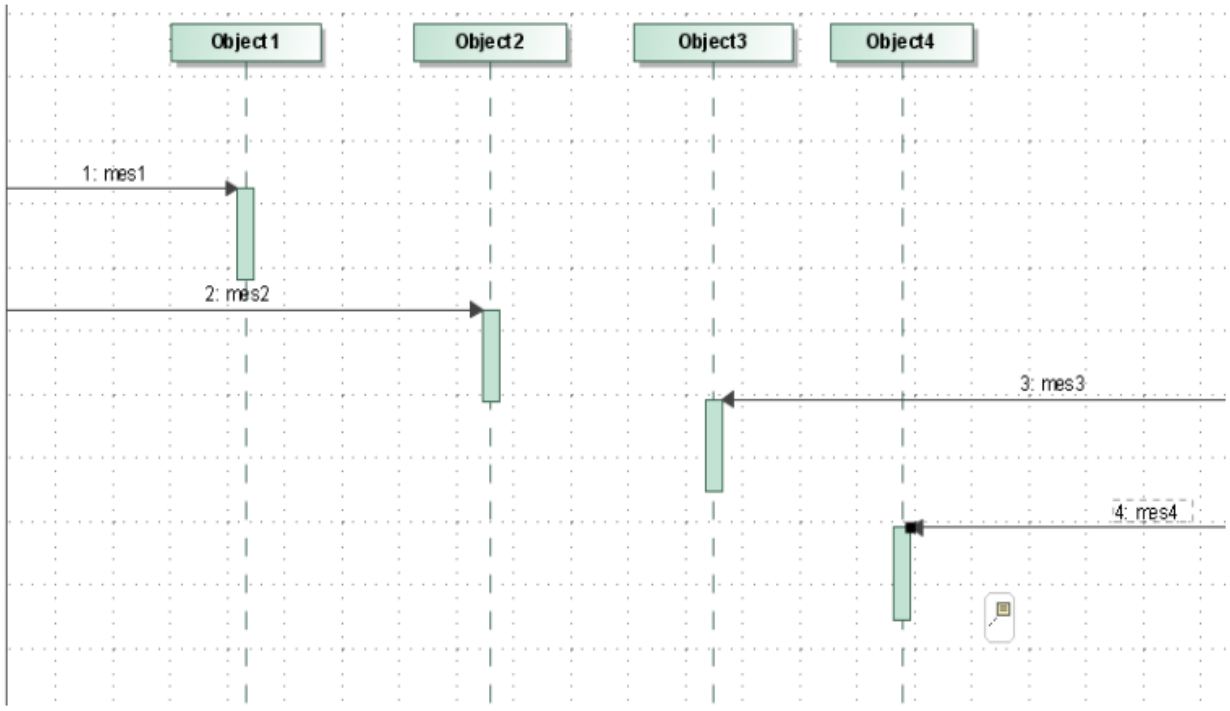


Gate (EA) and Reply Message (MagicDraw, CEA, or CSM).
The Diagonal Message and Reply Message will be connected to the nearest diagram boundary.

EA (Before Conversion)



MD (After Conversion)



Gate (EA) and Direction of Diagonal Message (MagicDraw, CEA, or CSM).

Anything else connected to a Gate will also be removed, except the tail of a Sequence Message. For example, if the head of a Sequence Message is connected to a Gate, the Message will be removed. If the tail of a Message is connected to a Gate, but the head is connected to anything other than a Lifeline line, the Message will be removed.

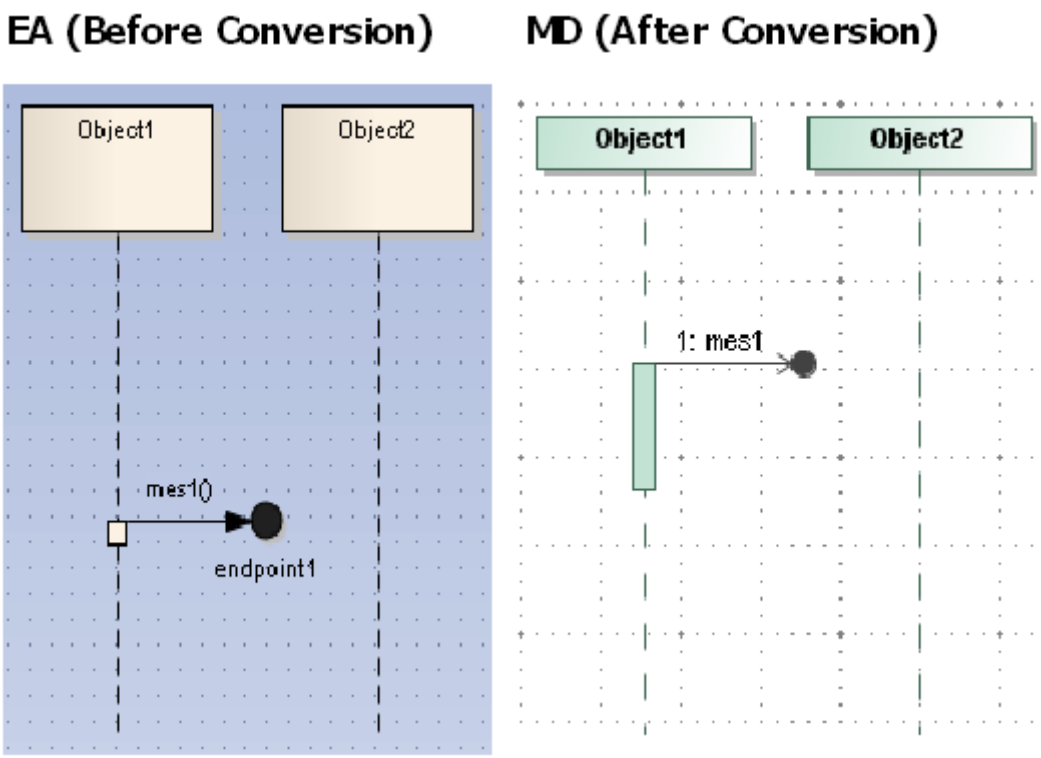
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Endpoints

A Sequence Message whose head connected to an Endpoint and tail connected to a Lifeline line in EA will be transformed into a Lost Message. The Endpoint element itself will be removed. Once the process has been completed, the following transformation message will open:

```
Removed element <xmi:id>: EndPoint.
```

A Message whose head connected to a Lifeline line and tail connected to an Endpoint will be transformed into a Found Message.



Endpoint.

Anything else connected to an endpoint will also be removed, except the head of a Sequence Message. For example, if the tail of a Sequence Message is connected to an endpoint, the Message will be removed.

If the head of a Message is connected to an endpoint, but the tail is connected to anything other than a Lifeline line, the Message will be removed. Once the process has been completed, the following transformation message will open:

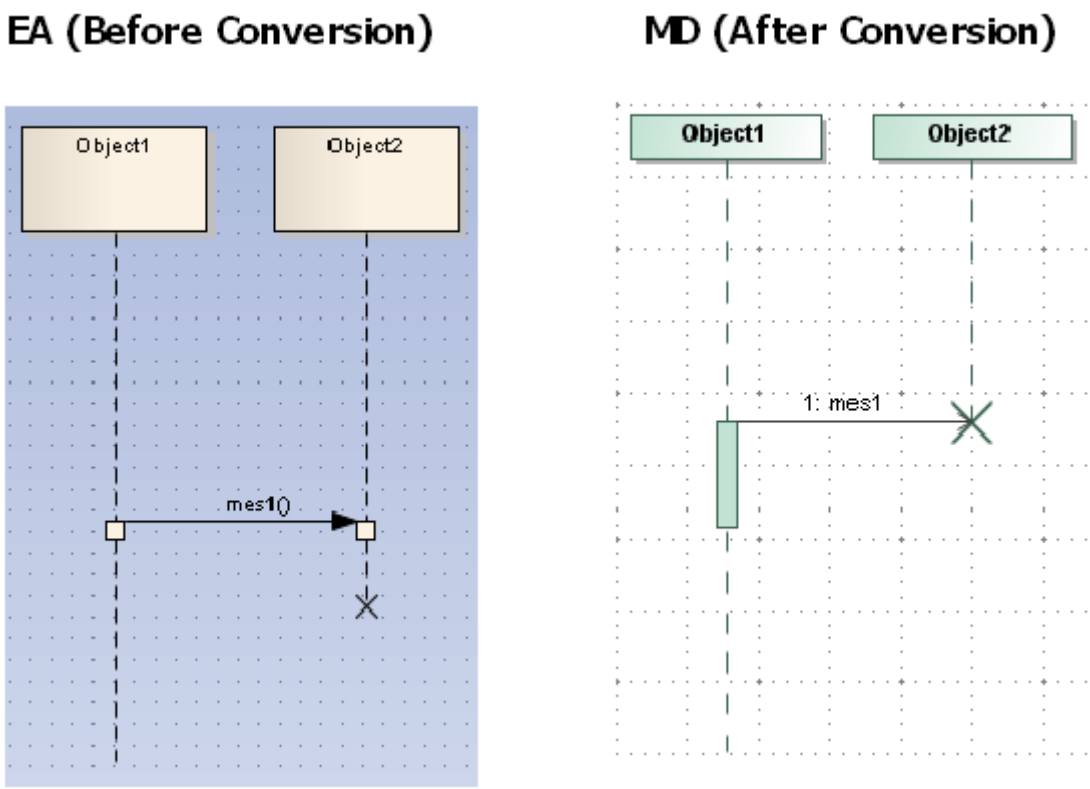
```
Removed element <xmi:id>: Invalid Message. Source and Target of the Mes- sage
are not connected to any Lifeline.
```

Delete Messages

A Sequence Message whose property *Lifecycle* is set to *Delete* (that causes the Lifeline targeted by the Message to end at some range after the point of contact), will be transformed into a Delete Message. The Lifeline connected to its head will end at the point of contact, and all Messages will be removed after that point of contact of the Lifeline. MagicDraw, CEA, or CSM will report the following transformation message once each process has been completed:

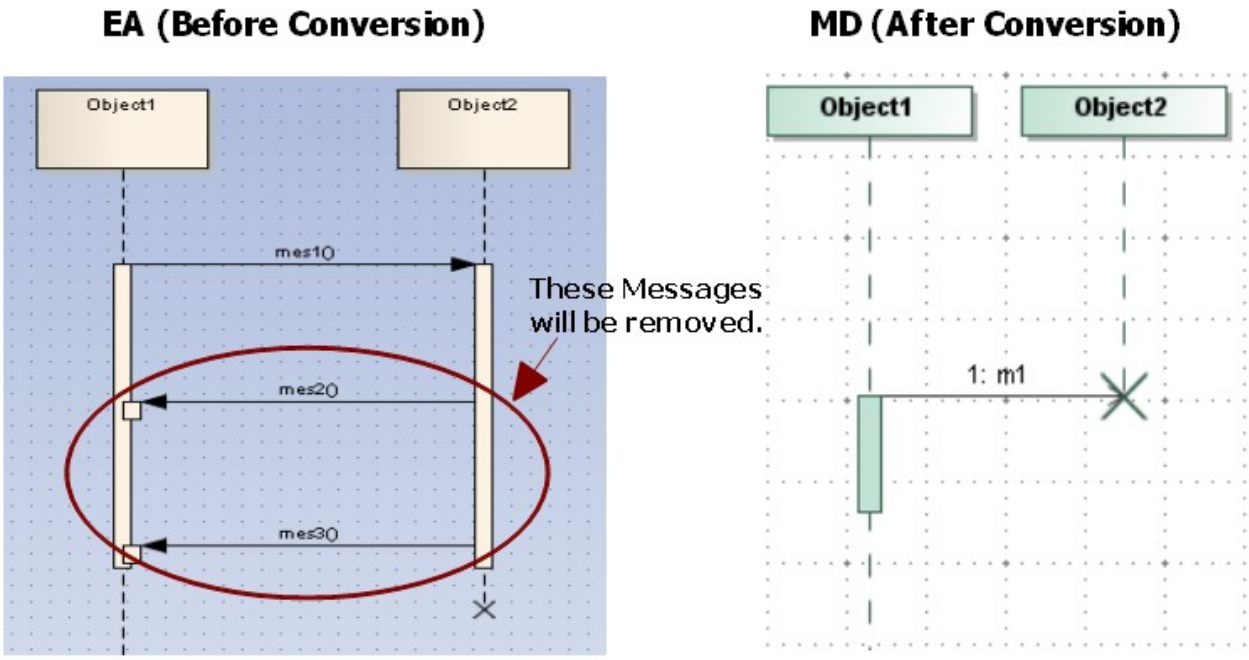
```
Removed element <xmi:id>: Message is under Delete Message.
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Delete message.

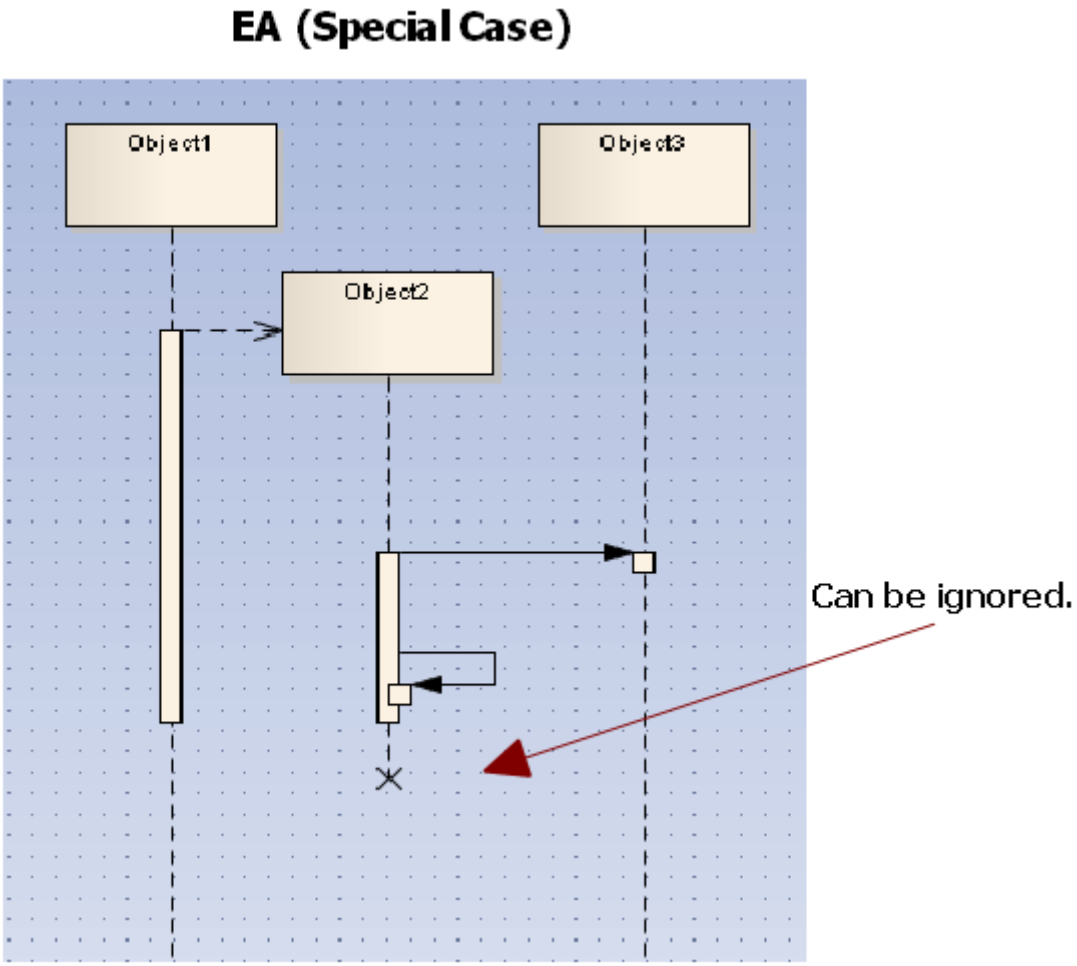
Under certain circumstances, a Lifeline connected to the head of a Delete Message does not end at the point of contact. Instead, an Activation will be created and started from the point of contact. The Lifeline will then end at the end of the Activation. However, that Activation should not interact with any Message.



Special case of delete message.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Under certain circumstances, an **X** sign (normally drawn after a Delete Message) in EA will be drawn on a Lifeline line whose Lifeline is being pointed by a Create Message. If this is the case, the **X** sign has no significant meaning and can be ignored.



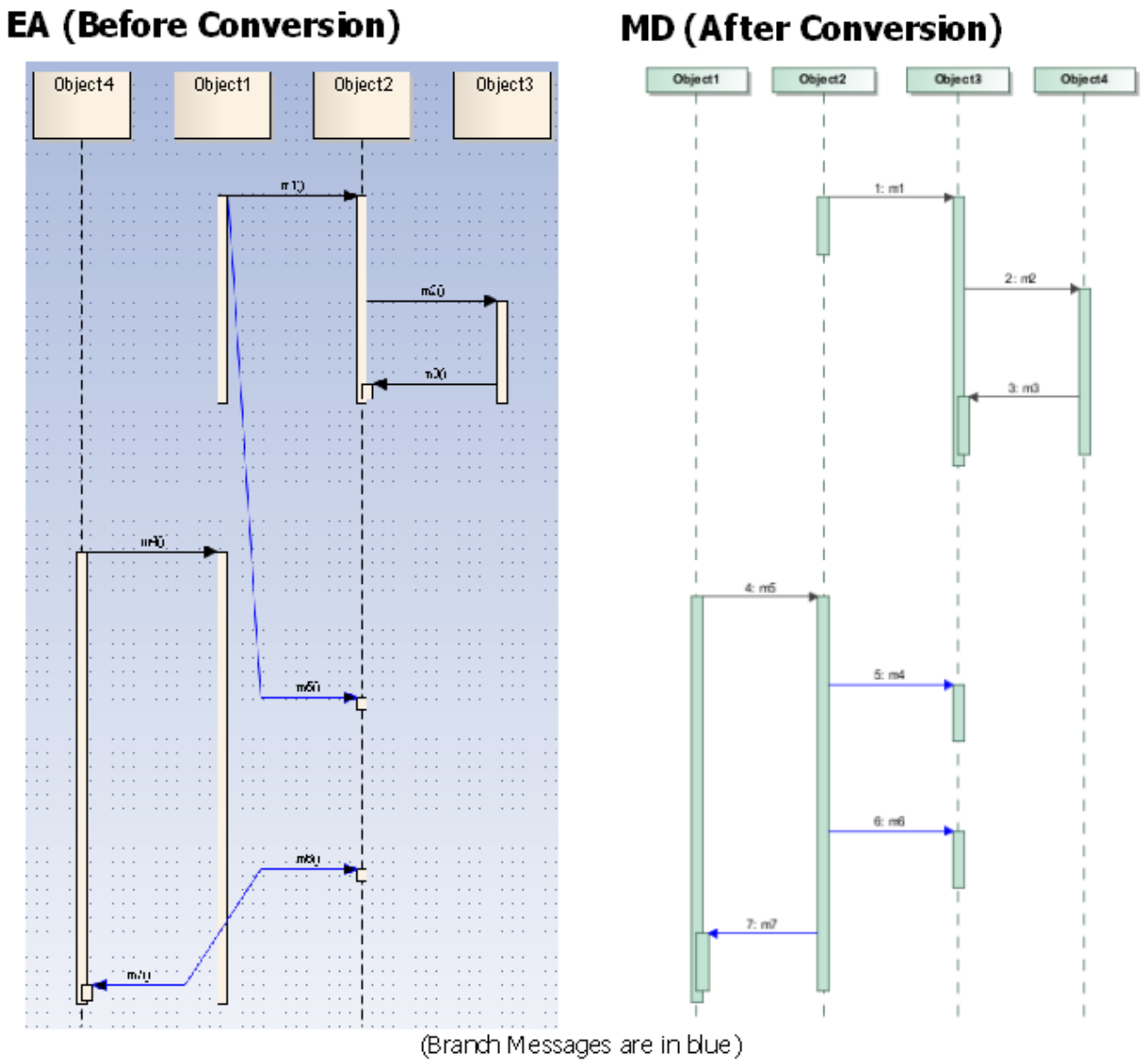
The X sign.

Branch Messages

If the **Branch with previous Message** option of a Sequence Message is enabled, the tail of the message connect to the tail of its previous Message on the same Lifeline line. The message will be called *Branch Message* from that time on.

Every Branch Message in EA will be transformed into a normal Message in MagicDraw, CEA, or CSM.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Branch Message.

After the transformation process has been completed, the following transformation message will open:

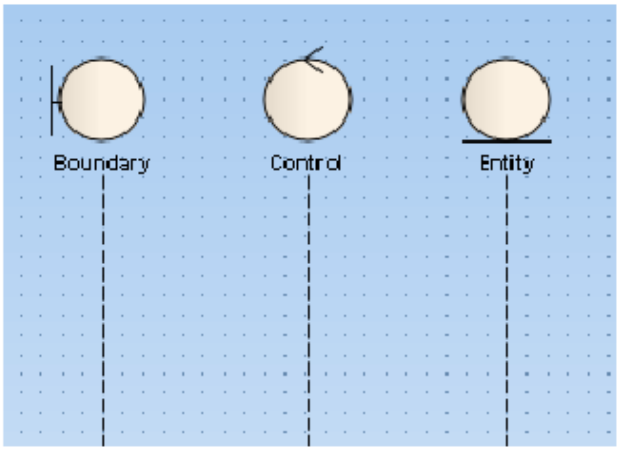
```
Updated element <xmi:id>: Branch Message updated to regular Message.
```

Boundaries, Controls, and Entities

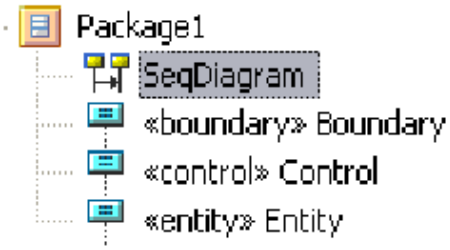
The Boundary, Control, and Entity elements in a Sequence diagram context in EA are Lifeline elements whose stereotype property types are set to *Boundary*, *Control*, or *Entity*. They will be imported like any other normal Lifelines. However, an XMI exported from EA has an «entity» stereotype problem. This means the Lifeline with the «entity» stereotype applied is unable to show the stereotype icon. You can solve this problem by placing an «entity» stereotype in *EA_Profile.xml.* The converted model can then use it instead of the one from EA.

EA (Before Conversion)

Display picture

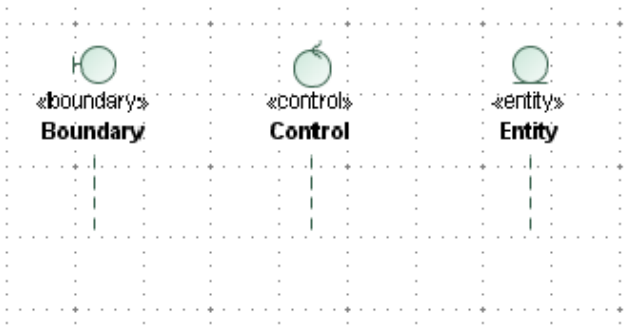


Containment Tree

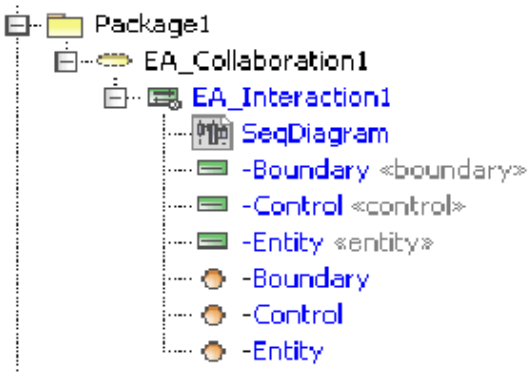


MD (After Conversion)

Display picture



Containment Tree



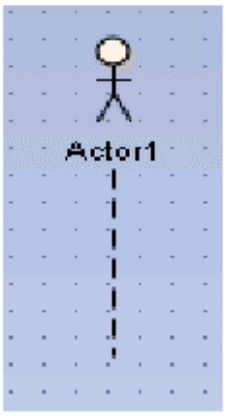
Boundary, Control, and Entity.

Actors

An Actor element in the Sequence diagram context in EA is not a regular Lifeline element. It is a special Lifeline element whose property type is *uml:Actor*. It will be transformed into a regular Actor and a Lifeline element will be created to represent it. The figure below shows an Actor (see page 646)element in the MagicDraw, CEA, or CSM Containment tree.

EA (Before Conversion)

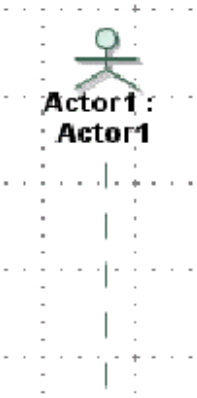
Display picture



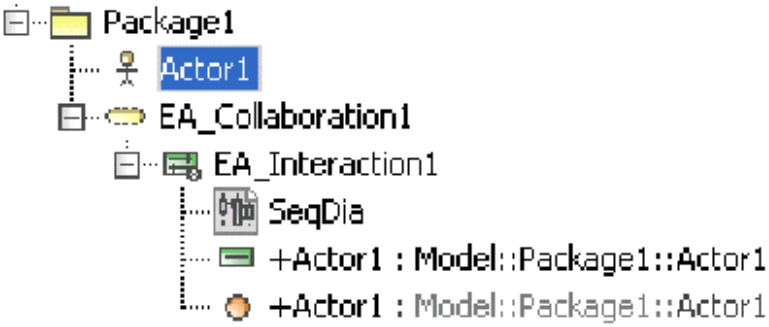
Containment Tree

MD (After Conversion)

Display picture



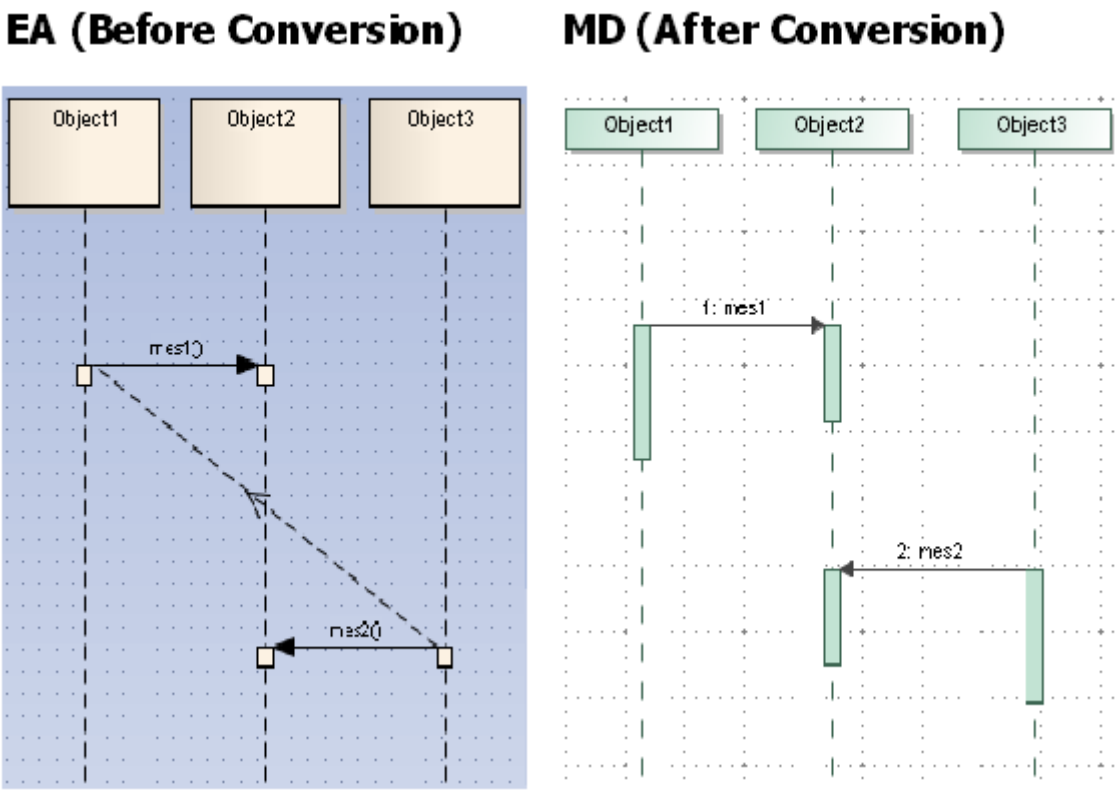
Containment Tree



Actor.

General Ordering

Not every General Ordering element in EA will be imported, as MagicDraw, CEA, or CSM does not support it in the current release.



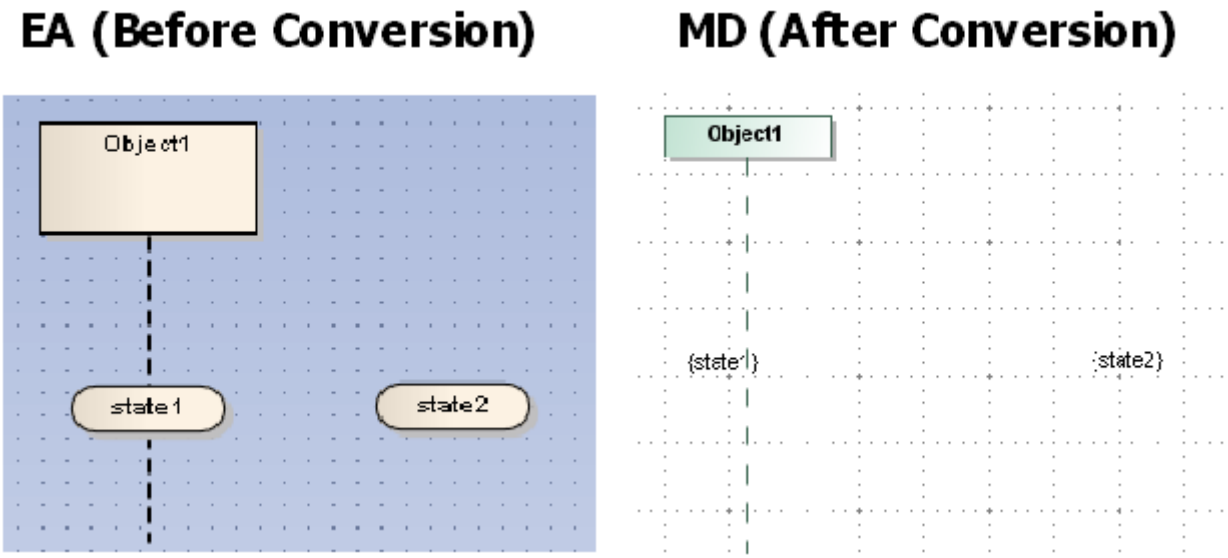
General ordering.

After completing the transformation process, the following transformation message will open:

```
Updated element <xmi:id>: General Ordering will not be imported.
```

State Invariant

A State Invariant in EA will be transformed as is. MagicDraw, CEA, or CSM does not support State Invariants in the current release.

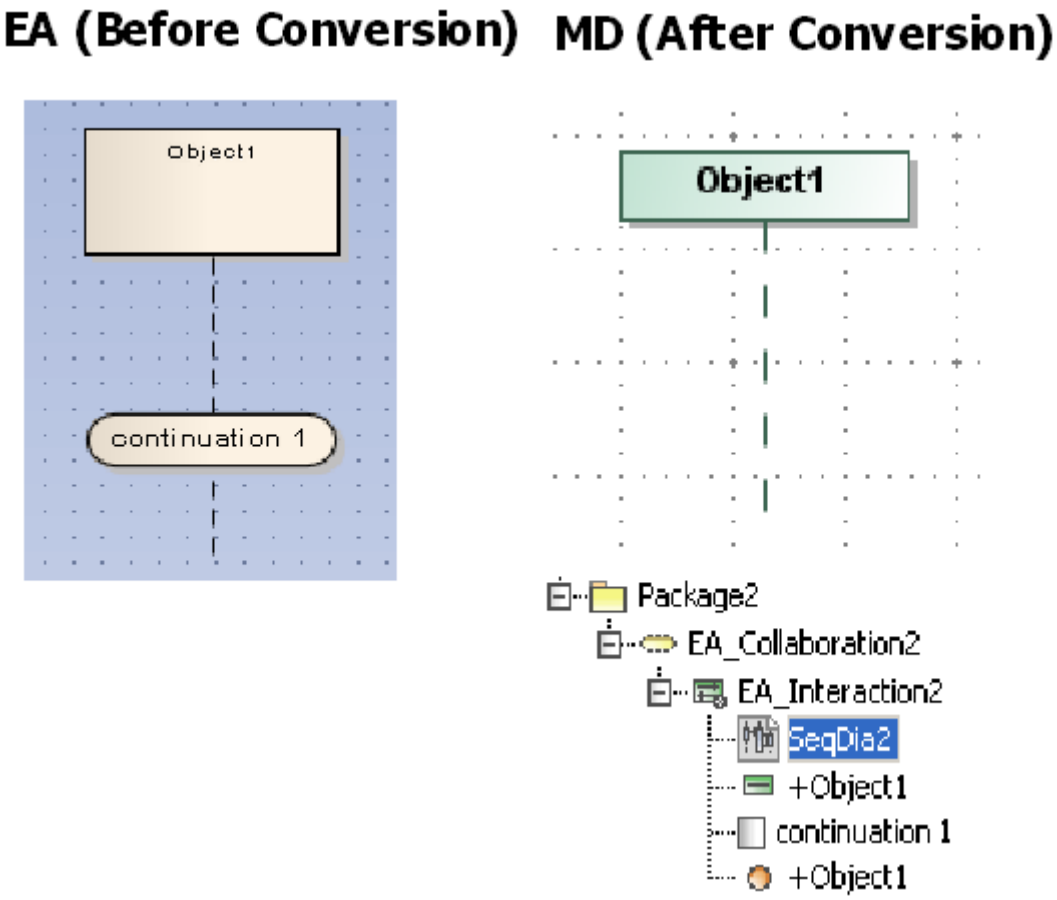


State Invariant.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Continuation

Continuations from EA can be imported to MagicDraw, CEA, or CSM. They are viewable in the Containment Tree(see page 693), but without a picture displayed in the diagram. MagicDraw, CEA, or CSM does not support Continuations in the current release.



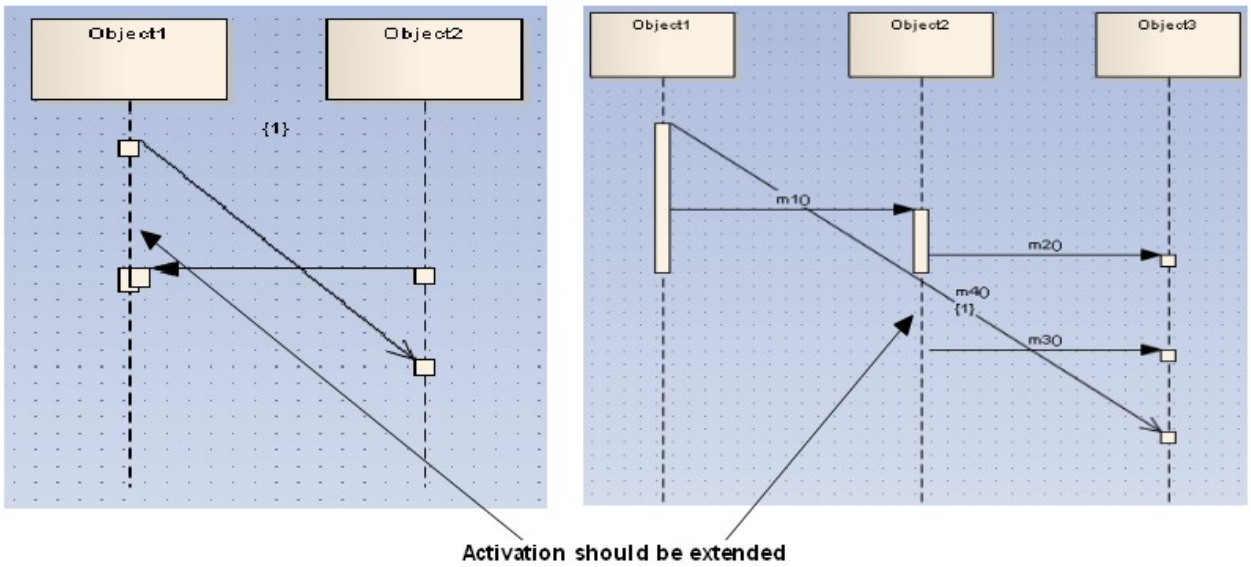
Continuation.

After completing the transformation process, the following transformation message will open:

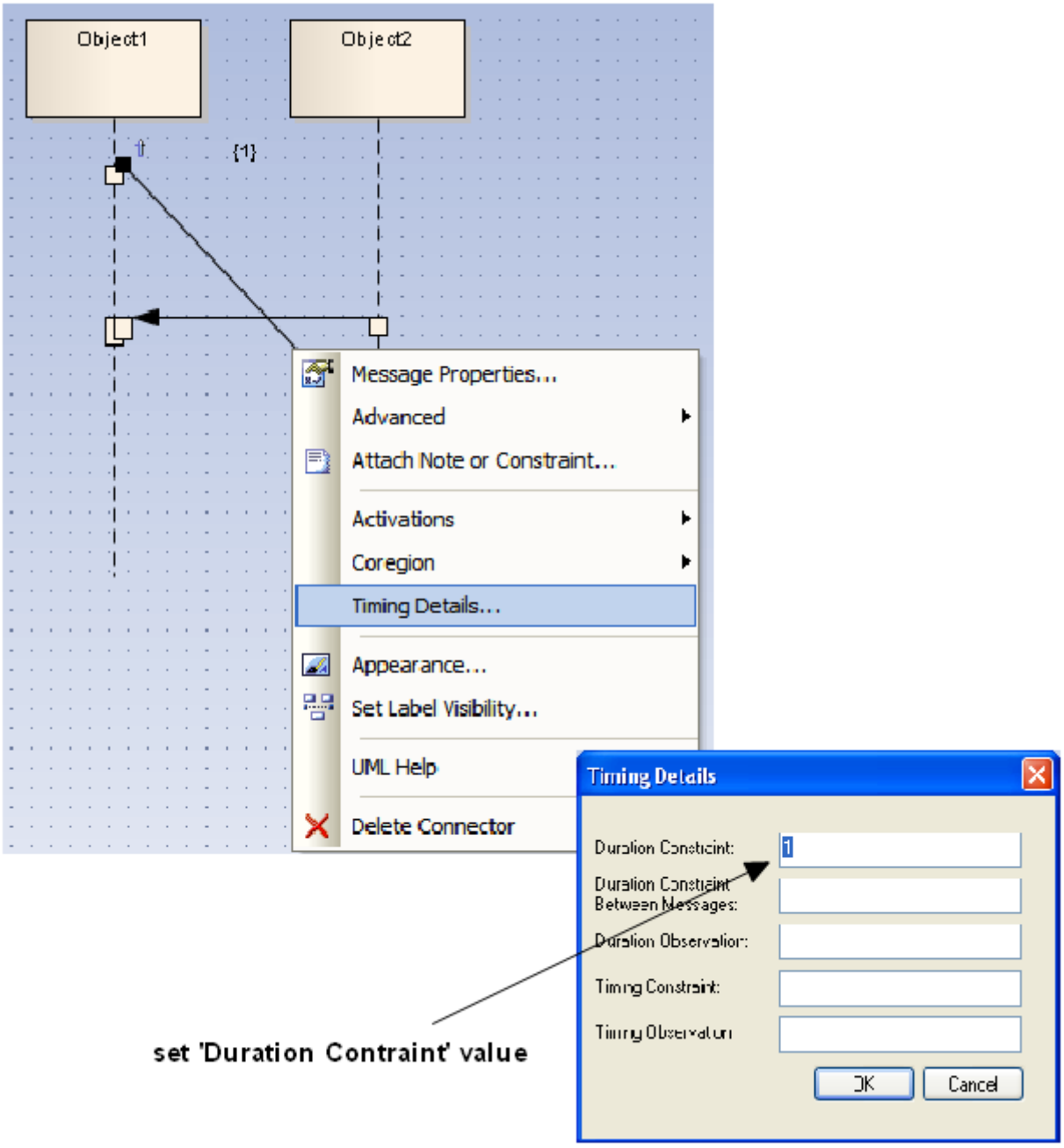
```
Updated element <xmi:id>: Continuation will not be displayed.
```

Diagonal Message

A Diagonal Message is a Message whose destination's height is adjustable. A diagonal Message may change Activations.



To create a Diagonal Message in EA, you must specify the *Timing Details* property of the Message and add a numeric value to the Duration Constraint input field.



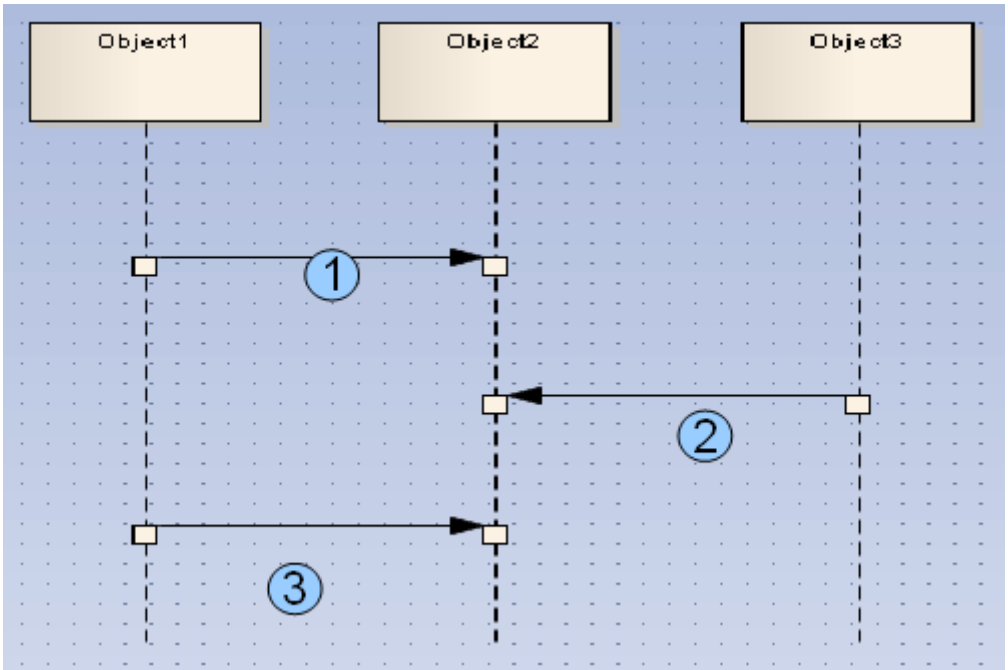
Creating a Diagonal Message.

Synchronous Message behavior

The placement of Synchronous Messages in EA affects the way Activations are created.

Order

A Message in EA has a *Sequence Number* that indicates the order of the message in the diagram. This information can be found in the exported XMI file. Knowing it helps predict how Activations will be created. The order starts from the top and goes downward, so the first Message in the diagram is the one drawn at the top of the diagram. Its Sequence Number will be set to '1'.

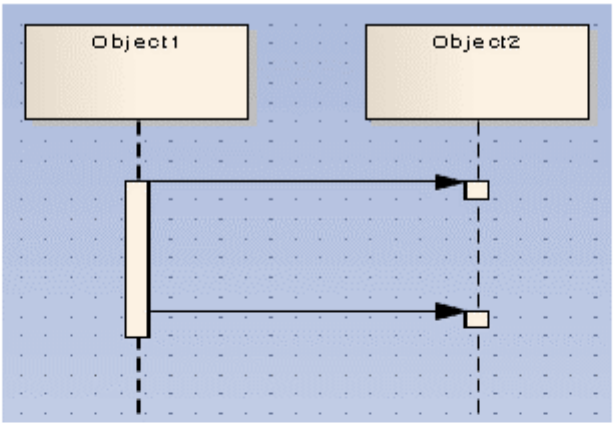


Message sequence number.

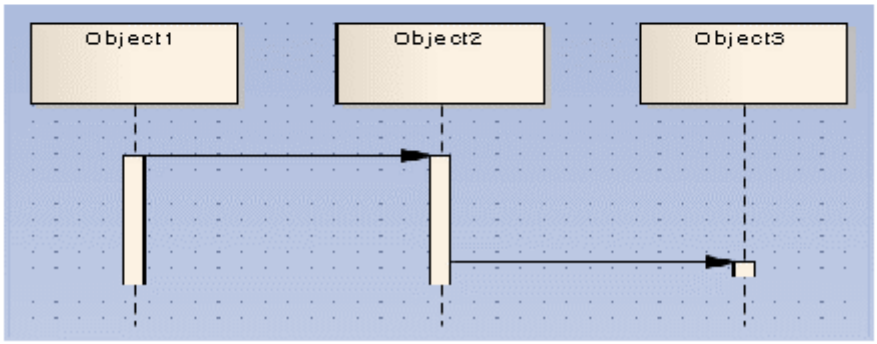
Process

Generally, in EA, any two Synchronous Messages will be in the same process if they meet the following conditions:

- Both of their tails are placed on the same Lifeline.
- The head of the upper Message and the tail of the lower Message are on the same Lifeline.



condition 1

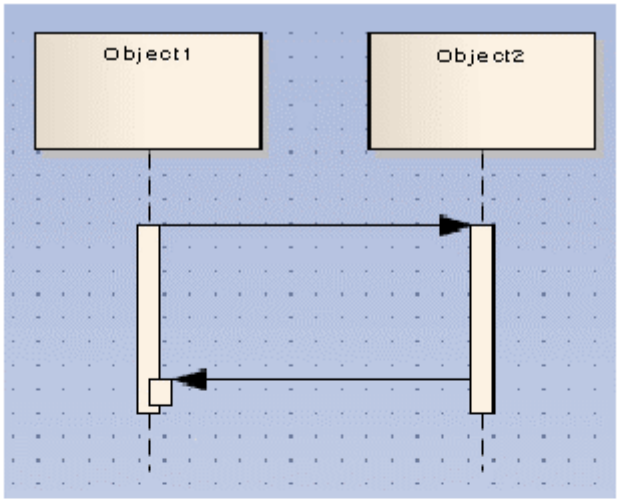


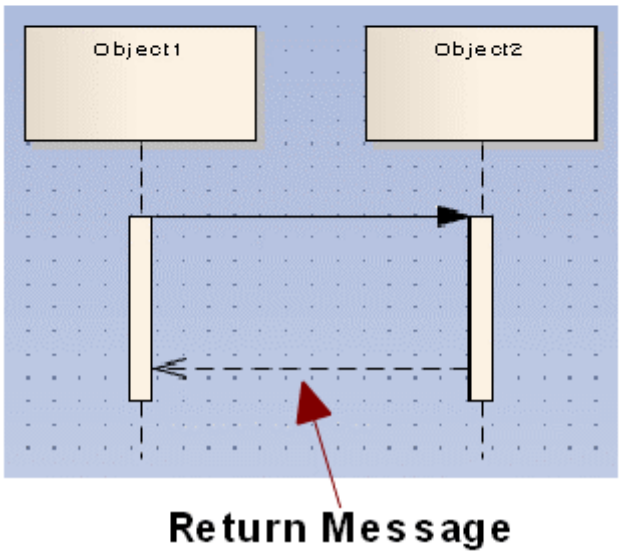
condition 2

Process.

Activation level

The Activation Level starts from level 0. It will increase in increments of 1 as an ongoing process that receives a Message that is not a Return Message (Reply Message in MagicDraw, CEA, or CSM).

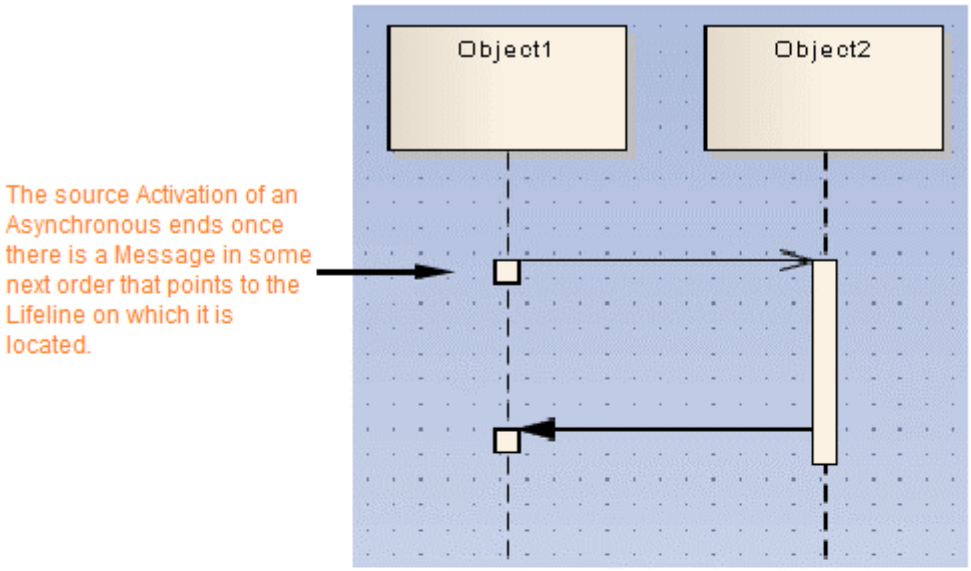


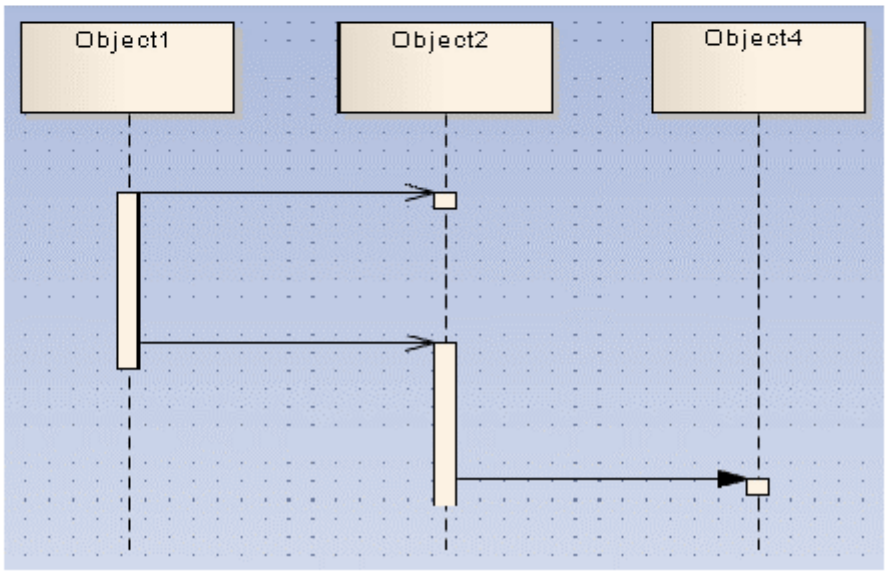


Activation level.

Asynchronous Message behavior

If a Message is an Asynchronous Message in EA, its source Activation will end if there is no other Message in the same group with a higher Sequence Number. Its source will be attached to the same Lifeline.

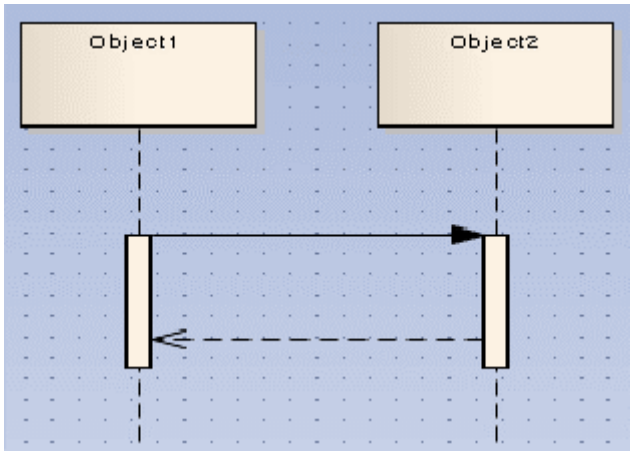




Asynchronous Message.

Return Message behavior

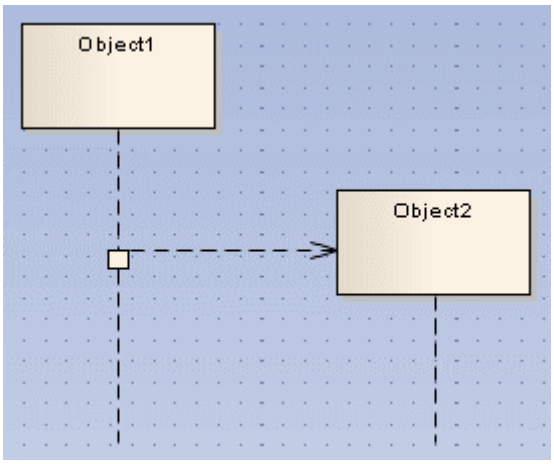
A Return Message in EA is called a Reply Message in MagicDraw, CEA, or CSM. When it is pointed to an Activation, it will not create an Activation at the top of the existing Activation, unlike other normal Synchronous Messages.



Return message.

Activation options

You can control how Activations behave at some level through the Message options. You can access Message options by right-clicking any Message, and then selecting the **Activations** option. However, MagicDraw, CEA, or CSM does not support Activation options in the current release.



Activation options.

Related pages

- Sequence diagram(see page 304)

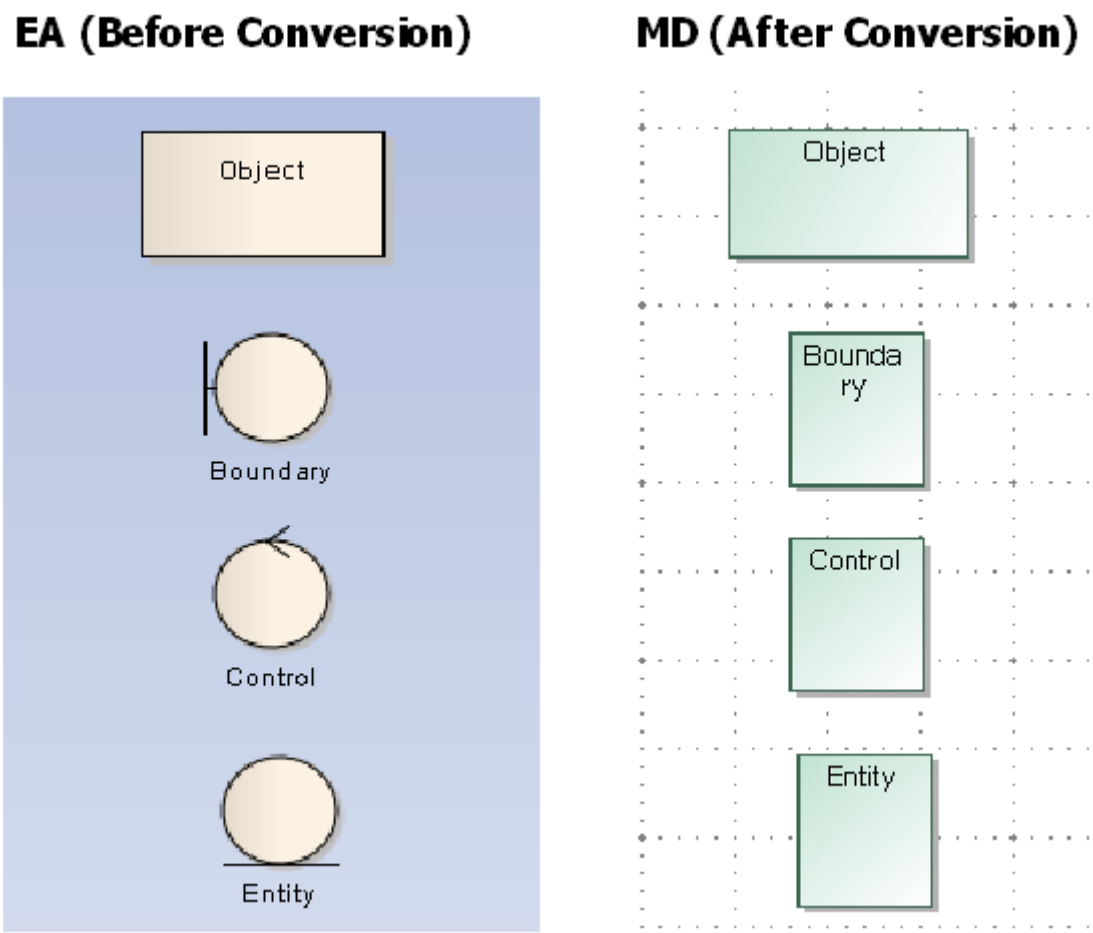
Communication diagram elements

On this page

- Object, Boundary, Entity, and Control(see page 161)
- Exception Elements(see page 163)
- Association(see page 164)
- Message(see page 166)
- Operations(see page 167)

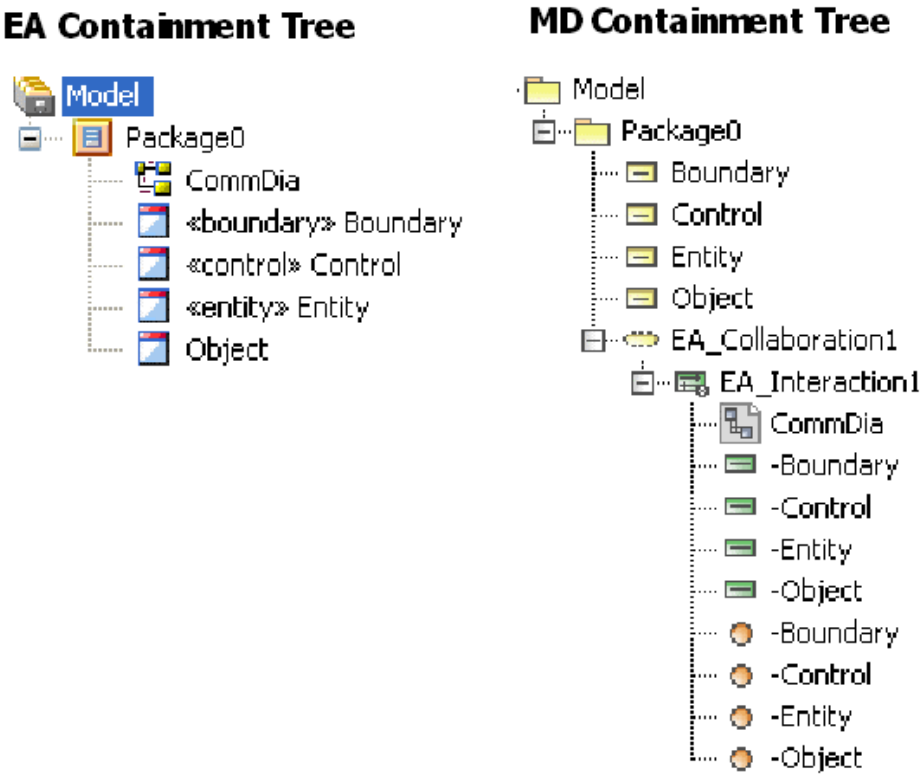
Object, Boundary, Entity, and Control

All Object, Boundary, Entity, Control elements in EA are InstanceSpecifications(see page 735). After conversion, their UML element types will remain, and a Lifeline (see page 746)and an OwnedAttribute element will be created to represent each of them in the Communication diagram(see page 302) in which they occur.



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Object, Boundary, Entity, and Control (diagram view).



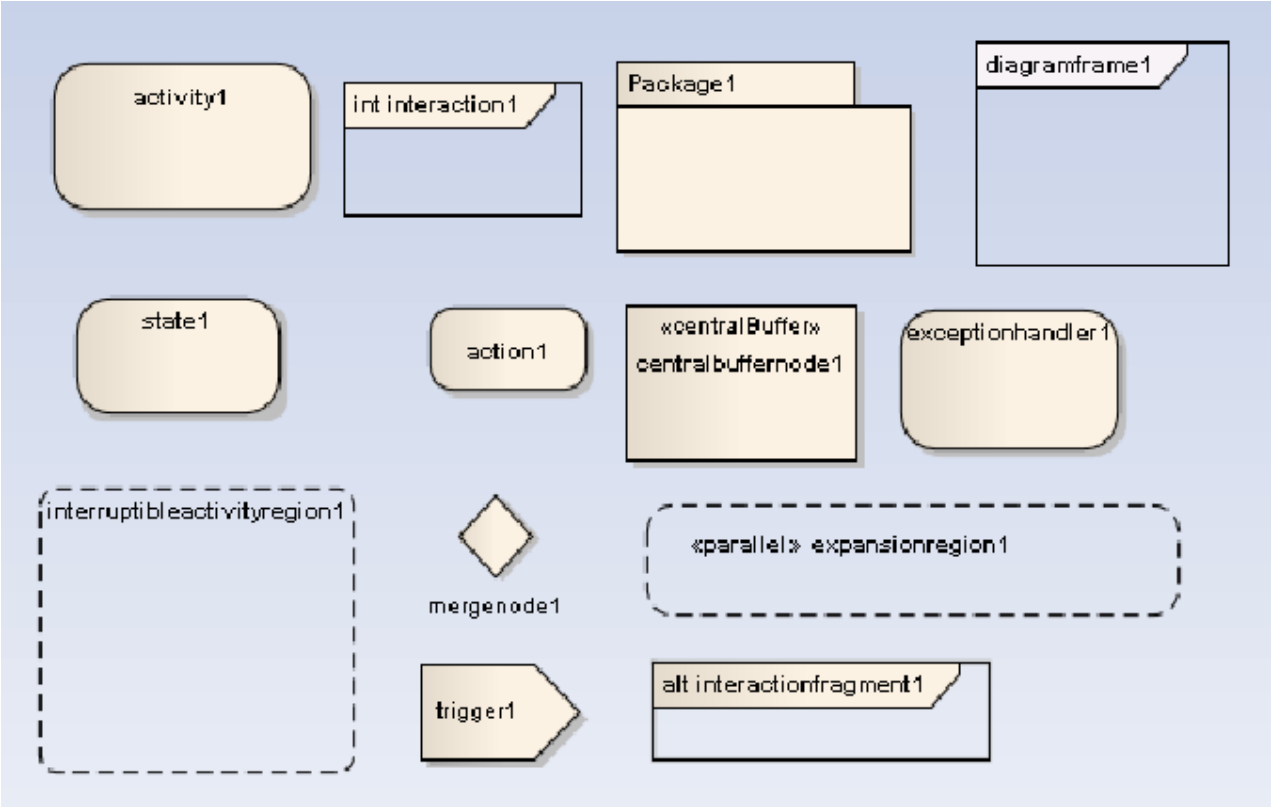
Object, Boundary, Entity, and Control (Containment tree).

Other elements that can be drawn in a Communication diagram will be handled in a similar manner.

Exception Elements

Some of the elements that can be drawn in a Communication diagram using EA are not supported by MagicDraw, CEA, or CSM. Consequently, their displaying parts will not be imported. These elements include Package(see page 800), Activity(see page 311), Action(see page 629), DiagramFrame(see page 523), State(see page 848), Interaction(see page 739), ExceptionHandler(see page 711), CentralBufferNode(see page 798), InterruptibleActivityRegion(see page 851), MergeNode(see page 798), Trigger, ExpansionRegion(see page 788), and InteractionFragment(see page 739).

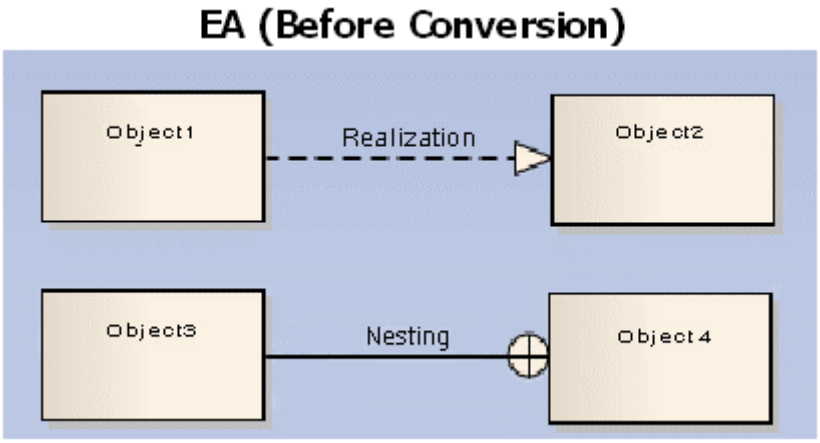
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Exception elements.

Realization and Nesting

The Realization element type in EA is *uml:Realization*. The Nesting element type in EA is exported in XMI as *uml:Dependency*. The Realization and Nesting lines in EA are not supported in the Communication diagram; therefore, only their model data will be imported, not the displaying parts.



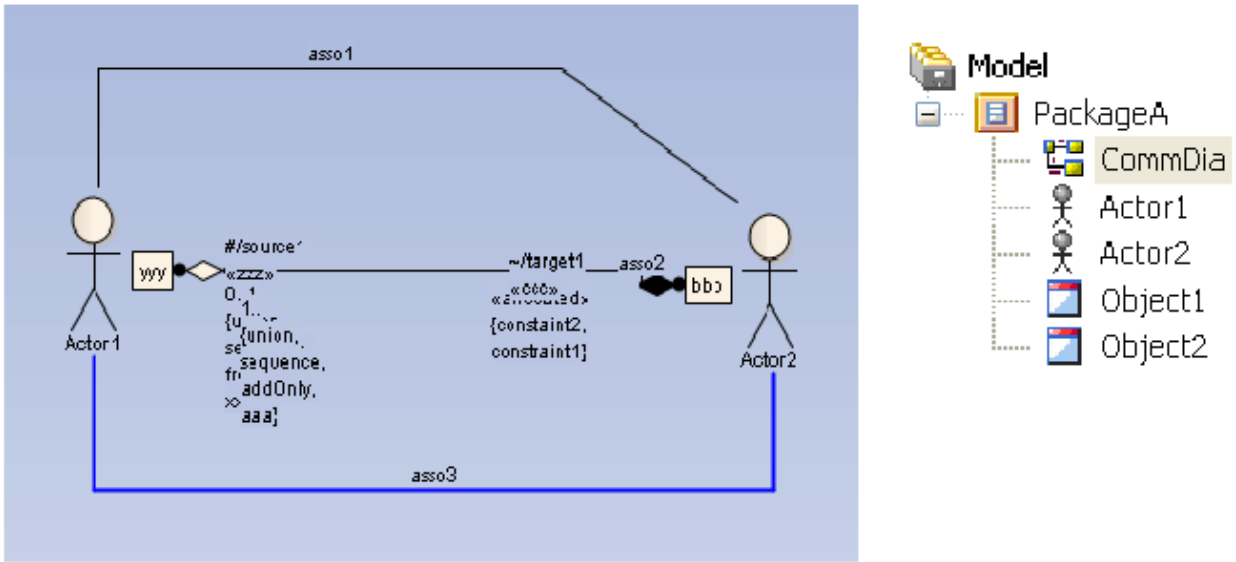
Realization and Nesting.

Association

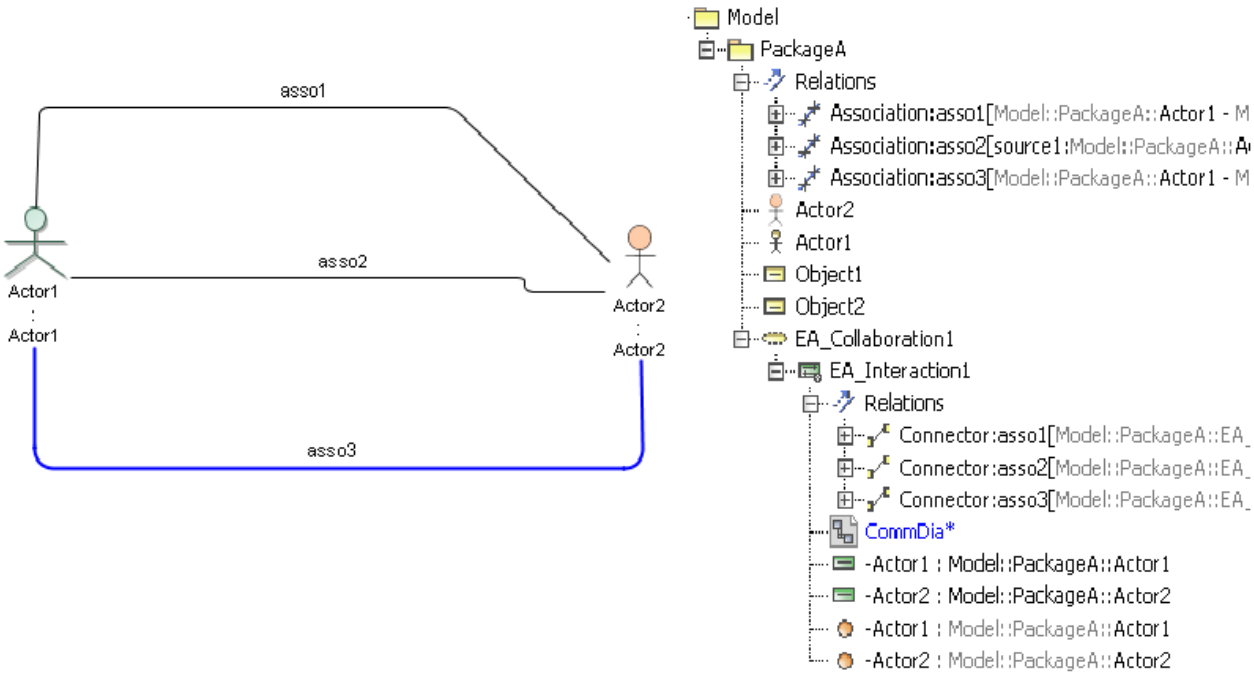
Every Association relationship drawn in a Communication diagram in EA will have a Connector line created for each of them. The elements attached to both ends of the Association (see page 654)line will have a Lifeline element created to represent each of them. The Association lines and the elements attached to them will not be removed. However, the elements that will be shown in the diagram frame will be the Lifeline elements and the Connector lines created to represent them.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

EA (Before Conversion)



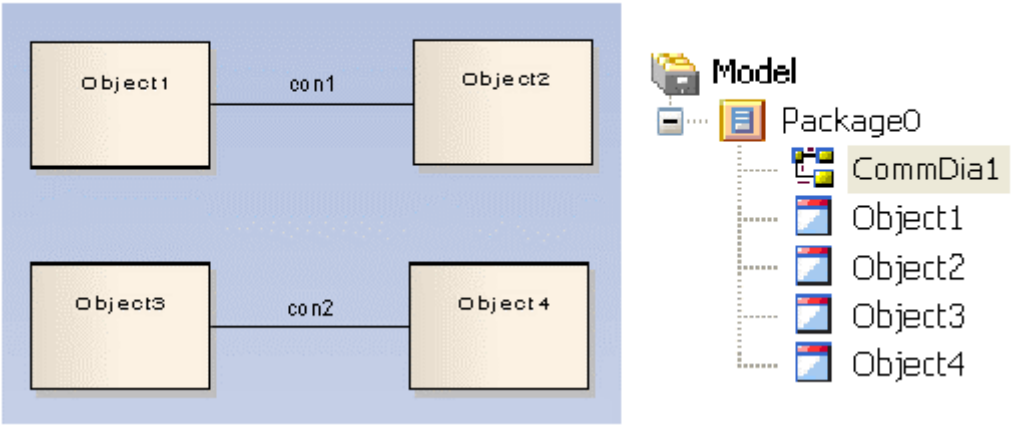
MD (After Conversion)



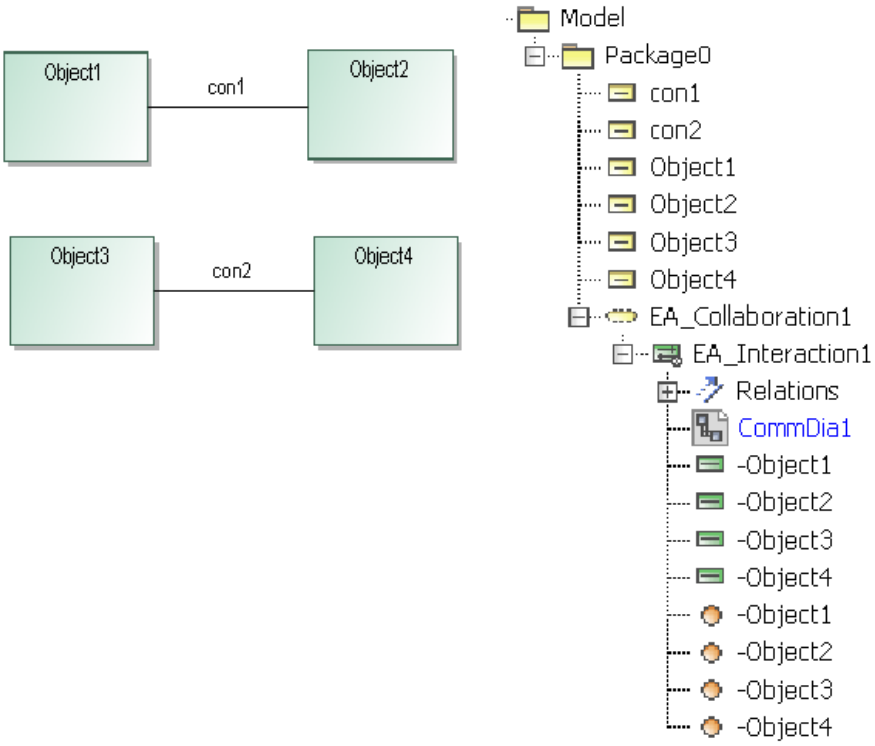
Association.

If an Association line is connected to the InstanceSpecification elements, its data will be removed. If an Association line is linked between two InstanceSpecification elements, it will be transformed into an InstanceSpecification (see page 735)element. This is one of the constraints belonging to the Communication diagram.

EA (Before Conversion)



MD (After Conversion)

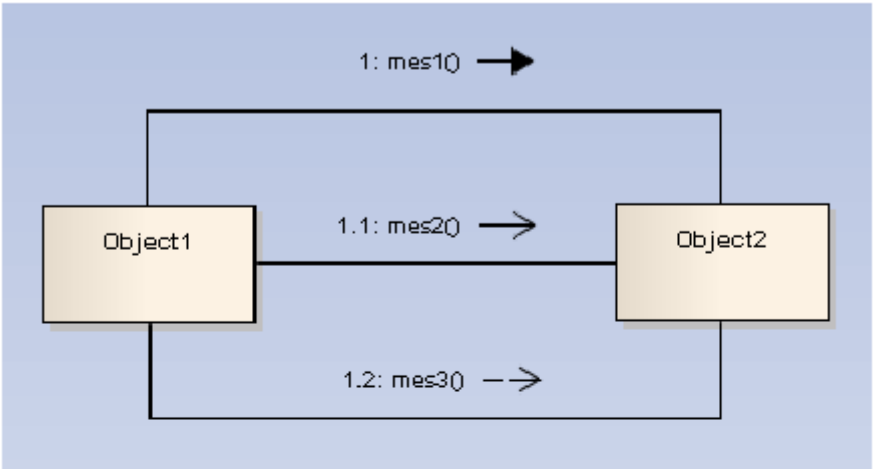


Association Line between InstanceSpecifications.

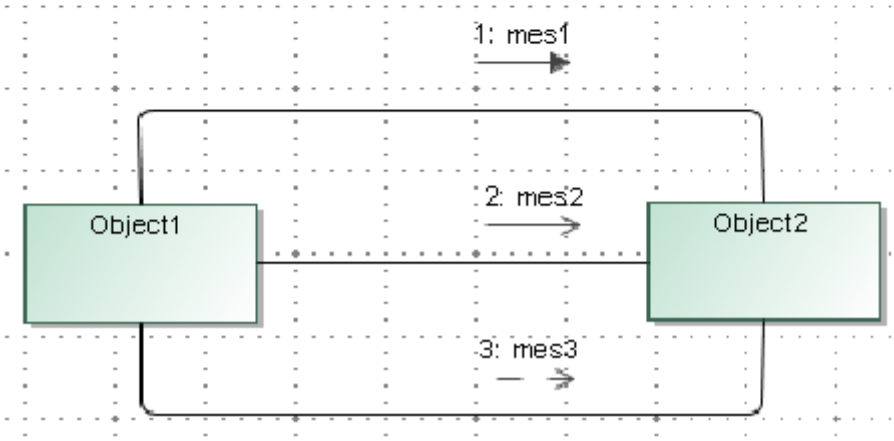
Message

Messages can be created on Connectors and will be imported to MagicDraw, CEA, or CSM.

EA (Before Conversion)



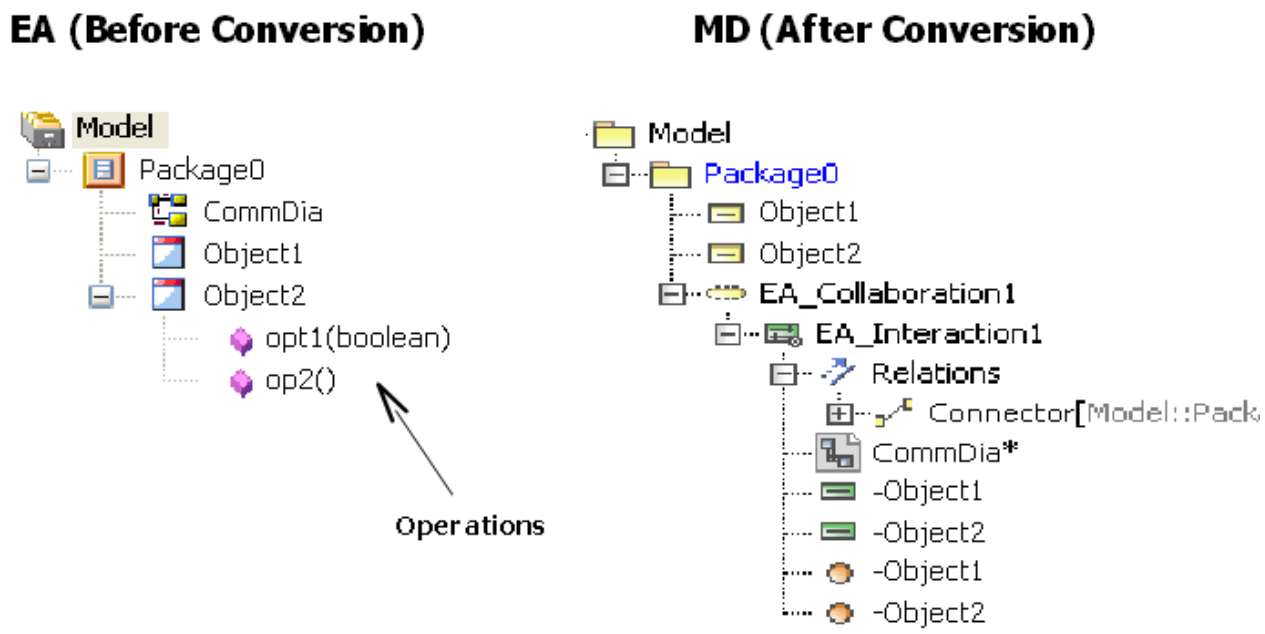
MD (After Conversion)



Message.

Operations

An InstanceSpecification cannot contain Operation elements. If the XMI file from EA has some InstanceSpecification elements containing Operation elements, those Operations will be removed.



Operations.

Related pages

- Communication diagram(see page 302)

State Machine diagram elements

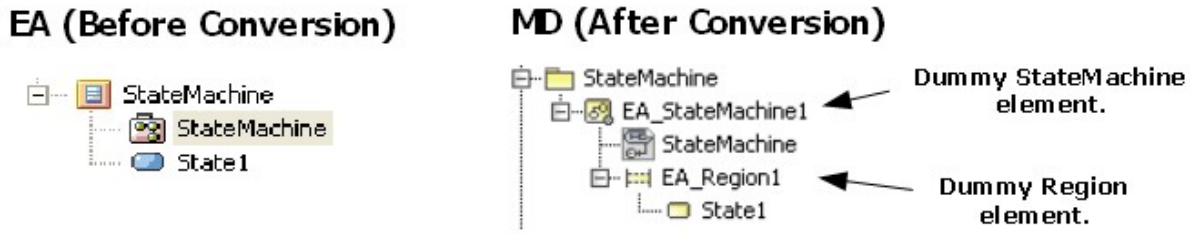
On this page

- StateMachine(see page 168)
- State(see page 169)
- StateMachine placed on a diagram(see page 172)
- Object(see page 172)
- Synch(see page 174)
- EntryPoint / ExitPoint(see page 175)
- Object Flow connecting State Machine elements(see page 176)
- Information Flow connecting State Machine elements(see page 177)
- Trigger(see page 178)
- Self Transition(see page 178)
- Removed element(see page 179)

StateMachine

After conversion, a dummy StateMachine element will be created. The dummy StateMachine is either created by the XMI exporter from EA or by EA Import. A StateMachine diagram(see page 308) will be placed inside the dummy StateMachine and a dummy Region element will be created to contain all of the StateMachine elements.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

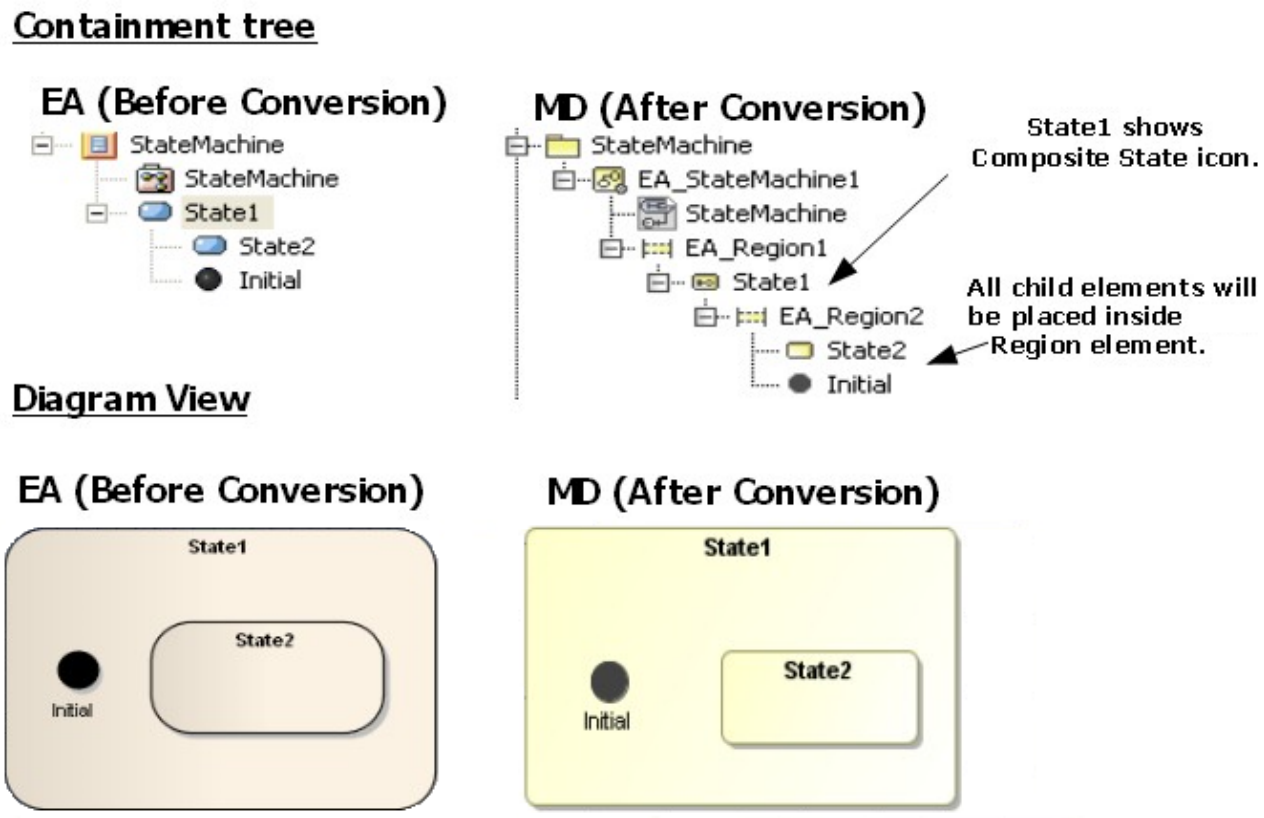


A dummy StateMachine element and a dummy Region element.

State

State containing other elements

A State element containing other elements will be transformed to a Composite (see page 848)State(see page 848). All of the contained elements will be placed inside the Region (see page 851)element of the State element.



State Containing Other Elements.

State Containing StateMachine

If a State element contains a StateMachine element, the StateMachine element will be brought out and placed at the same level of the Region element of the State element.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Containment tree

EA (Before Conversion)

MD (After Conversion)

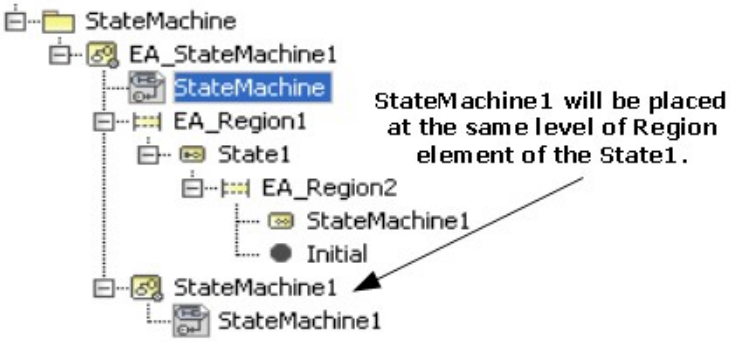
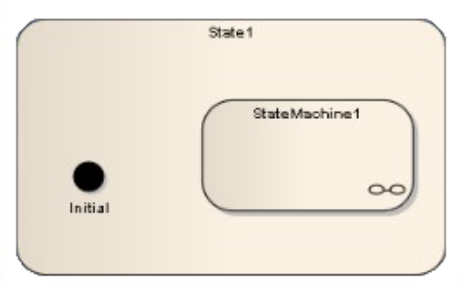
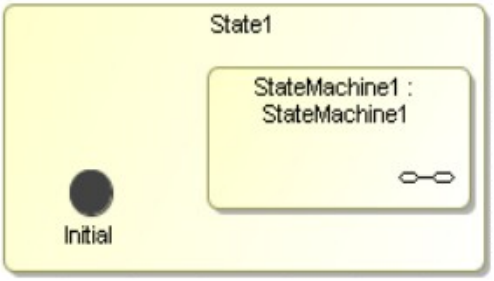


Diagram View

EA (Before Conversion)



MD (After Conversion)

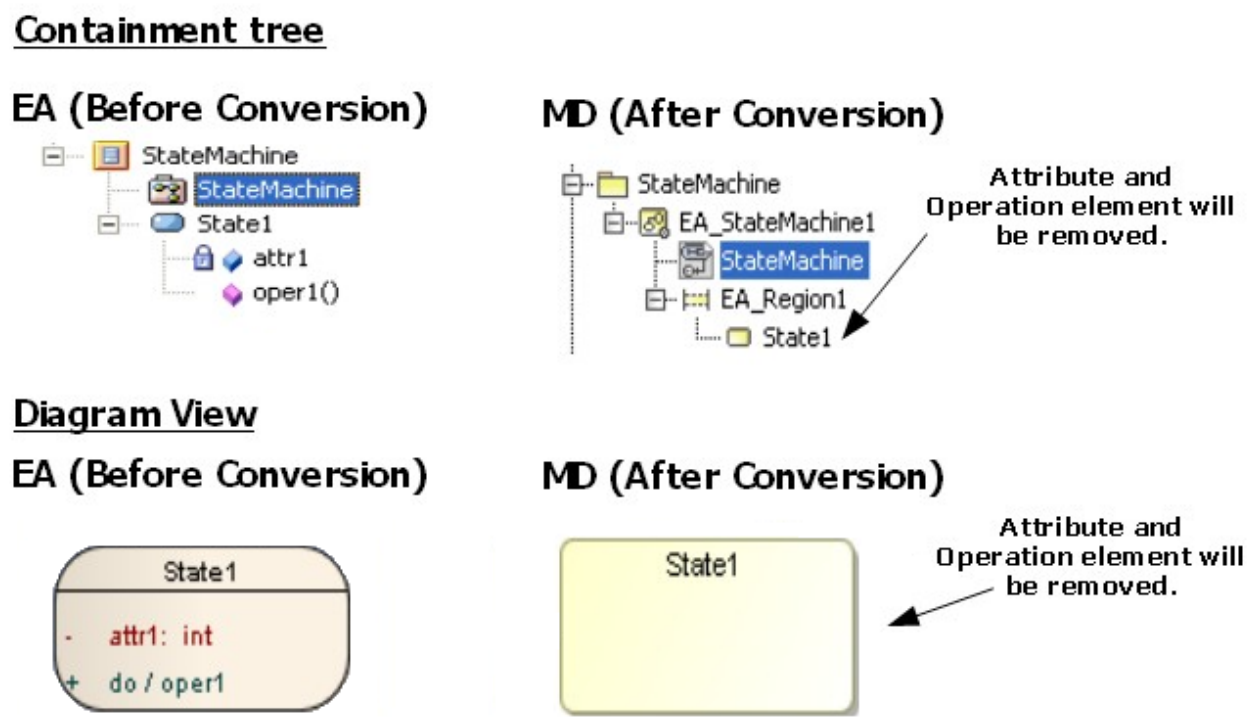


State Containing StateMachine.

State containing Attribute and Operation

If a State element contains Attribute and Operation elements, the Attribute (see page 672)and Operation (see page 677)elements will be removed from both the Diagram pane(see page 24) and Containment tree(see page 693).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



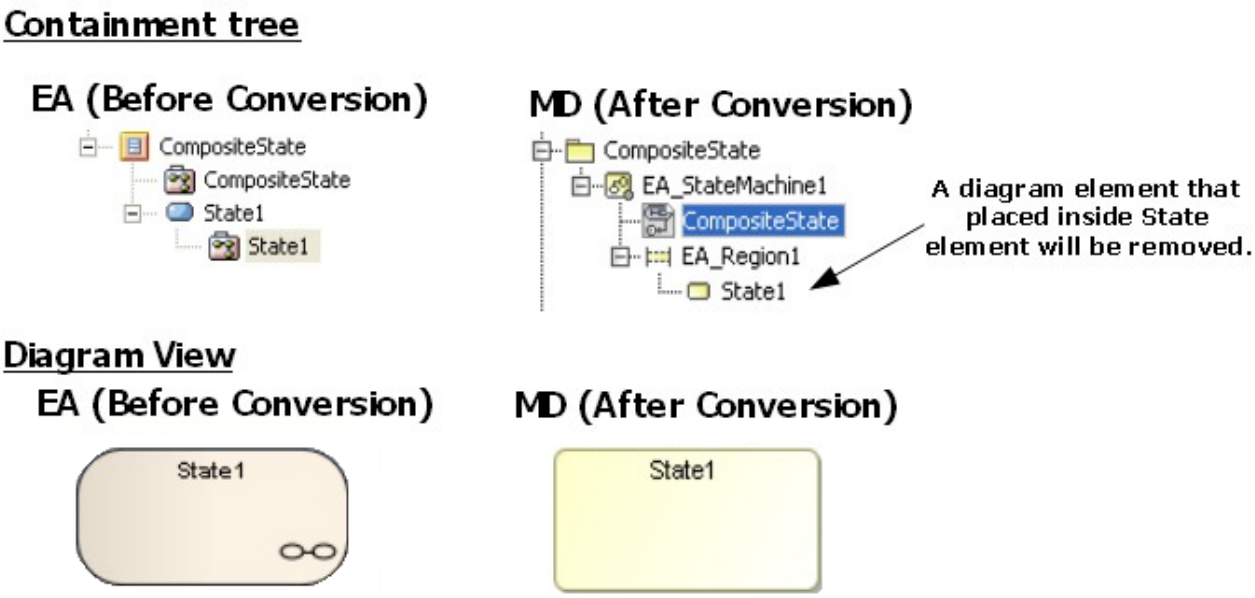
State Containing Attribute and Operation.

The following transformation messages will open:

- Removed element <xmi:id>: State cannot contain Attribute.
- Removed element <xmi:id>: State cannot contain Operation.

State containing Diagram Element

If a diagram element is placed inside a State element, it will be removed.



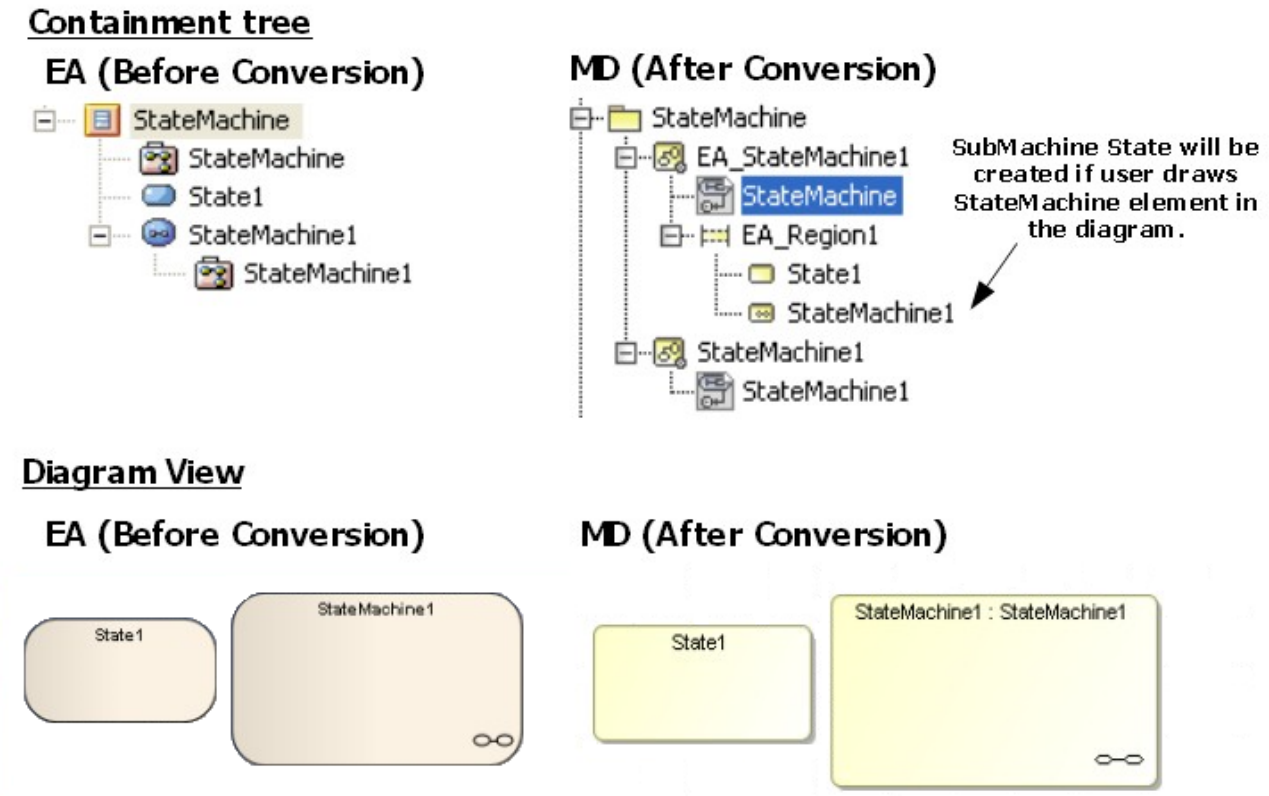
State Containing Diagram Element.

The following transformation message will open:

```
Removed element <xmi:id>: State cannot contain diagram element.
```

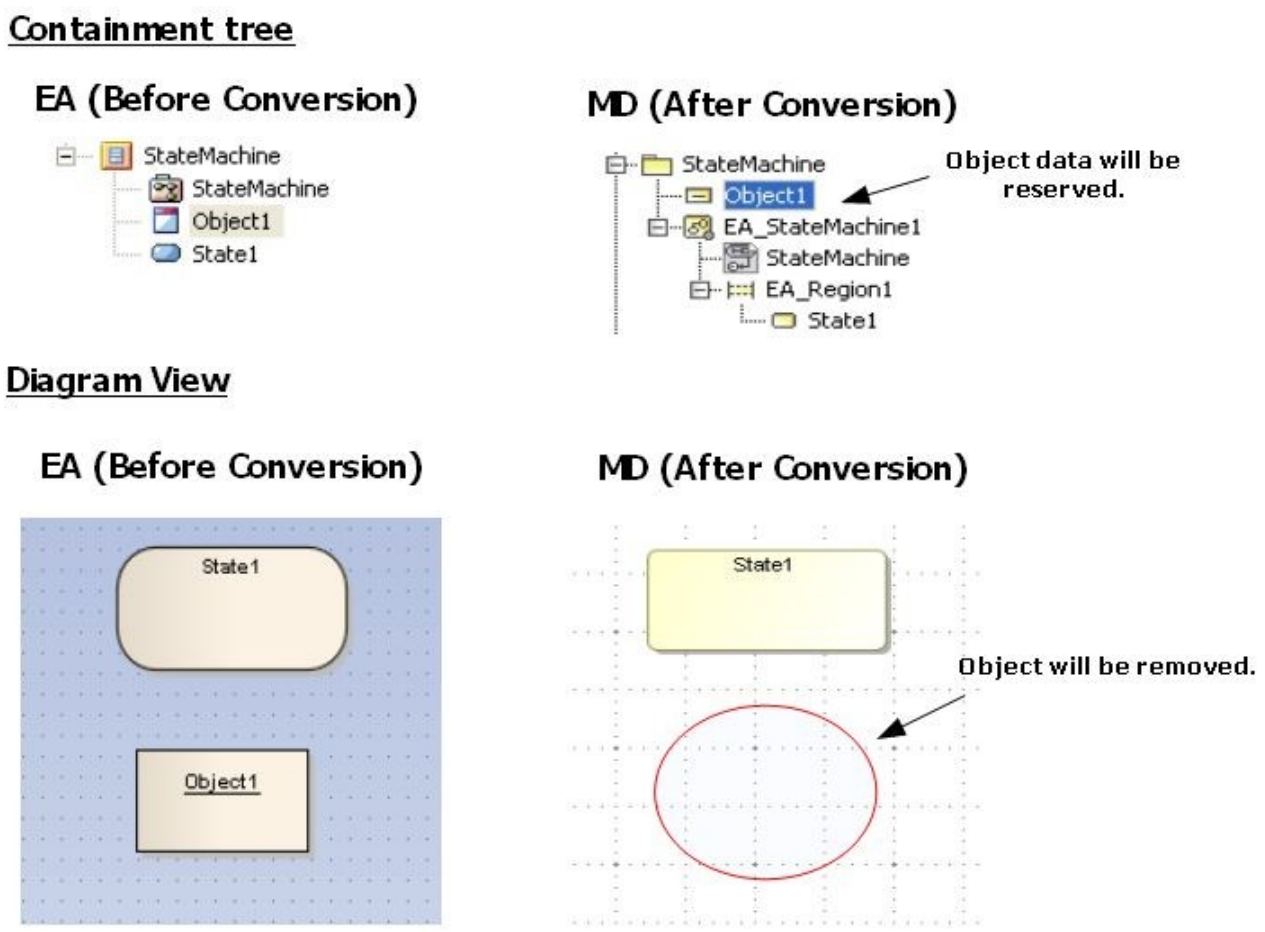
StateMachine placed on a diagram
If a StateMachine element is drawn in a StateMachine diagram, an additional SubMachineState will be created to represent the StateMachine.

> **Note**
> A SubMachine State is a State whose SubMachine property is set to a StateMachine.



StateMachine Represented by a SubMachine State.

Object
An Object element placed in a State Machine diagram will be removed from the Diagram view. However, its data will be preserved in the Containment tree as an InstanceSpecification(see page 735).



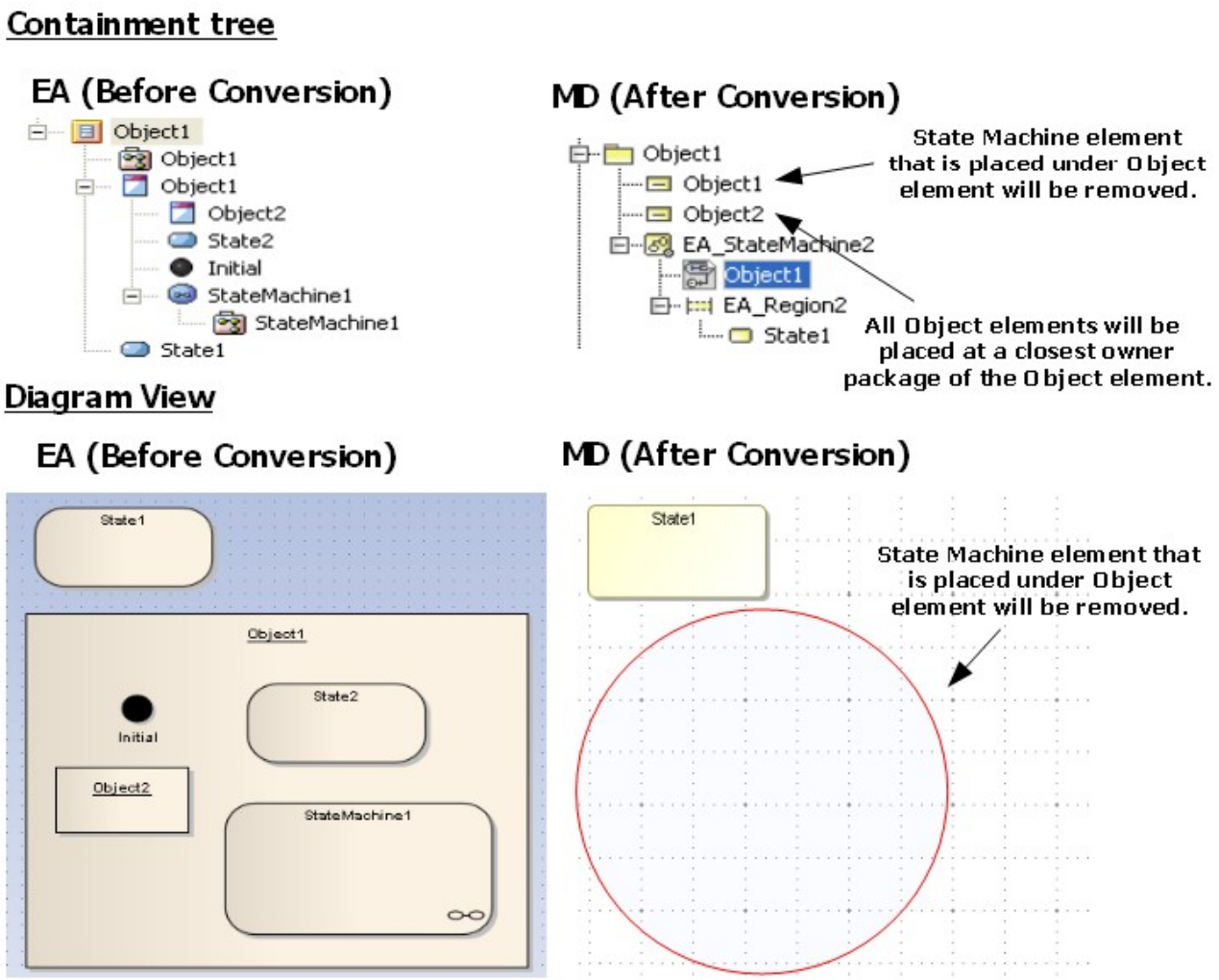
Object.

The following transformation message will open:

```
Removed view <xmi:id>: The view represents element that does not support in
State Machine Diagram.
```

Object containing a State Machine element

In EA, an Object element can contain a State Machine element. After it has been imported to MagicDraw, CEA, or CSM, the State Machine element placed inside an Object element will be removed both from the Diagram pane and Containment tree. However, the Object element data will be preserved. All of the Object element data will be placed at the closest owner package of the Object element.



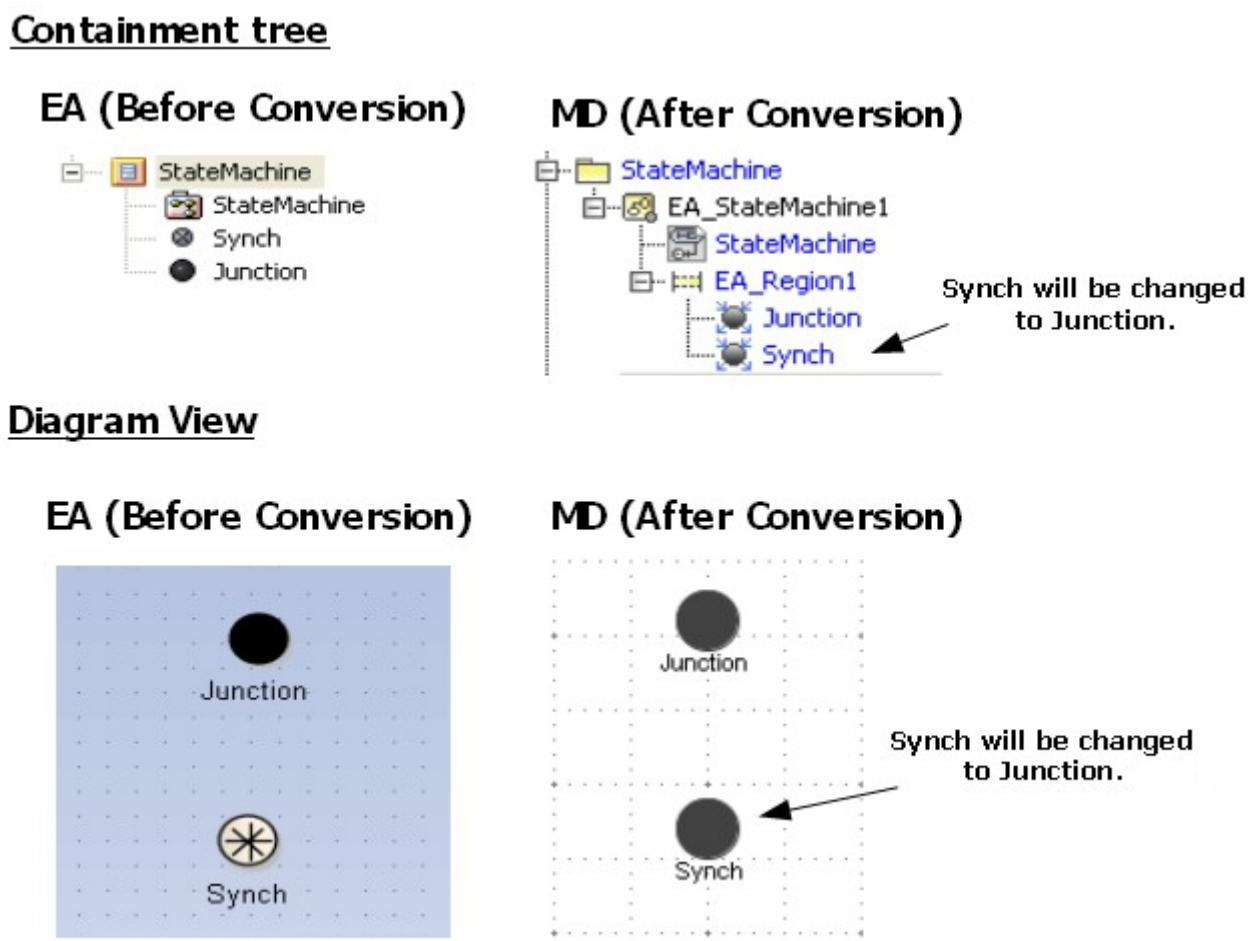
Object Containing State Machine Element.

The following transformation message will open:

```
Removed element <xmi:id>: Invalid element. Instancespecification can not
contain element from State Machine.
```

Synch

A Synch element will be transformed to a Junction element.



Synch.

The following transformation message will open:

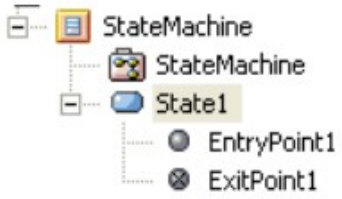
```
Updated element <xmi:id>: Synch updated to Junction.
```

EntryPoint / ExitPoint

An Entry or Exit point placed inside a State (or StateMachine) element will be placed outside a Region element of its parent element in the Containment tree. It will be relocated to the nearest boundary of its parent element in the Diagram pane.

Containment tree

EA (Before Conversion)



MD (After Conversion)

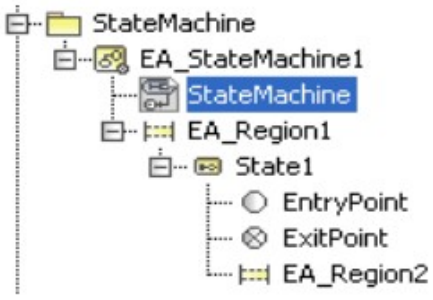
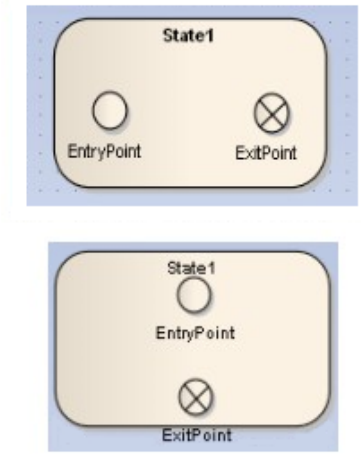
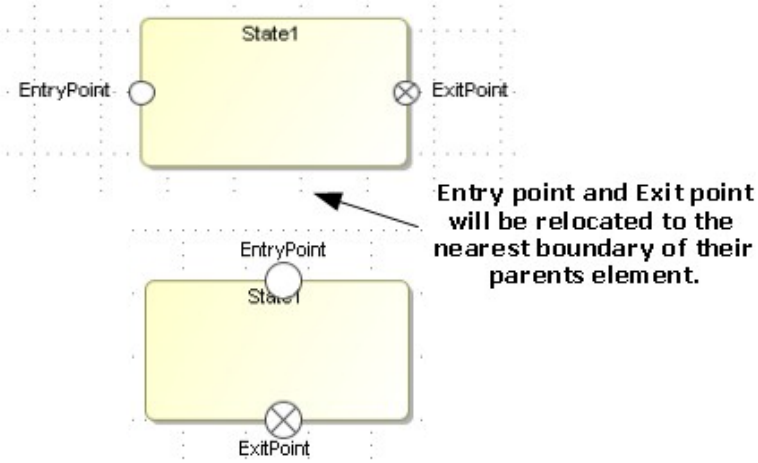


Diagram View

EA (Before Conversion)



MD (After Conversion)



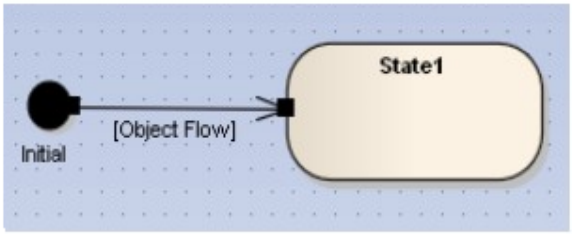
Entry/Exit Point.

Object Flow connecting State Machine elements

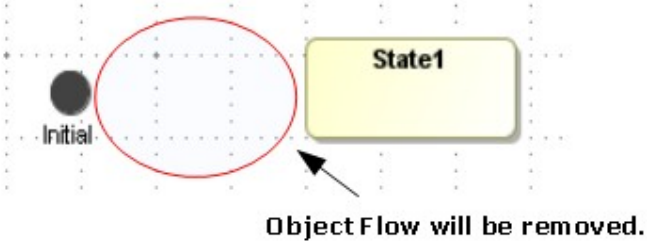
An Object Flow connecting the elements in a State Machine will be removed from both the Diagram view and Containment tree.

Diagram View

EA (Before Conversion)



MD (After Conversion)

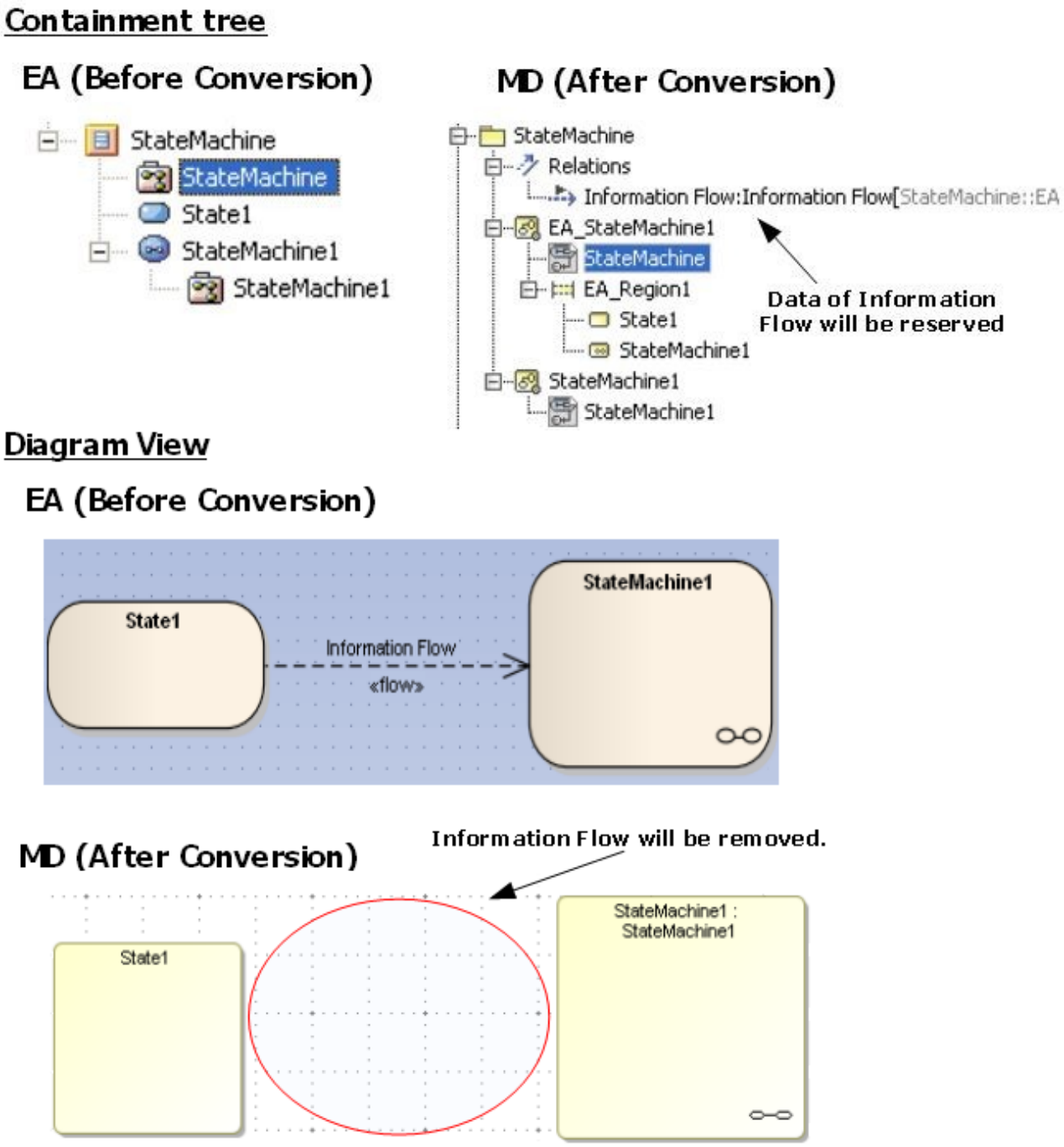


ObjectFlow Connecting State Machine Elements.

The following transformation message will open:

```
Removed element <xmi:id>: Invalid ObjectFlow. Source or Target of
theObjectFlow are connected to element from State Machine.
```

Information Flow connecting State Machine elements
An Information Flow connecting the elements in a State Machine will be removed from the Diagram pane; however, its data will be preserved.



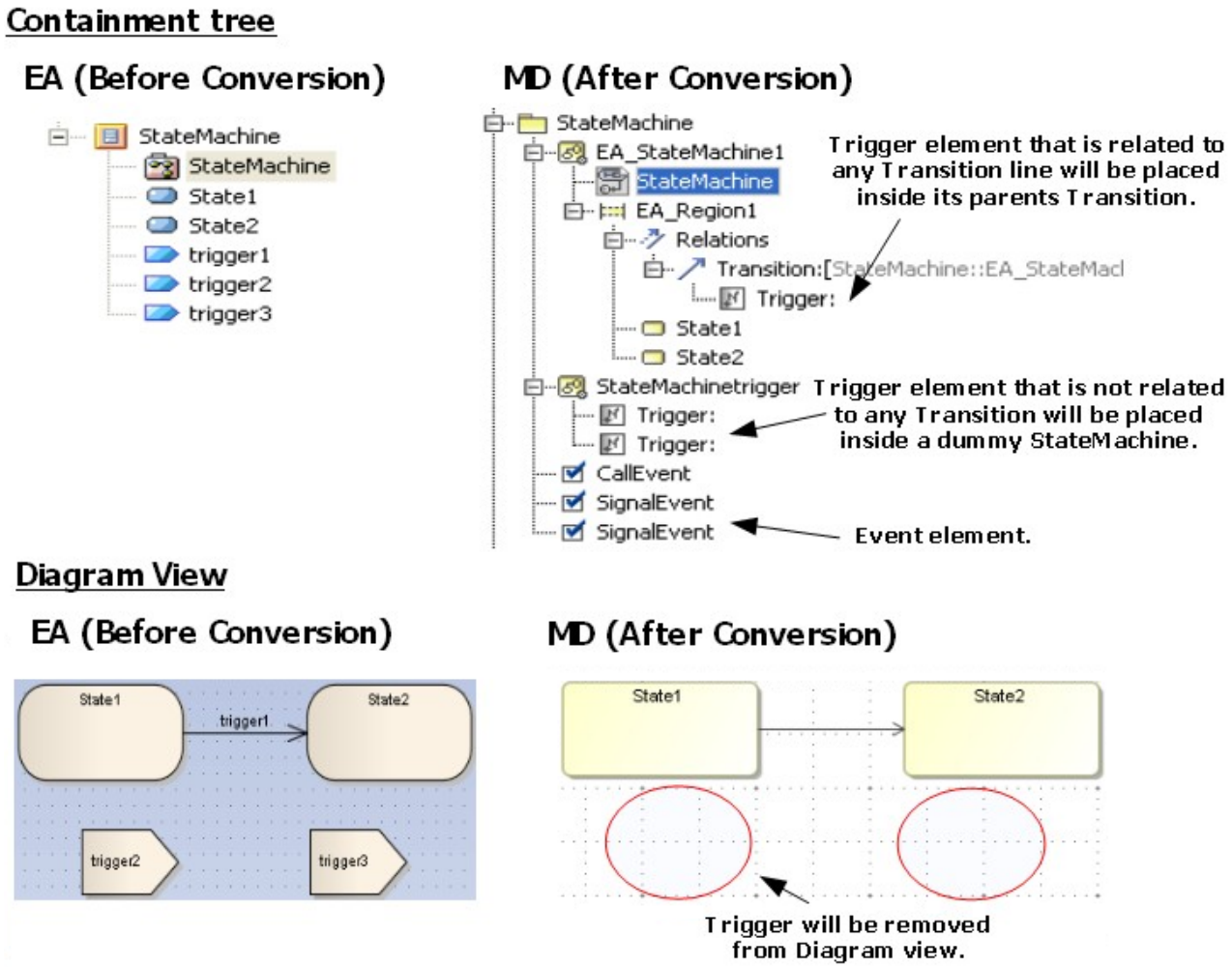
Information Flow, Trace, and Dependency.

The following transformation message will open:

```
Removed element <xmi:id>: The view represents element that does not sup- port
in State Machine Diagram.
```

Trigger
If there is a Trigger element unrelated to any Transition line, a dummy StateMachine will be created to hold the Trigger element. The dummy StateMachine will be named after the parent package name concatenating with 'trigger'.

In the case of a Trigger element related to Transition, the data of the Trigger element will be placed inside its parent Transition(see page 872). Each Trigger element will be given an event type, represented by an Event element. After conversion, the Event will be placed at the closest owner package of the Event(see page 708).



Trigger.

Self Transition
Most of the elements in a StateMachine diagram can have a self transition, except the Initial, Final, and History elements. The self transition in these three elements will be removed from the Diagram pane, but the data of the self transition will be preserved in the Containment tree.

Containment tree

EA (Before Conversion)

MD (After Conversion)

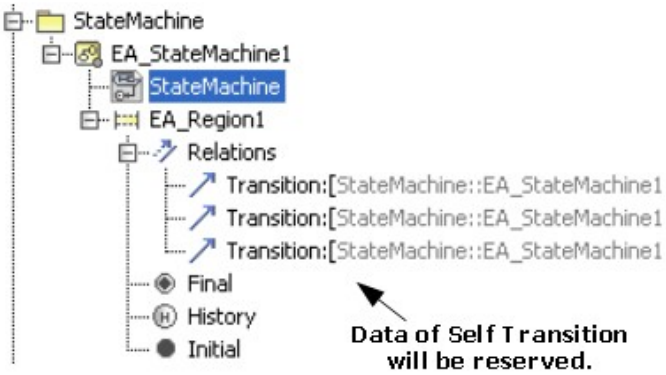
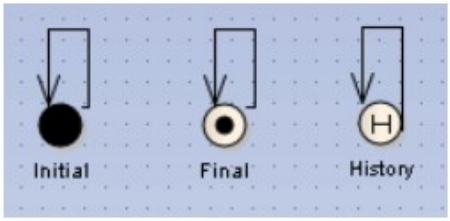
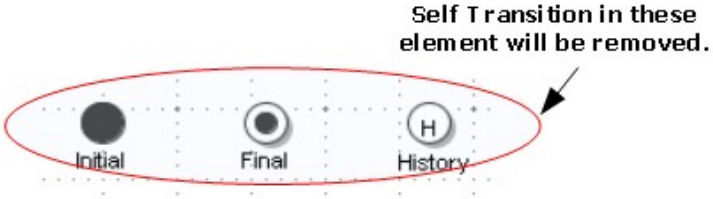


Diagram View

EA (Before Conversion)



MD (After Conversion)



Self Transition.

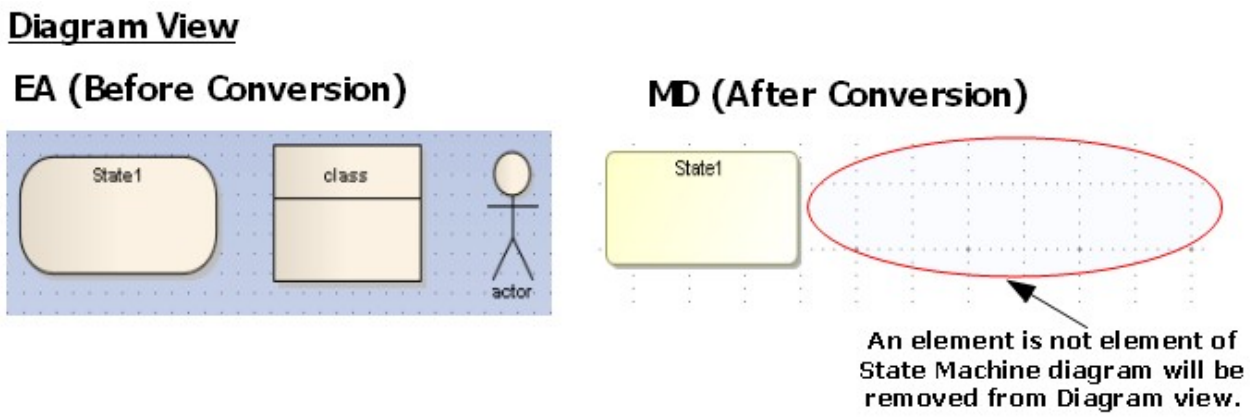
The following transformation messages will open:

- `Removed view <xmi:id>: Self Transition does not support forInitial.`
- `Removed view <xmi:id>: Self Transition does not support forFinal.`
- `Removed view <xmi:id>: Self Transition does not support forHistory.`

Removed element

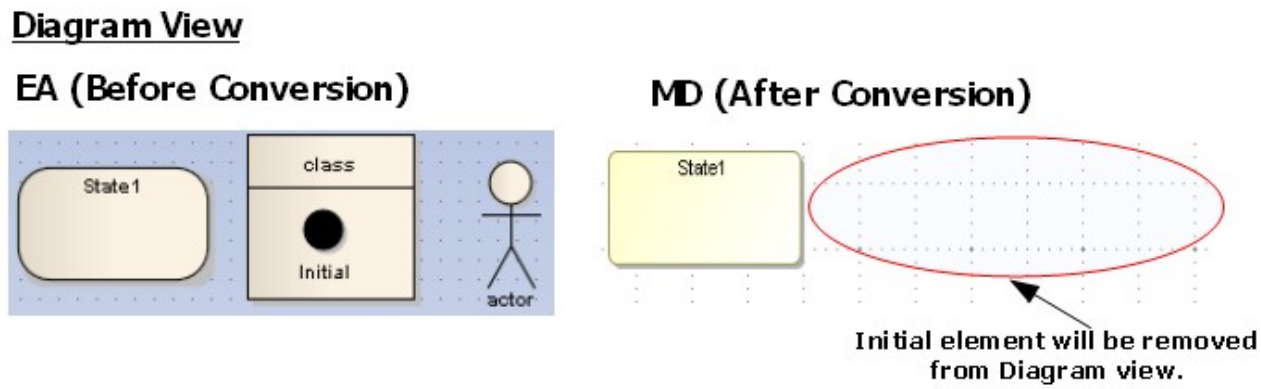
Any element which is not the element of a State Machine diagram (such as Class, Actor, Use-case, or Action) drawn in the State Machine diagram will be removed from the Diagram pane. However, its data will be preserved in the Containment tree.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Removed element.

If an element which is not the element of a State Machine diagram has a child element and is drawn in the State Machine diagram, the element will be removed from the Diagram pane. However, its data will be preserved in the Containment tree.



Removed Element with a Child Element

The following transformation message will open:

```
Removed view <xmi:id>:The view represents element that does not support in
State Machine Diagram.
```

Related pages

- State Machine diagram(see page 308)

Composite Structure diagram elements

On this page

- Import elements(see page 181)
- Conversion details(see page 181)
- Transformation report(see page 194)

This section presents additional EA specific Composite Structure diagram information.

Import elements

EA differs from MagicDraw, CEA, or CSM in the Composite Structure diagram's content elements design. The table below shows the differences by focusing on the EA's elements and how they will be transformed or converted into a format MagicDraw, CEA, or CSM can correctly load and display. The element names shown in the table are the same in both EA and MagicDraw, CEA, or CSM GUI. The contents in the brackets ([...]) are the XMI element type references.

Enterprise Architect	MagicDraw, CEA, CSM
Interaction	
Class [uml:Class]	Class [uml:Class]
Interface [uml:Interface]	Interface [uml:Interface]
Part [uml:Class] Part [uml:Property]	Part [uml:Property]
Port [uml:Port]	Port [uml:Port]
Collaboration [uml:Collaboration]	Collaboration Use [uml:Collaboration]
Expose Interface [not exist]	
Connector [element type is not exist]	Connector [uml:Connector]
Assembly [element type is not exist]	
Delegate [element type is not exist]	Connector [uml:Connector]
Role Binding [uml:Dependency]	Role Binding [uml:Dependency]
Represents [uml:Dependency]	Dependency [uml:Dependency]
Occurrence [uml:Dependency]	Dependency [uml:Dependency]

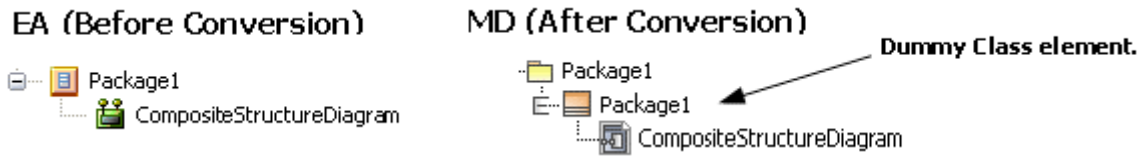
Conversion details

Dummy class

In MagicDraw, CEA, or CSM, the Composite Structure diagram(see page 325) needs a Context element to contain itself. However, in EA there is no Context element. After conversion, a dummy Class (see page 666)element will be created to represent the Context element of Composite Structure diagram.

A dummy Class will be named after the closest owner package of the Composite Structure diagram. The Composite Structure diagram and all of the Composite Structure elements will be placed inside the dummy Class element.

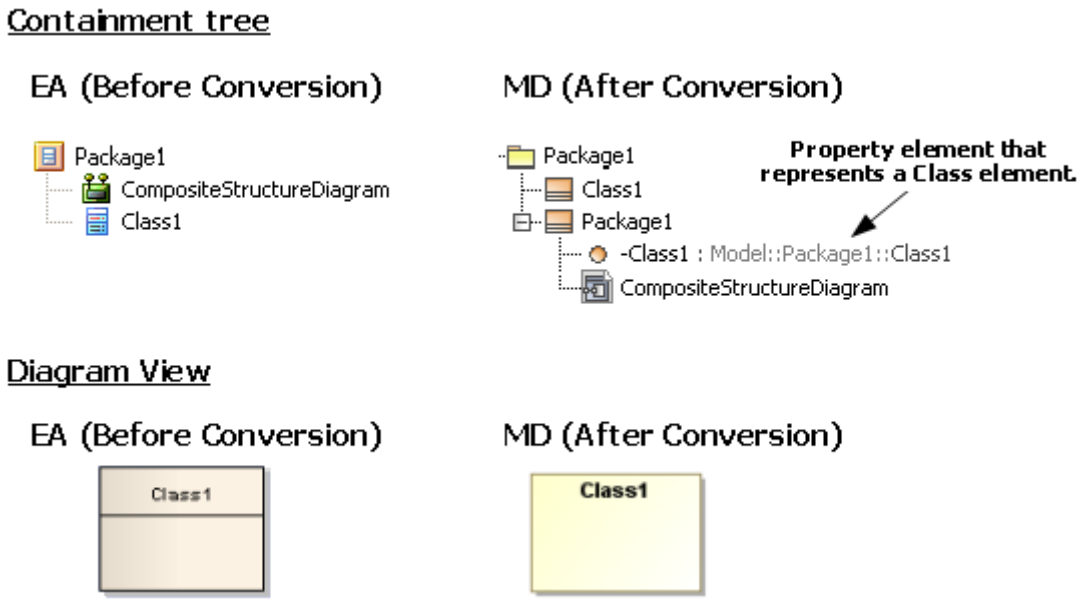
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Dummy class.

Class

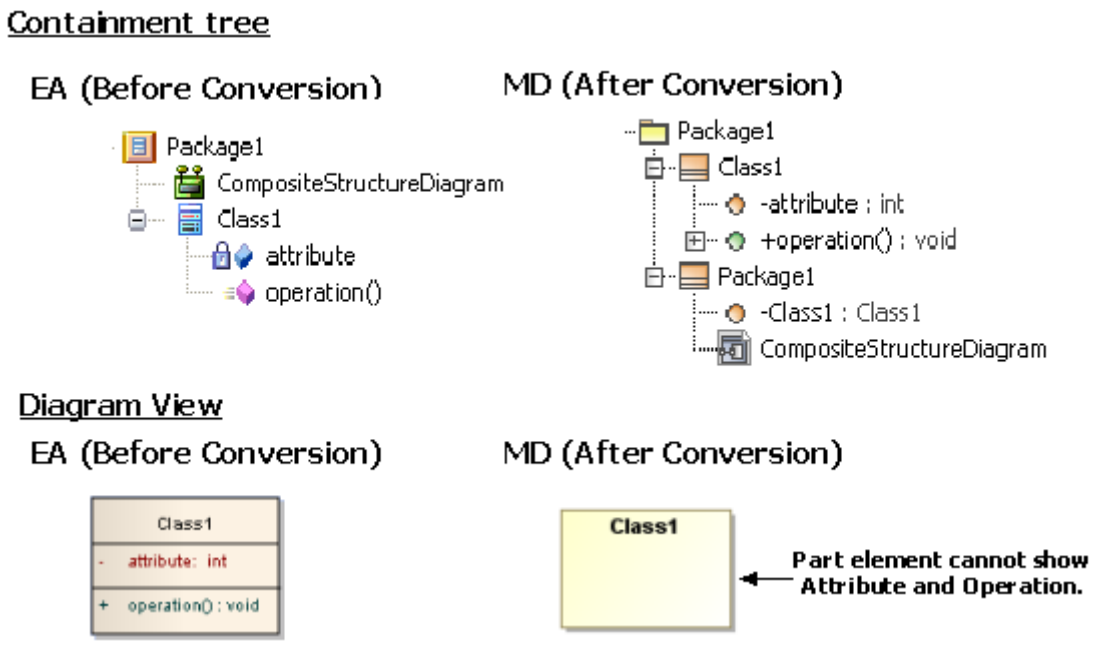
The Class element data will be normally copied even if the Class element is drawn in a Composite Structure diagram. A Property element will be created to represent the Class element. The Property element will be named after the Class element.



Property Representing a Class Element.

Class with Attributes and Operations

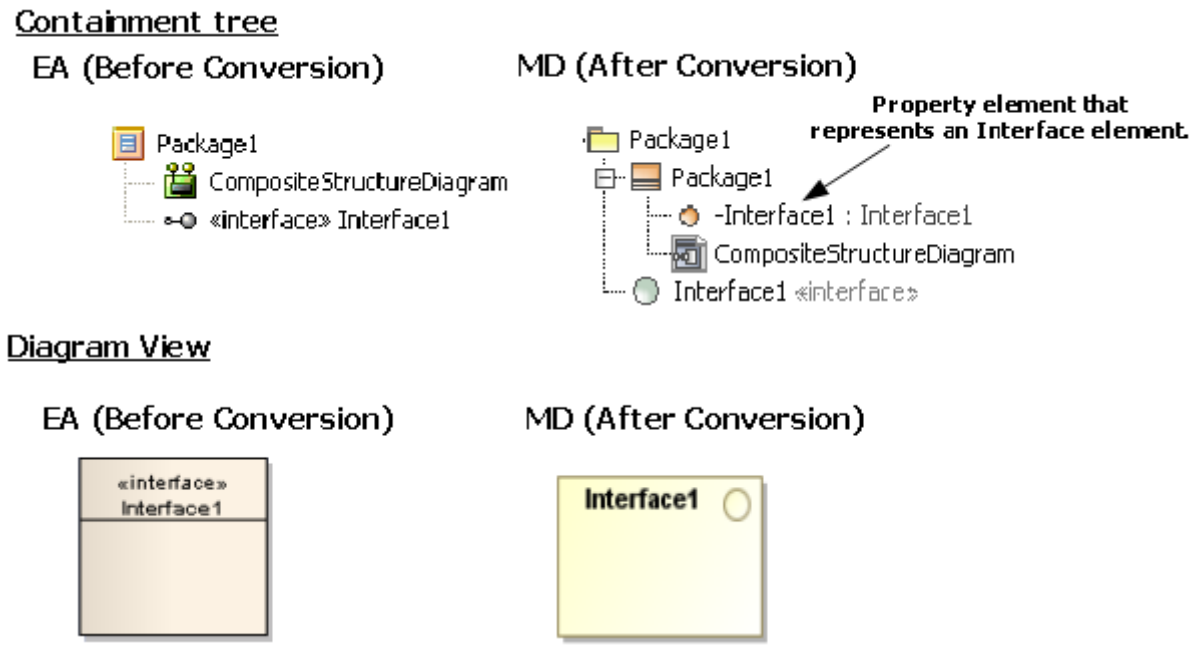
The Class view in EA will be converted to a Part (see page 804)in MagicDraw, CEA, or CSM. Attributes (see page 672)and Operations (see page 677)will not be shown in the Composite Structure diagram.



Part Element after Conversion.

Interface

The Interface (see page 741)element data will be normally copied even if the Interface element is drawn in a Composite Structure diagram. A Property element will be created to represent the Interface element. The Property element will be named after the Interface element.



Property Representing Interface Element.

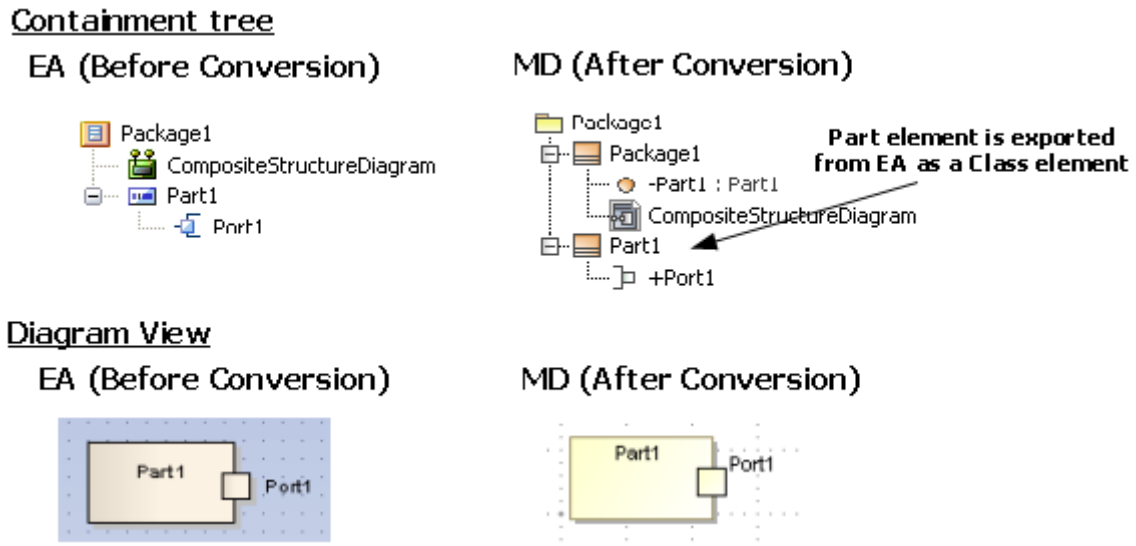
Part

There are two types of Parts(see page 804):

- Part with a port; and
- Part that sets a type to other elements.

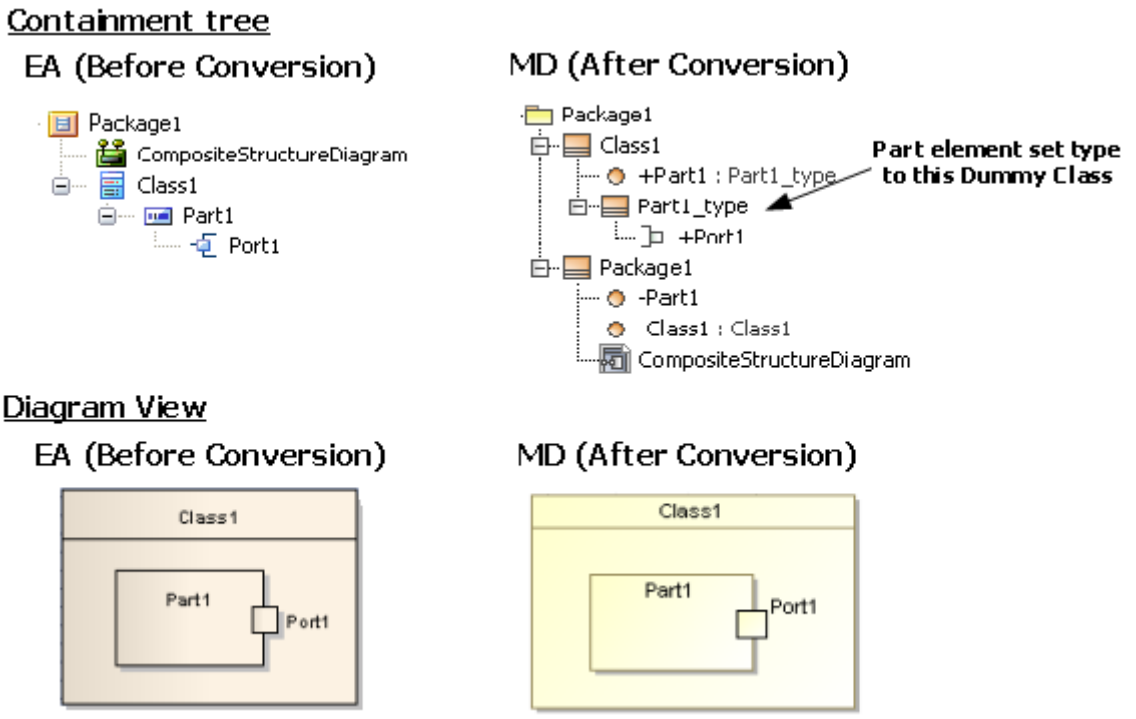
Part with Port

If you assign the type of a Part element to a Class, Component, or Node, the Part element can have a Port (see page 809). If a Part is not nested to any element, the Part element will be exported from EA as a Class. Therefore, this Part element can have a Port.



Part that does not Nest Element(s) with Ports.

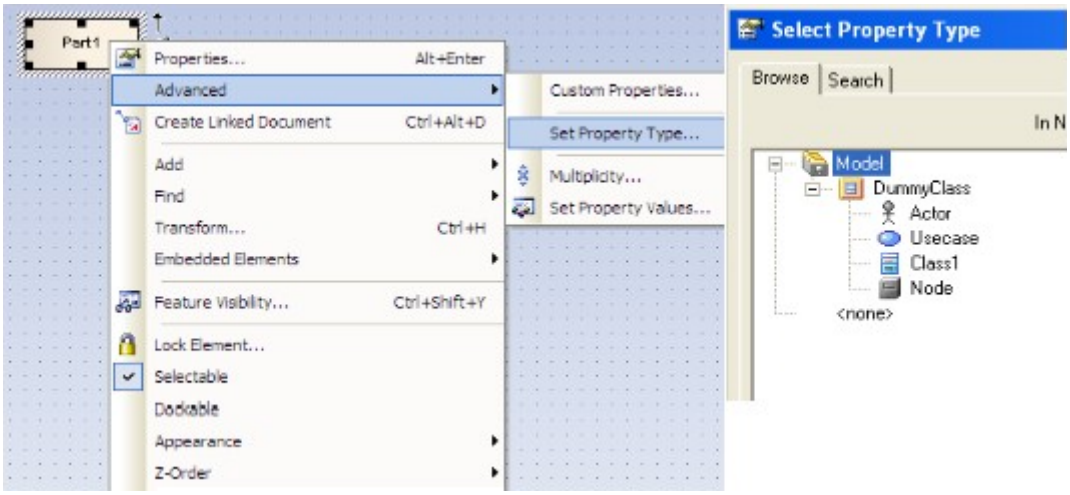
If a Part is nested to an element and the type is not set, then the Part type will be used to set type to a dummy Class. This particular Part can have a Port. A dummy Class will be created at the same level of the Part element that sets type to it. The dummy Class will be named after the Part concatenating with *_type*.



Part that Nests Element(s) with Post.

Part that sets type to other elements

The type of a Part can be set to another element (such as a Class, Actor, Component, or Usecase) by right-clicking the Part element and selecting **Advanced** > **Set Property Type** > **Select Property Type.**

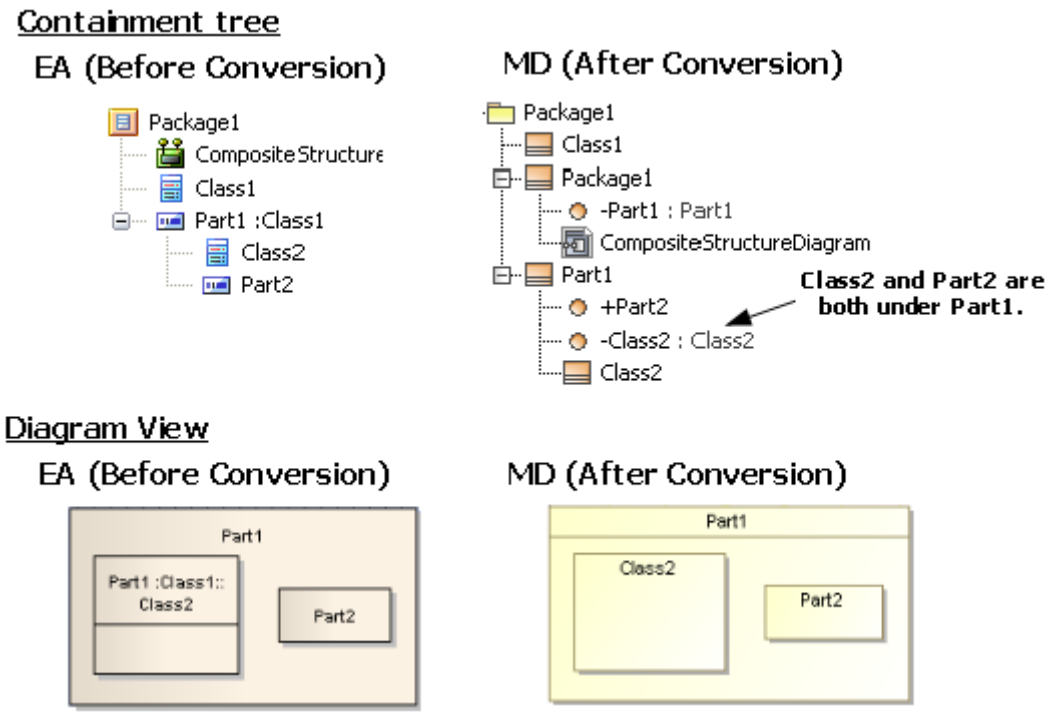


Setting Part Type.

If the Part type is set to a type that cannot be the owner of a port, the port will be removed.

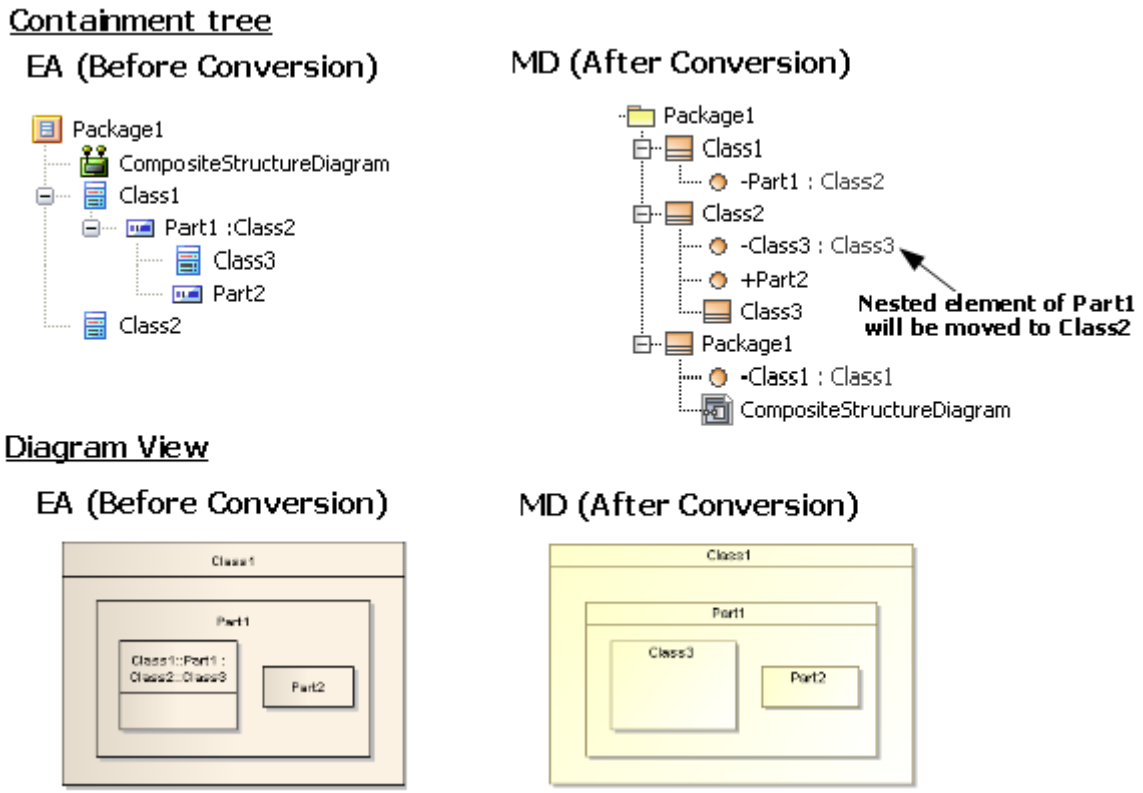
In MagicDraw, CEA, or CSM, the Property element data cannot have any elements. However, in EA, it can have elements. Therefore, if you draw a Part element and it has an element inside it, that particular element will be relocated to an element, which is the Part type.

If a Part is not nested to any element, the Part element will be exported from EA as a Class. The element inside the Part will not be relocated. If that particular Part assigns a type to other elements, its type will always be set to Class.



Part that Sets Type to Other Elements.

If a Nested Part whose type is set to another element (such as a Class, Actor, or Usecase) and this Part contains Nested elements, all of the Nested elements of the Part will be relocated to the element that is the type of this Part.



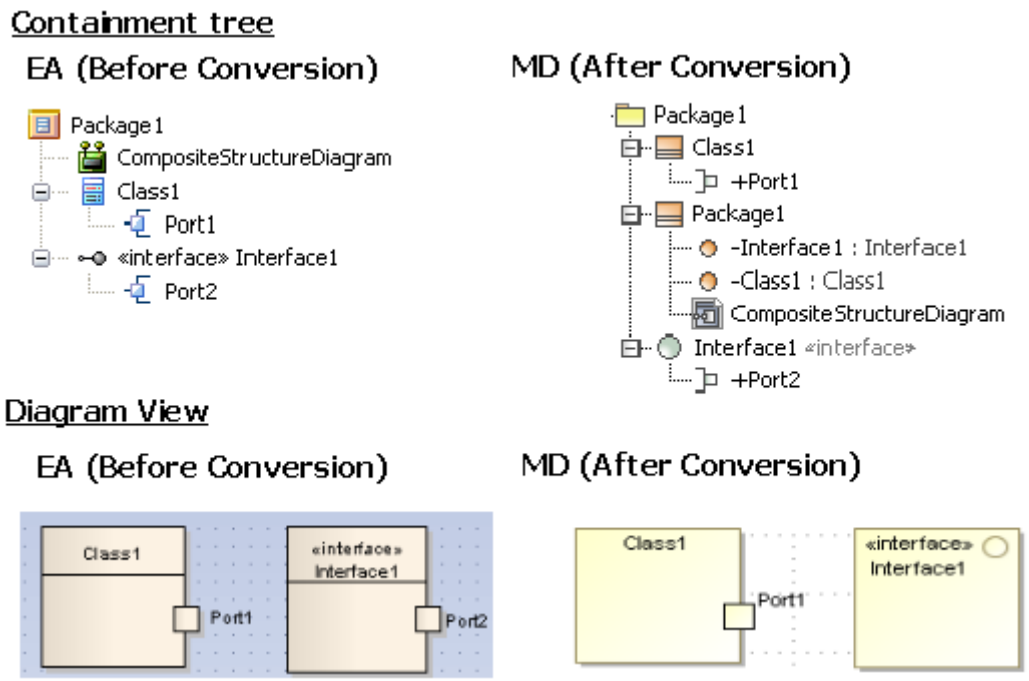
Part that Sets Type to Other Elements has Nested Element Inside Itself.

Port

In MagicDraw, CEA, or CSM, a Property element that can have a Port is the Property that sets type to a Class, Component(see page 685), or Node(see page 782), and other elements cannot have a Port. However, in EA, most of the elements can have a Port.

If a Port is created with an element that is not a Class, Component or Node, the Port will be removed from the diagram view. However, its data will be preserved.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



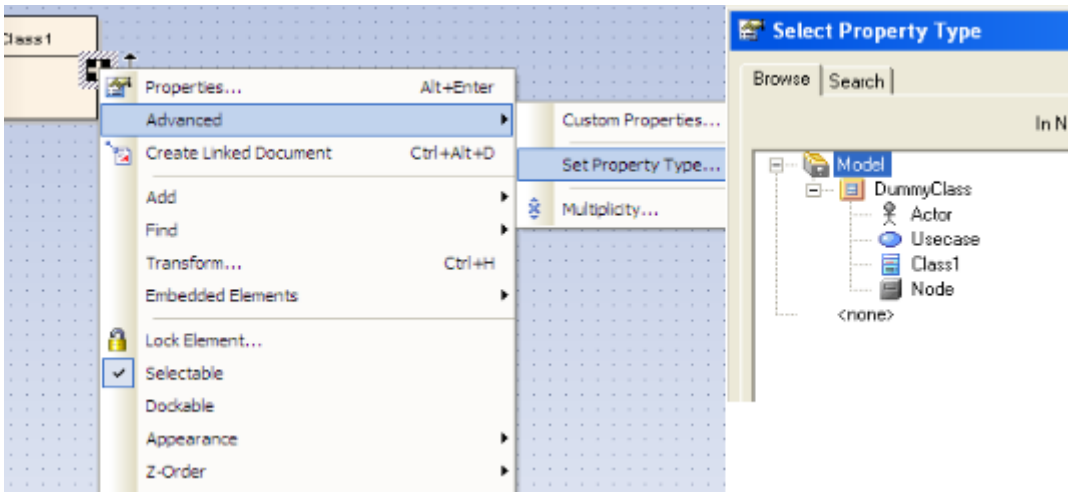
Only Class, Component, and Node Can Have a Port.

The following transformation message will open:

```
Removed view <xmi:id>: Invalid Port. Port can be added to Part that its type
is set to Class, Component and Node only.
```

Port whose type is set to other elements

You can set the type of Port element to another element (a Class, Actor, Component, or Usecase) by right-clicking the Port element and selecting **Advanced** > **Set Property Type** > **Select Property Type.**

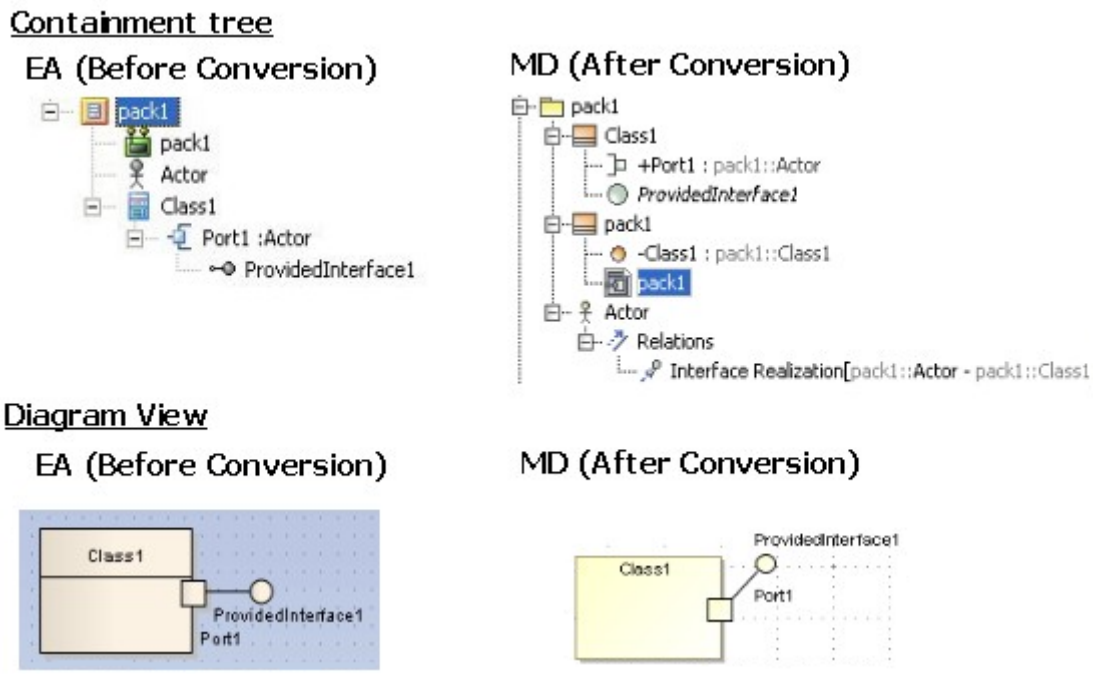


Setting Port Type.

Port whose type is set to another element and containing Expose Interfaces

If an Expose Interface is created on a Port and the Port type is set to another element, the Expose Interface data will be relocated to the Port's type element. For example, if the Port1 type is set to Actor,

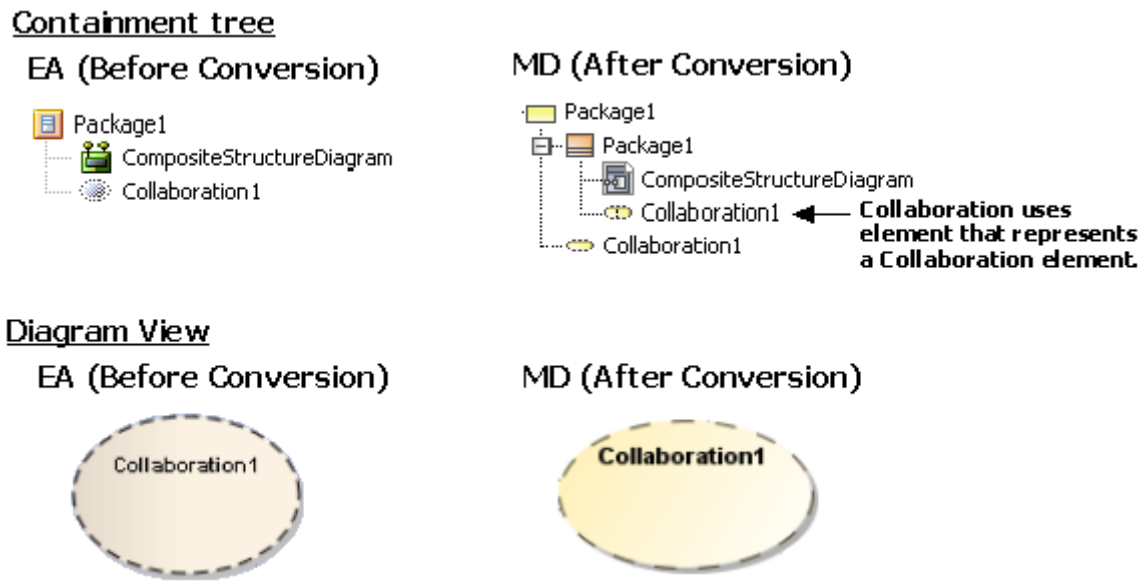
an Interface Realization (the Expose Interface data) will be placed inside the Actor (see page 646)after conversion.



Port that Sets Type to Other Element Contains Expose Interface.

Collaboration

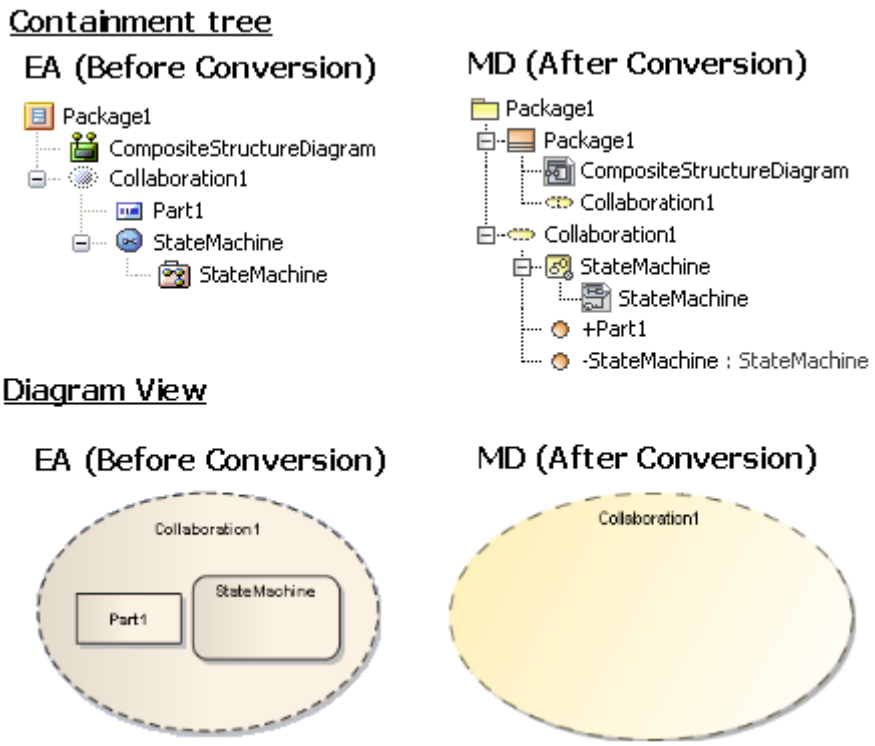
The Collaboration (see page 680)element data and its nested data will normally be copied. However, if the Collaboration element is drawn in a Composite Structure diagram, a Collaboration use element will be created to represent the Collaboration element. The Collaboration use element will be named after the Collaboration element.



Collaboration use.

Collaboration containing other elements

A Collaboration can contain only Property, Activity, State Machine, and Interaction. Elements other than these will be removed. If any element is drawn inside the Collaboration, it will be removed from the diagram view.



Collaboration Containing Other Elements.

The following transformation messages will open:

- `Removed element <xmi:id>: Invalid Element. Collaboration can contain Property, Activity, State Machine and Interaction only.`
- `Removed view <xmi:id>: Invalid Element view. Collaboration cannot be contained in any element view.`

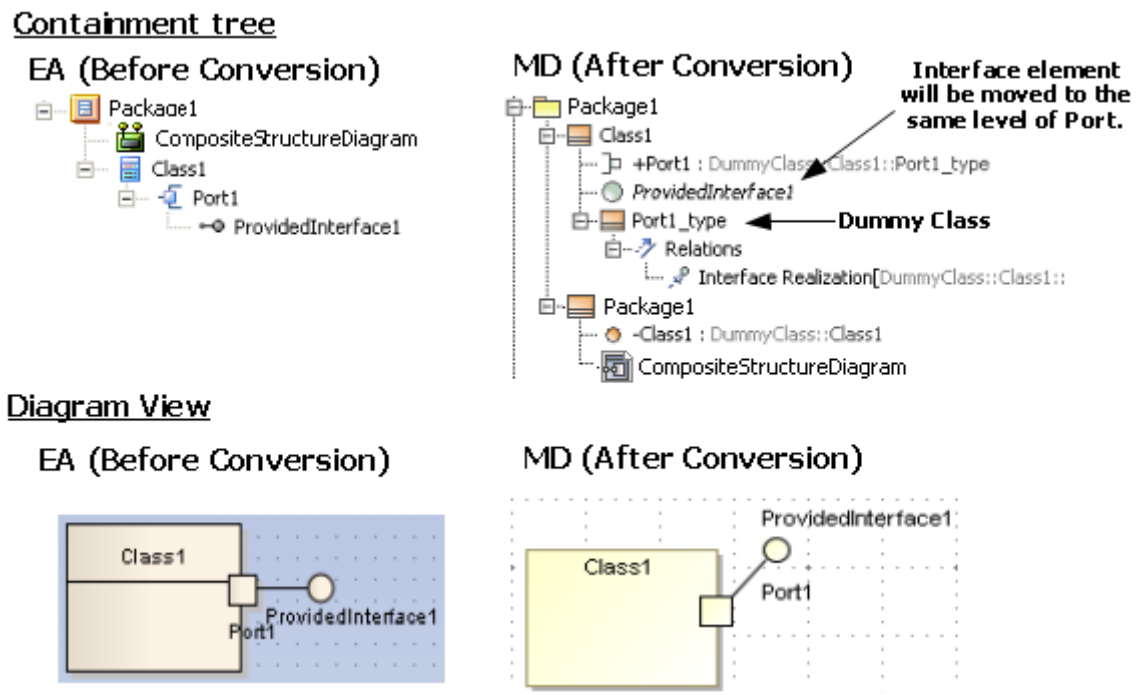
Expose Interface

Expose Interfaces are the Provided and Required interfaces in MagicDraw, CEA, or CSM. You can draw a Provided Interface or Required Interface in a Port only and set the type of the Port to another element (an Actor, Use Case, or Class). You can create that particular Port on a Part element whose type is set to a Class, Component, or Node only.

Expose Interface with Port

If you draw an Expose Interface in a Port and the Port does not assign a type to any elements, that particular Port will be used to set type to a dummy Class. The dummy Class will be created and named after the Port concatenating with *_type*. The Interface element will be moved to the same level of the Port.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Expose Interface with Port.

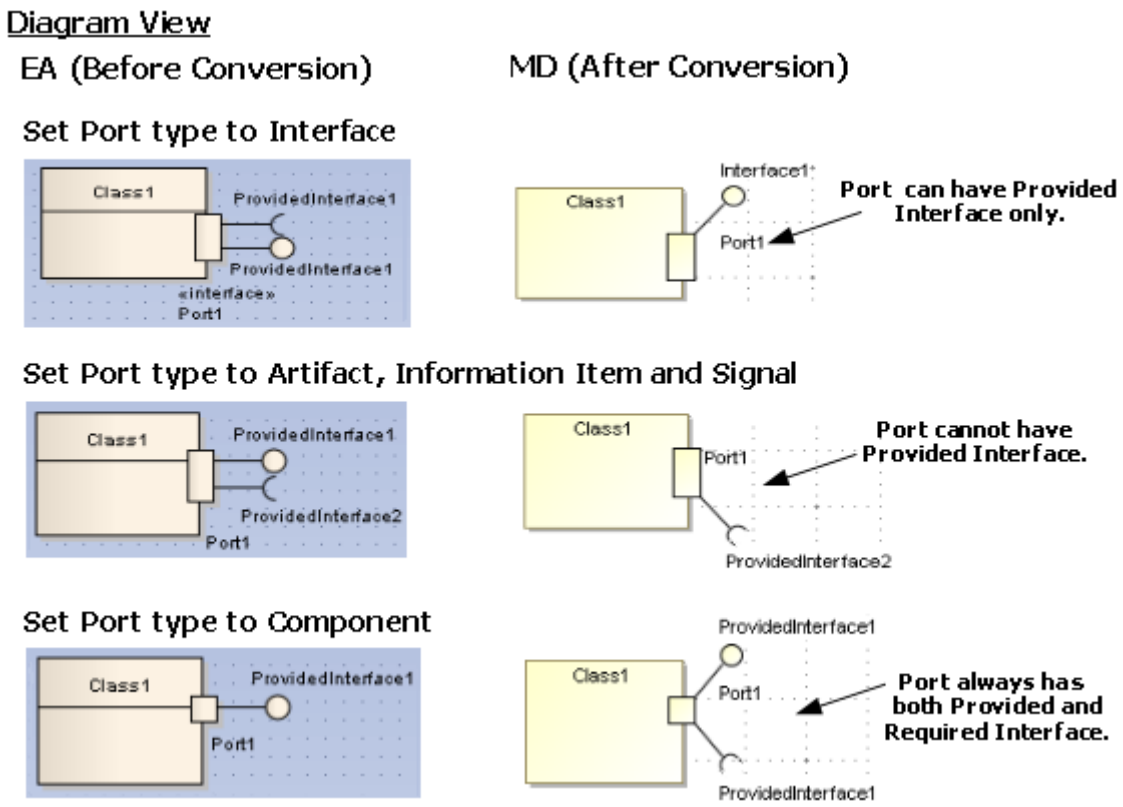
If an Expose Interface is created on a Port and the type of the Port is set to another element, the Expose Interface data will be relocated to the Port's type element (see Port (see page 186)whose type is set to another element and containing Expose Interfaces for more information).

In MagicDraw, CEA, or CSM, some elements can be used as a Port type. These elements can have either a Provided Interface or Required Interface, or both.

The following is a list of the Port's type elements with specific conditions:

- Interfaces can only have a Provided Interface.
- Artifacts (see page 649)cannot have a Provided Interface.
- Information Items cannot have a Provided Interface.
- Signals (see page 711)cannot have a Provided Interface.
- Components always have both Provided and Required Interfaces.
- The other elements can have both Provided and Required Interfaces.
- After conversion, if the type of a Port is set to an Interface and the Port has an Expose Interface, only the Provided Interface will be shown and the Interface Realization data will be removed.
- Artifacts, Information Items, and Signals cannot hold Interface Realization. If the type of a Port is set to one of them, the Provided Interface will be removed. However, this does not affect the Required Interface.
- If the type of a Port is set to a Component and the Port has an Expose Interface, it will always show both the Provided and Required Interfaces.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

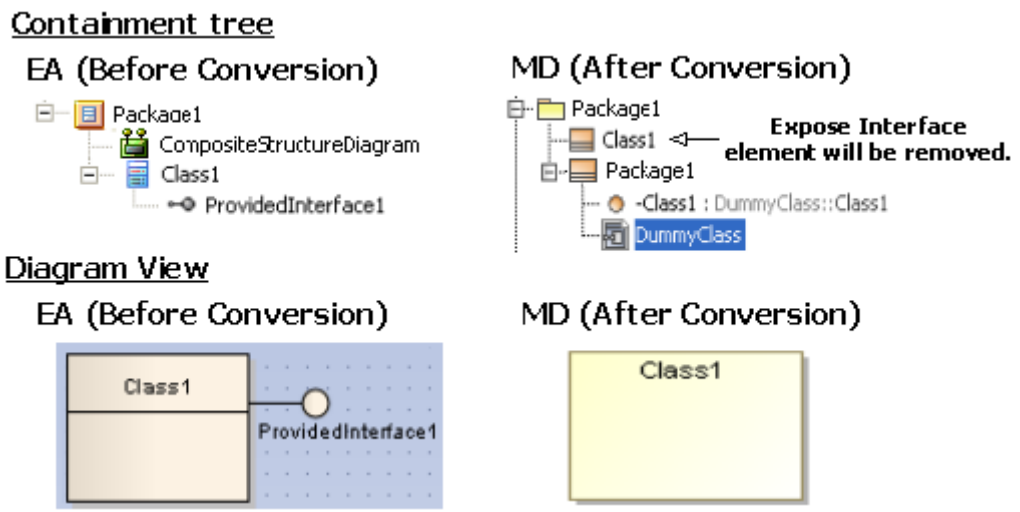


Port's Type Element with Specific Conditions.

The following transformation message will open:

```
Updated element <xmi:id>: Provided Interface conflicts with Port type. The
Port type is updated to an owner of Interface Realization.
```

If an Expose Interface is created with an element that is not a Port, the element will be removed both from the Containment tree and diagram pane.

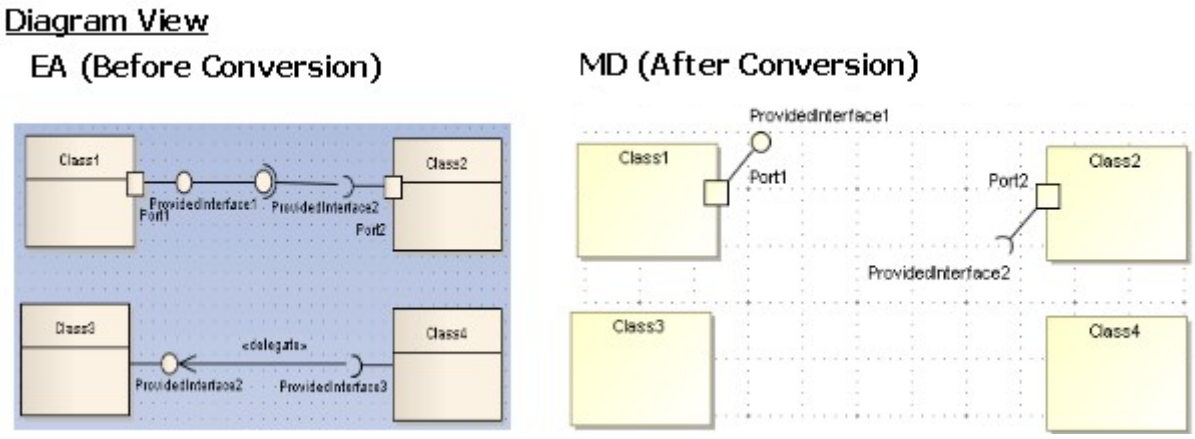


Expose Interface with Element that is not Port.

The following transformation message will open:

```
Removed element <xmi:id>: Invalid Element. Expose Interface can be added to
Port only.
```

Expose Interface with relationship
In MagicDraw, CEA, or CSM, the Provided and Required Interfaces cannot be connected with any relationship. If there is any relationship connecting to the Expose Interface, it will be removed from both the Containment tree and diagram pane.

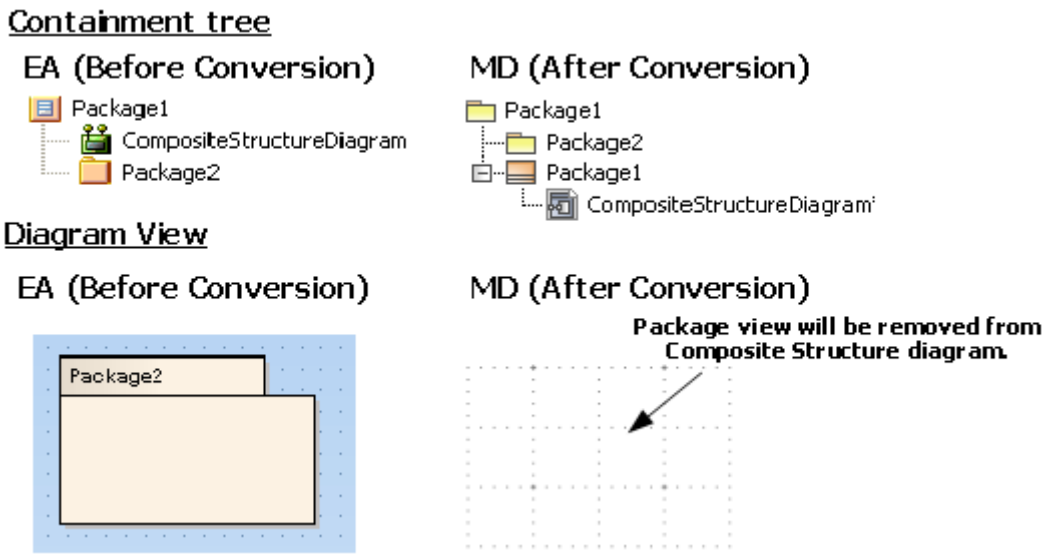


Expose Interface with Lines.

The following transformation message will open:

```
Removed element <xmi:id>: Invalid Element. Expose Interface cannot be
connected with the element.
```

Package
In MagicDraw, CEA, or CSM, a Package (see page 800)cannot be drawn in a Composite Structure diagram. If it is drawn in EA, it will be removed from the diagram once it has been converted to MagicDraw, CEA, or CSM.



Package.

Assembly
In MagicDraw, CEA, or CSM, there is no Assembly line. It will be updated to a Connector(see page 686).

The following transformation message will open:

```
Updated element <xmi:id>: Assembly updated to Connector.
```

Updating the Assembly Line to the Connector Line.

Dependency

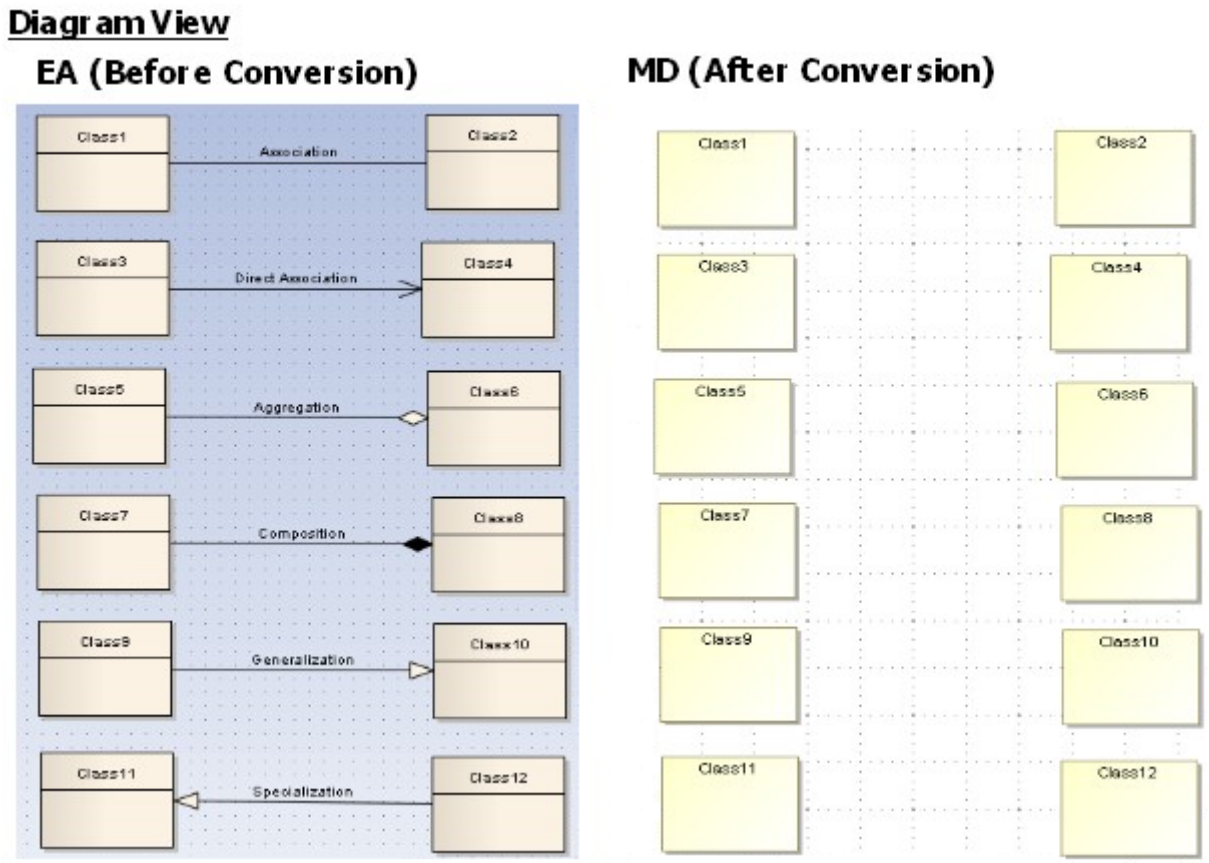
There are many relationships that EA exporter exports to Dependency(see page 699) as the following elements:

- Delegate
- Role Binding
- Represents
- Occurrence
- Nest
- Derive
- Import
- Instantiate
- Usage
- Realize
- Trace

The above relationships will be shown in the diagram view as Dependencies with a stereotype.

Removed relationships

If an Association, Direct Association, Aggregation, Composition, Generalization, or Specialization is created in a Composite Structure diagram, it will be removed from the diagram view, but its data will be preserved.



Removing lines.

The following transformation messages will open:

- `Removed view <xmi:id>: Association cannot be shown in Composite Structure diagram.`
- `Removed view <xmi:id>: Direct Association cannot be shown in Composite Structure diagram.`
- `Removed view <xmi:id>: Aggregation cannot be shown in Composite Structure diagram.`
- `Removed view <xmi:id>: Composition cannot be shown in Composite Structure diagram.`
- `Removed view <xmi:id>: Generalization cannot be shown in Composite Structure diagram.`

Transformation report

A report containing the conflicts solved during transformation, along with other transformation information (such as special mapping and removal of some irrelevant data), is required to be provided to the users.

The following is a list of the transformation messages:

- `Removed view <xmi:id>: Invalid Port. Port can be added to Part that set type to Class, Component, and Node only.`

- Removed element <xmi:id>: Invalid Element. Collaboration can contain Property, Activity, State Machine, and Interaction only.
- Removed view <xmi:id>: Collaboration cannot be contained in any element view.
- Removed view <xmi:id>: UseCase cannot be contained in any element view.
- Removed element <xmi:id>: Invalid Element. Expose Interface can be added to Port only.
- Removed element <xmi:id>: Invalid Element. Expose Interface cannot be connected with the element.
- Updated element <xmi:id>: Assembly updated to Connector.
- Updated element <xmi:id>: Provided Interface conflicts with Port type. The Port type is updated to an owner of Interface Realization.
- Removed view <xmi:id>: Association cannot be shown in CompositeStructure diagram.
- Removed view <xmi:id>: Direct Association cannot be shown in Composite Structure diagram.
- Removed view <xmi:id>: Aggregation cannot be shown in Composite Structure diagram.
- Removed view <xmi:id>: Composition cannot be shown in Composite Structure diagram.
- Removed view <xmi:id>: Generalization cannot be shown in Composite Structure diagram.

Related pages

- Composite Structure diagram(see page 325)

Interaction Overview diagram elements

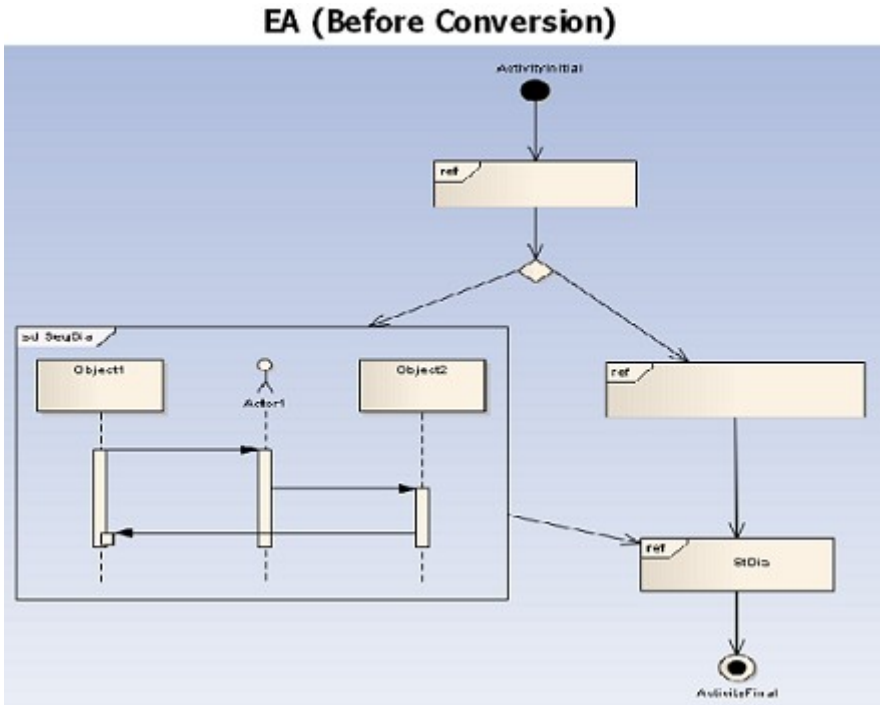
On this page

- Interaction Overview Diagram(see page 195)

The following sections describe additional EA specific Interaction Overview diagram information.

Interaction Overview Diagram

An Interaction Overview diagram(see page 332) is one of four types of Interaction diagrams. The other three are Timing(see page 403), Sequence(see page 304), and Communication (see page 302)diagrams. An Interaction Overview diagram, just like an Activity diagram(see page 311), visualizes a sequence of activities. Most of the notation elements for Interaction Overview diagrams are the same as those for Activity diagrams (such as Initial, Decision, Fork, Join, and Final Nodes(see page 782)). Interaction Occurrences and Interaction elements are two new elements in the interaction overview diagrams.



Interaction Overview Diagram.
After conversion, the Interaction Overview diagram will be placed within an Activity element.

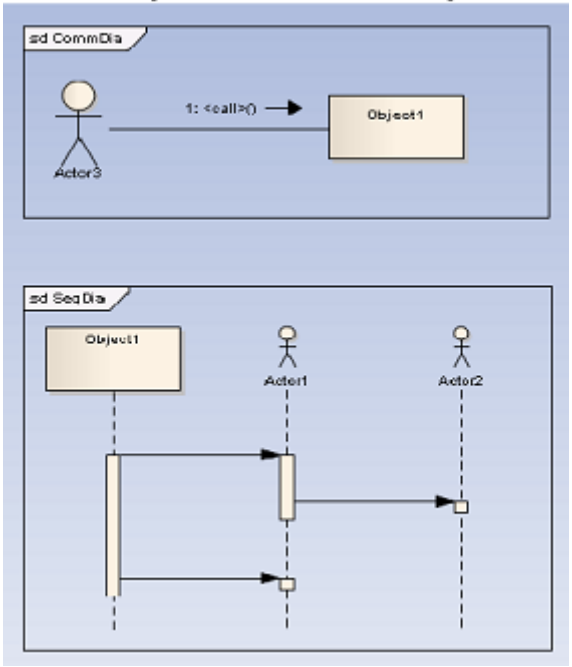
Interaction Overview Diagram Placement.

Interaction Element

Interaction elements display an inline Interaction diagram (Interaction Overview, Timing, Sequence, or Communication diagram). In EA, Interaction elements can be created to display other diagrams than those classified as Interaction diagrams.

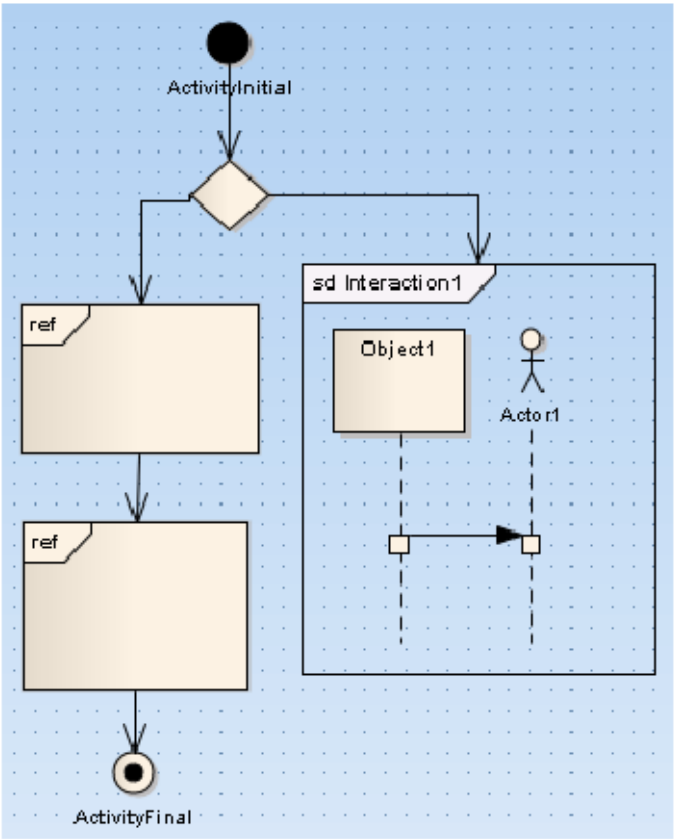
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

EA (Before Conversion)

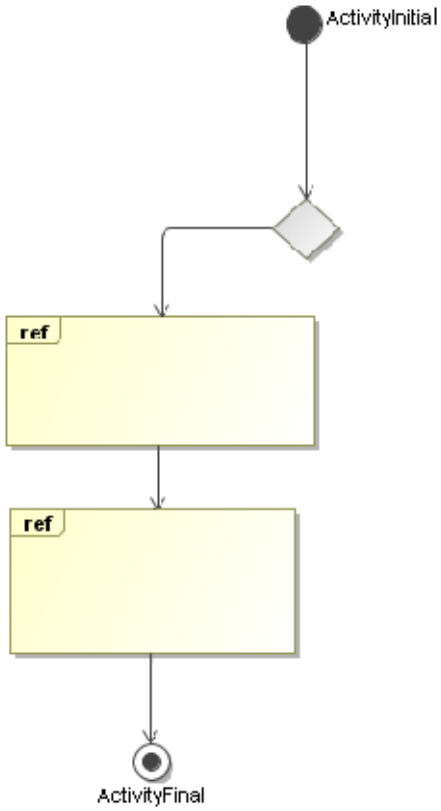


Interaction element.

EA (Before Conversion)



MD (After Conversion)

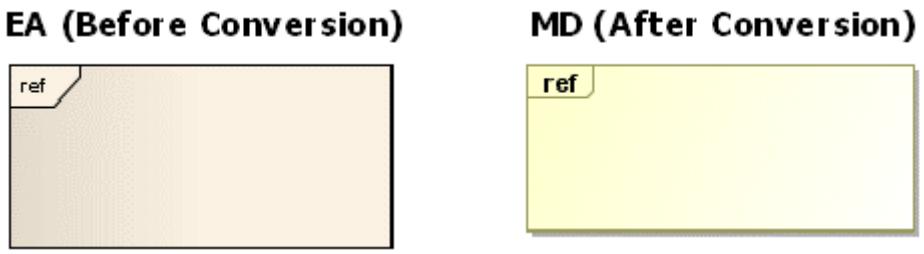


Interaction Element in Interaction Overview Diagram.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Interaction Occurrence

Interaction Occurrence elements refer to an existing Interaction diagram. They are visually represented by a rectangular frame, which can be created by dragging the Interaction diagram from the project Browser to an Interaction Overview diagram and selecting the **Interaction Occurrence** command from the shortcut menu.



Interaction Occurence element.

Related pages

- Interaction Overview diagram(see page 332)

Known Limitations and Constraints

The following table shows the conversion limitations and constraints in EA exported XMI and EA Import.

No.	Constraint name	Description
1	N-ary Association	Due to the difference between the EA exported XMI and MagicDraw, CEA or CSM XMI, the view of N-ary association(see page 660) will not be imported.
2	Diagram Legend	MagicDraw, CEA, or CSM does not have a similar element. Diagram legends will not be imported.
3	Diagram Note	The element most similar to Diagram Note is the MagicDraw, CEA and CSM diagram information table (**Option** > **Show diagram info**). However, in EA it will be mapped to the **Option** > **Show diagram details**. So Diagram Notes will not be imported to MagicDraw, CEA or CSM.
4	Text size	The size of text displayed in EA and MagicDraw, CEA, or CSM may vary. The text size in EA is mostly larger.
5	Word wrap	Due to size constraints in text displayed in element blocks, the result of word wrap in EA and MagicDraw, CEA, and CSM is likely to be different.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

No.	Constraint name	Description
6	Contact point between element and link	The position of both ends of a link element cannot be mapped to MagicDraw, CEA, or CSM because the last segments of the link element in MagicDraw, CEA, or CSM always point to the center of connected elements.
7	Display alternative image	In EA, you can display an element in a diagram by using an alternative image. MagicDraw, CEA, or CSM will not import that particular image and will use a default shape instead.
8	Link label position	A Link element, such as an Association, (see page 651)has many labels, including multiplicity and role name labels. MagicDraw, CEA, or CSM will not import the position of these labels and will use a default position instead.
9	Pin position	A Pin (see page 805)position in MagicDraw, CEA, or CSM may have been slightly moved from its original position in EA. If the pin is placed at the corner of its containing element, it will be shifted a little away from the corner (mostly in a clockwise direction).
10	Nested CallBehavior- Action	A CallBehaviorAction (see page 633)element nested with another CallBehaviorAction element. The outermost part of the element will remain. The other will be removed.
11	DataStoreNode inner element	A DataStoreNode element containing Activity Diagram related elements. Every element inside that particular DataStoreNode element will be removed. Only the DataStoreNode element will remain.
12	Object inner element	An Object element containing Activity Diagram related elements. Every element inside that particular Object element will be removed. Only the Object element will remain.
13	Lifeline position	The Lifeline (see page 746)position in MagicDraw, CEA, or CSM will not correspond to the original position in EA. The position is fixed.
14	Diagonal Sequence Message	MagicDraw, CEA, and CSM do not support Diagonal Sequence Messages.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

No.	Constraint name	Description
15	Sequence Activation Options	MagicDraw, CEA, and CSM do not support manipulating Sequence Activation through Sequence Activation Options.
16	State contains diagram	A diagram element placed inside a State (see page 848)element will be removed.
17	Region in Orthogonal State	EA exported XMI contains incorrect information when: • more than one region has identical names. • a region was created and then removed before exporting to XMI. This will cause an unexpected result after importing it into MagicDraw, CEA, or CSM.
18	Assembly Relationship in Composite Structure diagram	Assembly relationships in EA are exported to XMI as Connectors. They will then be imported to MagicDraw, CEA, or CSM as Connectors(see page 686).
19	Interaction elements as diagram frame	MagicDraw, CEA, and CSM do not import Interaction elements displayed as a diagram frame in an Interaction Overview(see page 332) diagram.
20	Message timing details	Duration Observation, and Timing Observation are not imported to MagicDraw, CEA, or CSM.
21	Concurrent State Regions in StateMachine	Adding and removing multiple Concurrent State Regions to and from StateMachine in EA can cause the EA XMI to be incorrectly exported. If the EA exported XMI is in this state, the result of the StateMachine(see page 848) imported to MagicDraw, CEA, or CSM cannot be determined. The same problem also occurs if there are multiple Concurrent State Regions with the same name.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

No.	Constraint name	Description
22	Combined Fragment	Adding and removing multiple Interaction Operands to and from Combined Fragment in EA can cause the EA XMI to be incorrectly exported. The problem can be fixed by importing the EA XMI back to a new project in EA and exporting it back before importing it to MagicDraw, CEA, or CSM.
23	Problem occurs during conversion of Sequence diagram(see page 304).	Importing EA XMI to MagicDraw, CEA, or CSM sometimes results in the following error message: “Problem occurred during conversion of Sequence diagram”. If this occurs, please try to import the XMI back to a new project in EA and export it back again before importing it to MagicDraw, CEA, or CSM.

Exporting UML models

MagicDraw allows exporting projects to the following file formats:

- **UML XMI 2.4 file.** You can export a project to file formats that are supported by MagicDraw.
- **MagicDraw Native XML file.** You can export a project to an .xml file format.
- **EMF Ecore file.** You can export either the whole project or selected packages to an .ecore file.
- **MOF XMI file.** You can export a project to an MOF (both CMOF and EMOF) XMI file.
- **Eclipse UML2 (v1.x, v2.x, v3.x, v4.x, v5.x) XMI file.** You can export a project to an Eclipse based UML2 (v1.x / v2.x / v3.x / v4.x / v5.x) compatible XMI file.

To export a project

1. On the main menu, click **File** > **Export To.**
2. Select a file format you want to export your project to.
3. The following actions depend on the dialog that corresponds to the selected file format.

Exporting projects as Eclipse UML2 (v1.x / v2.x / v3.x / v4.x / v5.x) XMI files

The export of a MagicDraw model to an Eclipse based UML2 (v1.x / v2.x / v3.x / v4.x / v5.x) compatible XMI file enables the interchange of the UML2 models for further manipulations and transformations with the most popular MDA tools. You can also export your project as a template.

To export a project as an Eclipse UML2 (v1.x / v2.x / v3.x / v4.x / v5.x) XMI file

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

1. Open a project you want to export as an Eclipse UML2 (v1.x / v2.x / v3.x / v4.x / v5.x) XMI file.
2. On the **File** menu, click **Export To** and select one of the following command:
   - **Eclipse UML2 (v1.x) XMI file**.
   - **Eclipse UML2 (v2.x) XMI file**.
   - **Eclipse UML2 (v3.x) XMI file**.
   - **Eclipse UML2 (v4.x) XMI file**.
   - **Eclipse UML2 (v5.x) XMI file**.
3. Specify a location for exported project files.
4. Click **Export**.

- A project exported as an Eclipse UML2 (v1.x) XMI file is saved with the *.uml2* file extension.
- A project exported as an Eclipse UML2 (v2.x / v3.x / v4.x / v5.x) XMI file is saved with the *.uml* file extension.

To change export property values

1. From the **Options** menu, select **Environment**.
2. Click one of the following options group:

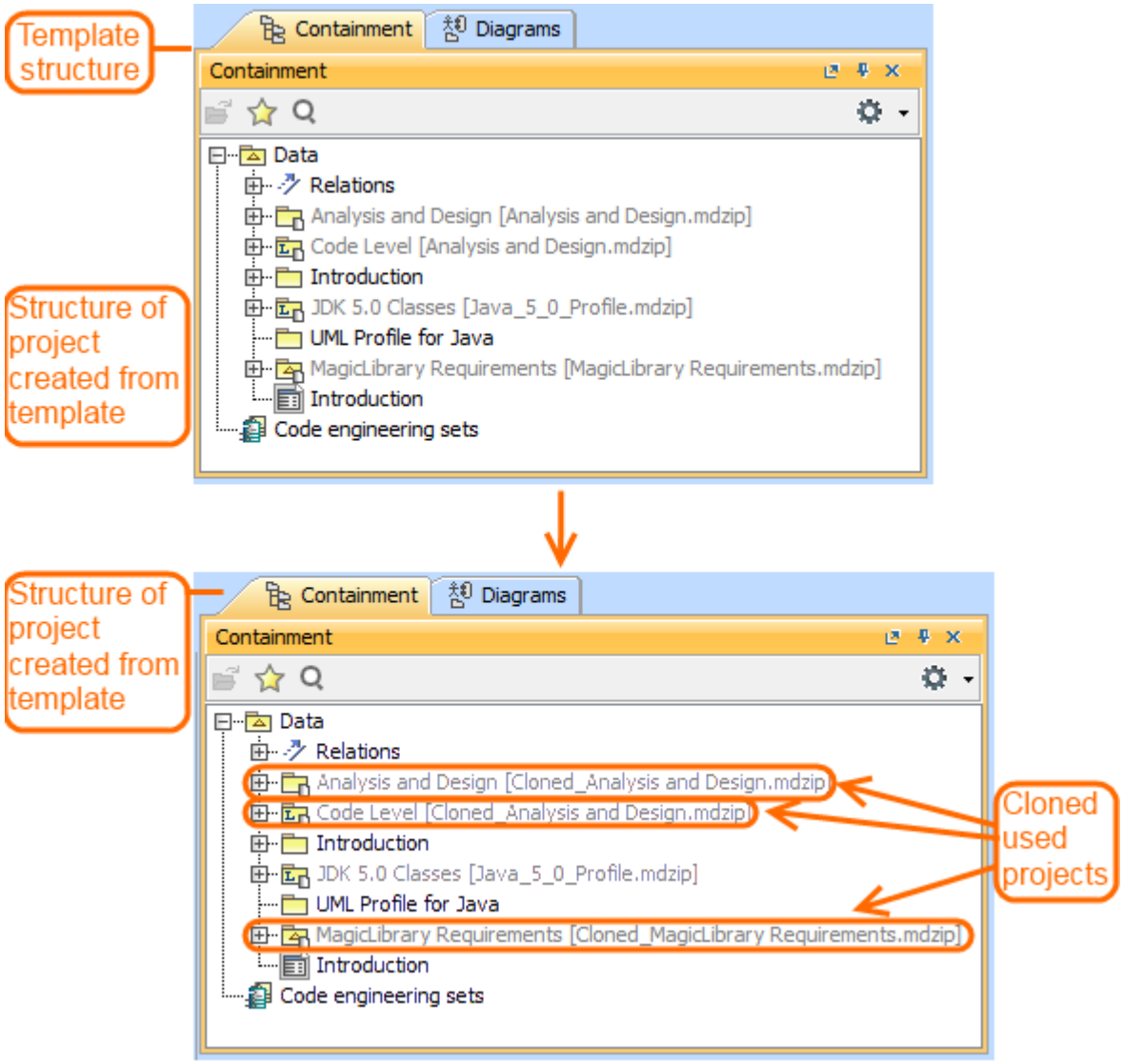
   - **Eclipse UML2 (v1.x) XMI file**.
   - **Eclipse UML2 (v2.x) XMI file**.
   - **Eclipse UML2 (v3.x) XMI file**.
   - **Eclipse UML2 (v4.x) XMI file**.
   - **Eclipse UML2 (v5.x) XMI file**.

3. In the property list, specify the desired property values. You can see descriptions of each property in the description area below the property list.

Creating and using project templates

Project template is a customized project pattern. It serves as a starting point for creating a new project in a predefined format.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Related pages

- Exporting projects as templates(see page 203)
- Creating custom project templates(see page 204)
- Creating new projects from templates(see page 206)

Exporting projects as templates

You can save (or export) the created project as a template and then use the same project for creating new projects.

> ⚠ This functionality is available in Standard, Professional, Architect, and Enterprise editions.

To export a project as a template

1. Open the project you want to export as a template.
2. From the **File** menu, select **Export To** > **Template**. The **Export Template** dialog opens.
3. Type the name and the description of the template.
4. Click **OK**. The project is exported as template.

- According to the template name in the **Export Template** dialog, the *.mdzip* file will be created in *<MagicDraw installation folder>\templates*. For example, type the name *LibraryTemplate* to save file as *LibraryTemplate.mdzip*.
- As of MagicDraw 17.0.3, when creating a new project from template, you can clone the custom used projects that exist in the template. Read more in Creating new projects from templates(see page 206).

Creating custom project templates

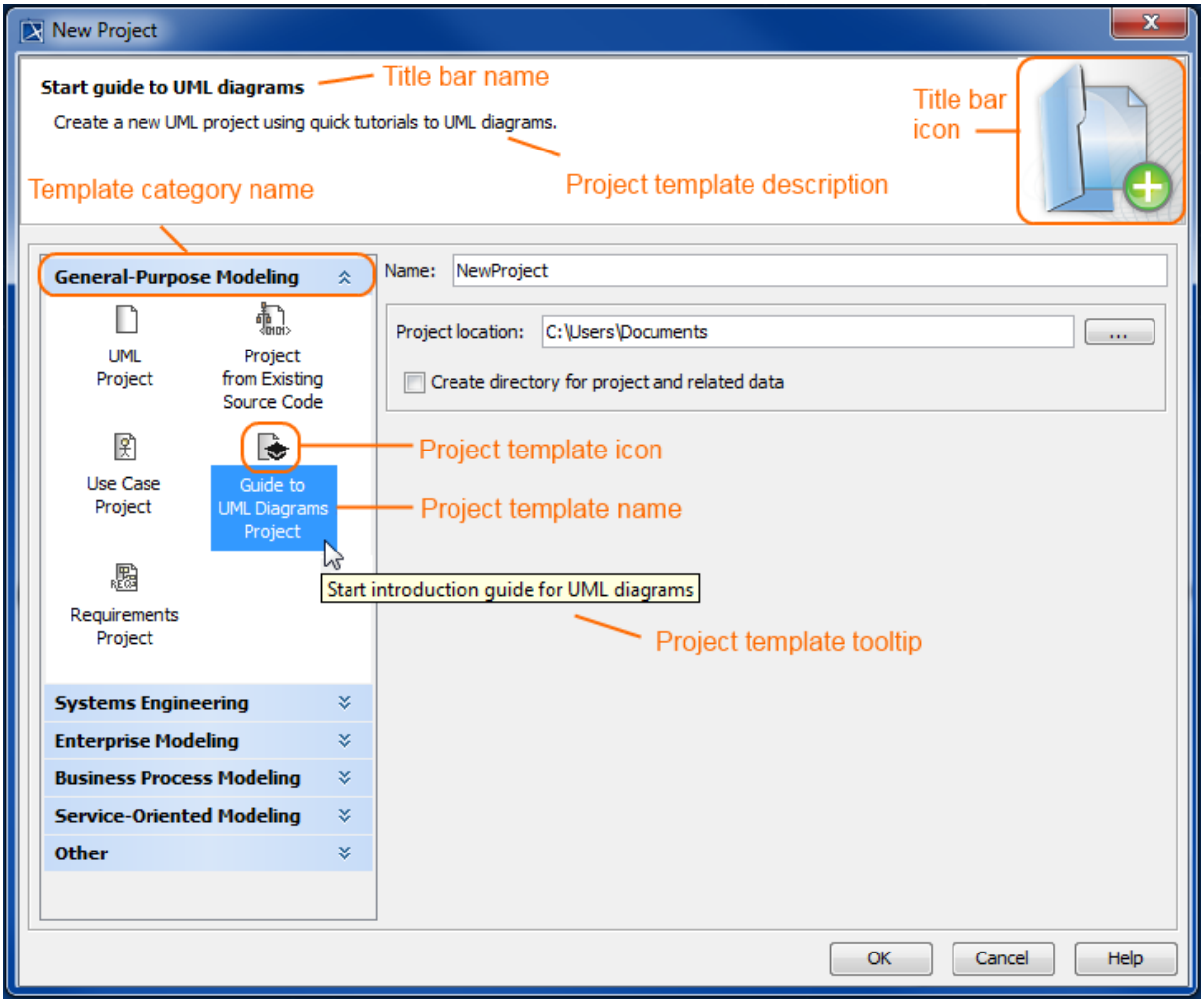
You can create your custom project template that appears in the **New Project** dialog.

Each project template is saved in a separate folder in *<MagicDraw installation directory>\templates*. The template folder should consist of the following files:

- A MagicDraw project file containing predefined initial project data.
- An *.xml* file describing project template data. See the following figure wherein all customizable template data are depicted.
- An image of a dialog title bar icon that is displayed in the title bar of the **New Project** dialog. The recommended maximum image size is 80 x 80 pixels. Recommended image file formats are *.gif* or *.png*.
- An image of a template icon that is displayed in a particular project template category of the **New Project** dialog. The recommended maximum image size is 24 x 24 pixels. Recommended image file formats are *.gif* or *.png*.

Folder name, MagicDraw project file name and *.xml* file name must be identical.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



While creating a project template, you can also create a category wherein your template can be located. If you need to have several project templates under the same category, in the .xml file, define the same category name for these templates.

To create a custom project template

---

1. If MagicDraw is running, close it.
2. In *<MagicDraw installation directory>\templates* create a new folder for your template files.
3. Paste a MagicDraw project file and icon image files to the created folder. The project file name should be the same as the folder name.
4. In the same folder, create an .xml file and name it the same name as a folder name. The .xml file content should be as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<custom_template>
<icon>Project_Template_Icon.gif</icon>
<iconLabel>Projecttemplatename</iconLabel>
<perspective>MagicDrawperspectivename</perspective>
<tooltip>Projecttemplatetooltiptext</tooltip>
<banner_title>Titlebarname</banner_title>
<banner_description>Projecttemplatedescription</banner_description>
<category>Templatecategoryname</category>
<banner_icon>Title_Bar_Icon.png</banner_icon>
```

```
<cloneModules/>
</custom_template>
```

5. Save the file and start MagicDraw. Your created project template is displayed in the **New Project** dialog in the category defined by you. Select your template and create a new project.

> The easiest way to create a custom template is to copy and paste the **Guide to UML Diagrams Project** folder with the different name and edit its inner files. This folder comes with the default MagicDraw installation and is located in *<MagicDraw installation directory>\templates*.

Creating new projects from templates

> This functionality is available in Standard, Professional, Architect, and Enterprise editions.

By default, the following templates are available in the **New Project** dialog: C#, C++, CIL, CORBA IDL, Guide to UML Diagrams Project, Java, Metamodeling, RUP, Use Case modeling, WAE, and WSDL.

A new project created from the template will contain specific structure, model elements, and stereotypes that were predefined in the template. As of version 17.0.3, when creating a new project from the template, you can select to clone the custom used projects that exist in the template.

To create a new project from template

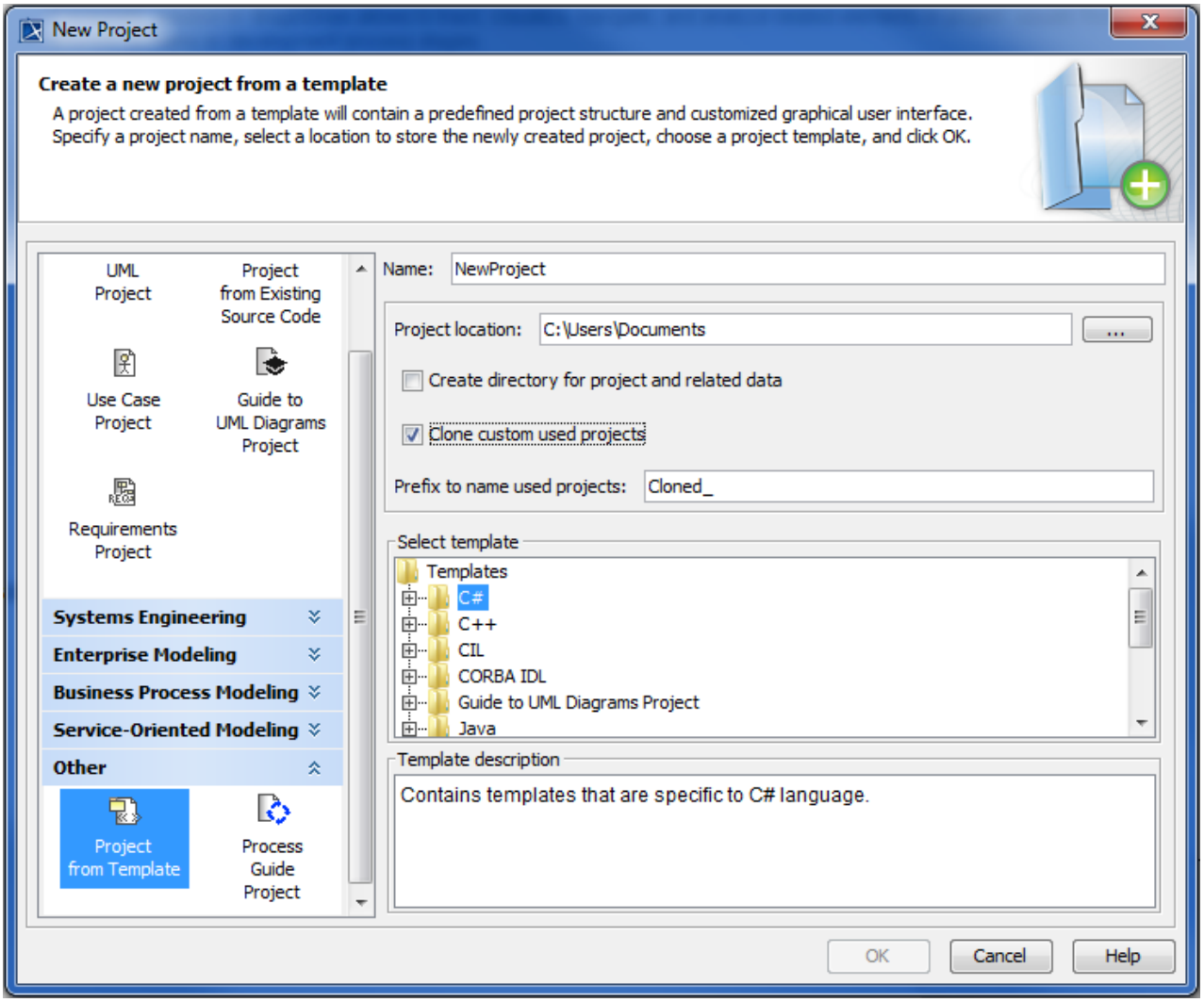
---

1. Do one of the following:
   - From the **File** menu, select **New Project**.
   - On the main toolbar, click the **New Project** button.
   - Press Ctrl + Shift + N.

   In all cases, the **New Project** dialog opens.

2. Expand the **Other** group and select the **Project from Template** icon.
3. Specify the new project name in the **Name** box.
4. Click the button to select the location in your file system to store the new project.
5. Select the template from the templates tree and click **OK**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The following table provides descriptions of the **New Project** dialog with the *Project from Template icon* selected under the **Other** category.

Element name	Type	Description
Other	Category	Select the **Other** category to see the list of the available templates.
Project from Template	Icon	Select the **Project from Template** icon to create a new project from a template.
Name	Text box	Type the name of the new project to create from a template.

Project location	Text box	Click the button to select a new project location. The **Select Location** dialog opens.
Create directory for project and related data	Check box	Click the check box to create a new folder for the new project. The folder name will be the same as the specified project name.
Clone custom used projects	Check box	Click to clone used projects from template project. In the new project from template, the copies of the used projects will be created.
Prefix to name used projects	Text box	Type the prefix for the name of each cloned used project. The default value is *Cloned*._ For example, if a used project of the template project is named *Requirements,* then the name of the cloned used project in the newly created project is *Cloned_Requirements*. The text box is available only if the **Clone custom used projects** check box is selected.
Select template	List	Select template from the list. The list represents the folders and files that are located in *<MagicDraw installation folder>\templates*.
Template description	Text area	The predefined template description is displayed. Note that this field is not editable.

You can also import the desired template into your project. Select **File** > **Import From** > **Another Project** for this.

Related pages

- Exporting projects as templates(see page 203)
- Creating custom project templates(see page 204)

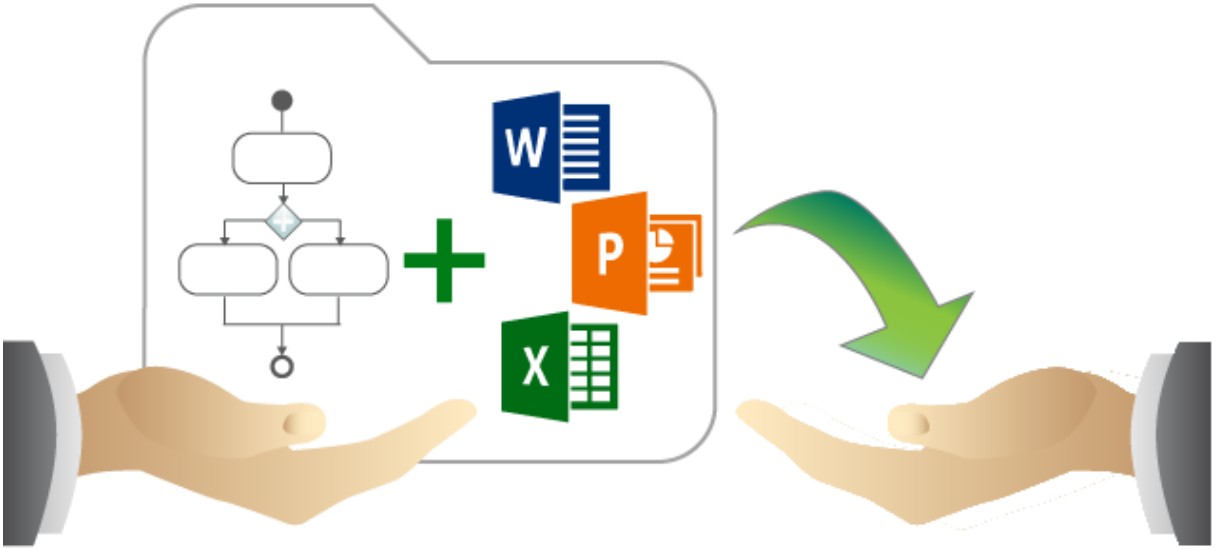
Projects with file attachments

You can attach any type of file from the local file system to your project.

Attached files are bundled with the project and remain attached after transferring the project, keeping project references to these files not broken.

The contents of the attached file can be modified and the changes are instantly saved to the project. Different versions of a server project can refer to different versions of the same attached file.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Related pages

- Attaching, reattaching, and detaching files(see page 209)
- Working with attached files(see page 211)

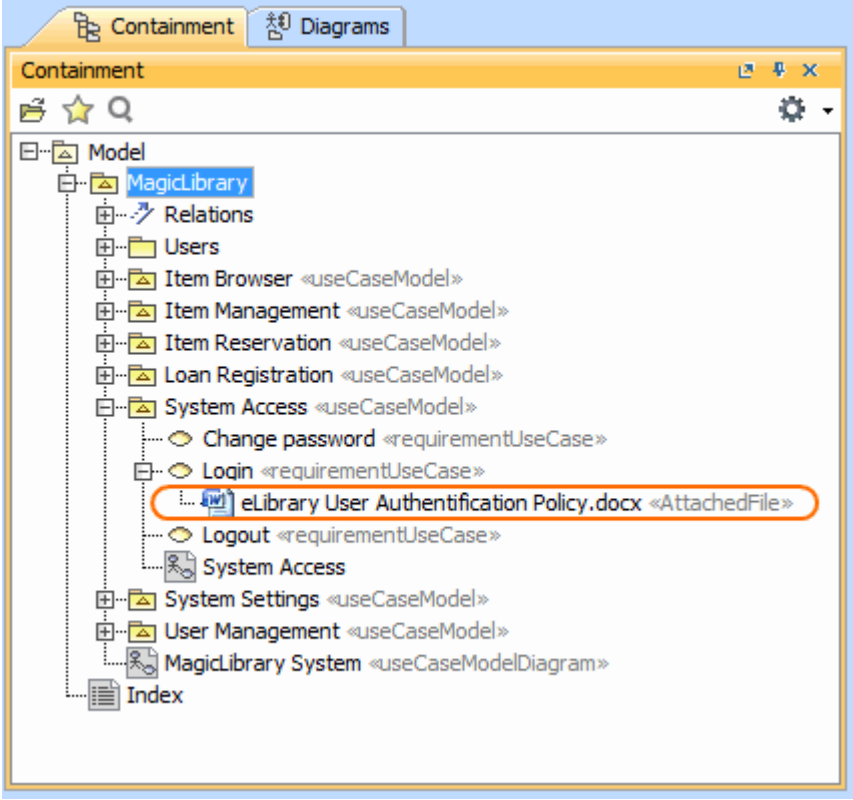
Attaching, reattaching, and detaching files

On this page

- Attaching files(see page 210)
  - Dragging the file icon to the diagram pane(see page 210)
  - Dragging the file icon to the Containment tree(see page 210)
  - Using a command from the shortcut menu of the element selected in the Containment tree(see page 211)
- Reattaching files(see page 211)

When a file is attached to a project, such file becomes a model element called Attached File.

Attached Files are displayed in the Model Browser.



Attaching files

You can attach files in one of the following ways:

- - Dragging the file icon to the diagram pane(see page 210)
  - Dragging the file icon to the Containment tree(see page 210)
  - Using a command from the shortcut menu of the element selected in the Containment tree(see page 211)

Dragging the file icon to the diagram pane

To attach a file by dragging the file icon to the diagram pane

1. In the file system, select the icon of the file you need to attach.
2. Drag it to the diagram pane.

The shape of the newly created Attached File element appears on the diagram pane. The diagram owner becomes the owner of the file as well.

Dragging the file icon to the Containment tree

To attach a file by dragging the file icon to the Containment tree

1. Drag the file icon onto an element in the Containment tree.

2. In the shortcut menu, select **Create Attached File**.

A new Attached File element appears in the Containment tree. The element on which you dragged the file icon becomes the owner of the file.

Using a command from the shortcut menu of the element selected in the Containment tree

To attach a file by using a command from the shortcut menu of an element

1. In the Containment tree, right-click a possible owner for the file to be attached to.
2. In the shortcut menu, select **Create Element** > **Attached File**.
3. In the **Select File to Attach** dialog, select the file and click **Attach**.

A new Attached File element with its name in the edit mode appears in the Containment tree. If you do not need to change the file name, press Enter. Otherwise, type a new name and then press Enter.

> Regardless of the way used to create an Attached File element, it possesses the same default properties, such as **Attached At**, **Modified At**, **Size**, **Author**, and **File**. You can see these properties, their descriptions, and values in the **Specification** window of the Attached File element. If you need to replace the attached file, you can do this using the **File** property value editor.

Reattaching files

To reattach the file

1. In the Containment tree, select the attached file you need to replace.
2. Press Enter to open its **Specification** window.
3. In the **Quick Filter** box, type *file*.
4. Click the cell of the **File** property value, then click .
5. In the **Select File to Attach** dialog, select the new file to attach, and click **Attach**.
6. In the **Specification** window, click **Close**.

The file is reattached, and the **Modified At** property value of the appropriate Attached File element is updated.

If you no longer need the attached file, you can simply remove it from the model just like any other element. Select it in the Containment tree and press Delete. If you have its shape on the diagram, select the shape and press Ctrl+D.

Working with attached files

The actions you can perform with attached files are listed below:

- Modifying the contents(see page 212)
- Inserting into HTML text(see page 212)
- Exporting back to the local file system(see page 213)
- Creating image shapes(see page 213)

Attached files can also be included in reports. The attached file is saved in the file system together with the generated report. In this report, any hyperlink to that attached file defined in the model is replaced with reference to the appropriate file in the file system.

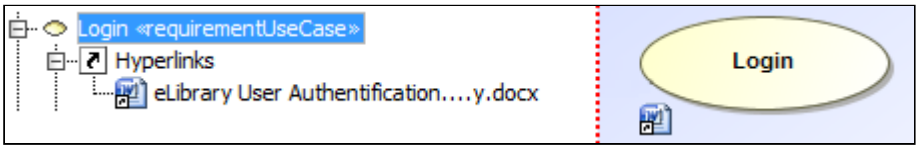
Modifying the contents

Attached files can be opened directly from MagicDraw and reviewed or modified with appropriate programs (documents – with MS Word, worksheets – with MS Excel, HTML files – with any HTML editor).

To open an attached file, do one of the following

- In the Containment tree, double-click the attached file.
- In the Containment tree, right-click the attached file and select **Open File**.
- Double-click either the element that has an active hyperlink to the attached file, or the shape of that element (see the following figure).

The file is opened in the appropriate program in your operating system. You can review or modify it.



After closing the modified file, you can choose whether to save it in MagicDraw or not. Saving the latest version of the file in MagicDraw updates the **Modified At** property value of the appropriate Attached File element.

If the attached file is read-only, the changes made to it cannot be saved. To allow modifying the file, do one of the following:
- If the file is attached to used project, change the accessibility mode of this used project from *read-only* to *read-write*.
- If the file is attached to server project, lock the Attached File element for edit.

Inserting into HTML text

Attached image files can be inserted into HTML texts and displayed on diagrams.

To insert an attached image file into HTML text

1. In the Containment tree, right-click the attached image file and select **Copy Image URL**.
2. On a diagram pane, draw a text box with the HTML mode on.
3. Open the **Advanced HTML Editor** dialog(see page 1060).
4. On the toolbar, click either

   - **Insert image** button, then paste the URL into **Picture source** field.
   - **Insert attached image** button, then find and select the attached image file in the list.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

> If you are working with a server project, make sure that the path to the image location is accessible from any computer that has Internet connection. Otherwise, the image won't be displayed, when the project is opened on another computer.

5. Click **OK** twice to close both the **Picture Properties** and **Advanced HTML Editor** dialogs.

The inserted image appears on the diagram. Now if you transfer the project to another computer, the diagram continues displaying the image.

You can also insert images that are not yet attached to the project model. Read more about Inserting images into HTML text(see page 1062).[59]

Exporting back to the local file system

To export the attached file back to the local file system

---

1. In the Containment tree, right-click the attached file and select **Save File As** .
2. Select the location in the local file system and click **Save**.

The file is saved in the local file system.

Creating image shapes

You can use an attached files of images (*.gif*, *.jpg*, *.jpeg*, *.svg*, or *.png*) to create an image shapes on the diagram pane(see page 24).

To create image shape using attached image file

---

1. In the Containment tree(see page 32) select already attached file or attach a new one. How to attach a file in the Containment tree >>[60]

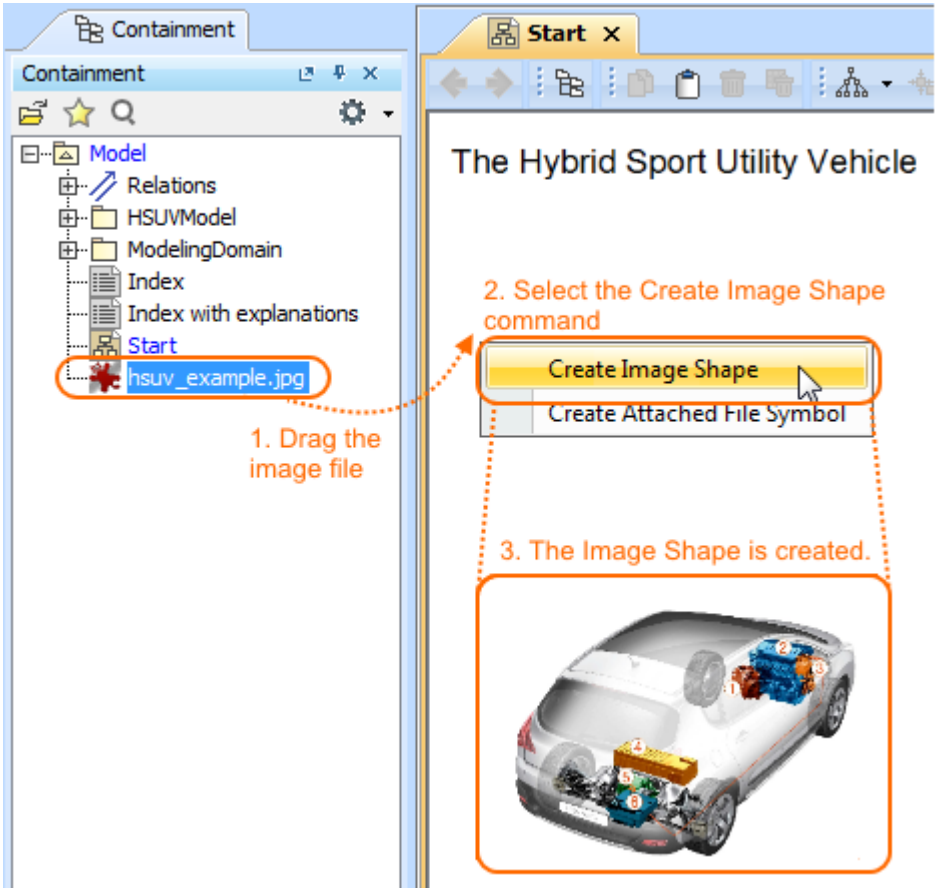
   > **Image formats**
   > You can attach images of the following formats: *.gif*, *.jpg*, *.jpeg*, *.svg*, or *.png*.

2. Drag attached file element on the diagram pane(see page 24) (1 step in the figure below).
3. From the opened menu, select the **Create Image Shape** command (2 step in the figure below). The image shape is created (3 step in the figure below).

   > **Image does not update**
   > Image shape is not synchronized with attached file. If you replace an image in the attached file, the image shape on the diagram pane will not update.

---

59 http://docs.nomagic.com/display/MD190SP4/Inserting+an+image+into+HTML+text
60https://docs.nomagic.com/display/MD190SP4/Attaching%252C+reattaching%252C+and+detaching+files#Attaching,reattaching,anddetachingfiles-Attachingfiles



Setting project options

Use the **Project Options** dialog to do the following:

- Specify general project-specific options.
- Specify summarizing information (for example, diagram author, diagram creation and modification dates) that will be displayed on each diagram.
- Specify symbol property styles for shapes, paths, diagrams, and stereotypes within the project.

  > ⓘ For the instructions how to create, edit, clone, import /export, or remove symbol property styles, please refer to Style Engine.

- Change default element property values.

  > ⓘ For the instructions about setting the default element property values, please refer to Default Property Values.

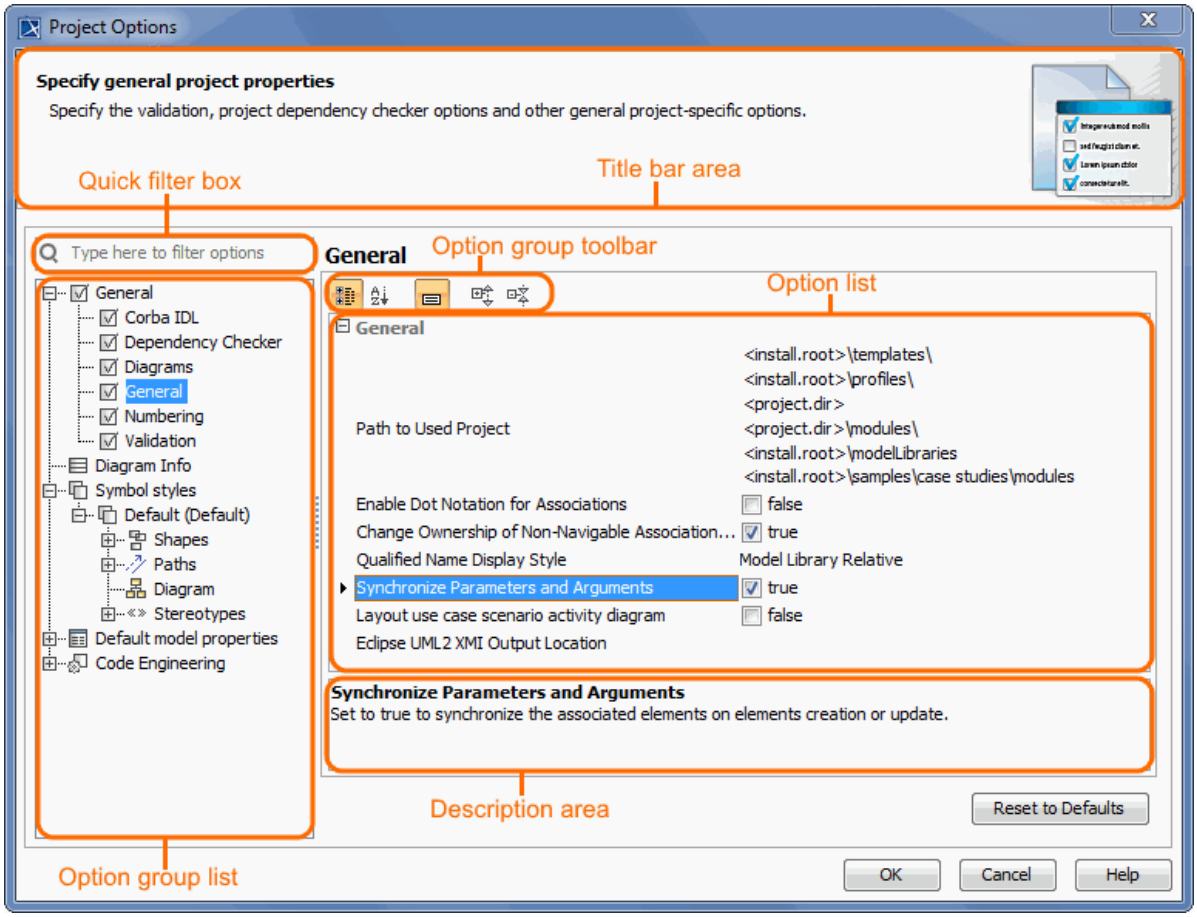
- Set general code generation or code reversing options as well as code formatting styles for selected programming languages.

The **Project Options** dialog includes option groups, each designated for one of the above mentioned features and containing lists of corresponding options. Groups are represented using the tree structure.

An option value can be simply changed by typing a new value, setting a value to true / false, or selecting a value from the list. Click the **Reset to Defaults** button to restore default option values.

To open the **Project Options** dialog

- From the **Options** menu, select **Project**.



Changing options

In order to change a desired project option, first of all you need to find it in the **Project Options** dialog. These options are grouped into categories to make it easier for you to locate them. Use the q(see page 918)uick filter(see page 918) to find find the desired option in the list. Filtering is performed in the all available options, not only in the selected option group.

To make sure you selected the right option, first read its description and see if the effect of changing the option value is what you expect to be done.

Make sure that the Show Description mode is turned on in the **Project Options** dialog. To turn the Show Description mode on or off, click the Show Description button on the following toolbar in the **Project Options** dialog.

To read the option description

1. Click an option that you want to change.
2. Read the option description in the area below the options list.

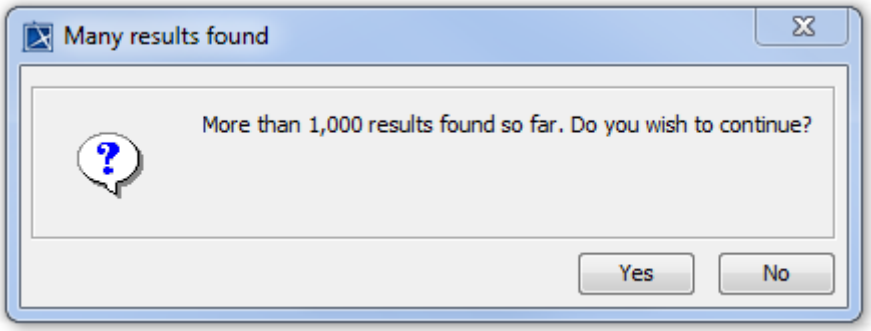
Searching

The search of the modeling tool is designed for search within model elements, as well as element properties. You can also perform a quick search without defining any advanced options, or you can replace any text quickly using **Find and Replace** type of search.

You can read more about types of search dialogs in the related pages.

The search is performed as a background task, therefore while searching, you may still read the model and browse it. If you make changes in the model, the search stops and presents incomplete search results in a **Search Results** tab of the **Model Browser**. You need to restart the search to get the full list of results.

If there is a large number of search results to display (more than 1000), the following message appears.



Click **Yes** to continue searching, or click **No** to stop searching and correct the query.

Related pages

- Find dialog(see page 217)
- Quick Find dialog(see page 223)
- Find TODO(see page 229)
- Find and Replace dialog(see page 229)
- Searching in diagrams(see page 231)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Find dialog

On this page

- Main area of the Find dialog(see page 219)
- Options area of the Find dialog(see page 221)
- Advanced search options(see page 222)

The **Find** dialog provides a quick way to search for specific model elements. Use it to search for elements by name, applied stereotype name, tag value and documentation content. The dialog can be opened from several different places in your modeling tool. To complete a simple search, use the main area of the dialog, or click the **Options** button, to specify criteria that will reduce your search results to only applicable ones.

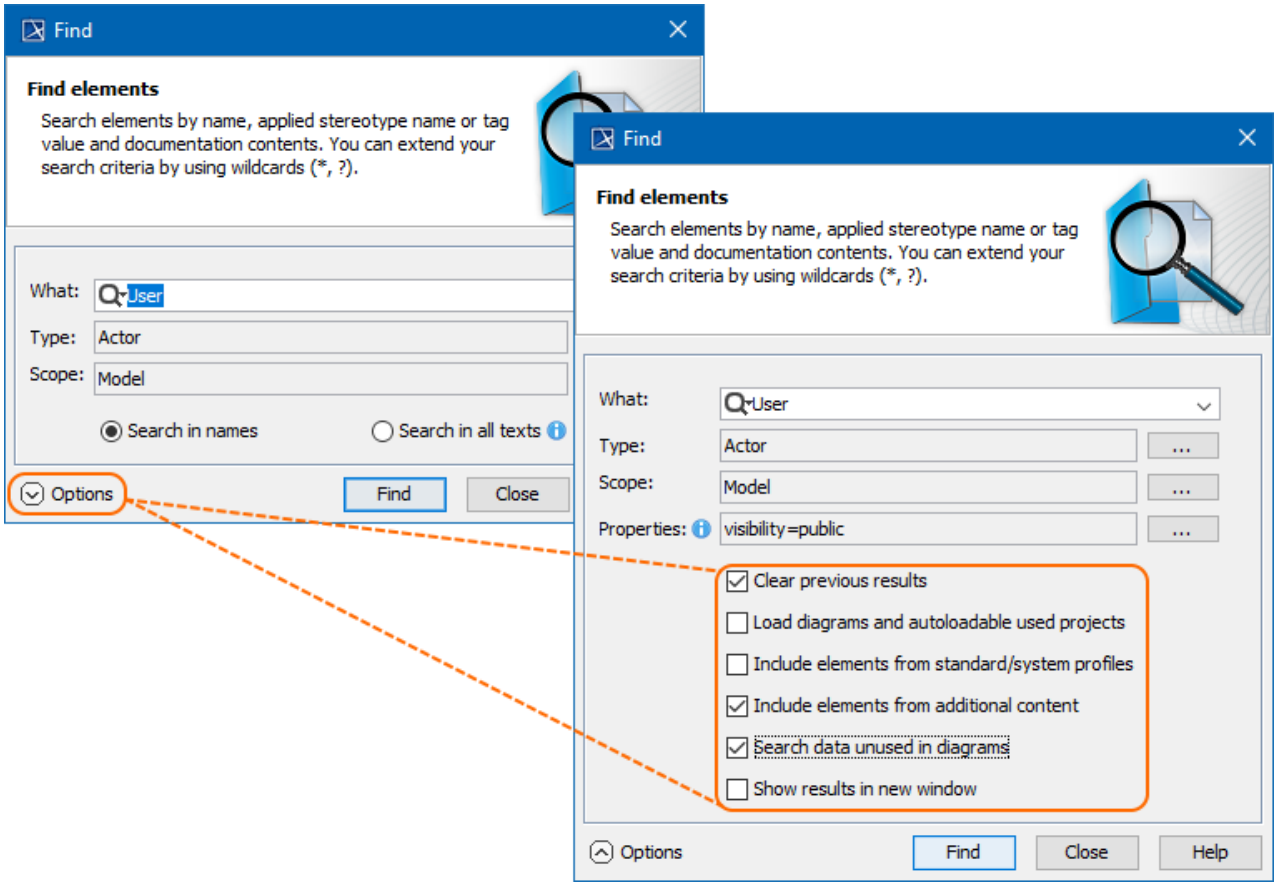
To search for elements using the Find dialog

---

1. Open the **Find** dialog in one of the following ways:
    - In the main menu, click **Edit** > **Find**.
    - In the main toolbar, click .
    - Press Ctrl+Shift+F.
2. Specify the search criteria in the main area of the dialog(see page 219).
3. If you would like your search to be more specific, click the **Options** button to expand the **Options** area(see page 221), and select the desired options.
4. When you finish specifying your search criteria, click the **Find** button.

After completing the steps above, the search results are displayed in the **Search Results** tab of the model browser.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.


Find
Find elements
Search elements by name, applied stereotype name or tag value and documentation contents. You can extend your search criteria by using wildcards (*, ?).
What:
User
Type:
Actor
Scope:
Model
Search in names
Search in all texts
Options
Find
Close
Find
Find elements
Search elements by name, applied stereotype name or tag value and documentation contents. You can extend your search criteria by using wildcards (*, ?).
What:
User
Type:
Actor
Scope:
Model
Properties:
visibility=public
Clear previous results
Load diagrams and autoloadable used projects
Include elements from standard/system profiles
Include elements from additional content
Search data unused in diagrams
Show results in new window
Options
Find
Close
Help

The Find dialog with the extended Options area.

Searching tips

- To search for textual information in an active diagram, click in the diagram toolbar or press Ctrl+F.
- To quickly access your search results, add them to Favorites or save them as the contents of a Smart Package.

Click to learn how to add search results to Favorites

1. In the **Search Results** tab of the Model browser, select the element(s) you want to add to Favorites. Press and hold the Shift button to select multiple items.
2. Click on the toolbar of the **Search Results** tab. Then select **Add Selected to Favorites** from the menu.

The selected search results are now added to Favorites. They can be accessed by clicking and selecting the desired element under the **Go to Favorites** group.

Click to learn how to save search results as the content of a Smart Package

1. Click on the toolbar of the **Search Results** tab containing the search results you want to add to a Smart Package.
2. From the open menu, select one of the following:
   - **Save Query as Smart Package** if you want the content of a newly created Smart Package to be dynamic. The content of the Smart Package will then be updated according to your search. The specified search criteria will be saved as the **Query** property value.
   - **Save Results as Smart Package** if you want the content of a newly created Smart Package to be static. In this case, your model is not queried, and new elements meeting the specified search criteria are not automatically included in the Smart Package.
3. In the open dialog, select the Package that will own the created Smart Package.

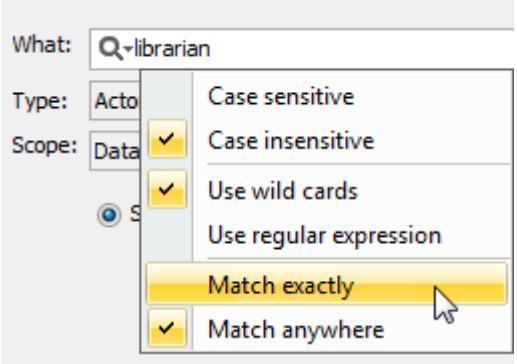
A new Smart Package containing all the search results from the **Search Results** tab is created under the selected owner.

Main area of the Find dialog

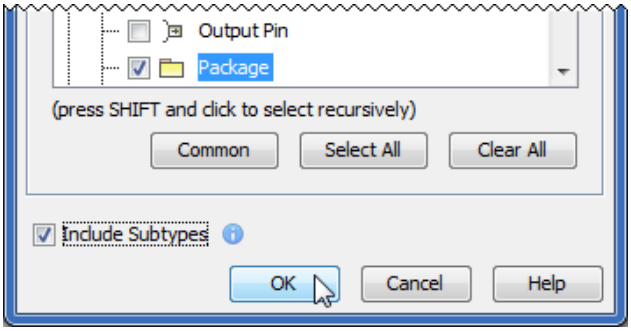
Use the main area of the **Find** dialog to specify the most common search criteria, such as an element name or the scope of the search. The table below provides detailed descriptions of all the elements in the main area.

Dialog element name	Dialog element type	Description

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

What	Text box with drop- down list	The **What** box allows you to search for model elements by the text used in their names or textual properties. To search for desired model elements, enter the text you want to find. To search in element names, select the **Search in names** check box. To search in textual properties including Documentation, select the **Search in all texts** check box. Note that both of these options are available only if the **Options** area is collapsed. To make your search more specific, click as illustrated below.  You may use wildcard characters (,?) in your search phrase. "" means zero, one, or more characters. If your search phrase is "dd", all the elements containing "dd" in their names can be found, e.g. "Address" and "ThreadDeath". "?" means exactly one character. If your search phrase is "?dd", only the elements with "dd" in their names, beginning with the second character (e.g., "Address"), can be found. "ThreadDeath" cannot be found in this case, because it contains more than one character before the search phrase "dd". Be sure to check the **Match exactly** command when using wildcard characters!

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Type	Text box with ...	Click ..., and select one or more element types from the list provided in the open dialog. You have the option to include standard and custom subtypes of the selected element type in the search results. For example, if you need to search for packages along with profiles, models, smart packages, and other custom subtypes of the Package type, select this type, and then select the **Include Subtypes** check box. 
Scope	Text box with ...	The scope refers to the package of your model in which the content search will be performed. Click ... and select the desired package. Specify the search scope before opening the **Find** dialog by performing one of the following: • Right-clicking a package in the Model Browser, and selecting **Find** from its shortcut menu. • Selecting a package in the Model Browser and pressing Ctrl + F.
Properties	Text box with ...	The box is available only when the **Options** area is expanded. Click ..., select the properties applicable to your search and specify their values. The property list in the open dialog contains only the properties of the element types selected in the **Type** box. Select at least one element type to make the button available. Keep in mind that the property list contains primitive value properties, e.g. string, boolean, integer, and a non-primitive value property named **Applied Stereotype**.

Options area of the Find dialog

The **Options** area allows you to select from a number of additional search options that will make your search more specific and narrow down the search results.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Dialog element name	Dialog element type	Description
Clear Previous Results	Check box	Select to remove all previous search results from the **Search Results** tab in the Model Browser.
Load diagrams and auto-loadable used projects	Check box	Select to load all elements to be included in the search if the model has diagrams or used projects that are not loaded. If the load mode of the used project is **Manual load**, the used project will not be loaded and elements from it will not be included in the search.
Include elements from standard/system profiles	Check box	Select to include elements from standard/system profiles.
Include elements from additional content	Check box	Select to include additional elements from Smart packages if Smart packages fall into the search scope. By default, this check box is not selected and elements from Smart packages are excluded from search results. This enables you to avoid search results being duplicated by results outside of Smart packages.
Search data unused in diagrams	Check box	Select to search for only those elements that do not have symbols in any diagram. Note that the check box is unavailable if the **Find** dialog is opened by using the **Find in Diagram** command in the diagram shortcut menu.
Show results in new window	Check box	Select to display the results of every subsequent search in a new **Search Results** tab of the Model Browser.

Advanced search options

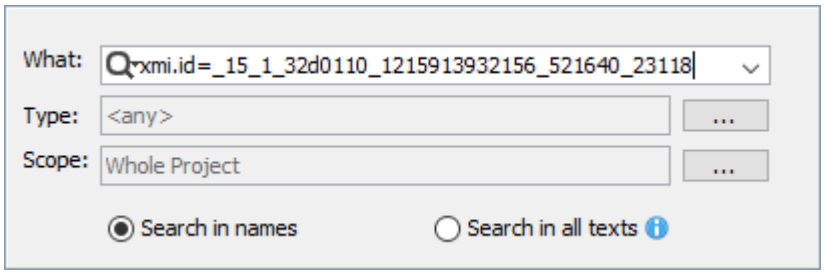
The **Find** dialog offers a number of advanced search options allowing you to narrow down search results. Below are some specific cases illustrating how search criteria can be defined.

To find an element by an element ID

1. Open the **Find** dialog.
2. In the **What** box, enter the prefix "xmi.id[61]=" followed by the ID of the element you want to find.

61 http://xmi.id

3. Click the **Find** button.



To find abstract Classes

1. Open the **Find** dialog.
2. Click ... next to the **Type** box, and select **Class**.
3. Expand the **Options** area if it is not yet expanded.
4. Click ... next to the **Properties** box, select the **IsAbstract** property, and set its value to *true*.
5. Click the **Find** button.

To find the elements with a specific stereotype and a certain phrase in their documentation

1. Open the **Find** dialog.
2. Click ... next to the **Type** box, select the element type you want to find (e.g, **Actor**) and select the **Include Subtypes** check box.
3. Expand the **Options** area if it is not yet expanded.
4. Click ... next to the **Properties** box, select the **Applied Stereotype** property, and select a custom stereotype (e.g., «BusinessActor») as its value.
5. In the same dialog, select the **Documentation** property and enter a search phrase as its value.
6. Click the **Find** button.

Related pages

- Find dialog(see page 217)
- Quick Find dialog(see page 223)
- Find TODO(see page 229)
- Find and Replace dialog(see page 229)
- Searching in diagrams(see page 231)

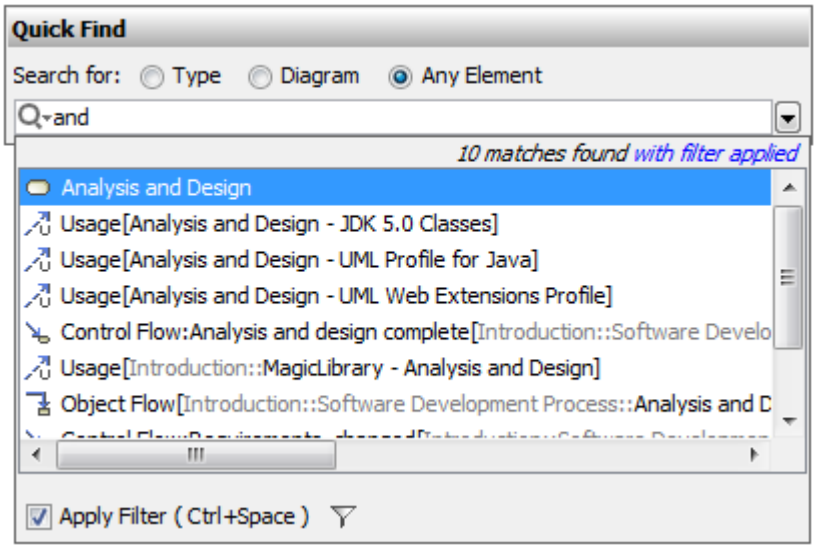
Quick Find dialog

You can use a quick search to find an element by its name. A very simple and easy to use dialog is designed to perform a quick search. In this dialog, you just need to type a name of the element you are searching for and select, if it's a type, diagram, or element.

To open the **Quick Find** dialog, do one of the following

- On the toolbar of the **Containment**, **Diagrams**, **Inheritance**, or **Model Extensions** tab, click .

- On the main menu, click **Edit** > **Quick Find**.
- Press Ctrl + Alt + F.
- On the main toolbar, click the arrow nearby the button and then select **Quick Find**.



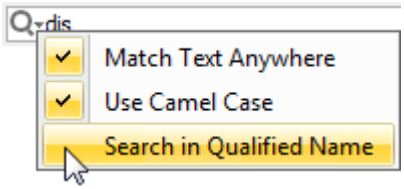
Search results are displayed in the drop-down list, which opens when you start typing the search phrase. Click the element you need in this list or correct the search query to perform a new search. The element will be selected in the Containment tree on the Model Browser as well as on the active diagram, if the element has symbols on the active diagram.

The following table provides descriptions of the **Quick Find** dialog elements.

Dialog element name/ image	Dialog element type	Description
Search for	Options group	• Click **Type** to search only in types, that is, classes, interfaces, use cases, and others. • Click **Diagram** to search only in diagrams. • Click **Any Element** to search in all elements of your project.

Search By Name	Text box with a drop-down list	Type a phrase to search for in element names. See the search results in the drop-down list. You may use wildcard characters (*,?) in your search phrase: • "*" substitutes any range of characters. So if your search phrase is "*dd", all the elements containing "dd" in their names can be found. For example, "Address" and "ThreadDeath". • "?" substitutes exactly one character. So if your search phrase is "?dd", only the elements with "dd" in their names starting from the second character can be found. For example, "Address". "ThreadDeath" cannot be found in this case, because it contains more than one character before the search phrase "dd". Be sure the **Match Text Anywhere** command is unchecked when using wildcard characters!

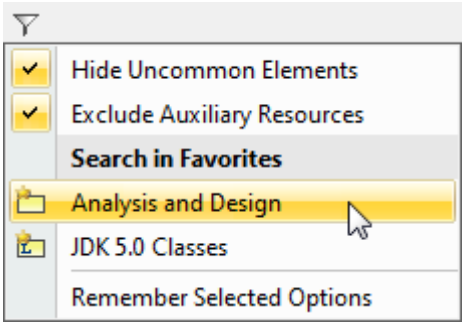
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

	Menu button	Click the button and customize search options.  **Match Text Anywhere** • To search for matching phrase in any part of the element name, click the command to add the check mark. • To explicitly search from the beginning of element name, click the command to remove the check mark. **Use Camel Case** • To search for matching phrase typed in camel case, click the command to add the check mark. • To search for matching phrase typed in regular case, click the command to remove the check mark. You can use camel case as follows: • Type only the first letters (either capitalized or not) of each word, for example, "and" or "AND" to find "Analys is and Design". • Type skipping spacers, for example, "cv3" to find "CV-3". • Perform partial search, for example, "ibd" to find SysML Internal Block diagram. Be sure the **Match Text Anywhere** command is checked! **Search in Qualified Name** • To search for matching phrase in qualified names of elements, click the command to add the check mark. • To search for matching phrase in element names only, click the command to remove the check mark.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Apply Filter (Ctrl+Space)	Check box	Click to clear the check box, if you need to remove the search scope filter. If you clear the **Apply Filter** check-box once, your selection is remembered for the next time.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Filter Options	Menu button	Click the button and customize search scope filter options. Be sure the **Apply Filter** check box is selected! **Hide Uncommon Elements** To reduce the search results list by hiding uncommon elements, click the command to add the check mark. Elements, such as Interface, Association Class, Class, and Component, remain in the list.  **Exclude Auxiliary Resources** To reduce the search results list by excluding elements contained in auxiliary resources, click the command to add the check mark. This option does not hide the following: • UML datatypes (placed under MagicDraw Profile) and UML2 Metamodel (placed under UML Standard profile) only for UML project. • UML2 Metamodel, SysML value types, and stereotypes for SysML project. • Elements with <<**ignoreAuxFilter**>> stereotype applied and all its elements recursively. How to apply stereotype >>(see page 857)

		Search in Favorites Select one or more favorite packages as the search scope. Be sure you have at least one package marked as favorite! **Remember Selected Options** To make the **Quick Find** dialog remember the specified search scope filter options for the next time it opens, click the command to add the check mark. The same scope filter settings appear in autocompletion lists and the element Selection dialog as well.

Find TODO

Command **Find TODO** performs a quick search for all *To Do* values.

You can specify *To Do* values in the element Specification window, the **To Do** property cell. You can use the **To Do** property cell for adding the notes about the work you need to do. Then you can quickly find the elements that have the **To Do** properties defined.

Results of this kind of search are displayed in the **Search Results** tab on the Model Browser.

To start the **Find TODO** search, do one of the following

- On the main menu, click **Edit** > **Find TODO**.
- On the main toolbar, click the arrow nearby the button and then select **Find TODO**.

The **Search Results** tab opens with results represented in it.

Find and Replace dialog

The Find and Replace functionality allows replacing one specified model value with another. You can change the values for the following properties:

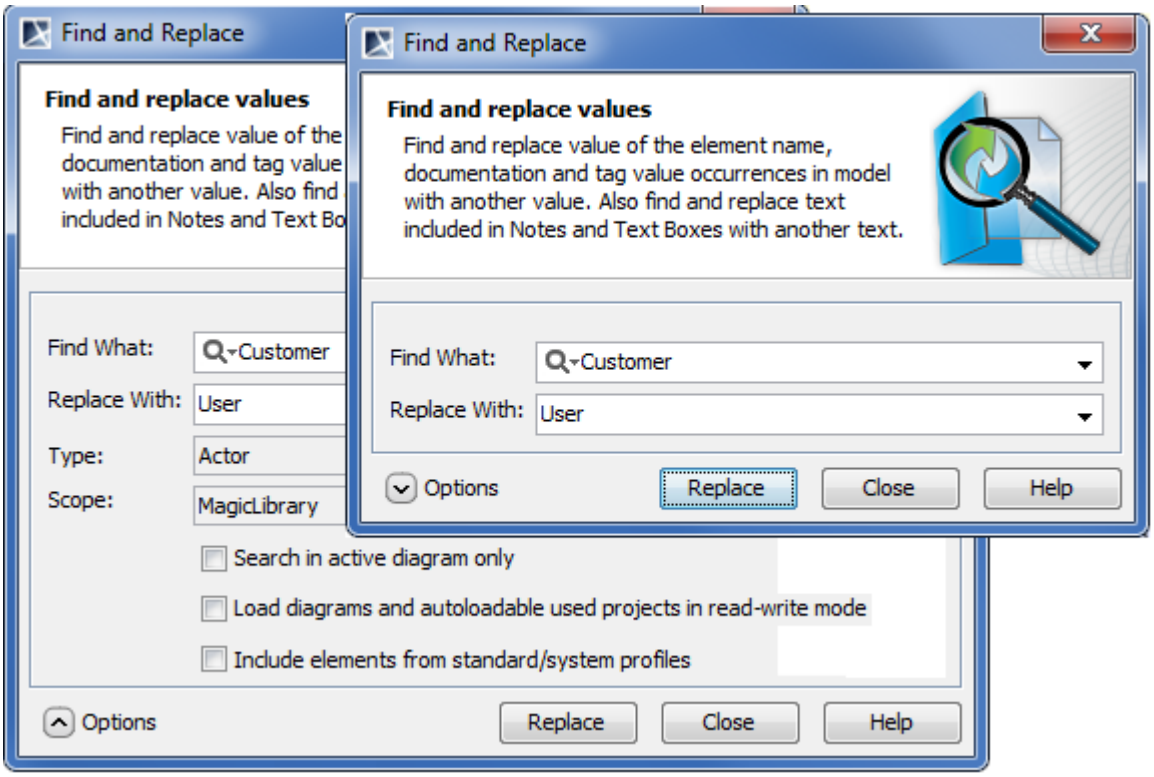
- Names
- Documentation
- Tag values
- Text included to Notes
- Text included to Text Boxes
- Expressions

To open the **Find and Replace** dialog, do one of the following

- On the main menu, click **Edit** > **Find and Replace**.
- Press Ctrl + R.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- On the main toolbar, click the arrow by the Find button , then select **Find and Replace**.



The following table provides descriptions of the **Find and Replace** dialog components.

Dialog element name	**Dialog element type**	**Description**
Find What	Text box with drop- down list	Type or select from the list to search the value of the element to be replaced.
Replace With	Text box with drop- down list	Type or select from the list the value of element that will replace the found value.
Options area		
Type	Text box with	Click the button and select one or more element types from the proposed items in the opened dialog, or leave the default value <any>.

Scope	Text box with	Click the button and select a package wherein the content search will be performed.
Search in active diagram only	Check box	Select to search the symbols in the active diagram only. The check box is not available, if all diagrams are closed or if the active diagram is empty.
Load diagrams and auto-loadable used projects in read- write mode	Check box	Select to load all elements to be included in the search if the model has diagrams or used projects that are not loaded. Used project will not be loaded and elements from it will not be included into the search, if the load mode of that used project is **Manual load**.
Include elements from standard/ system profiles	Check box	Select to include elements from standard/system profiles.

You may see an error when trying to replace a value with an invalid one.

- For example, if the value type is *boolean*, you may replace values from *true* to *false* or from *false* to *true*, but if you will try to replace the value *true* or *false* to other, for example, to *Motor,* an error message will be displayed.
- You will not be allowed to change the *Integer* value to *String* when it is a part of the value. For example, if you have the *120* value, trying to replace *20* with *AB* will cause an error.

Searching in diagrams

The **Find in Diagram** command enables you to search for textual information in all types of diagrams, including tables, matrices and maps. Use this command to find a specific text phrase in a diagram and navigate through the highlighted search results.

To find text in a diagram

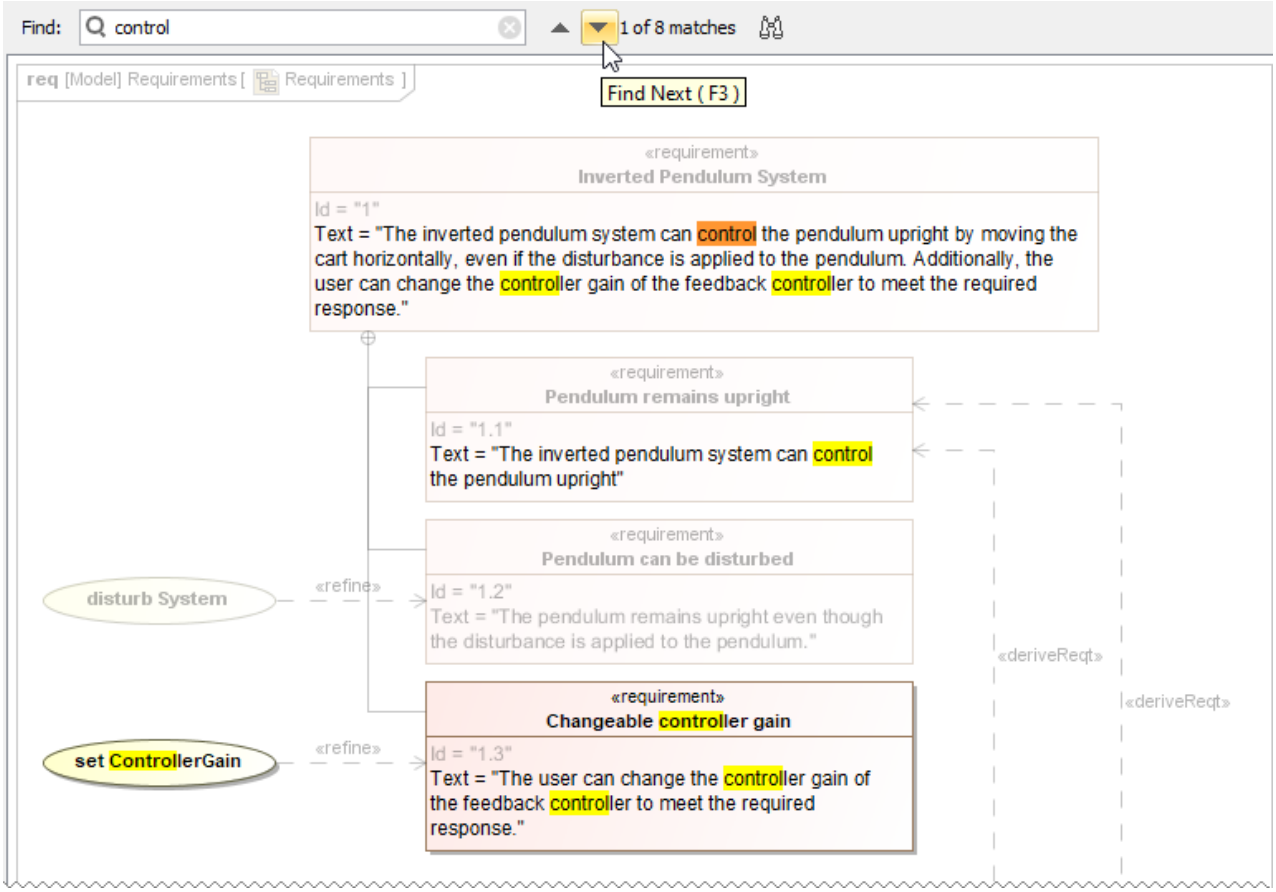
1. Open the diagram in which you want to perform the search.
2. Do one of the following:
   - Click in the diagram toolbar.
   - Press Ctrl+F.
3. In the search box, type the text you want to find.
4. Use (Shift+F3)/ (F3) to navigate through highlighted search results.

Searching tips

- You can search in several open diagrams at once. Typing a search phrase in a subsequent diagram does not undo the search results in a previous diagram.
- When you press Ctrl+F with multiple open diagrams, the search starts in an active diagram.
- To search for textual information in the whole model, click in the diagram search bar and specify the search criteria in the Find dialog(see page 217).

After typing a search phrase, the search results are highlighted in yellow and the diagram or table scrolls to the first search result (highlighted in orange), as shown below. Diagram symbols or table rows that do not contain your search phrase are shaded. A diagram symbol is also shaded if the search phrase is found in the symbol compartment.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



How to use the Find feature in diagrams and navigate the search results.

Related pages

- Searching(see page 216)
  - Find dialog(see page 217)
  - Quick Find dialog(see page 223)
  - Find TODO(see page 229)
  - Find and Replace dialog(see page 229)

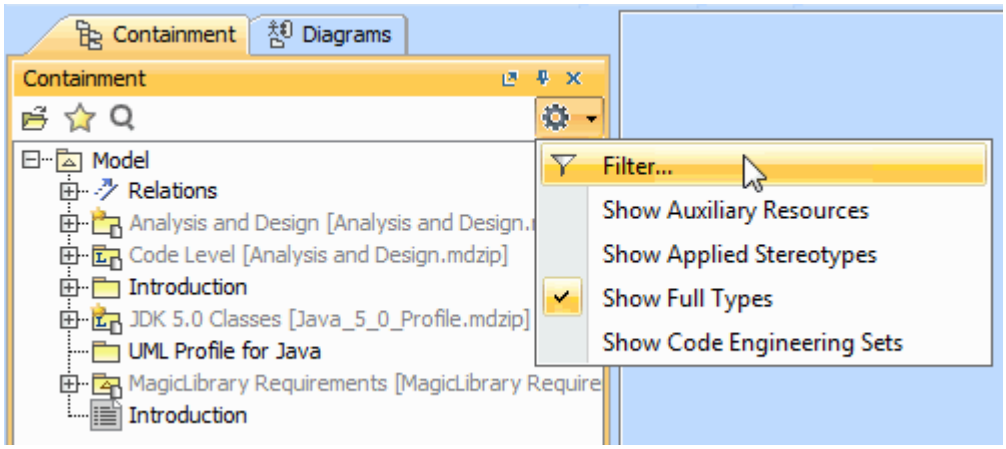
Filtering

The modeling tool filtering mechanism allows filtering types of model elements in particular lists. For example, you can filter the types of model elements that are displayed in the Model Browser, or you can select them when searching.

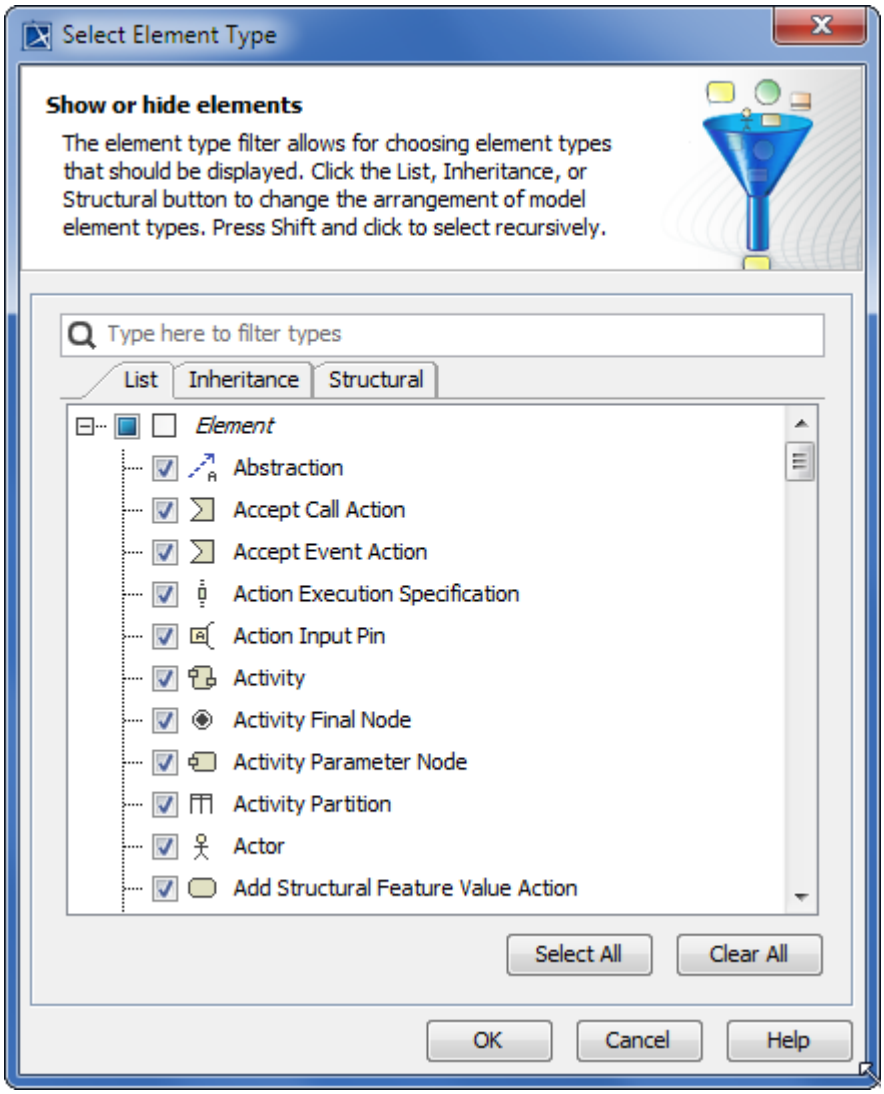
To filter the types of the model elements that are displayed in the **Containment** tab(see page 32)

1. In the Model Browser, open the **Containment** tab (in case it was closed or another tab was open).

2. Click and select **Filter**.



The Select Element Type(see page 1075) dialog opens.



3. Do one of the following:

- Click to clear the check box next to the model element type, which you do not want to be displayed in the **Containment** tab.
- Click to select the check box next to the model element type, which you want to be displayed in the **Containment** tab.

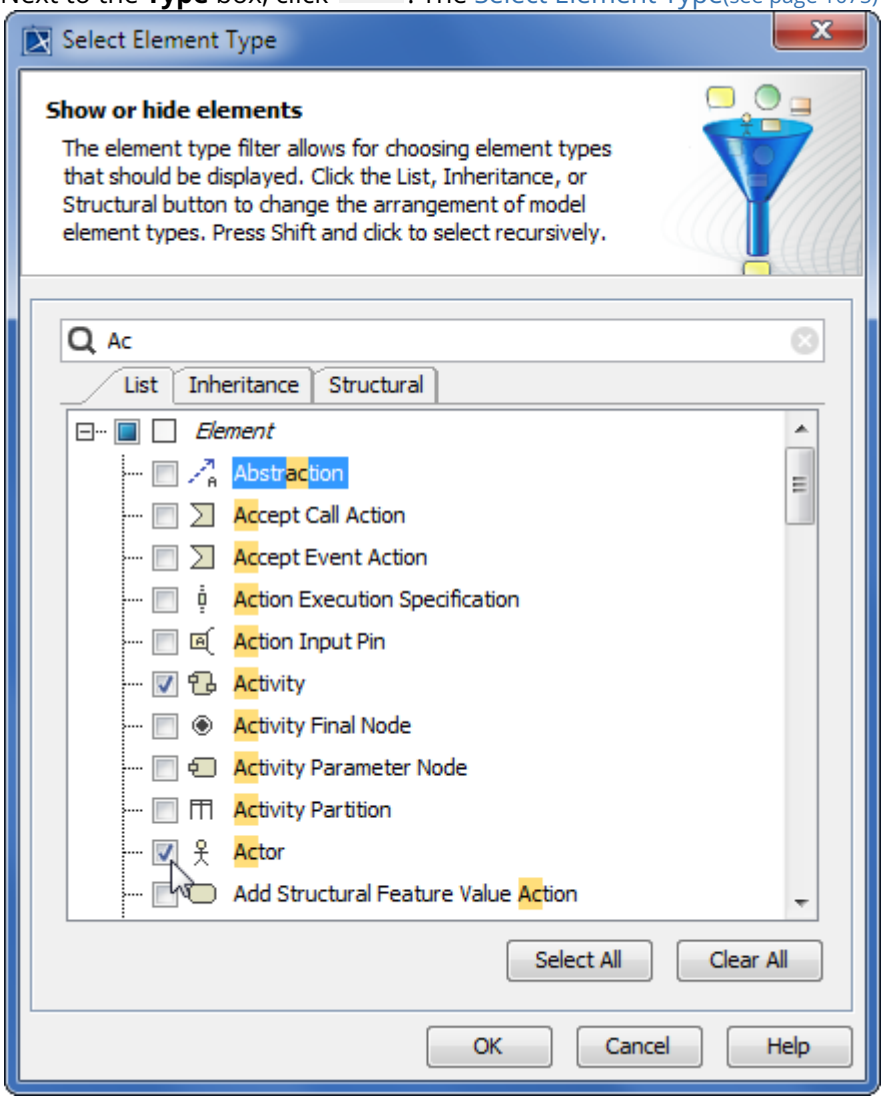
4. Click **OK**. Items in the **Containment** tab are now filtered.

> All elements in the **Containment** tab are displayed in hierarchical order. If you clear the *Model* check box, no elements will be displayed in the **Containment** tab. The same is valid for all element types – if you clear the check box next to the type of element that has inner elements in the Model Browser, you won't be able to see these inner elements.

To filter the types of model elements when searching(see page 216)

---

1. From the **Edit** main menu, select **Find**. The **Find** dialog opens.
2. Next to the **Type** box, click ... . The Select Element Type(see page 1075) dialog opens.



3. Click to select the check box of the model element type that you want to search for.

4. Click **OK**. You are now back to the **Find** dialog then click **Find**. The **Search Results** tab opens in the Model Browser.

Related page

- Selecting element types(see page 1075)

Project partitioning

In life, all large things are built from smaller parts. The same statement applies for large projects. For large models having several weakly dependent parts, it is advisable to split them into several separate project files (or project resources in a collaborative environment), also called used projects.

Project partitioning opens up the additional possibilities, such as:

- Reusing the same model part (e.g. library) in several projects.
- Separate administration of different project parts. You can define different access rights in the collaborative environment for each used project. Separate (teams of) developers can be assigned for each resource, clearly establishing responsibility boundaries and preventing inadvertent modifications, when these projects are used in the read-only mode.
- Separate versioning. Each used project can have its own version history with its own tagging and branching, life cycle of development, feature-freezes, and stable releases.
- Increased performance on very large projects. Whenever used projects are marked as loaded manually, user can selectively load just the necessary parts of the project. Or he/she can open the isolated used project separately.

A model can be decomposed to separate projects. A used project is a part that can be separated from the main project and can be used for a specific purpose.

MagicDraw supports the following used project accessibility modes:

- Read-only
- Read-write

The decision to use a project as read-only or read-write depends on the maturity of this project and the organization ownership or responsibility rules for the projects using other projects. If the library in the used project is complete (changes to it are not expected / likely / possible) it should be use in read-only mode. If a project is developed by a separate team and this team is responsible for this project and the project is reused in another project, it is recommended to use this project as read-only. This prevents inadvertent changes in the project by another team.

In case a project is actively developed and evolves together with the projects that are using it, this project can be used in read-write accessibility mode. In this case you must be careful and remember that your changes to the used project will be reflected in other projects. Usage of Teamwork Server might be advisable in this case. And, of course, there can be mixed usage situations - when a project is used as read-only in some projects and as read-write in others.

How to partition your project read in the following pages:

- Decomposing model(see page 237)
- Composing model(see page 242)
- Managing used projects(see page 245)
- Automated project usages(see page 251)
- Maintaining decomposed model integrity(see page 254)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Validating project integrity(see page 264)

Decomposing model

Model decomposition has a package level granularity. Smaller elements cannot be split into separate projects. Basically, each project package could be partitioned into a separate projects, however this is excessive.

The decision on how to split a model into parts should be made carefully. You should isolate model parts which form logically complete pieces of a structure (e.g., a subsystem, code library, or profile) and are not very dependent on each other.

When there are many one-way dependencies to some model part (e.g., parts **A**, **B**, **C** depend on a part **D**, but part **D** does not depend on any of the parts **A**, **B**, **C**), this part is a good candidate for a placement into a separate project.

When one big project is used to store all the modeling information (e.g., use case models, high level architectural models, detailed implementation level class, sequence, state, and other information), it may be useful to partition the models according to the modeling domains (i.e., use cases in one used project, architectural models in another, implementation level models in yet another). This allows unloading unnecessary used projects while working on one part (and improving the performance), but still retain the relationships between domains and load used projects on demand.

> ⚠ Avoid decomposing a model into parts which have circular dependencies. That is, A ↔ B or A → B → C → A situations.

Usually programmers are very adept at splitting large code bases into libraries. The very same criteria should be applied for splitting large models into separate projects.

The MagicDraw project decomposition functionality allows two important possibilities:

- Working without all used projects loaded.
- Using the project in the read-write accessibility mode.

Used projects are often exploited for storing profiles, however a used project is not a profile, and it is important not to mix them. Any model part can be stored in a separate project.

Related pages

- Exporting packages to new projects(see page 237)
- Sharing project data(see page 239)
- Indexing of unloaded projects(see page 240)

Exporting packages to new projects

> ⚠ This functionality is available in Standard, Professional, Architect, and Enterprise editions only.

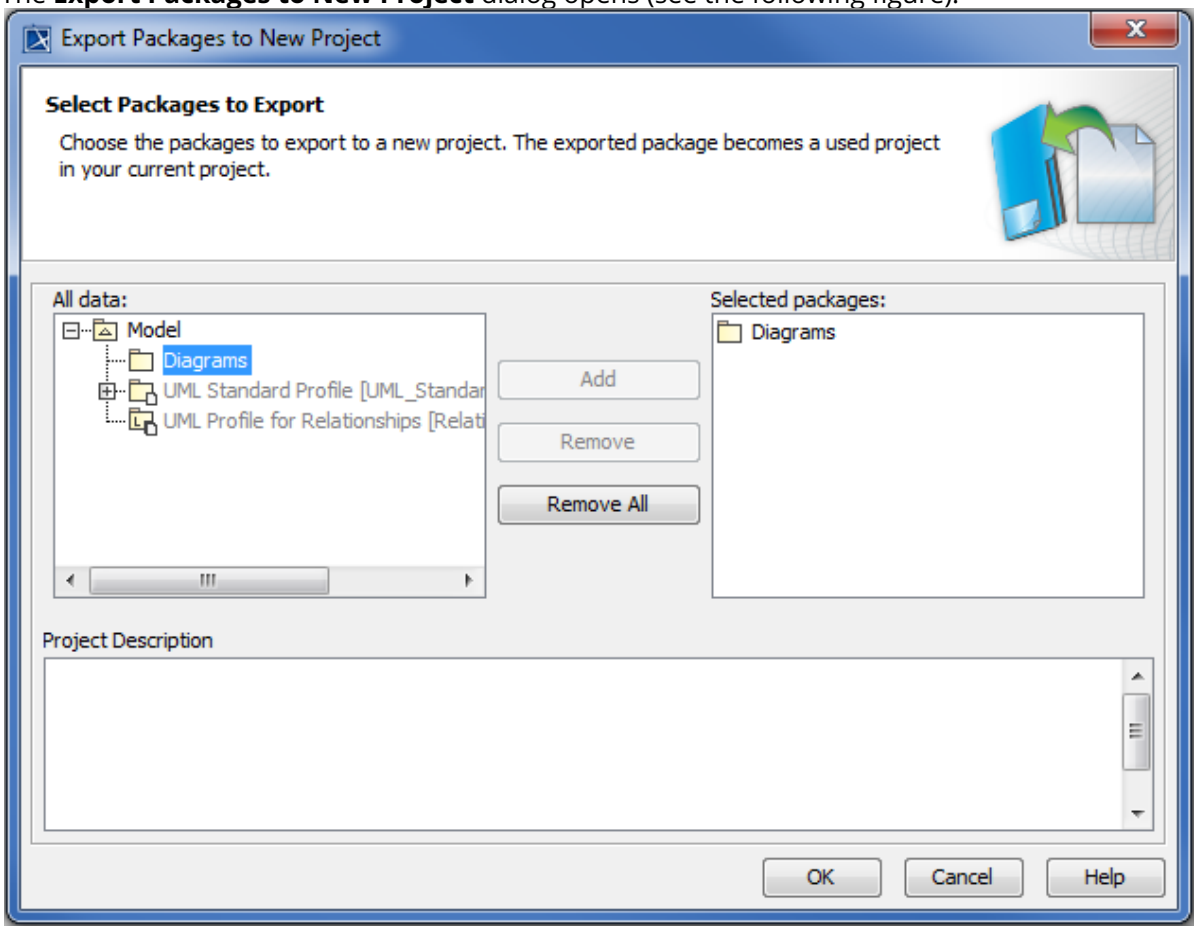
Using the **Export Packages to New Project** dialog, you can partition the model and save the content of a selected package as a separate project. Once exported, the package and its containing elements are read-only, and the project name is displayed in brackets next to the package name in the Model Browser.

To export a package as new project

---

1. Do one of the following:
   - From the **File** menu, select **Export To** > **Project Usage**.
   - In the Model Browser, select one or more packages you want to save as a separate project. Right-click it (them) and in the shortcut menu select **Project Usages** > **Export Packages to New Project**.

   The **Export Packages to New Project** dialog opens (see the following figure).



2. In the **All data** list, select the package you want to save as a separate project and click the **Add** button. The package is added to the **Selected packages** list.
3. If needed, type a description of the exported packages in the **Project Description** box. This description is displayed in the documentation of the package.
4. Click **OK**.

Before exporting selected packages to a separate project, MagicDraw checks for dependencies between the model and the exported part of the model. If any error occur, you have to resolve it, otherwise recovered elements may appear in the new project. The dependency resolution process is the same as for resolution of dependencies between shared and not shared parts of the project.

When dependencies are resolved, MagicDraw asks to specify the new project location and name.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

This action consist of 3 elementary steps:

- Saving model elements into a separate file.
- Sharing the entire contents of this project.
- Using the project in the main project.

Alternatively, if you have several small, related projects, you can join them into a larger, partitioned project to work with all the information in one place. This can be done by using the **Use Project** button on the **Used Projects** dialog (to open the dialog, select **Options** > **Project Usages**).

> ⚠ Only packages can be exported to new projects. To export an element or diagram, you must move it to a package and then export the package.

Sharing project data

> ⚠ The shared packages are available only for Teamwork server or local projects. The Used Project(see page 1441) is alternative for shared packages in the Teamwork Cloud envirenment. This functionality is available in Standard, Professional, Architect, and Enterprise editions only.

Not all contents of a used project are visible in the main project. The used project has a shared part and the part that is not shared. Only contents of the shared part are visible in the main project. The concept “shared” is similar to the public/private attributes of a class in programming languages (e.g., Java).

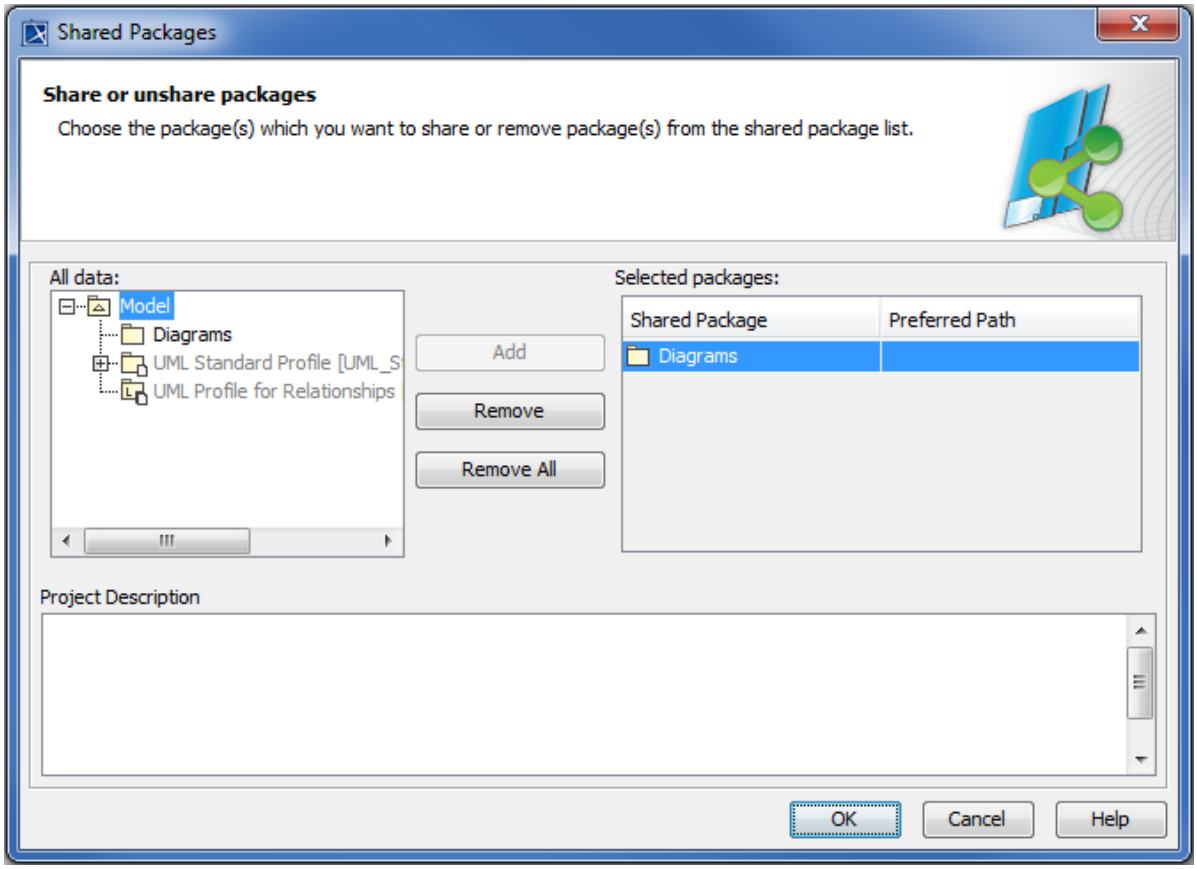
To designate packages of the project as shared

---

1. Select package.
2. Do one of the following:
   - From the **File** menu, select **Share Packages**.
   - From the package shortcut menu, select **Project Usages** > **Share Packages**.

   The **Shared Packages** dialog opens.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



3. If not all desired packages are showing in the **Selected packages** list, select them in the **All data** list and click the **Add** button
4. Click **OK**.

The selected package including its owned elements is shared.

When a project is used in another project, all shared packages of the used project appear in that project. Moreover, each shared package can have a different location in the Containment tree. Any available package in a project can be used as a container for the shared package of the used project.

Additionally, each shared package can provide a “Preferred Path” that can be used as a hint where to mount a package when it is used in other project. No “Preferred Path” means that the package will be mounted directly under the root package Data. Profiles are typically mounted directly under the root package Data, however this can be changed.

> Shared package “util” from the used project can be mounted on the “com::company” path in the main project in order to form the “com::company::util” path. The Preferred Path of the Shared Package can be used in this case to serve as a hint for the suggested location where to mount the package.

Used projects form a recursive data structure — the main project uses one or more other projects; these projects in turn can use other projects; those other projects can use another set of projects etc. All model pieces from these projects are gathered and connected into the integral model, which is shown in the Model Browser when the main project is opened.

Indexing of unloaded projects

If you have a large project and, due to performance issues, some used projects are not loaded, you may use indexing to work with the project smoothly. Indexing can be considered as an intermediate form of work, falling between working with a fully loaded used project and working with a not-loaded used project.

> **Proxy**
> A **proxy** is a lightweight surrogate that carries only the name and kind of the model element information. These proxies appear in the place of the original used project elements, and are dedicated to maintaining model integrity (so that there are no dangling ends of relationships, types of properties do not disappear, and so on), while the used project is not accessible (for example, you do not have permissions) or is not loaded.

When a used project is unloaded/not loaded in the project, only proxies of necessary elements are shown in place of the used project. Using indexing allows you to show more proxy elements from the unloaded project. These proxies can be used as normal model elements in the project using them without loading the used project. Keeping the used project not loaded saves a considerable amount of computer resources. If you ever need full information from that used project, you can load it at any time.

A typical case where this functionality is useful is outlined below:

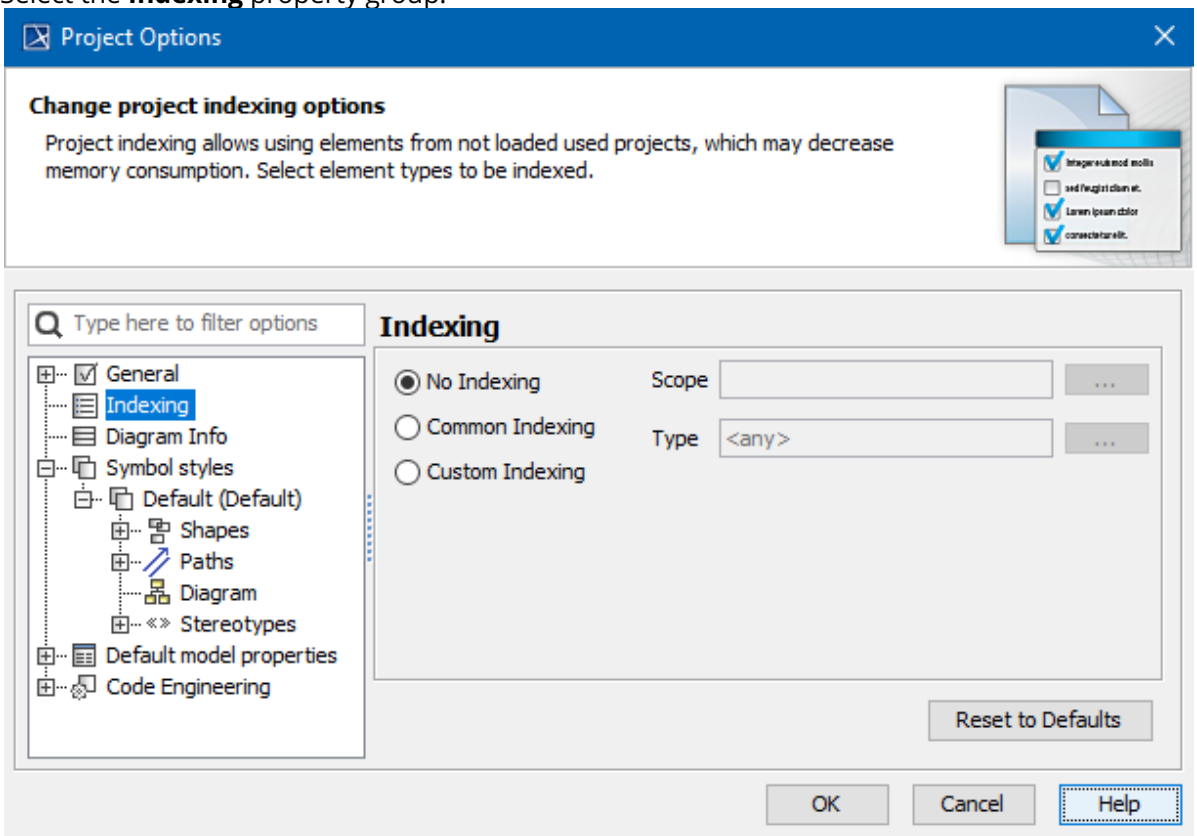
A large software library is used in a project. Only certain classes are important for the main project: some library classes are set as types of properties in the model classes, some model classes inherit from the library classes, etc. In this case, the sub-elements of the library classes (their properties and methods) are not important, and we can index classes in the library and keep it not loaded in the main project. Indexing will still show all the classes in the main project from the library.

To specify elements for indexing

1. Open the used project (that is, not loaded) separately from the using project.
2. From the **Options** menu, select **Project**. The **Project Options** dialog opens.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

3. Select the **Indexing** property group.



4. Select the **Common Indexing** option. This enables indexing of the used project and determines what information will be indexed.
5. When common indexing is chosen, classifiers and their inheritance relationships will be indexed. If you want more elements to be indexed, select the **Custom Indexing** option and fine-tune which element types (properties, methods, etc) should be indexed. The more elements you select, the more elements will be accessible in the project using them as proxies. However, your gains in performance from the used project unloading will also diminish. Thus, a balance is needed when customizing the index. It is usually better to use the common indexing variant. Click **OK**.
6. In the project using the indexed project, go to the options of the used indexed project (from the main menu, select **Options** > **Project Usages**) and select the **Use Index** check box.

Related pages

- Maintaining decomposed model integrity(see page 254)

Composing model

You can compose your model by using parts of models stored in other projects:

- Using other projects in a project(see page 242)
- Importing models(see page 245)

Using other projects in a project

When a project is used in another project, shared elements of the used project become visible and accessible in the Containment tree of the project that uses them as if they were a part of the project.

Using local projects

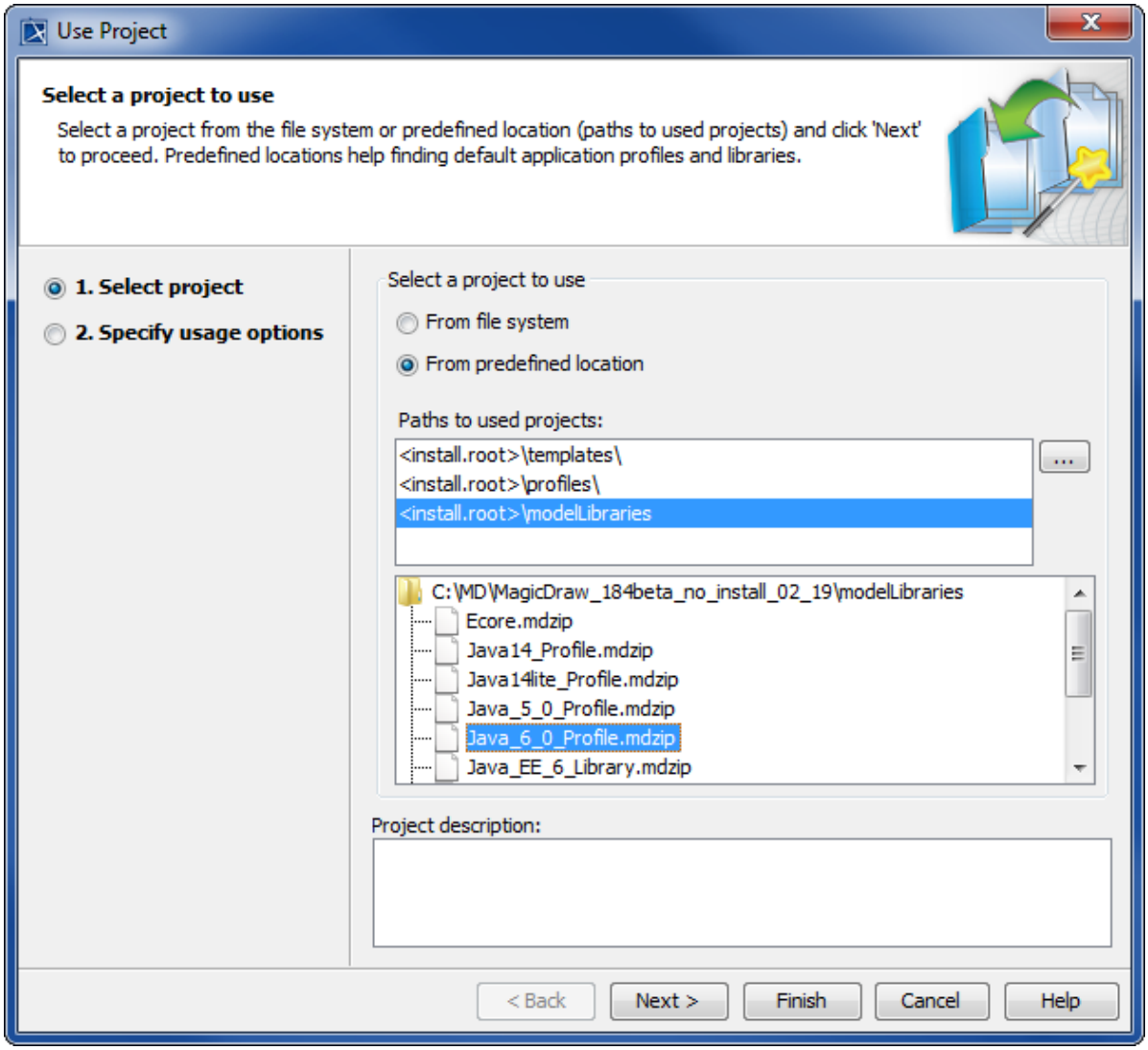
To start using a project

---

1. Do one of the following:
   - From the **File** menu, select **Use Project** > **Use Local Project**.
   - From the **Options** menu select **Project Usages** and click the **Use Project** button.

   The **Use Project** wizard opens.
2. In the **Select project** step, select the project you want to use from the file system or from the predefined location:
   - If you want to use your custom project from a custom location, select to use a project **From file system** and click ... next to the **Project file** box to navigate to the desired file.
   - If you want to use the project coming with modeling tool installation, select to use a project **From predefined** location, then select a path from the **Paths to used projects** list and the project to use from the list below.

     > You can customize the **Paths to used projects** list and add your own paths by creating path variables(see page 99).
3. Click the **Next** button.



4. In the **Specify usage options** step, specify the project usage options(see page 1441), such as the accessibility mode, load mode, and mounting locations of shared packages.
5. Click the **Finish** button.

After your project starts using another project, the elements of the used project are still stored separately:

- Used project elements — in the used project file.
- Project elements — in the project file.

Modifying project accessibility

The accessibility mode of the used project can be specified in the **Used Projects** dialog. By default, projects are used in the read-only mode.

To change the accessibility mode of a used project

1. From the **Options** menu, select **Project Usages**. The **Used Projects** dialog opens.
2. In the tree on the left side of the open dialog, select the used project and click the **Read-only** or **Read-write** option in the **Accessibility** area.

3. Click **OK**.

If you need to change the contents of the project used in the read-only mode, you can open it separately from the project that uses it. For this, click the **Open Project** button in the same dialog.

Related pages

- Importing models(see page 245)
- Managing used projects(see page 245)

Importing models

Importing models from used projects

If you no longer need to have the set of elements in a separate project, you can import these elements into the main project.

To import a used project into the main project

1. In the Model Browser, select the package of this used project and open the shortcut menu.
2. Select **Project Usages** > **Import**.

All the elements of this used project are copied into the main project, and links to this used project are removed from the main project. The unlinked project and its elements continue to exist in the file system or server after the import.

- If the used project you want to import uses other projects, after the import these projects become directly used in the main project.
- The Standard/System Profiles cannot be imported according to our tool policy.

Importing models from other projects

Another way of composing your model, is to import a model from another project.

To import a model from another project

1. In the main menu, select **File** > **Import From** > **Another Project**.
2. Select a project you want to import.
3. Click **Import**. All elements from another project will be imported.

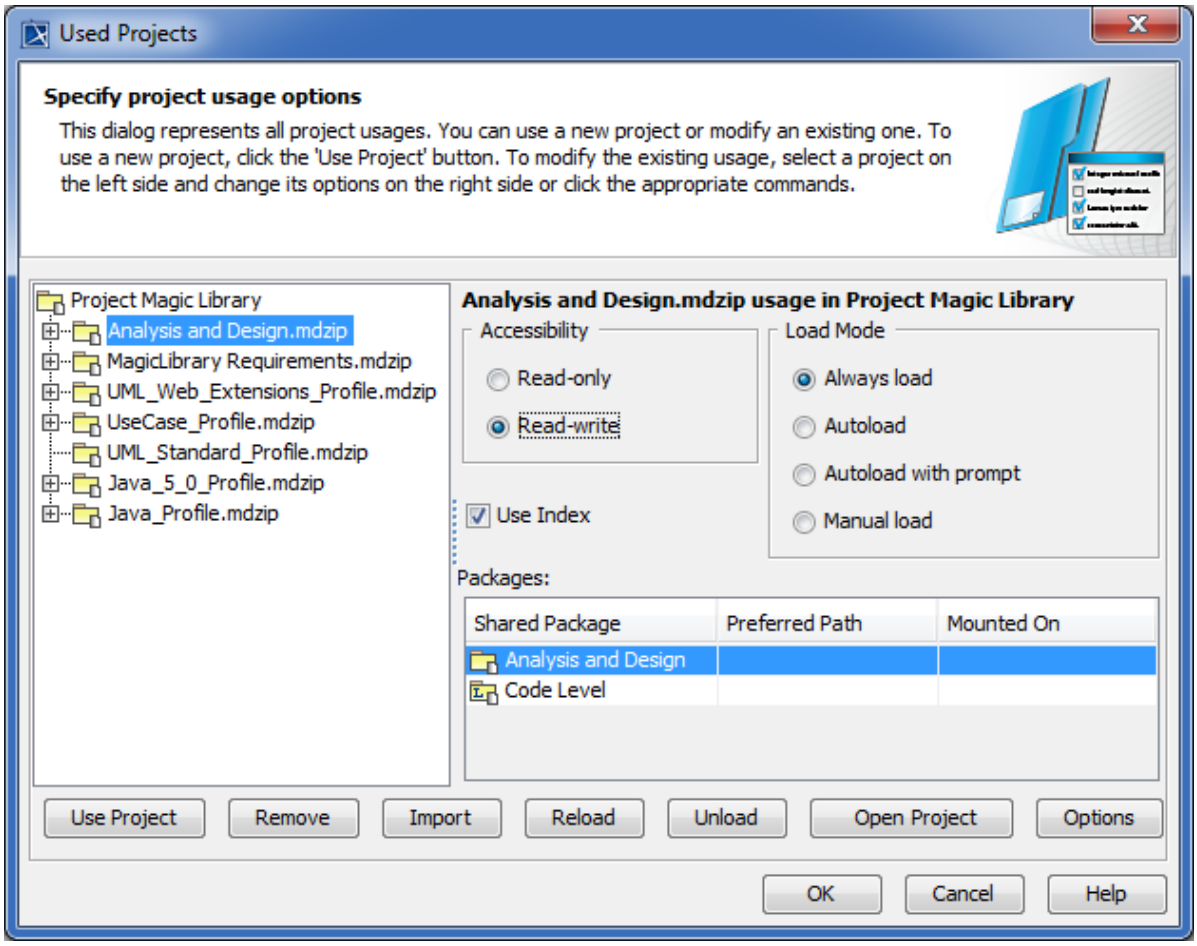
   If the model that you are importing also uses projects, these projects will be reused directly in your project after the model import.

Managing used projects

You can manage usages of an open Teamwork server or local project in the **Used Projects** dialog. To learn how to manage Teamwork cloud project usages, go to Managing project usages(see page 1441).

To specify project usage options

1. Do one of the following:
   • From the **Options** menu, select **Project Usages**.
   • Right-click a used project in the Containment tree(see page 32) and, from its shortcut menu, select **Project Usages** > **Options**.
   The **Used Projects** dialog opens.



2. Do one or both of the following:
   • Specify the following project usage options:

Dialog element name	Description
Accessibility	• **Read-only** - the project is loaded for reading only and cannot be modified in the main project. Elements of the used project can be used and referenced in the main project. • **Read-write** - the project can be edited directly in the main project. Elements of this used project can be used, referenced, and modified in the main project.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Dialog element name	Description
Load Mode	• **Always load** (default) - the selected used project is always loaded when the main project is opened. • **Autoload** - the selected used project is not loaded when the main project is opened. MagicDraw monitors user activities in the project and loads the particular used project on demand by the project. • **Autoload with prompt** - this mode is similar to the **Autoload** mode, except that MagicDraw asks the user for a confirmation before loading the used project. • **Manual load** - the selected used project is not loaded when the main project is opened. The model integrity is not broken, as all required elements of the used project exists, just simplified versions (that is, loaded as proxies) of the elements are used in the project. This load mode is recommended for all used projects that are stable or rarely modified. Reload the main project after changing the load mode of the selected used project.
Use Index	Select this check box to load indexed elements from the used project that is not loaded. This option is available only if the used project is not loaded. Using indexed elements in the project increases the performance.
Packages	this is the list of all shared packages of a used project and paths for mounting these packages in the main project.

- Use the following commands:

Command	Description
Use Project	Click to select projects for usage in the open project and specify the usage options. The Use Project(see page 242) wizard opens.
Remove > **Remove with References**	Click to remove the selected used project from the project. After removal, all the elements of the used project disappear from the Containment tree, and all the references to these elements are cleared from other model places.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Command	Description
Remove > **Remove, keep references**	Click to remove the selected used project and keep model references across it in the model. There are two cases when you may need to remove the used project this way. **Case #1**: You may need to replace a used project with the equivalent project carrying the same elements (but with slightly different content) and retaining all the references. In this case, removing the initially used project, but keeping references creates an automated project usage that temporarily carries the model level references. This automated usage is invalid, since it is unconfirmed. You can confirm it by replacing the initially used project with another project(see page 255). **Case #2**: You may need to refactor projects that are created with earlier MagicDraw version than 17.0.3 and remove excessive project usages. These versions of MagicDraw have only user-defined project usages. Because of this, a project usually had direct user defined usages to indirectly used projects besides the indirect usages through directly used projects. So both “project → directly used project → indirectly used project” and “project → indirectly used project” usage paths were present. In this case it may be advantageous to remove the surplus usage “project → indirectly used project” by clicking the Remove, keep References button.
Import	Click to import the contents of the selected used project to the main project. The Import button is available if the selected used project: • Is not a MagicDraw native project • Is locked (valid for server projects only) • Is not locked by another user (valid for server projects only) The same used project can be imported only once. After the import, all the elements of the used project are copied to the main project. The inner structure of the used project is included into the project as a package. The imported project is not deleted from the file system. Standard and system profiles cannot be imported.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Command	Description
Reload	Click to reload the selected used project in the main project. This button is available only when the selected used project is loaded.
Unload	Click to unload the selected used project from the main project leaving only used elements loaded. It is recommended to unload rarely used projects in order to reduce a memory usage while working with large projects. This button is available when a selected used project is loaded.
Open Project	Click to open the selected used project for editing.
Options	Click to open the **Project Options** dialog to manage styles and element display settings of the selected used project. It allows to change element styles in the used project even without opening it.

3. Click **OK**.

Related pages

- Reloading used projects in a project(see page 249)
- Working with partially loaded projects(see page 249)

Reloading used projects in a project

The best way to access the latest changes in a used project is to reload it. All modifications made in the used project become visible after it is reloaded in the open project. Alternatively, used projects are reloaded automatically when the open project is reloaded, or the project is updated on the server.

To reload the used project

---

1. In the Containment tree, right-click the package of the used project.
2. From the shortcut menu, select **Project Usages** > **Reload**.

> ⚠ If you open a used project for editing, be sure to save any changes you make. The modifications made appear after reloading the used project in the main project.

Working with partially loaded projects

To increase modeling efficiency when working with very large projects, you can work with partially loaded used projects. This may help decrease memory consumption.

> Here, “large” is tens of thousands of classes and other complex elements. If counting all small elements, such as properties, methods, method parameters, several hundreds of thousands of elements is considered a large project. Note that diagrams are large elements. A project consisting of 1000 or more complex diagrams should be considered large.

You can unload a particular used project at any time when working with a large partitioned project.

To unload a used project from a project

- Right-click the selected used project, and select **Project Usages** > **Unload**.

Not loaded (or unloaded) used projects can be loaded in a project anytime. For more information about managing used projects, see Managing Used Projects(see page 245).

To load a used project in a project

- From the unloaded used project's shortcut menu, select **Used Projects** and then **Load Used Project.**

When the used project is neither loaded nor unloaded, there can be some model elements left from the used project. These elements are not editable and are characterized by a small M letter in the upper left corner of their icon (see the following figure).

Notation of unloaded used project elements (fragment of Containment tree)

These are the so-called “proxy” elements of the original elements from the used project. Instead of the original model elements, the proxy is a lightweight surrogate that carries only the name and kind of the model element information. These proxies are left in place of the original used project elements and are dedicated to maintaining model integrity (ensuring that there are no dangling ends of relationships, types of properties do not disappear, etc.) while the used project is not accessible or not loaded.

There are 4 used project loading modes:

- **Always load** (default). In this mode, used projects are always loaded when you open the project. They can be unloaded if necessary.
- **Autoload**. A used project is not loaded while the project using it is loaded. However, our modeling tool monitors user activities in the project and tries to guess when you might want to use the model piece from the unloaded used project. For example, if you search and find usages/dependencies, reports, metrics, transformations, or code engineering actions with a scope that touches the unloaded used project, our modeling tool will load that used project.
- **Autoload with prompt**. This mode differs from the **Autoload** mode in that our modeling tool asks you before loading the used project.
- **Manual load**. A used project is not loaded while the project using it is loaded. It can be loaded using the **Load** used project command (described above).

To change the used project loading mode

---

1. From the **Options** menu, select **Used Projects.** The **Used Projects** dialog opens.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

2. In the used project's tree, select a used project.
3. In the **Used Project Load Mode** area, select the appropriate loading mode.

> Frequently Used Projects should be set in the **Always load** mode.

Occasionally used projects should be set in **Autoload** mode (or **Autoload with prompt** to have more control on loading behavior). Very rarely used projects can be set in the **Manual load** mode. Another frequent case where used projects can be set to **Manual load** mode occurs when the used projects represent a software library that is not expected to change.

Related pages

- Maintaining decomposed model integrity(see page 254)

Automated project usages

Automated usages of other projects in the main project are the usages fully managed by our modeling tool. For example, it can create, remove, and rearrange as necessary without user intervention.

Version 17.0.3 and later

Version 17.0.3 introduces a new type of project usages, **automated project usages**. Our modeling tool fully manages them. It creates, removes, and rearranges as necessary without user intervention. They are not shown in the **Used Project** dialog(see page 245) as only **user-defined usages** are shown there.

Before version 17.0.3

In the earlier versions of our modeling tool, there was just one type of project usages. The user creates these **user-defined usages** and they carry additional parameters, i.e., whether the used project is used in read-only or read/write mode, the used project loading options, which used project version is required, etc.

When modifying projects with complex composition, always keep the following condition in mind:

> *Whenever there are model level references from elements in resource A (either a used project or the main project) to elements in resource B, there is a used project usage A → B.*

In earlier versions of our modeling tool, you could see this condition in action when working with used projects in read/write mode. Whenever you modified the contents of a used project so that it started to refer to another used project, and then you attempted to save it, our modeling tool would add an additional usage. You would then see the messages: "*Used project B was directly attached to A*" or "*Mounts were updated ...*".

However, in some cases, this system behavior caused undesirable consequences, as shown in the following pages:

- Easier refactoring of composite used projects(see page 252)
- Controlling dependency creation between used projects(see page 253)

Related pages

- Managing Used Projects(see page 245)
- Project Partitioning(see page 236)

Easier refactoring of composite used projects

One of the problems caused by the old approach described in Automated project usages(see page 251), was in refactoring composite used projects (used projects composed of sub-used projects). Whenever a composite used project was re-arranged by removing or re-importing sub-used projects, all of the projects or used projects using this used project more often than not became affected.

The following examples illustrate this problem.

Example 1
Consider a used project *ElectricComponents* with 3 sub-used projects: *Capacitors*, *Inductances*, and *Resistances*. Consider another project *RadioSet*, which uses the used project *ElectricComponents*.
Now, use several model elements from each sub-used project in the project *RadioSet*.

Old problematic behavior

Previous versions of our modeling tool would create these three direct user-defined usages:
RadioSet → *Capacitors,*
RadioSet → *Inductances,* and
RadioSet → *Resistances.*
Effectively, those versions were "freezing" the contents of the used project *ElectricComponents.* If you tried to refactor the used project *ElectricComponents* by removing its sub-used projects and re-importing their contents, the project *RadioSet* would be affected and start complaining about the missing used projects.
No earlier version of our modeling tool could rearrange the user-defined usages for fear of losing the user-specified used project composition data.

New behavior

As of version 17.0.3, a program creates automated used project usages instead of creating user-defined ones. So when the used project *ElectricComponents* is refactored, it keeps the same model elements but changes its structure. Furthermore, the automated usages in the project *RadioSet* are automatically rearranged.

Another types of problems
Now problems could occur only if the rearranged used project does not contain all the same elements it had before the rearrangement. Thus some automated usages may turn into unconfirmed ones. For more information about unconfirmed usages, see Controlling dependency creation between used projects(see page 253).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example 2
Another problem was “freezing” the used project's version. For instance, the project *RadioSet* uses the 100th version of the used project *ElectricComponents,* which in turn uses the 150th version of the sub-used project *Resistances.*

Old problematic behavior

Whenever the project *RadioSet* referenced the sub-used project *Resistances,* the direct usage would be created to the 150th version of it. If you wanted to migrate the project *RadioSet* to use the 200th version of the used project *ElectricComponents,* which used the 300th version of the sub-used project *Resistances,* a version conflict would occur, since the direct usage would be still pinned to the 150th version.

New behavior
As of version 17.0.3, automated usages do not carry information about the version and do not preclude the version update through the user-defined usage paths.

Complex Composition
If you already have projects with complex composition (having used projects or sub-used projects) that are prepared in any earlier version, you may want to refactor your used project structure to reap the benefits of this feature.

Related pages

- Resolving unconfirmed usages of packages(see page 255)

Controlling dependency creation between used projects

In earlier versions of our modeling tool, you could accidentally create unneeded or incorrect (from the user standpoint) dependencies between used projects in the read/write mode. Careless editing often caused cyclic module dependencies or inadvertent expansion of the scope of the other projects when the same used project was used in several other projects.

Automated project usages fix this problem.

Every automatically created usage must be backed by an existing user-defined used project usage. If it is not backed, the automated used project usage must be explicitly confirmed by the user. The usage cannot cause undesired effects in other projects while it is unconfirmed.

Automated used project usage $A \rightarrow B$ is called **unconfirmed** when there is no user-defined usage path from A to B (neither direct user-defined usage $A \rightarrow B$ nor user-defined usage path of arbitrary length $A \rightarrow \ldots \rightarrow X \rightarrow B$).

User-defined usages are always confirmed.

In ordinary circumstances, automated usages are not visible to the user. When the user modifies the main project’s model and adds dependencies to used projects or sub-used projects, these usages are proper. They are backed by the fact that the used project or sub-used project is attached to the project in some way; thus, the user-defined usage path refers from the main project to the used project or sub-used project.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

However, there are situations when automated used project usages are not backed and thus unconfirmed.

> ⚠ An unconfirmed automated used project usage is considered to be invalid.

This is just one of the project composition errors. Other project composition errors include similarity to recovered element errors, conflicting user-defined usage parameters, and cyclic dependencies.

Unconfirmed automated used project usages are caught by special automatic validation rules(see page 754). They are displayed in validation results as warnings or errors depending on the project composition. The validation results prompt the user to remedy the situation by either confirming or rejecting the usage.

Unconfirmed usages can occur in several cases. They are described as follows.

> ⓘ **Case #1**
> When you edit a model part belonging to a used project with read/write permissions, an additional model-level reference can be created either back to the project itself (cycle) or to another used project.
> These model-level usages may or may not be valid from the user point of view. Since the creation of usages is very easy, you can inadvertently create undesired usages. On the other hand, usages can be entirely benign. Our modeling tool cannot decide whether the usage is good or bad. For example, creating references from the used project *Implementation* to the used project *Requirements* is valid, while creating references in the opposite direction may be invalid since requirements usually are standalone, and implementation refers to them.

> ⓘ **Case #2**
> Reorganization or removal of sub-used projects may affect the projects or other used projects using the used project. This happens only in cases when a new refactored used project does not carry all of the contents that the old used project contained.
> Consider a used project *ElectricComponents* with three sub-used projects: *Capacitors, Inductances,* and *Resistances.* Also consider a project *RadioSet,* which uses the used project *ElectricComponents.* Now let's use one model element from each sub-used project in the project *RadioSet.* As a result, the three automated usages
> *RadioSet → Capacitors,*
> *RadioSet → Inductances,* and
> *RadioSet → Resistances*
> will be created.
> If the used project *ElectricComponents* was refactored by re-importing the sub-used project *Capacitors* and *Inductances* and completely separating the sub-used project *Resistances,* t hen when loading the project *RadioSet,* the automated usages *RadioSet → Capacitors* and *RadioSet → Inductances* will be automatically updated. However, the usage *RadioSet → Resistors* will remain unaddressed. It will appear as an unconfirmed used project usage, and you will have to confirm it, i.e., explicitly attach the used project *Resistances* to the project by creating the user-defined usage.

Maintaining decomposed model integrity

To maintain model integrity, there are two types of elements appearing in the model:

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Proxies
- Recovered elements

A **proxy** is a lightweight surrogate that carries only the name and kind of the model element information. These proxies appear in the place of the original used project elements, and are dedicated to maintaining model integrity (so that there are no dangling ends of relationships, types of properties do not disappear, and so on), while the used project is not accessible (for example, you do not have permissions) or is not loaded.

When you do not have granted “Read model” permission for a used project, this used project will not be loaded in a project where it is being used. To maintain the model integrity and keep model references that point to elements of “not accessible” used project, proxies are being displayed instead of referenced not accessible elements. That means that you will be able to see only the following information of these elements:

- Metaclasses of the element
- Names of the element
- Owners of the element

Recovered element is really an indication of the dangling reference. Appearance of these elements indicates that there are elements in the project (for example, elements in the main project or its used projects) that refer to the element in the used project that was previously there but currently cannot be found. It may be that the element was unsafely deleted/ removed/somehow made unavailable in the module. In such case, a modeling tool continues maintaining model integrity of unresolved reference by recovering missing elements - so called “Recovered element” appear in a place of the missing element where the original element existed in the past.

Read how to manage recovered elements in the following topics:

- Resolving unconfirmed usages of packages(see page 255)
    - Resolving unconfirmed module usages(see page 258)
- Managing unresolved references(see page 258)
    - Replacing recovered element with another element(see page 259)
    - Removing unresolved reference(see page 260)
    - Removing recovered element symbol(see page 261)
    - How to avoid triggering element recovery(see page 262)

Resolving unconfirmed usages of packages

For the overview of automated project usages and situations when they can become unconfirmed, see Controlling dependency creation between used projects(see page 253).

Unconfirmed project usages are caught by special automatic validation rules(see page 251) and displayed as a warning or error validation result. The severity depends on the exact model composition situation, but the solution methods are the same in both cases.

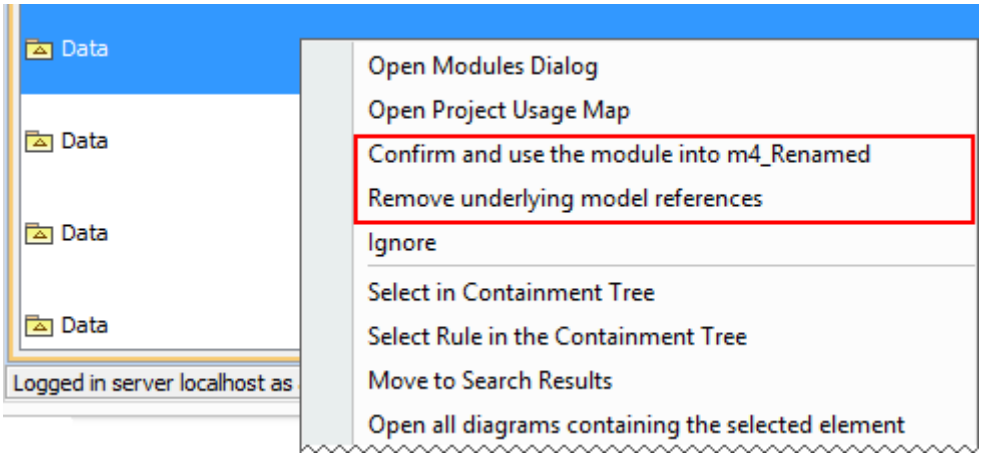
> **Model Usages**
> Since problematic project usages are not model elements, they cannot be visible in the Containment tree(see page 693). So when the model contains unconfirmed usages, the root package *Model* is highlighted as incorrect (regardless of which usage is actually problematic).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Model Usages

When there is an error level unconfirmed usage of some used project, then all the elements of the used project, that are referenced from other model places are shown as Recovered Elements(see page 258). These recovered elements are in turn flagged as errors. Hence single unconfirmed usage can cause a lot of error validation results: one for the unconfirmed usage itself (rule abbreviation - NCAMU) and one for each recovered element (rule abbreviation - REF). In this situation, the unconfirmed usage result should be examined first, because solving it may automatically solve all other results.

There are two ways to resolve an unconfirmed used project usage. The usage can be either confirmed or rejected.



Two ways for resolving unconfirmed used project usages.

Confirming usages

If the used project usage A → B is good and necessary according to the user policy, it can be confirmed, i.e., the user-defined usage is to be created instead of the current unconfirmed automated used project usage.

You can confirm an automated usage

- By using the specific command
- Manually

To confirm an automated usage by using the specific command

1. Right-click the automated usage in the validation results.
2. Select **Confirm and use the used project into <used project_name>** from the shortcut menu (see the preceding figure). The **Used project** wizard(see page 242) opens. The required used project is already selected in the first step of the wizard.
3. Click **Finish.** The necessary user-defined used project usage will be created.

To confirm an automated usage manually

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example
In this case, you can better control the usage creation process. So if you need, for example, to use not only a required used project but some other one that brings the required used project as a part of it, be advised to confirm the usage manually.
For example, let's say there are three unconfirmed usages from the project *RadioSet* to used projects *Capacitors, Inductances,* and *Resistances.* Instead of confirming each usage, you can create just one user-defined usage *RadioSet* → *ElectricComponents*, since the used project *ElectricComponents* brings in *Capacitors, Inductances,* and *Resistances* as sub-used projects.

Important!
However, this is the case when you must be sure that the user-defined usage will be created in the right place.

1. on the main menu, click **Options** > **Used Projects** . The **Used Projects** dialog(see page 245) opens.
2. Select a used project that is the usage target resource.
3. Click the **Used project** button. The **Used Project** wizard(see page 242) opens.
4. Follow the steps of the usual procedure (select a required used project and finish or continue to the second wizard step to provide more configuration options for used project usage and then finish).

Rejecting usages

If the usage A → B is not good according to the user policy, it must be rejected and removed.

You can reject an automated usage

- By using the specific command (for removing underlying model level references)
- Manually (by removing or redirecting underlying model level references to different elements)

To reject an automated usage by using the specific command

Model Usages
Use this way to reject a model-level usage that, for example, was created inadvertently.

1. Right-click the automated usage in the validation results.
2. Select **Remove underlying model references** from the shortcut menu (see the preceding figure). All underlying model-level references (the ones that cause this automated usage) will be removed.

This command is very similar to the **Clear Recovered Element Usages** command.

To remove an automated usage manually

In more complex cases, you may want to address each usage individually. Use dependency analysis means to examine what dependencies are there and review them on a case-by-case basis (see Analyzing usages and dependencies(see page 1192), Analyzing package dependencies(see page 1280), Displaying related elements(see page 508), Dependency matrix(see page 334), and so forth).

Useful commands for preparing to resolve unconfirmed used project usages

When you need to explore your project composition in detail before attempting to resolve unconfirmed usages, use the following commands for navigating:

- Open Used Projects dialog(see page 2215)
- Open Project Usage Map(see page 1511) for collaboration projects

Related pages

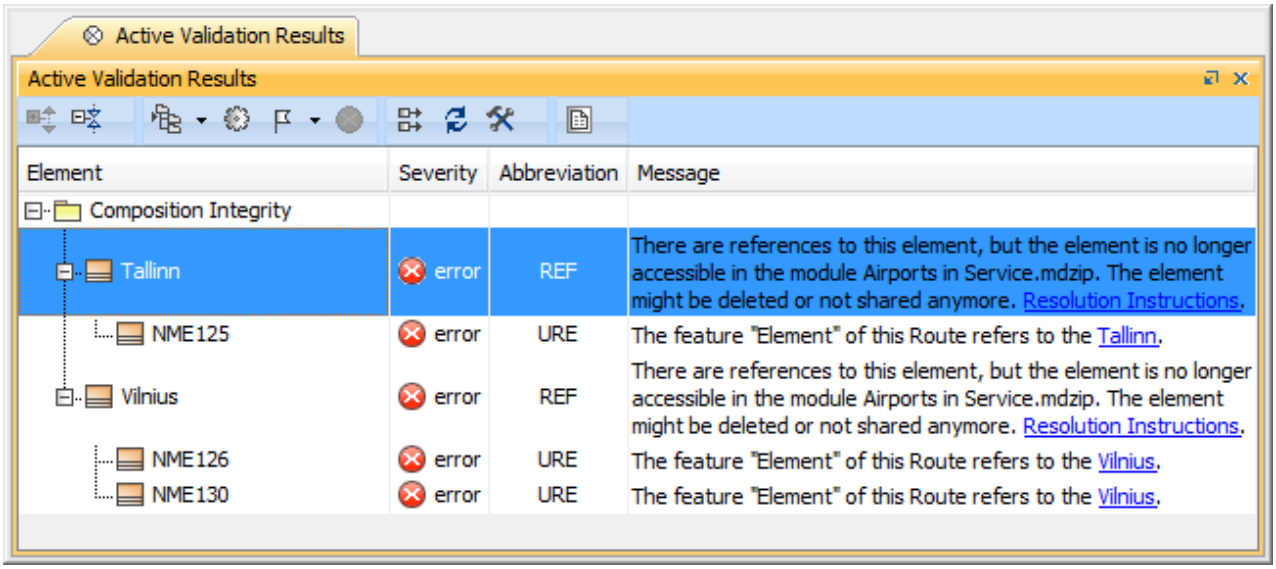
- Analyzing Usages and Dependencies(see page 1192)
- Analyzing Package Dependencies(see page 1280)
- Using other Projects in a Project(see page 242)
- Use Project Wizard(see page 1162)
- Project Usage Map(see page 1511)
- Validation(see page 1306)

Managing unresolved references

You can manage the unresolved references in the Model Browser or in the **Active Validation Results** panel. In the **Active Validation Results** panel, they are grouped by the recovered elements they refer to.

> ⚠ If you cannot see the recovered elements in the **Active Validation Results** panel, set the **Composition Inspection** option to **Advanced** in the **Project Options** dialog(see page 214). See, how to select the relevant level of project composition inspection(see page 264).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Representation of unresolved references grouped by recovered elements in the Active Validation Results panel

These are the methods of managing unresolved references using specific commands:

Replacing recovered element with another element

At times, you may decide to reference a different element rather than the recovered one in the model.

To replace a reference from a recovered element to another one

1. Do either:
   - In the Model Browser, right-click an element that refers to the recovered element. From its shortcut menu select **Validation** > **Usage of Recovered Element** > **Use Another Element**.
   - In the **Active Validation Results** panel, right-click an element that refers to the recovered element and whose abbreviation is URE. From its shortcut menu select **Use Another Element**.
2. In the element Selection dialog, select the element you want to use instead of the recovered element.
3. Click **OK**. The reference is now pointing to the selected element. Also, it is no longer unresolved.

> You can use the command for more than one element at a time.

> If some of the unresolved references are read-only, belong to the read-only module, are locked by another user, or still not locked, the command will skip these elements.

Alternatively, you can replace all references from a particular recovered element to another one using one operation.

To replace a recovered element with another one in all references

1. Do either:
   - In the Model Browser, right-click the recovered element and from its shortcut menu select **Validation** > **Usage of Recovered Element** > **Change Usages To**.
   - In the **Active Validation Results** panel, right-click the recovered element and from its shortcut menu select **Change Usages To**.
2. In the element Selection dialog, select the element you want to use instead of the recovered element.
3. Click **OK**. All the references are now pointing to another element. Also, they are no longer decorated as unresolved.

Related pages

- Removing unresolved reference(see page 260)
- Removing recovered element symbol(see page 261)
- Managing unresolved references(see page 258)

Removing unresolved reference

At times, a reference to some recovered element is no longer needed and can be removed.

To remove a reference to the recovered element, do one of the following

- In the Model Browser, right-click an element that refers to the recovered element. From its shortcut menu select **Validation** > **Usage of Recovered Element** > **Clean Usage of Recovered Element**.
- In the **Active Validation Results** panel, right-click an element that refers to the recovered element and whose abbreviation is URE. From its shortcut menu select **Clean Usage of Recovered Element**.

The reference to the particular recovered element is removed.

> You can use the command for more than one element at a time.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

> ⚠ If some of the referring elements are non-editable, belong to the read-only module, are locked by another user, or still not locked, the command will skip these elements.

You can remove all unresolved references (including all used modules) of a recovered element in one operation.

To remove all references to a recovered element, do one of the following

- In the Model Browser, right-click the recovered element. From its shortcut menu select **Validation** > **Usage of Recovered Element** > **Clean Usage of Recovered Element**.
- In the **Active Validation Results** panel, right-click the recovered element. From its shortcut menu select **Clean Usage of Recovered Element**.

All unresolved references to the particular recovered element are removed. As the unresolved references are cleared, the recovered element is removed as well.

> ⚠ Unresolved references will not be removed from the referring elements,if they are stored in a read-only module, are locked by another user, or still not locked.

Related pages

- Replacing recovered element with another element(see page 259)
- Removing recovered element symbol(see page 261)
- Managing unresolved references(see page 258)

Removing recovered element symbol

This is the case when a recovered element is depicted on diagrams of your project.

To remove a recovered element's symbol, do one of the following

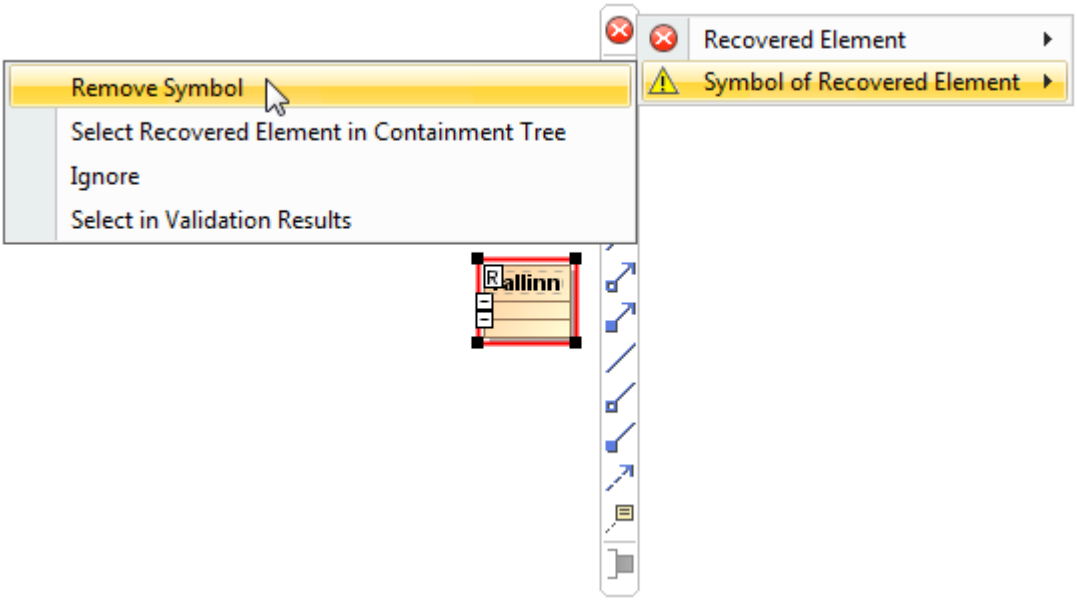
- On a diagram pane, select the symbol. On its smart manipulator toolbar click and then select **Symbol of Recovered Element** > **Remove Symbol**, as shown in the following figure.
- On the diagram pane, select the symbol and press Delete.
- In the **Active Validation Results** panel, right-click the symbol and select **Remove Symbol** from its shortcut menu.

The symbol of the recovered element is removed from the diagram.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

The same diagram can have many symbols of the recovered element. To remove all of them, repeat the same procedure for each symbol.

You can use the **Remove Symbol** command for more than one symbol at a time. If the diagram including the recovered element symbols is non-editable, belongs to the read-only module, is locked by another user, or still not locked, the command will not be applied.



Using smart manipulator toolbar to remove recovered element's symbol

Related pages

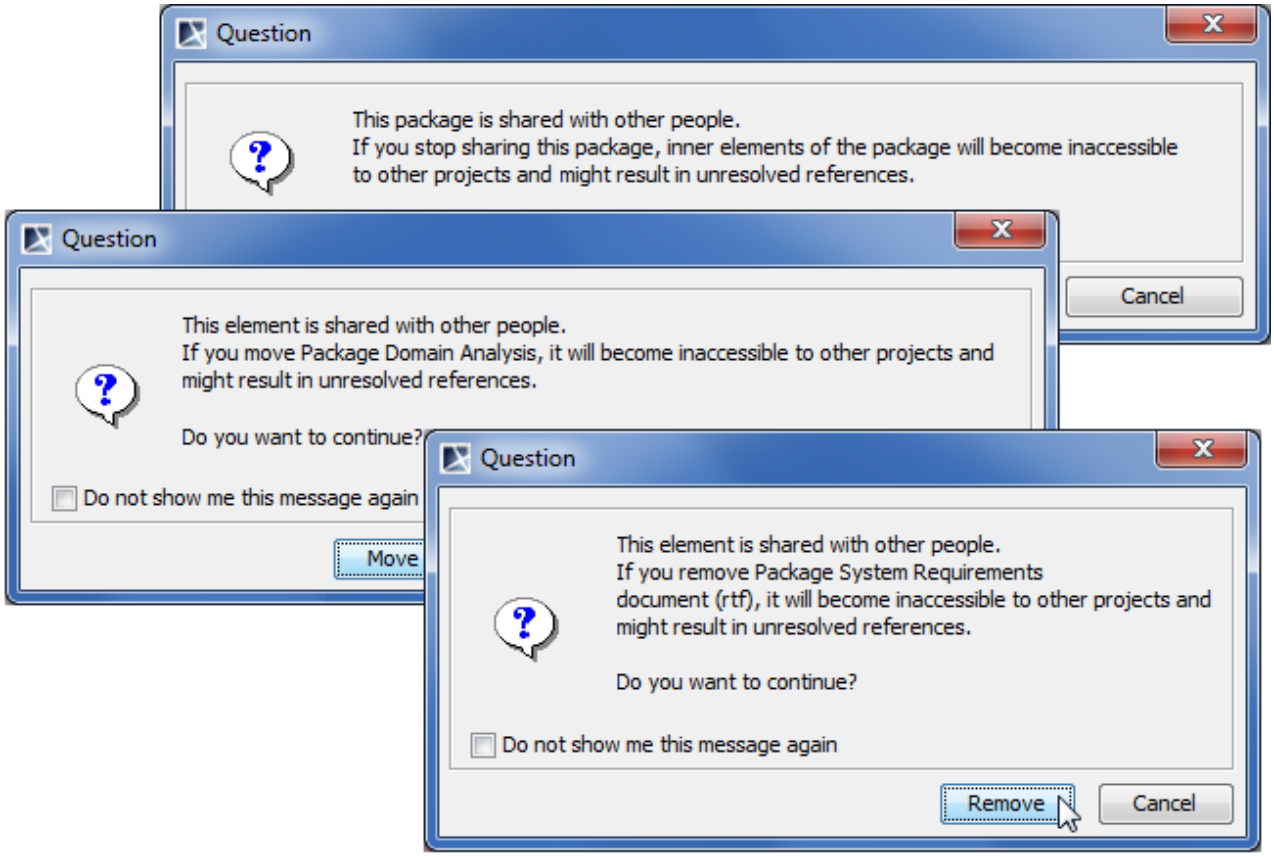
- Replacing recovered element with another element(see page 259)
- Removing unresolved reference(see page 260)
- Managing unresolved references(see page 258)

How to avoid triggering element recovery

The following recommendations will help you avoid triggering element recovery in your project:

1. Avoid unexpected modifications of shared elements, such as removing a shared element or moving the element to a non-shared location, etc.
2. Configure the modeling tool to prompt you for confirmation before proceeding with a potentially dangerous action, such as attempting to stop sharing a package, removing the content of a shared package, or moving elements from one module to another, when both modules are used in the read-write mode (shown in the following figure).
3. Select the relevant level of model integrity and project composition inspection.
4. Manage unresolved references as soon as possible, as they might produce other unresolved references. Remember that the fewer recovered elements your model has, the more valid it is.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

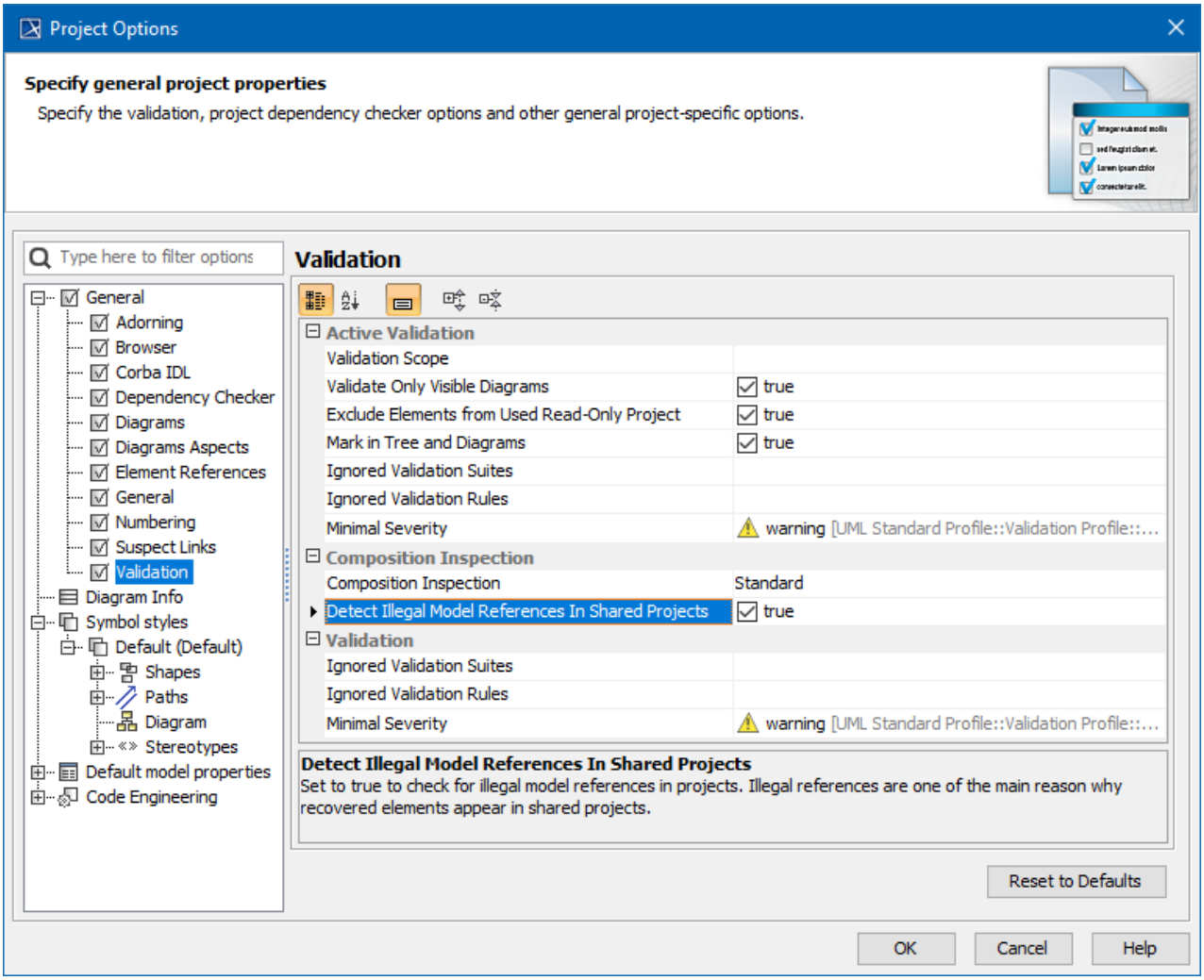


Examples of messages prompting to confirm potentially unsafe actions

To define a set of actions for modeling supervision to prevent recovered elements creation

---

1. Select **Options** > **Projects** to open the **Project Options** dialog(see page 214).
2. In the option group list, select the **General** > **Validation**.
3. In the **Validation** options list under the **Composition Inspection**, set the **Detect Illegal Model References in Shared Projects** property value to *true*.
4. Click **OK** when you are done.



To select the relevant level of project composition inspection

1. Select **Options** > **Projects** to open the **Project Options** dialog(see page 214).
2. In the option group list, select the **General** > **Validation**.
3. In the **Validation** options list under the **Composition Inspection**, specify the **Composition inspection** option value:
   - Select **Standard** if you need the active validation to search for issues *only* in the project, but not in the projects it uses.
   - Select **Advanced** if you need the active validation to search for issues in the project as well as in the projects it uses.
4. Click **OK** when you are done.

Validating project integrity

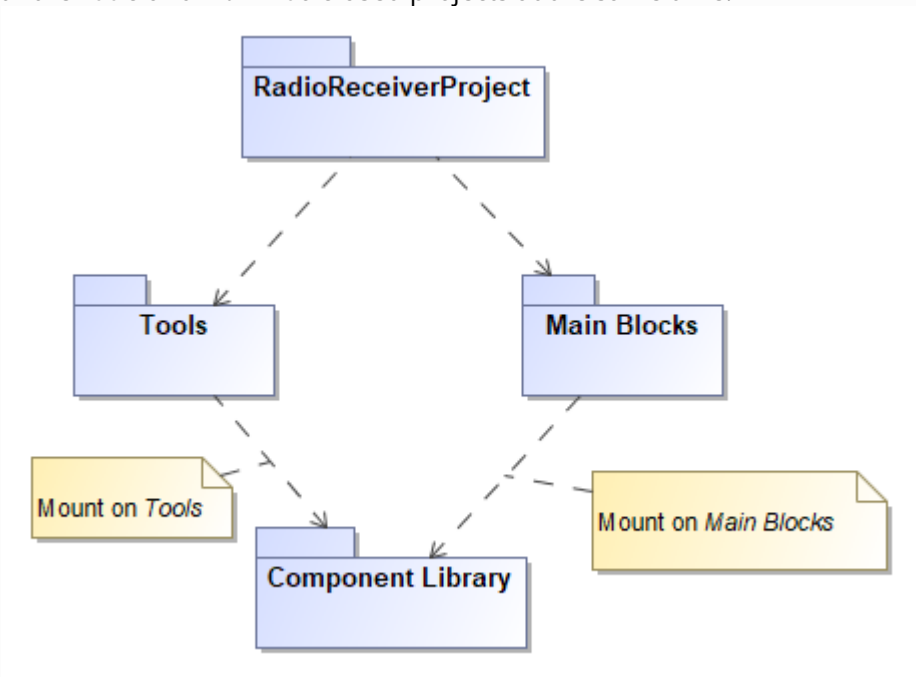
Some aspects are validated automatically, such as:

- Mount location inconsistencies in used projects
- Required version inconsistencies in used projects

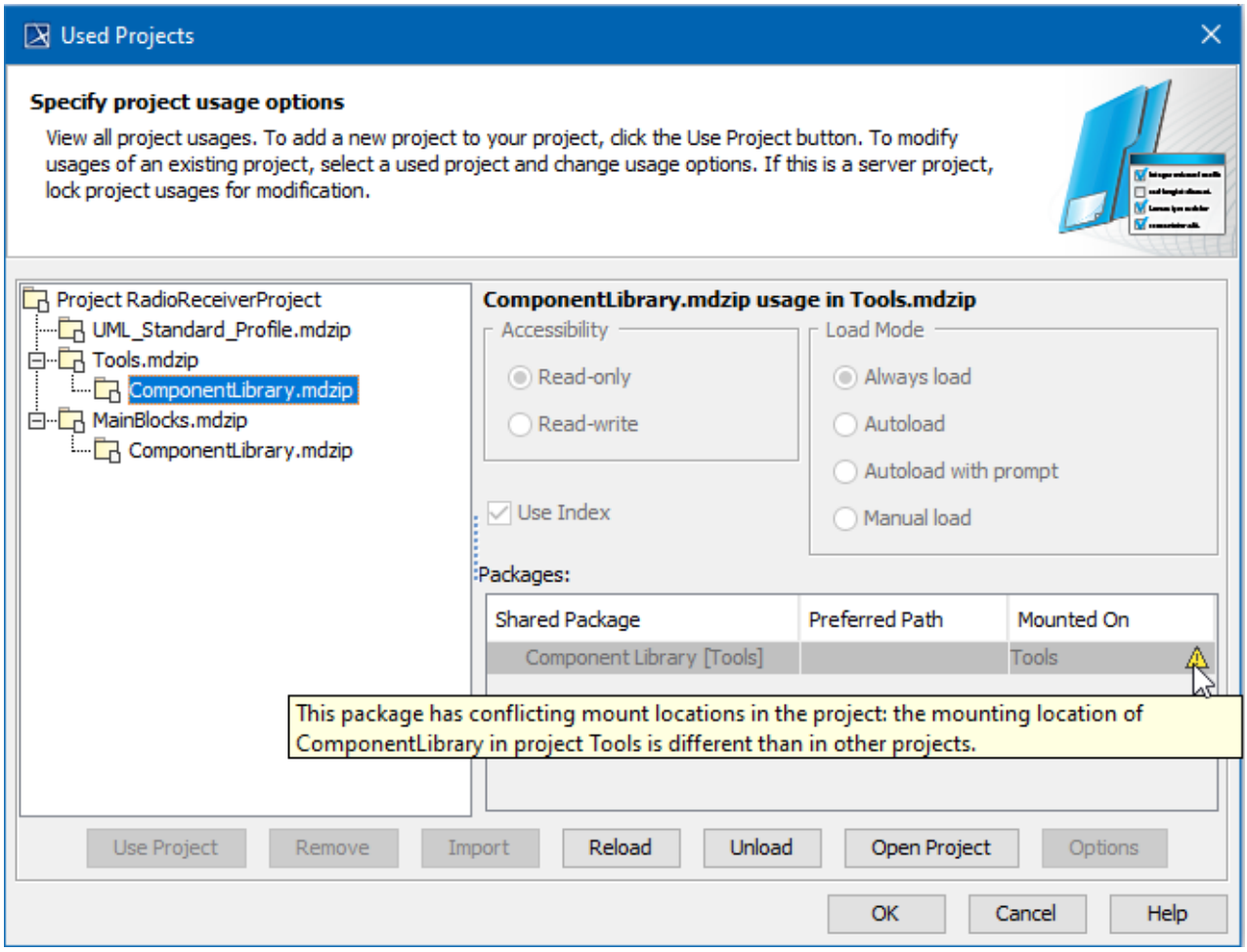
Mount location inconsistencies in used projects

This occurs when packages of the used project have different mount locations defined in separate projects.

Example
The used project *Tools* has mounted a package of the used project *ComponentLibrary*, while the used project *MainBlocks* has also mounted the same package *ComponentsLibrary*. As shown in the following figure, the mount location conflict occurs because the project *RadioReceiverProject* is using both the *Tools* and *MainBlocks* used projects at the same time.



You can solve the conflict in the **Used Project** dialog(see page 1441) by unifying the **Mounted On** value in both projects.



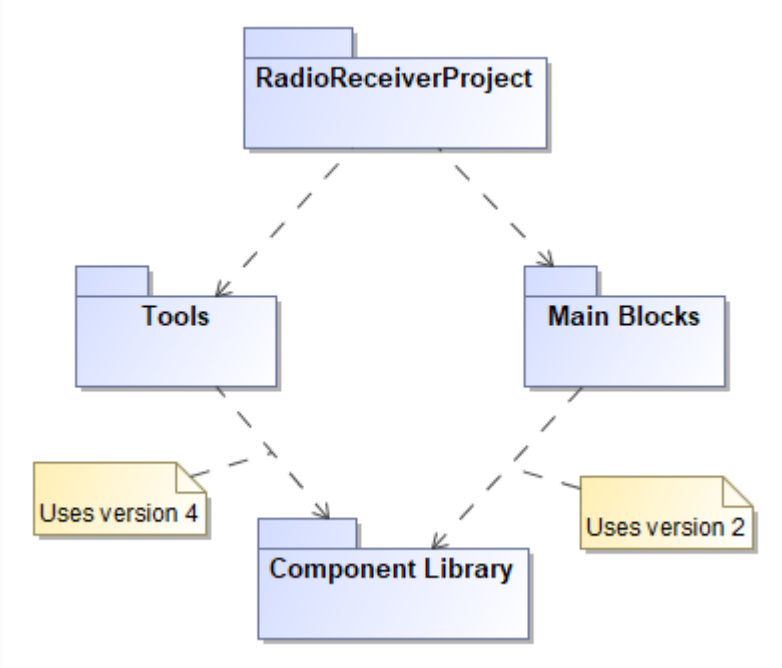
Example of conflicting mount locations warning displayed in Used Project dialog.

Required version inconsistencies in used projects

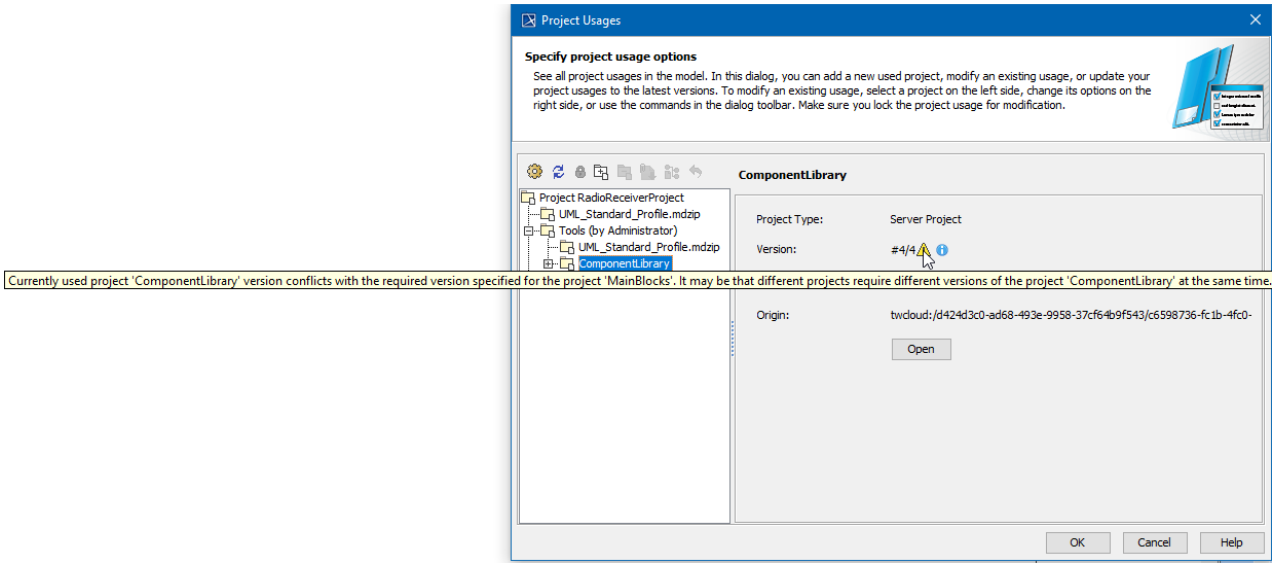
This occurs when a project directly and indirectly uses several different versions of the same used project at the same time.

Example

The used project *Tools* is using the 4th version of the used project *ComponentLibrary* and the used project *MainBlocks* is using the 2nd version of the used project *ComponentLibrary*. In the project *RadioReceiverProject*, a used project *ComponentLibrary* version conflict occurs as two different versions of this project are required at the same time (see the following figure).



You can solve the conflict in the **Used Project** dialog(see page 245) by unifying the required versions in both projects.

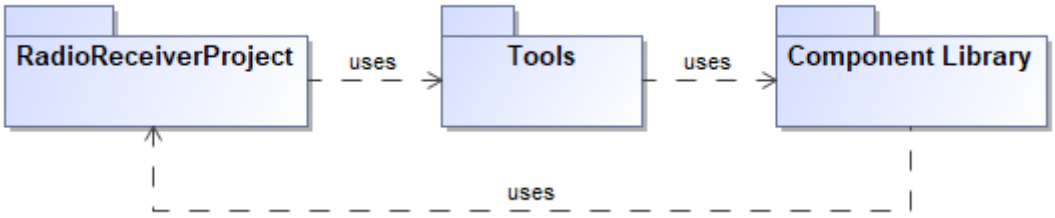


Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example of conflicting versions warning displayed in Used Project dialog

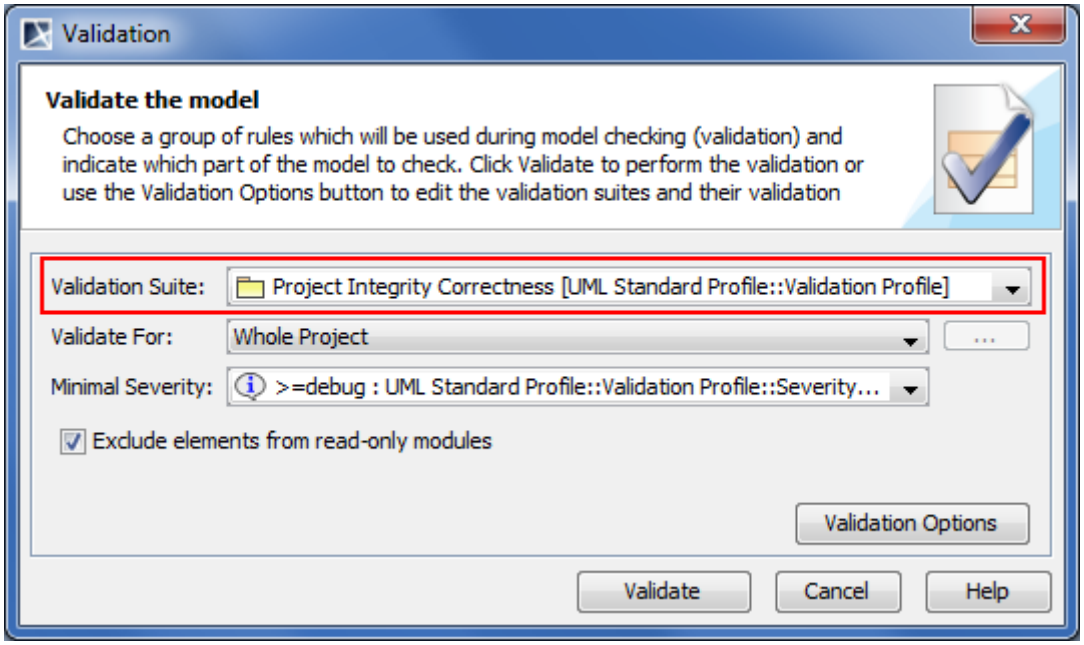
Cyclic dependencies among used projects

The figure below illustrates an example of cyclic dependency:



Example of cyclic dependency.

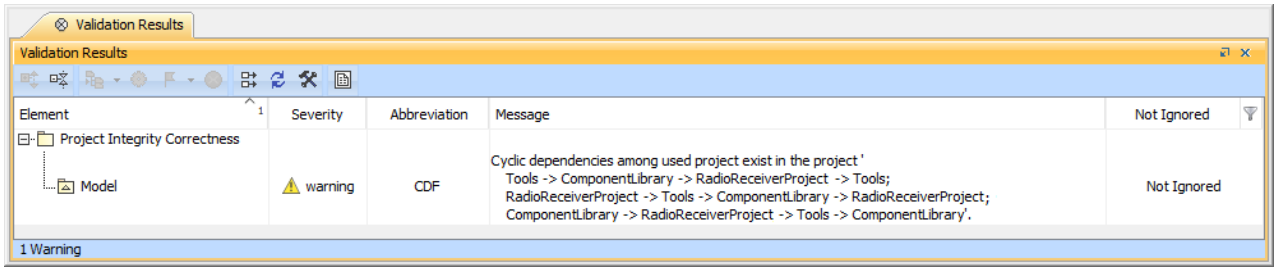
The Project Integrity Correctness suite includes the rules for **detecting** Cyclic dependencies among used projects. While validating (see page 1306)the model, use the **Project Integrity Correctness** validation suite to find cyclic dependency issues in your project composition.



Project Integrity Correctness suite

Violations of your project integrity will be displayed in the **Validation Results** panel(see page 1316).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Example of violations detected while checking project integrity correctness

You can resolve the cyclic dependency issues in the **Used Project** dialog(see page 245) or you can use the project usage map(see page 1511).

Related pages

- Validation(see page 1306)
- Validation Results Panel(see page 1316)
- Project Usage Map(see page 1511)
- Managing Used Projects(see page 245)

MOF support

MagicDraw allows export to/import from UML models from/to MOF (both CMOF and EMOF) XMI files. MOF 2.0 and 2.4 (MOF 2.4.1 to be more specific) are supported.

For more information about the MOF domain model, see **Meta Object Facility (MOF) Core Specification** for MOF 2.0[62] or MOF 2.4.1[63].

Related pages

- Exporting projects to MOF files(see page 269)
- Importing projects from MOF files(see page 271)

Exporting projects to MOF files

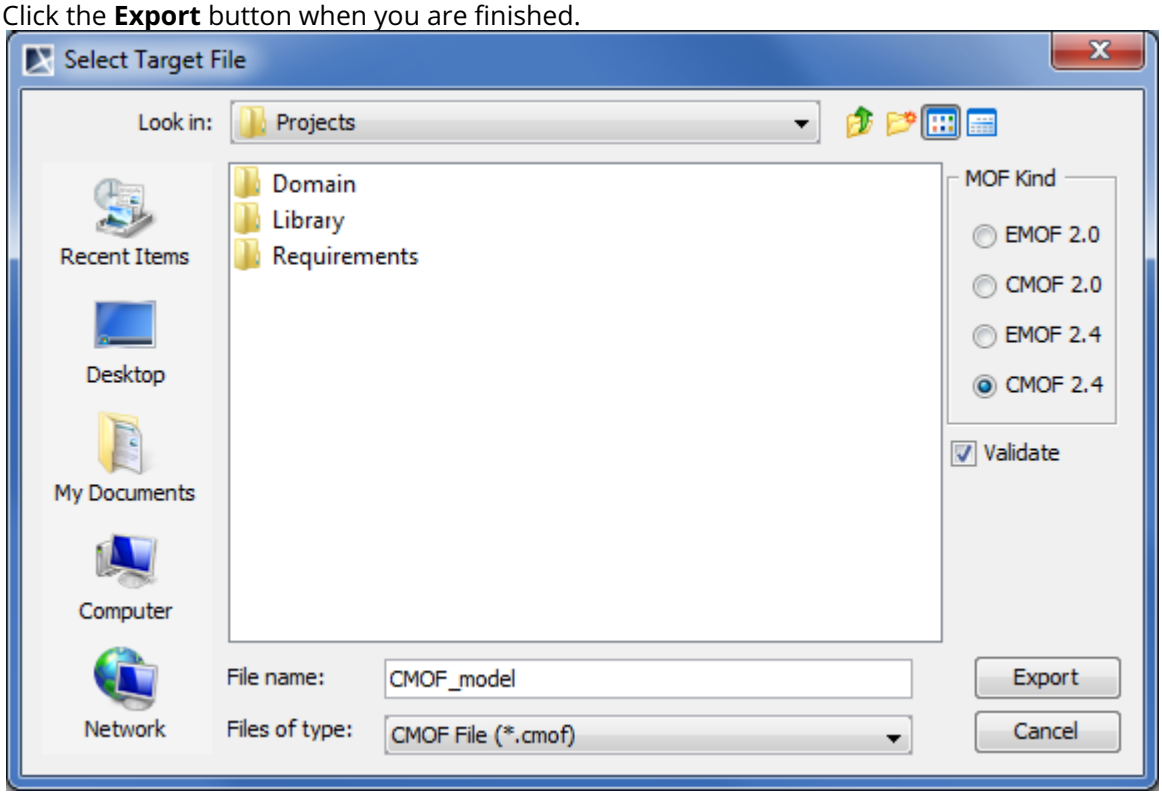
MagicDraw allows for exporting a selected project either to EMOF or the CMOF package. You can choose to export either the whole project or selected packages only. The model, all except auxiliary resources (for example, the UML Standard Profile package), will be exported to the selected MOF file.

62 http://www.omg.org/spec/MOF/2.0/PDF/
63 http://www.omg.org/spec/MOF/2.4.1/PDF/

Diagram data and model features available only in UML (behavioral models in particular) cannot be exported.

To export a project to a MOF file

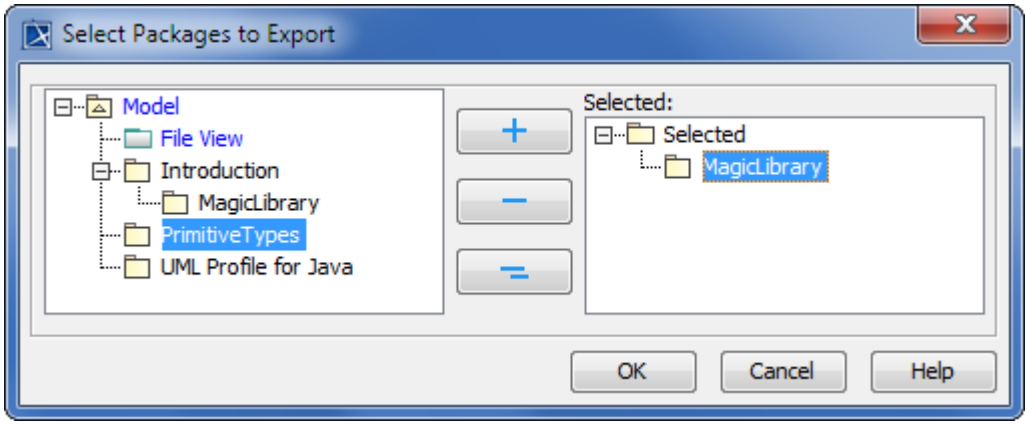
1. From the main menu, select **File** > **Export To** > **MOF XMI File** > **MOF Whole Model**. The **Select Target File** dialog opens (see the following figure).
2. Type the file name and select the location for the exported model.
3. On the right side of the dialog, select **MOF Kind**. The file type of the exported model changes according to your selection.
4. Click to select or to clear selection from the **Validate** check box.
5. Click the **Export** button when you are finished.



To export selected packages to a MOF file

1. From the main menu, select **File** > **Export To** > **MOF XMI File** > **MOF Selection**.
2. In the **Select Packages to Export** dialog, select packages you want to export and click **Add** to move them to the **Selected** list. Click **OK** when you are done.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



3. Perform the further steps as described in the previous export procedure (starting from step #2).

Exported elements validation
MagicDraw provides two validation suites (one for CMOF and one for EMOF) for validating models that are being exported to MOF files. These suites contain batches of rules to check exported elements.

Warnings about elements that were not exported (for example, diagrams, behavioral elements, or other) are displayed upon completion of the validation process. Validation does not preclude the model from being exported, the ineligible elements are simply skipped.

Importing projects from MOF files

To import a project from a MOF file

---

1. From the main menu, select **File** > **Import From** > **MOF XMI File**.
2. In the dialog, select the EMOF or CMOF file you want to import, and click **Open**. The file is imported as a separate project.

Ecore support

Ecore model is a model type supported by Eclipse Modeling Framework (EMF). This model type can be colloquially called the EMF model (even though EMF supports many types of models, e.g., UML models).

Ecore models can be used for various purposes:

- Metamodeling purposes, where its expressive power is roughly similar to EMOF (and even slightly higher than EMOF).
- Simple class modeling purposes, where Ecore model is used as a subset of UML.

Ecore models made using MagicDraw can be exported as Ecore models for the further processing (generating model repositories, code or XML parsing and storing, etc.) with other EMF tools.

Related pages

- Creating projects for Ecore modeling(see page 272)
- Ecore modeling(see page 275)

- Exporting projects to Ecore files(see page 283)
- Importing projects from Ecore files(see page 287)

Creating projects for Ecore modeling

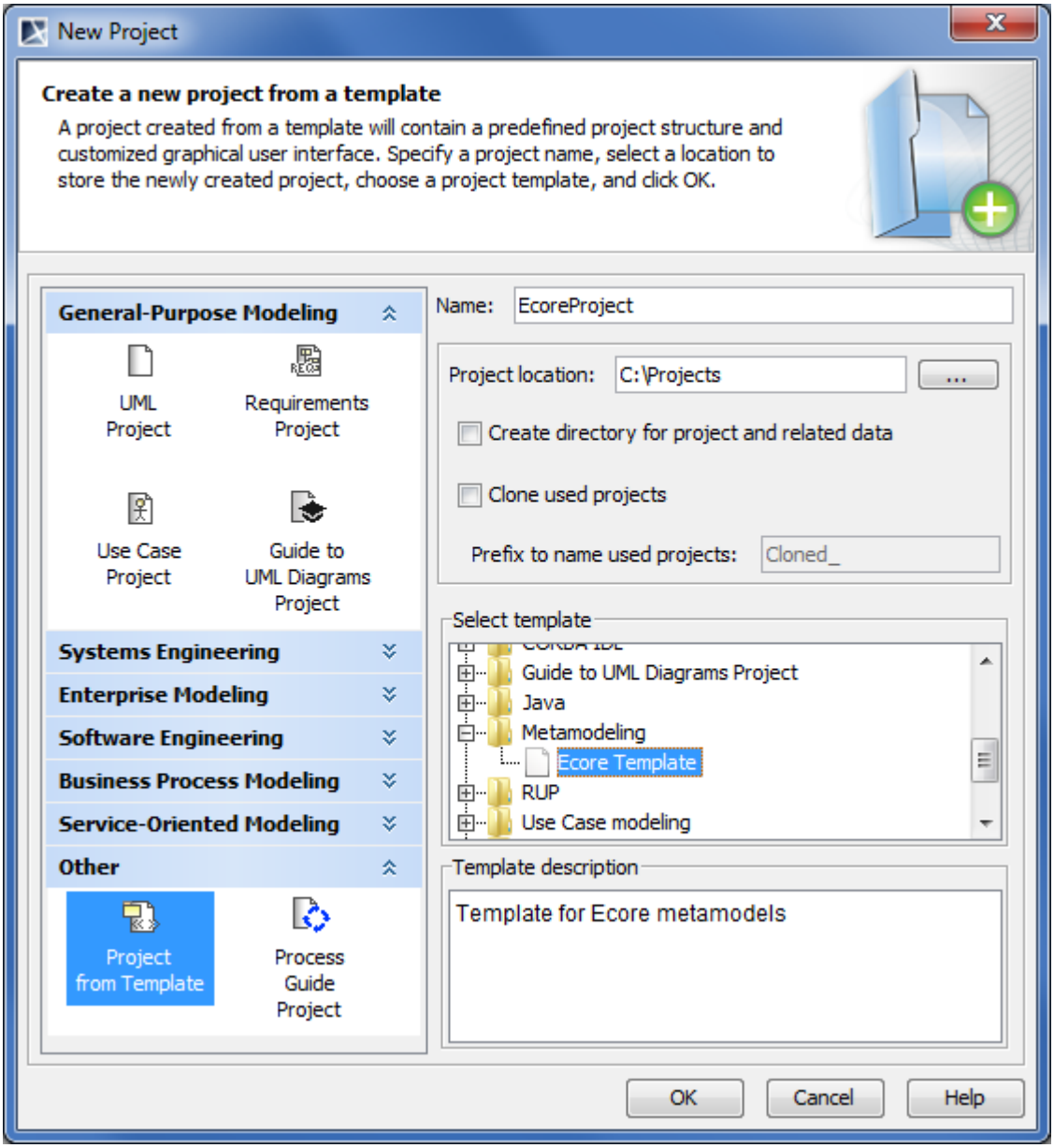
You can use the following ways to create a project for Ecore modeling:

To create a new project from the Ecore template

---

1. From the main menu, select **File** > **New Project**. The **New Project** dialog opens.
2. In the **Other** domain, select **Project from Template**.
3. Type a project name.
4. Specify the project location.
5. Under **Select template**, expand **Metamodeling** and then select **Ecore Template**.
6. Click **OK** when you are done.

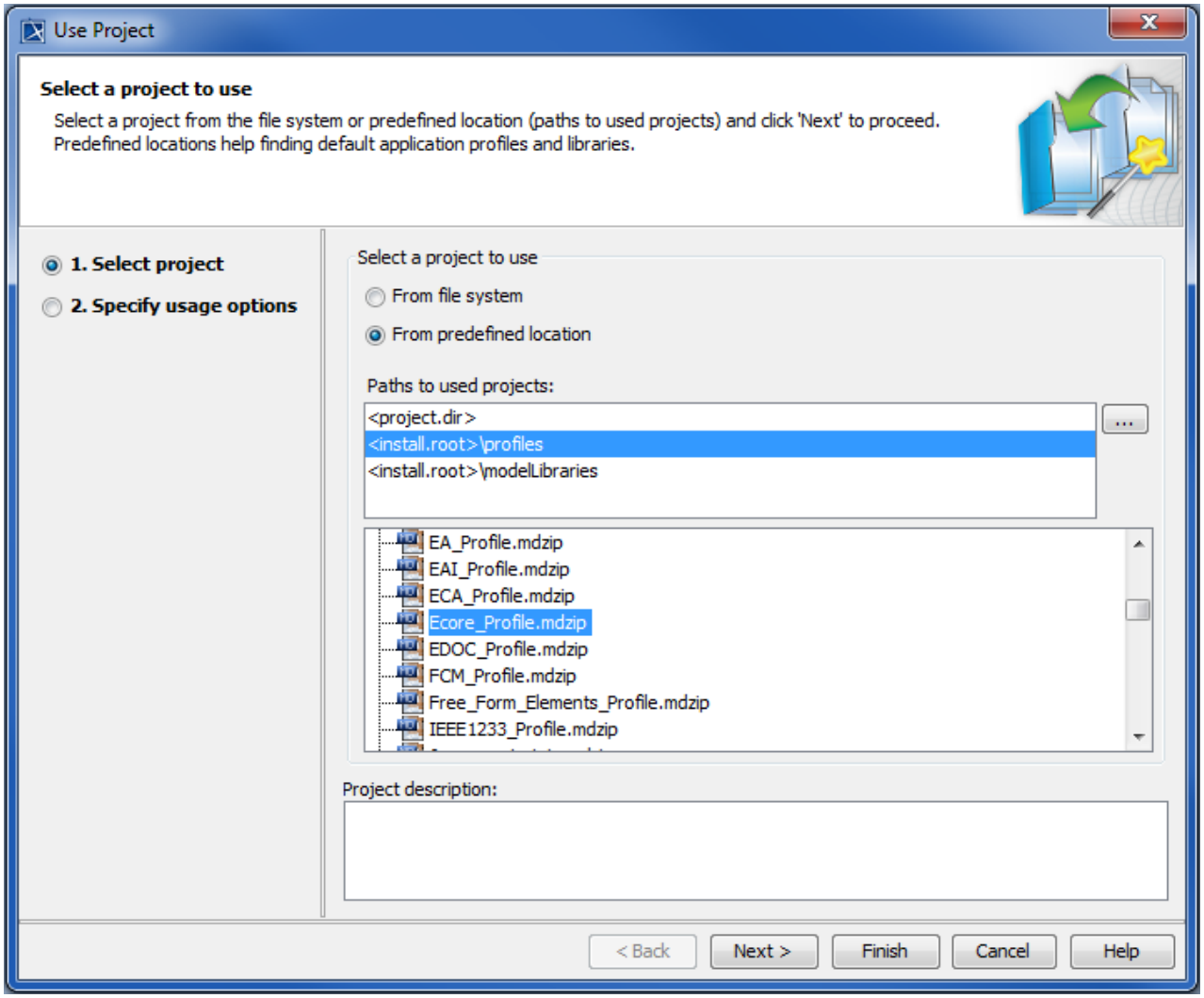
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



To apply the Ecore profile to a project

The Ecore profile can be applied only to existing UML or CMOF / EMOF projects.

1. From the main menu, select **File** > **Use Project...** The **Use Project** wizard opens.
2. Under **Select a project to use**, click **From predefined location**. The **Paths to used projects** list appears.
3. Select *<install root>\profiles*. The content of the *<install root>\profiles* folder is listed bellow.
4. Select *Ecore_Profile.xml* in the list.
5. Click **Next**, if you want to change used projects usage settings.
6. Click **Finish**.

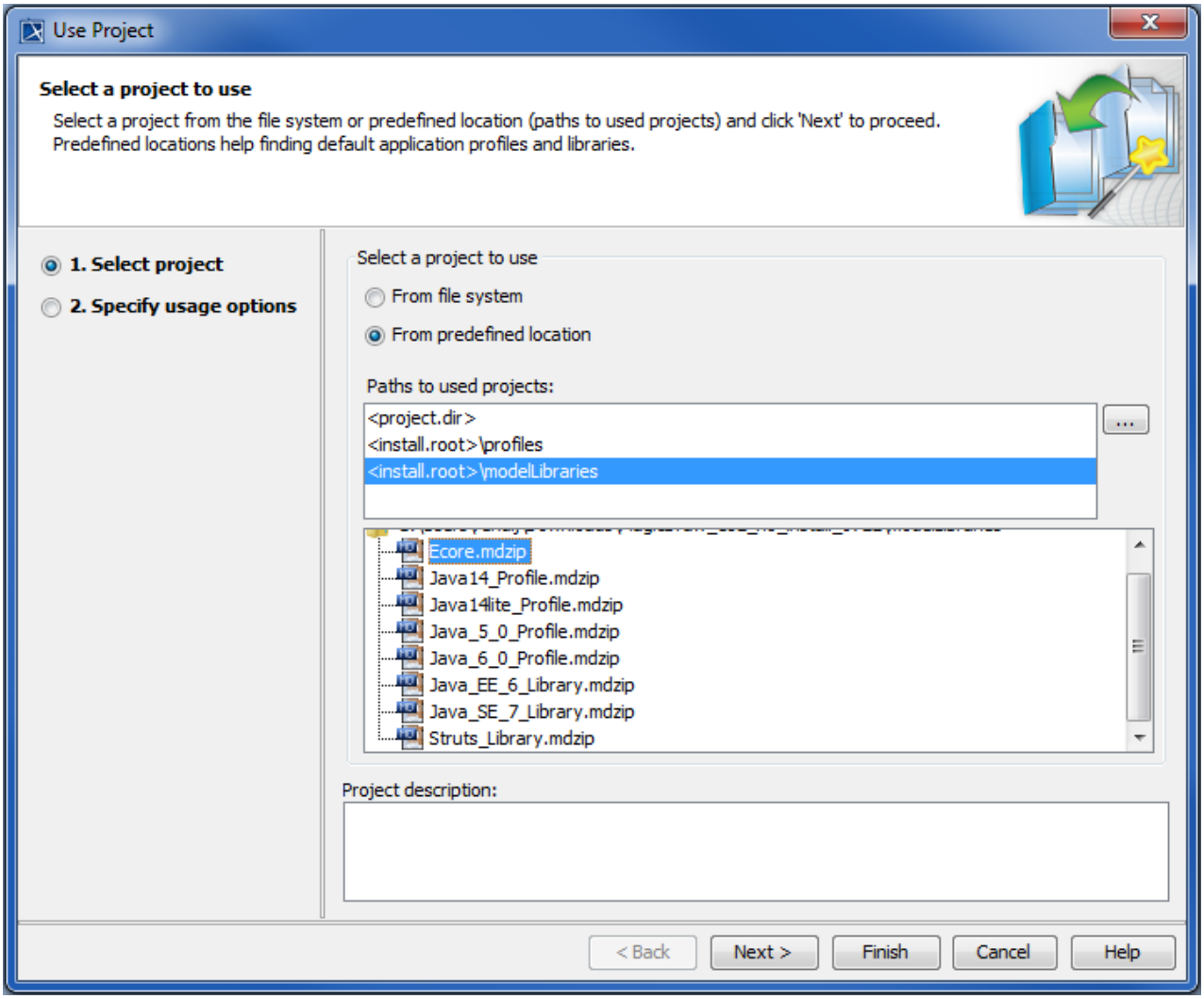


If your Ecore model references some standard Ecore elements (such as standard data types (for example, EShort) or standard metaclasses (for example, EStructuralFeature), you need to use the standard Ecore library in your project.

To use the standard Ecore library in a project

---

1. From the main menu, select **File** > **Use Project...** The **Use Project** wizard opens.
2. Under **Select a project to use**, click **From predefined location**. The **Paths to used projects** list appears.
3. Select *<installroot>\modelLibraries*. The content of the *<install root>\modelLibraries* folder is listed bellow.
4. Select *Ecore.mdzip* in the list.
5. Click **Next**, if you want to change used projects usage settings.
6. Click **Finish**.



Related pages

- Creating new projects from templates(see page 206)

Ecore modeling

Diagrams and elements in Ecore models

There are no specific diagrams for editing Ecore models in MagicDraw. You can use the same Class diagrams as you use for your UML models. Since Ecore is almost a subset of UML (with a few additions), familiar UML elements are used for the modeling. You can also develop Ecore models without using the Ecore profile at all. If your Ecore model uses only UML-specific information, you can develop it using plain UML and export it to Ecore without any problem.

Ecore is even more similar to EMOF. You can export the same model to both Ecore and EMOF.

Class, DataType, Enumeration, Package, Operation, Parameter have a direct one-to-one correspondence between UML and Ecore.

Ecore has two flavors of structural features, EAttribute and EReference, while UML has just one – Property. Fortunately a differentiation between an attribute and a reference is unambiguous and automatically resolved: the property, whose type is a data type is treated as EAttribute; the property, whose type is a class is treated as EReference. Hence the user does not need to worry about this - he/she can simply use properties.

There are no standalone Association and Generalization model elements in Ecore, but there is analogous information in Ecore: two EReferences, pointing to each other by their **opposite** property is equivalent to the association; the EClass::**eSuperTypes** property is equivalent to a generalization. Hence it is possible and meaningful to draw associations and generalizations in your model for exporting this information to Ecore.

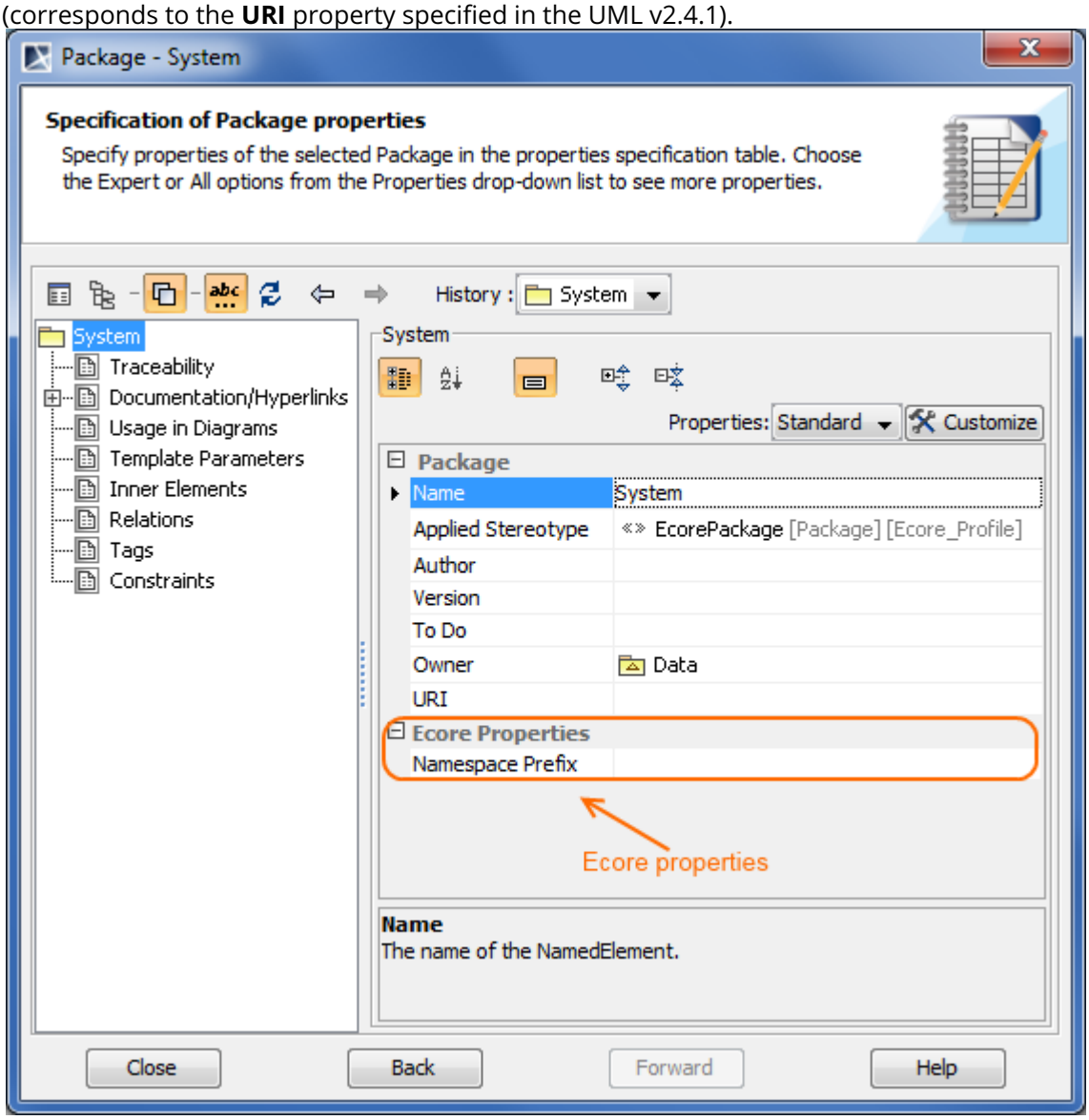
Ecore generics (templates) are also supported. You can use the UML template support to model Ecore generics. While the modeling is not trivial (and not one-to-one due to weak semantics of Ecore's EGenericType), it is possible to model all cases of template types, even ones with complexly nested type bounds like, for example, **SortedList<T extends Comparable<? super T>>**.
Your models can also contain any other UML elements, which are not present in Ecore. These elements are simply skipped during the export to Ecore. A warning is given about these elements.

Element properties in Ecore models

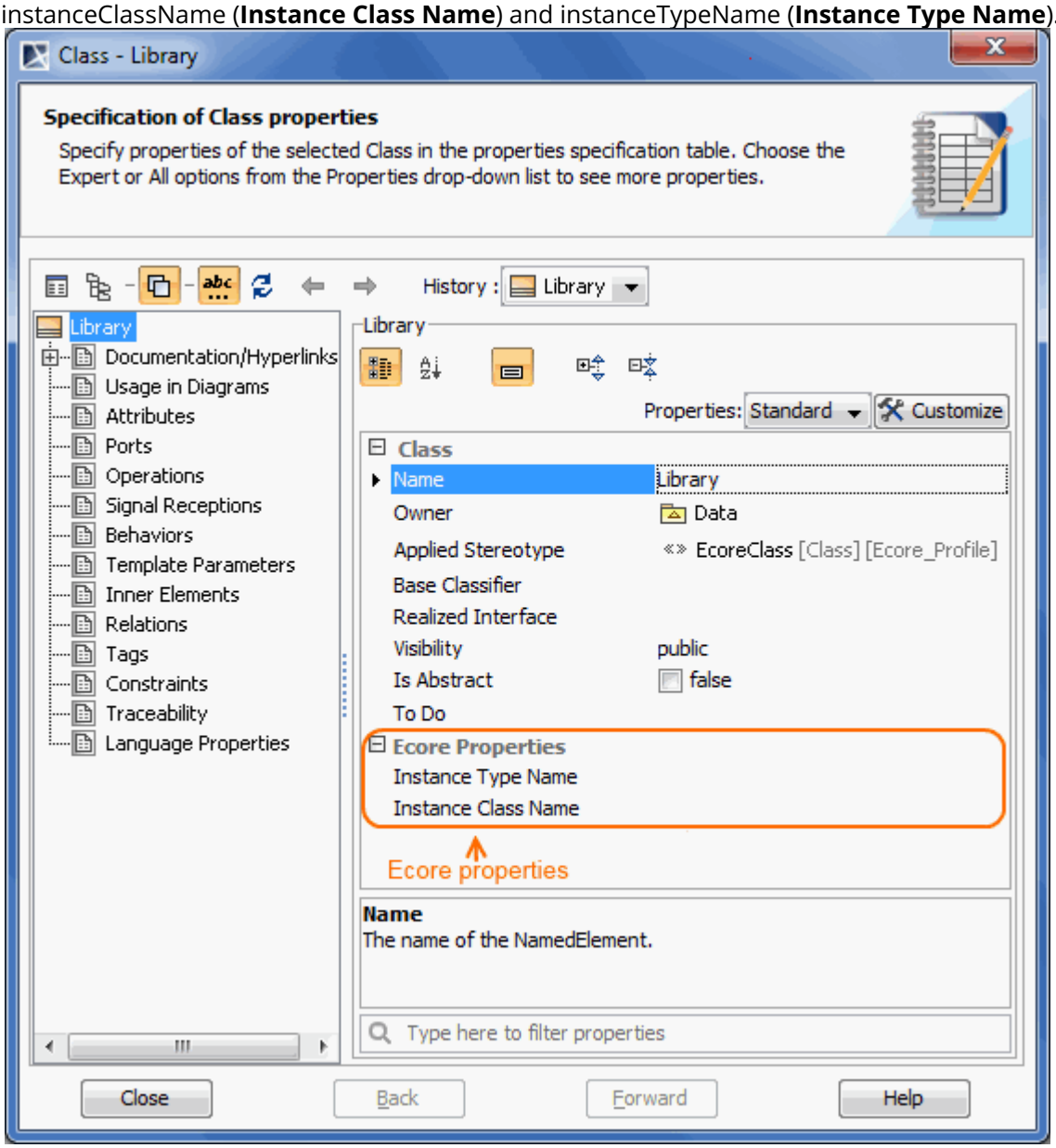
There are few Ecore-specific properties, which are brought in when the Ecore profile is used. These properties are used to capture Ecore specific information, not existing in UML. MOF-specific properties are also relevant for Ecore. These special properties are as follows:

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- The Ecore package has the additional properties: nsPrefix (**Namespace Prefix**) and nsURI (corresponds to the **URI** property specified in the UML v2.4.1).

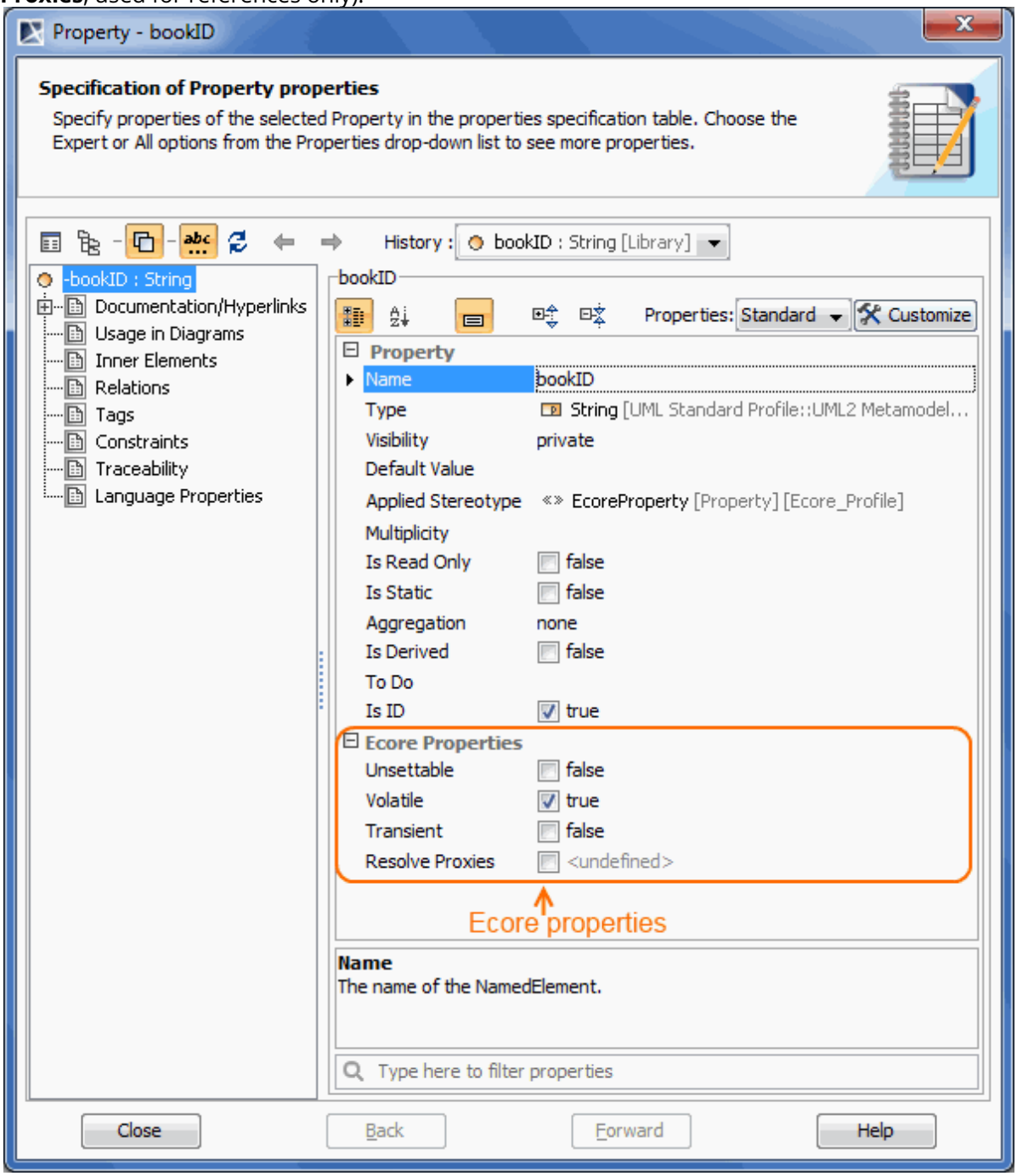


- The Ecore classifier (class, data type, enumeration) has the additional properties: instanceClassName (**Instance Class Name**) and instanceTypeName (**Instance Type Name**).



- Ecore attributes and references (modeled as the UML property) have the additional properties: volatile (**Volatile**), transient (**Transient**), unsettable (**Unsettable**), and resolveProxies (**Resolve**

Proxies, used for references only).



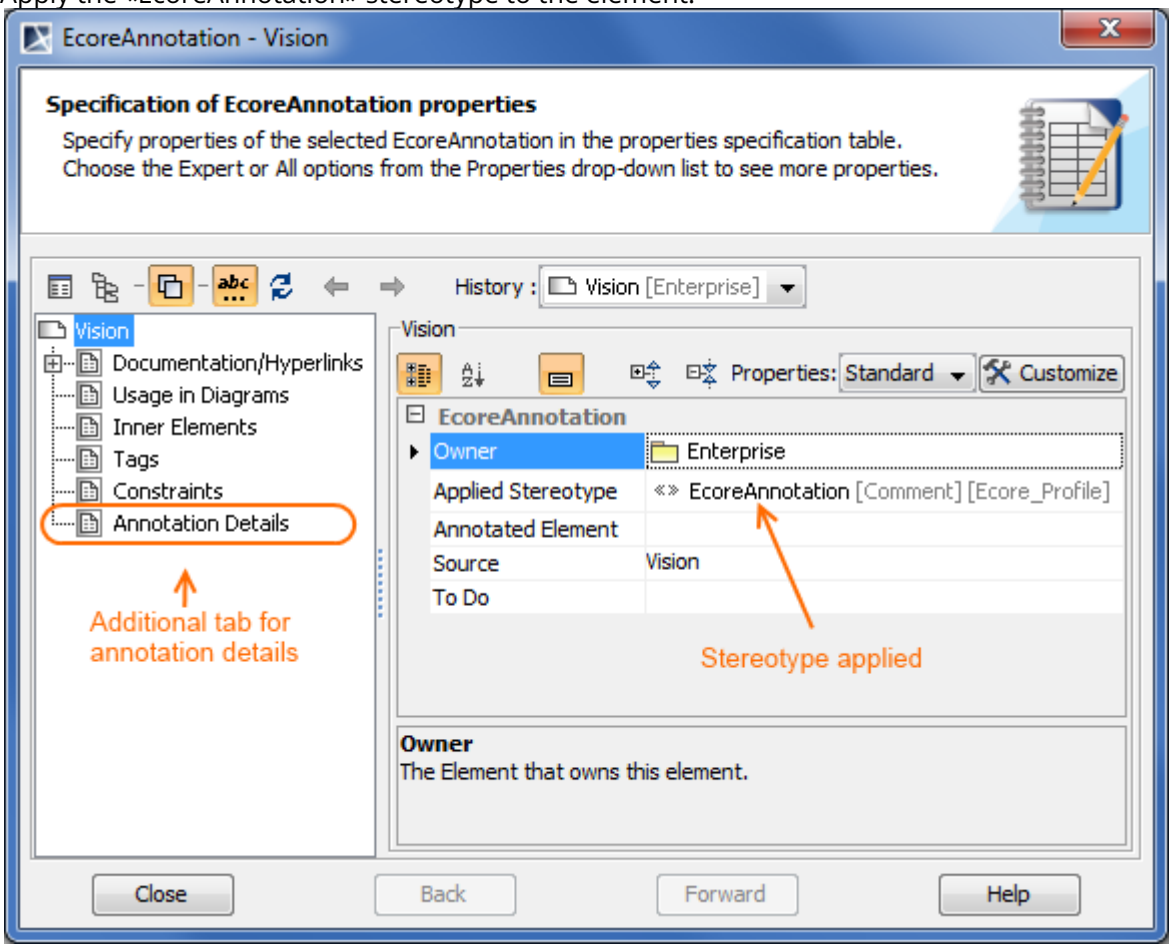
Annotation modeling

Ecore annotations are modeled as UML comments. For simple annotations no additional actions are necessary.

However, Ecore annotations have more powerful semantic than UML comments – they can have an internal substructure. In particular they can have an additional key-value map. For this additional information, there is a special «EcoreAnnotation» stereotype, that can be applied on an annotating comment. After applying the stereotype, the key-value map can be entered in a separate node of the annotating comment Specification window. Key-value pairs are stored as internal subcomment elements of the annotation.

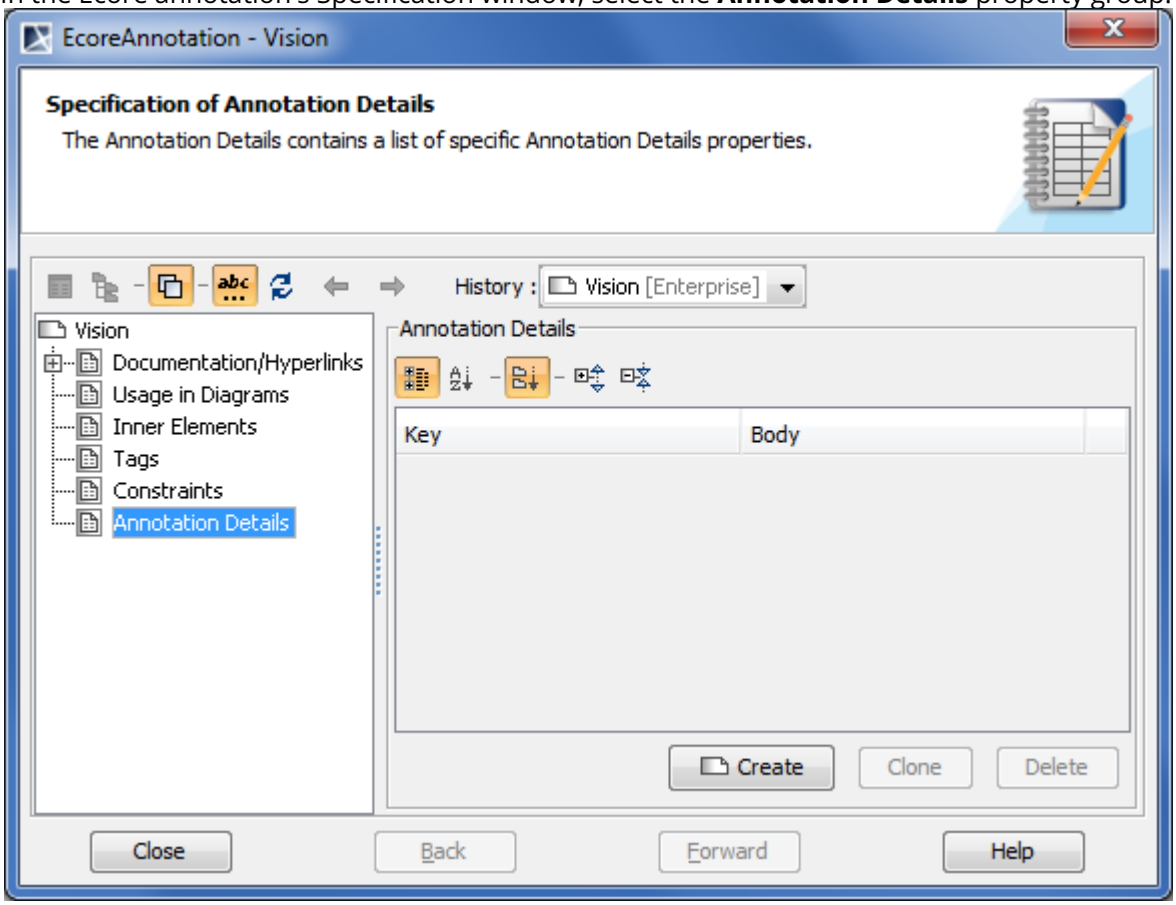
To create an Ecore annotation

1. In the Ecore model, create a Comment element.
2. Apply the «EcoreAnnotation» stereotype to the element.

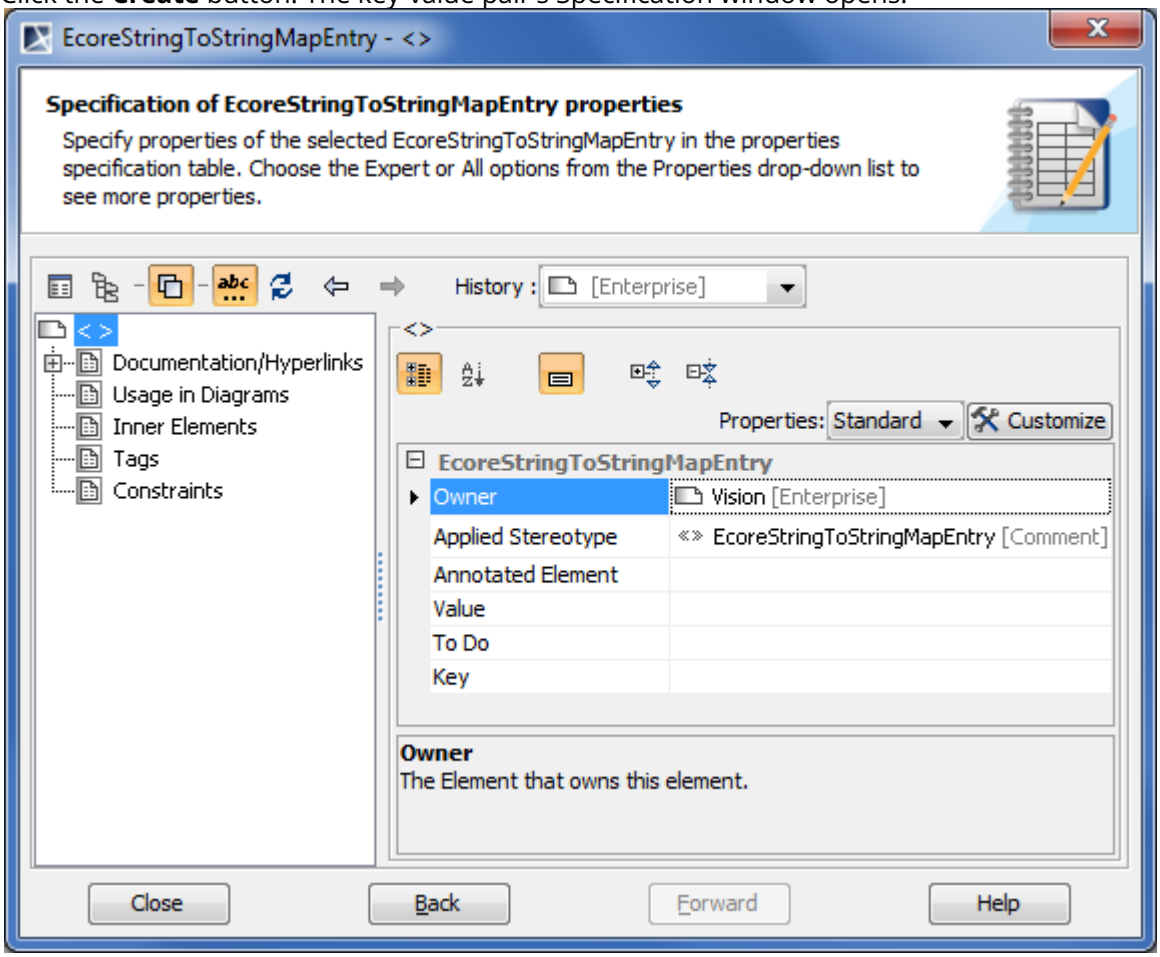


To create a key-value pair

1. In the Ecore annotation's Specification window, select the **Annotation Details** property group.

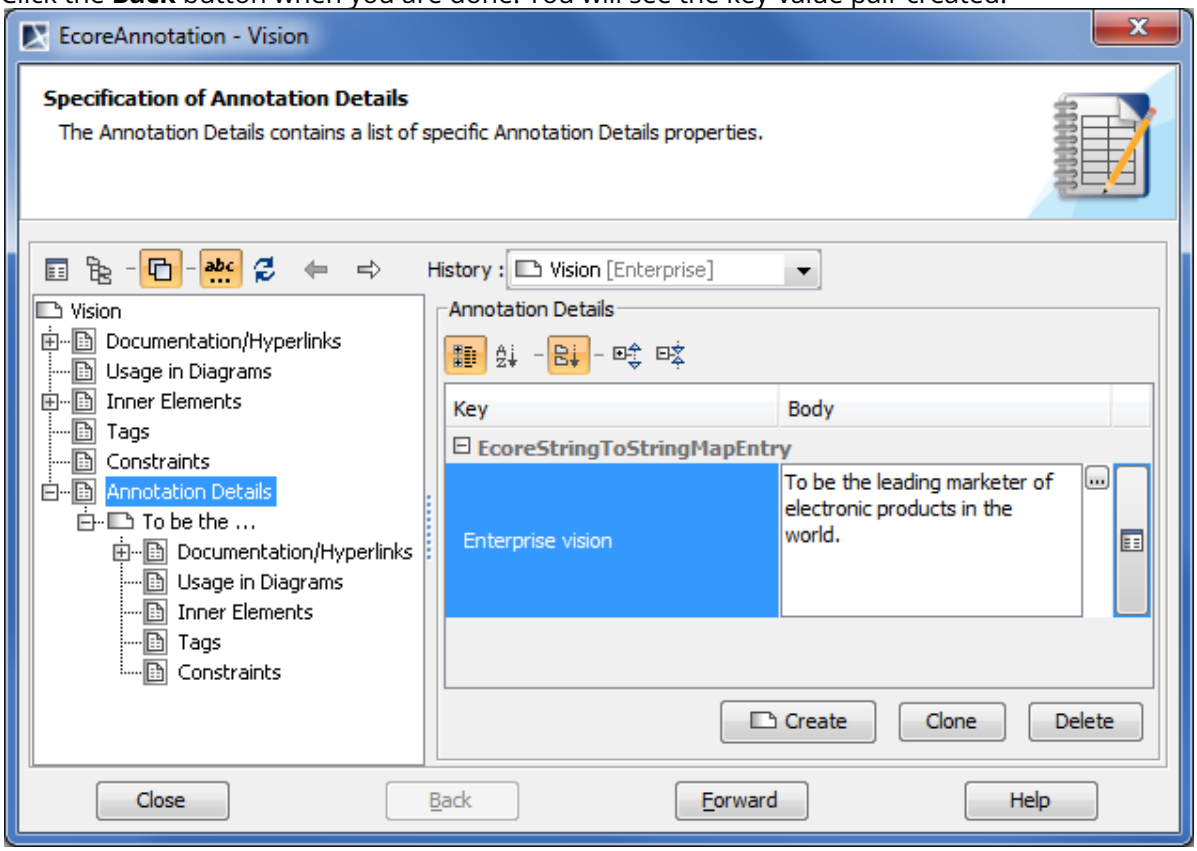


2. Click the **Create** button. The key-value pair's Specification window opens.



3. Enter values for both the **Value** and **Key** properties.

4. Click the **Back** button when you are done. You will see the key-value pair created.



Exporting projects to Ecore files

The Ecore model exporting is very similar to the EMOF / CMOF model exporting. After the Ecore model is created, you can export either the whole project or selected packages to an .ecore file.

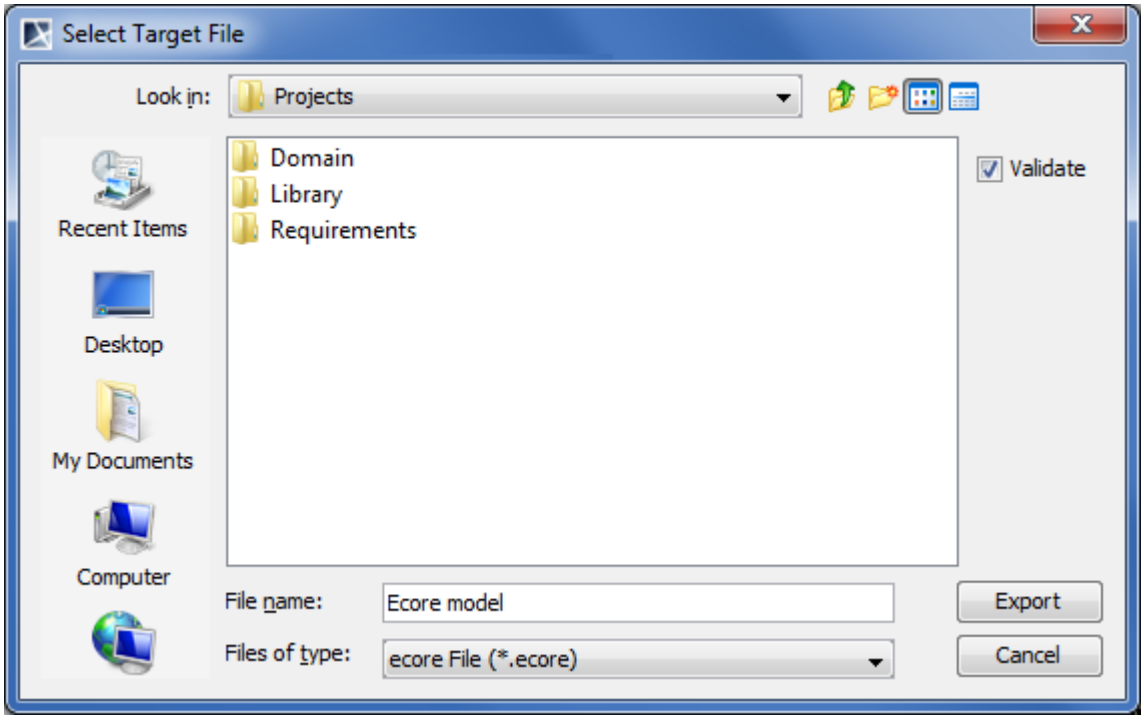
Exporting simple projects

To export a project to an Ecore file

---

1. From the main menu, select **File** > **Export To** > **EMF Ecore File** > **Ecore Whole Model**. The **Select Target File** dialog opens (see the following figure).
2. Type a file name and select a location for the exported model.
3. Click to select or to clear the **Validate** check box.
4. Click the **Export** button when you are finished.

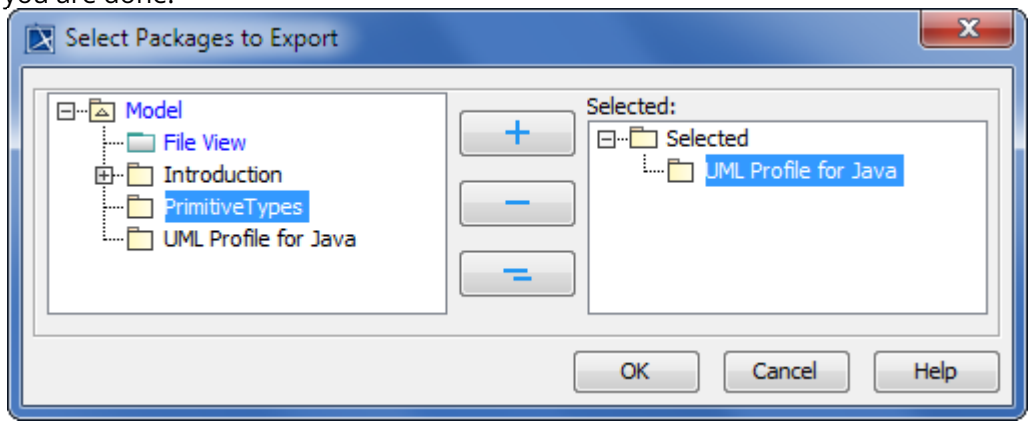
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



To export selected packages to Ecore file

---

1. From the main menu, select **File** > **Export To** > **EMF Ecore File** > **Ecore Selection**.
2. In the **Select Packages to Export** dialog, select packages you want to export and click **OK** when you are done.



3. Perform the procedure **To export a project to an Ecore file** starting from the **step #2**.

Exporting projects containing used projects

It is important to notice that any used projects in a MagicDraw project are not exported together with the project on its export to an Ecore file. Only references to these used projects are exported. Therefore the output of the project export is an Ecore file (model) containing the direct content of the MagicDraw project and referencing to other Ecore files (used projects) that are used in the MagicDraw project. For referencing to used projects, i.e., other Ecore files, Ecore references are used.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Each used project used by the project must be exported to an Ecore file individually. For this you have to open each used project as a project and then export it to an Ecore file.

It is strongly recommended to read the following paragraph before your very first attempt to export a MagicDraw project that uses used projects. Getting familiar with this information may help you to escape a serious problem that arises because of the nature of Ecore references.

Ecore references, crossing a resource boundary (when the element in one file references the element in another file), *are qualified-name-based, but not id-based* as in case of CMOF, EMOF, or UML. Hence when exporting a project with references to elements in other projects (used projects), the export tool must know the full path of elements in the used project's Ecore file. This information cannot be determined from UML model without additional information. For this the following approach is adopted:

1. **Save the used project after the export.** After a project is exported to an Ecore file, the qualified names (paths) of the shared elements are recorded in special stereotypes («EcoreExportServiceInformation») / tags (ecoreExportPath) of the project's shared packages. The side effect is that the project is modified during the export. To preserve this information for later usage, you need to save the project after the export.

2. **Export used projects before exporting the main project.** When a project that references to elements in one or more used projects is exported to an Ecore file, the export tool needs to know these elements' paths, saved on the appropriate used project export to Ecore. If this information is missing (e.g., in case the project is being exported before used projects are exported), the export tool tries to guess the correct path of each element in the used project and gives warnings about this. That is why used projects should be exported to Ecore before exporting the main project that uses these used projects.

Exported data type mappings

Standard UML data types are exported as standard Ecore data types. The following table shows which Ecore data type corresponds to which UML data type.

UML data type	Ecore data type
String	EString
Boolean	EBoolean
Integer	EInt
UnlimitedNatural	EInt
Real	EReal

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Standard data types from the MagicDraw profile are exported as Ecore types. The following table shows which Ecore data type corresponds which UML data type.

Data types in MagicDraw profile	Ecore data type
boolean	EBoolean
byte	EByte
char	EChar
date	EDate
double	EDouble
float	EFloat
int	EInt
real	EReal
long	ELong
short	EShort

The void data type is exported as an absence of the type.

References to Ecore model elements (standard datatypes like EInt, metaclasses like EStructuralFeature), defined in the standard Ecore library are exported as standardized Ecore references to Ecore metamodel elements (the resource identifier part of the Href is *http://www.eclipse.org/emf/2002/Ecore*).

Ecore elements validation

MagicDraw provides a validation suite for validating a model that is being exported to an Ecore file. This suite contain batches of rules to check exported elements. Warnings about not exported elements (for example, diagrams, behavioral elements, or other) are displayed after the validation process is completed.

The validation process does not preclude the model from being exported. Unsuitable elements are simply skipped.

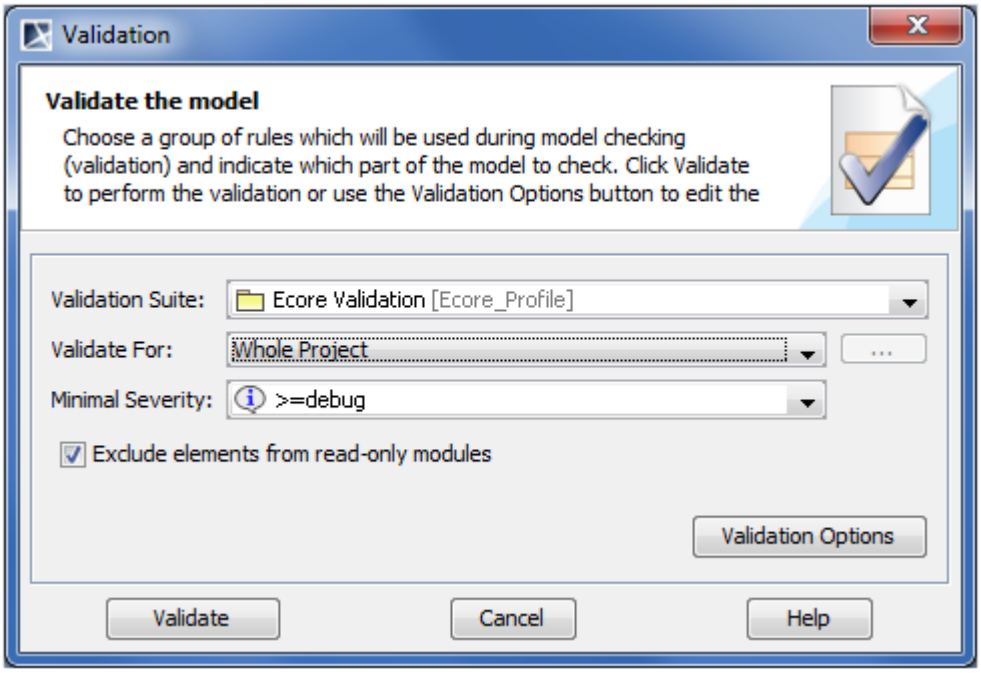
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

You can run the Ecore validation on a model (project or used project) export to Ecore. All UML elements that are not suitable for the Ecore, will be highlighted.

You can also run this validation suite at any time while you are developing an Ecore model.

To run the Ecore validation

1. From the main menu, select **Analyze** > **Validation** > **Validate**. The **Validation** dialog opens.



2. In the **Validation Suite** drop-down list, select **Ecore Validation**.
3. In the **Validate For** drop-down list, select the scope of the validation.
4. In the **Minimal Severity** drop-down list, select the level of severity.
5. Click **Validate**.

Importing projects from Ecore files

There is no possibility to import Ecore files directly. Ecore files should be imported using the EMOF import feature.

To import an Ecore file

1. In the Eclipse environment, open the *.ecore file and save it as *.emof.
2. Use the Importing projects from MOF files[64] procedure.

This way of importing Ecore files may cause some Ecore-specific model details that don't exist in EMOF to be lost.

64 http://documentation.nomagic.com/display/MD190SP4/Importing+projects+from+MOF+files

Working with standard profiles

All profiles and libraries that are bundled with MagicDraw products are considered as standard/system resources, non-modifiable and essential for the proper performance of the tool.

We highly recommend not to modify our provided standard profiles and libraries as it could cause problems on version updates, plugins, core malfunctions, and model corruptions.

Users will be warned on any intentional or unintentional attempt to modify profiles when:

- Opening profiles as projects.
- Using profiles in the read-write mode.
- Importing profiles into a project.
- Merging projects, and several other occasions.

Plugin and Profile Versions

Standard profiles are usually upgraded to support the latest versions of the specification of standards they represent (for example, SysML 1.2 or UML 2.4) in every MagicDraw release cycle. MagicDraw application, plugins code and behavior are modified accordingly to reflect these changes.

There is a high probability that the latest version of MagicDraw or plugins won't work with earlier or later profile versions and may cause unpredictable behavior or even model distortions. For example, SysML plugin 16.5 requires SysML Profile 16.5, as it could malfunction when using SysML Profile 16.0 or 16.6.

To protect users from such inconveniences, every MagicDraw project stores information about profiles or plugins versions that were used to create it and are required to load data correctly. Every standard profile has a version number. Normally, it's the same as MagicDraw (or plugin) version number.

MagicDraw requires using corresponding profile and software versions. If you use plugins or profiles that are obsolete, you will see a warning message. If you are missing certain plugins or profiles, Resource Manager with selected resources to install is launched automatically, so you can install missing resources in just few clicks.

If new versions of plugins are not purchased or you simply don't want to install it, but need to take a look at the project content, warnings may be ignored and such project may be loaded. In this case, proxy elements for missing profile elements will be created to retain missing references. Missing custom diagrams will be loaded as regular UML diagrams or will be restricted for review.

> Do not save such projects! Use them for preview only.

MagicDraw is backward compatible, therefore old projects will be loaded/converted without any warnings, if you have the newest versions of corresponding plugins and profiles.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Standard Profiles in Teamwork Server

Standard/system profiles are not added to Teamwork Server, because every user has recent versions installed locally. As profiles are non-modifiable, version control is not needed. It solves multiple profiles usage/modification/update issues for Teamwork and, at the same time, increases its performance, as standard profiles are not transferred via networks.

Updating UML Standard Profile due to migration to UML 2

> This information requires the urgent attention from users, who have migrated their old teamwork projects from 9.5 or earlier versions of Teamwork Server to any later version including 17.0 SP1.

It is highly recommended not to use standard/system modules (profiles and libraries) in the recent Teamwork Server version as they are not used beginning with the 16.6 version. Local standard modules are used instead of them.

If your teamwork projects still use old standard modules and it is a certain need to keep them in Teamwork Server, due to MagicDraw migration to UML 2, UML Standard Profile in MagicDraw 10.0 or later version must be updated manually. Please perform the following steps:

1. Login to Teamwork Server.
2. Open the UML Standard Profile as a teamwork project (on the main menu point to **Collaborate** > **Open Server Project**, in the opened dialog select the **Show Modules** check box, and choose the profile from the project list).
3. When the project is opened, lock it for edit recursively and import latest UML Standard Profile from *<MagicDraw installation directory>\profiles* folder (select **File** > **Import**).
4. A new UML 2 package is added to UML Standard Profile. Commit the profile to Teamwork Server.

Restoring a standard profile from the profiles backup folder

It is highly recommended not to use standard / system modules (profiles and libraries) in the recent Teamwork Server version as they are not used beginning with the version 16.6. Local standard modules are used instead of them.

If a standard profile was removed from Teamwork Server while upgrading the server and it is a certain need to keep the old standard profile, you may restore it from the profiles backup folder.

To restore a standard profile from the backup:

1. Start **Teamwork Administrator's Console**.
2. Click the **Projects** tab.
3. Click the **Import from Native Repository** button. The **Project Import Wizard** opens.
4. Click **Next**.
5. In the **Projects** list, select a profile you need to restore and click **Next**.
6. Select actions to resolve naming conflicts and click **Finish**.

Forward compatibility

The cross-version compatibility checking mechanism is implemented when a project with new feature usages is loaded with an older version of the modeling tool. Currently, the cross-version compatibility checking mechanism tracks feature usages of 19.0 SP1,19.0 SP2, and 19.0 SP3 versions that result in triggering a **Missing features detected** notification in earlier versions. The following table shows tracked features, the version they appear in, and whether the project can be committed/saved without any consequences.

Feature name	Version	Secure to commit/save
Hiding used projects(see page 1457)	19.0 SP1	No
Grant package permissions for the user group(see page 1400)	19.0 SP1	No
Pre-commit validation(see page 1341)	19.0 SP1	No
Custom row/column hierarchy in the dependency matrix(see page 340)	19.0 SP2	Yes
Contextual relationships in the dependency matrix[65]	19.0 SP3	Yes
Displaying classifiers on internal structure and behavior diagrams[66]	19.0 SP3	Yes
Custom hierarchy in tables(see page 431)	19.0 SP3	Yes
Lock-free Model Editing(see page 1409)	19.0 SP3	No

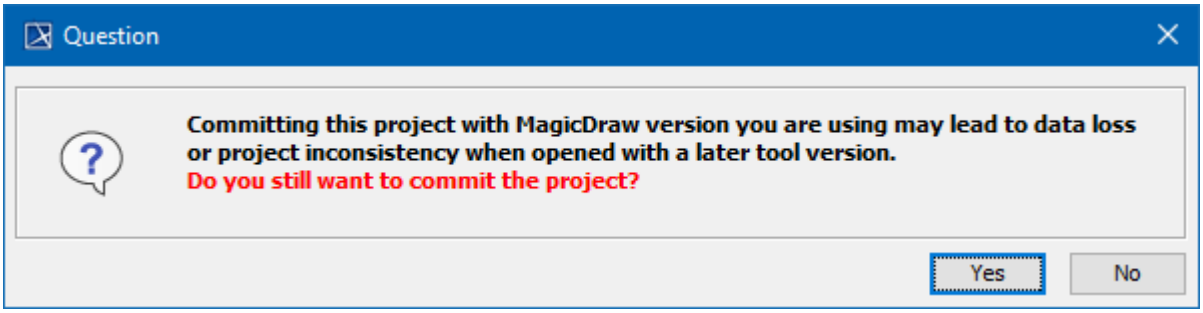
Important
Features required, but missing, in an earlier tool version (for example 19.0) are memorized by the cross-version compatibility checking mechanism. This process will continue with each SP release (i.e., SP2 might have features that are missing in both SP1 and 19.0 GA).

If you are working with features that are mentioned in the table above with status that it can cause inconsistencies after save/commit, this mechanism will track the missing features and show a warning message indicating possible consequences if you proceed. When you attempt to save or commit an

65 https://docs.nomagic.com/display/SYSMLP190SP4/Creating+contextual+relationships#Creatingcontextualrelationships-ContextualRelationshipsinDependencyMatrices
66 https://docs.nomagic.com/display/SYSMLP190SP4/Requirements+gap+and+coverage+analysis

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

opened project with a tool version that is missing features, this mechanism will track the missing features and show a warning message indicating possible consequences if you proceed.

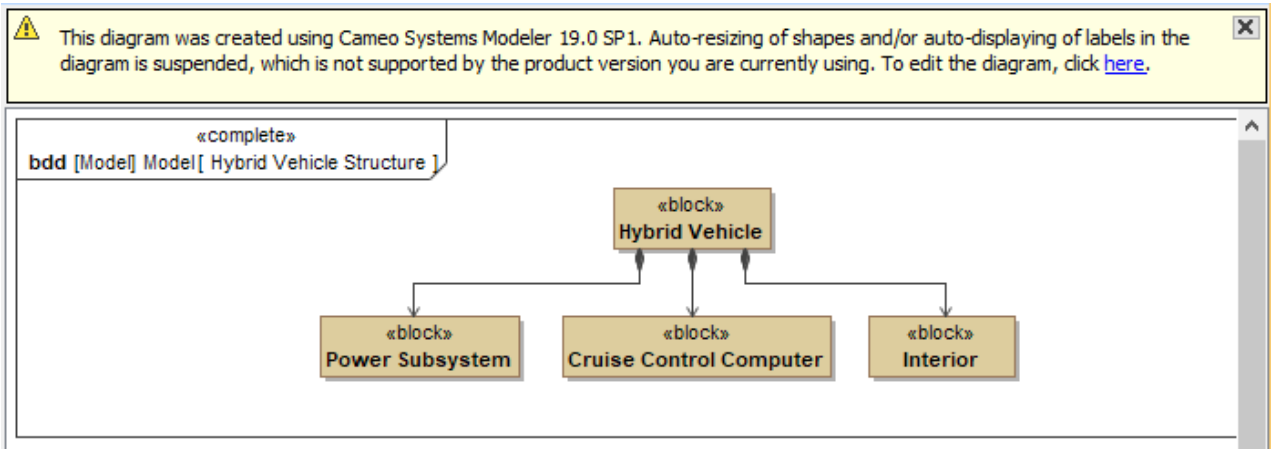


Diagramming features

Forward compatibility tracks the following diagramming features introduced in 19.0 SP1:

- Complete diagrams(see page 622)
- Vertical labels(see page 582)

When opening a project with tool version 19.0 (or 19.0 SP1 when the project was created with modeling tool 19.0 SP2 version), the usage of these features makes the diagram read-only. Additionally, you will be warned of possible consequences if you edit the diagram:



This mechanism prevents unexpected changes after opening a project with version 19.0 SP1.

CATIA Systems Synthesis

An integrated MDZipX plugin helps to improve the display of diagrams on the 3DEXPERIENCE platform by converting them to the MDZipX file.

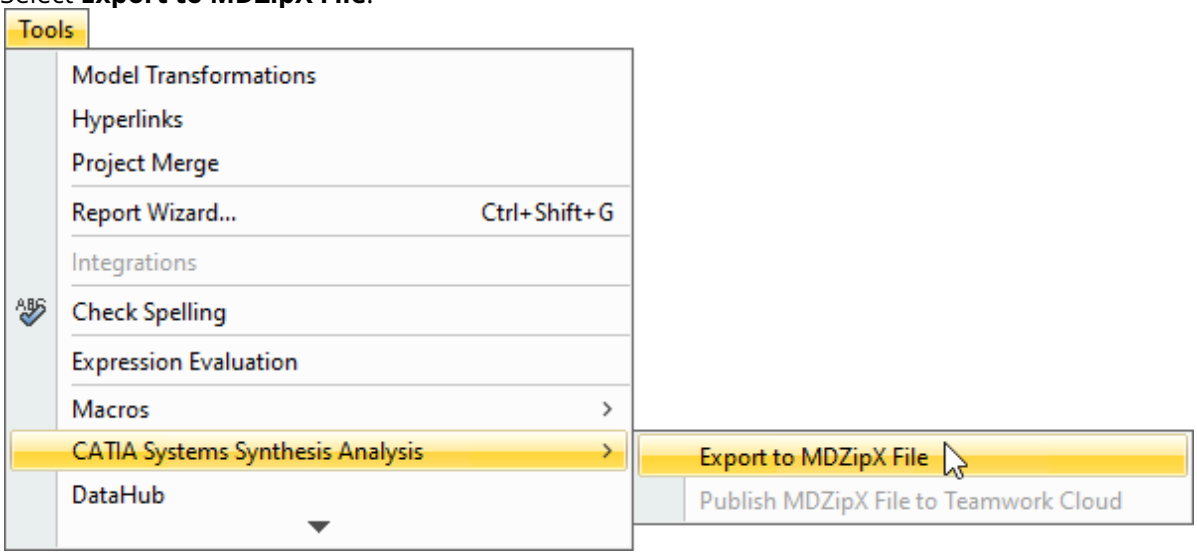
You can either generate the local .mdzipx file, or you can publish the .mdzipx file to Teamwork Cloud (TWC):

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Command	Description
Export to MDZipX File	The .mdzipx file contains: • the .mdzip file • the .svg file for each diagram
Publish MDZipX File to Teamwork Cloud	For the published TWC project, the .svg file is saved for each diagram of each revision.

To export to MDZipX

1. Click **Tools** > **CATIA Systems Synthesis**.
2. Select **Export to MDZipX File**.

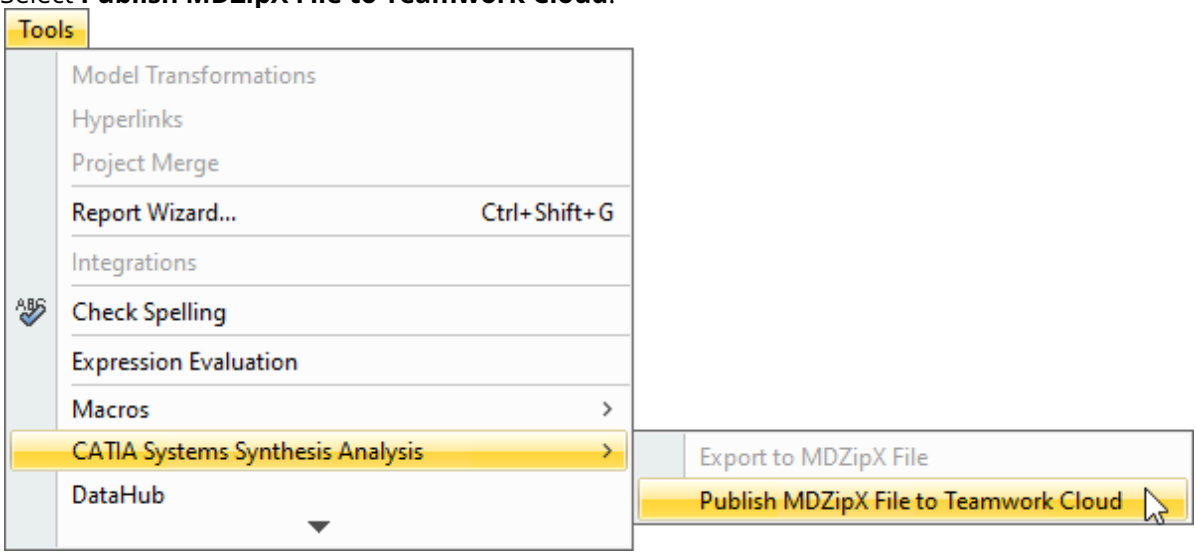


3. In the **Select target directory** dialog, specify the location for the MDZipX file on your file system and then click **Save**.

To publish MDzipX to Teamwork Cloud

1. Click **Tools** > **CATIA Systems Synthesis**.

2. Select **Publish MDZipX File to Teamwork Cloud**.



You can also enable the automatic publishing of the MDzipX file to Teamwork Cloud. If the option is specified, the MDzipX file will be automatically published for each diagram you have upon the project commit.

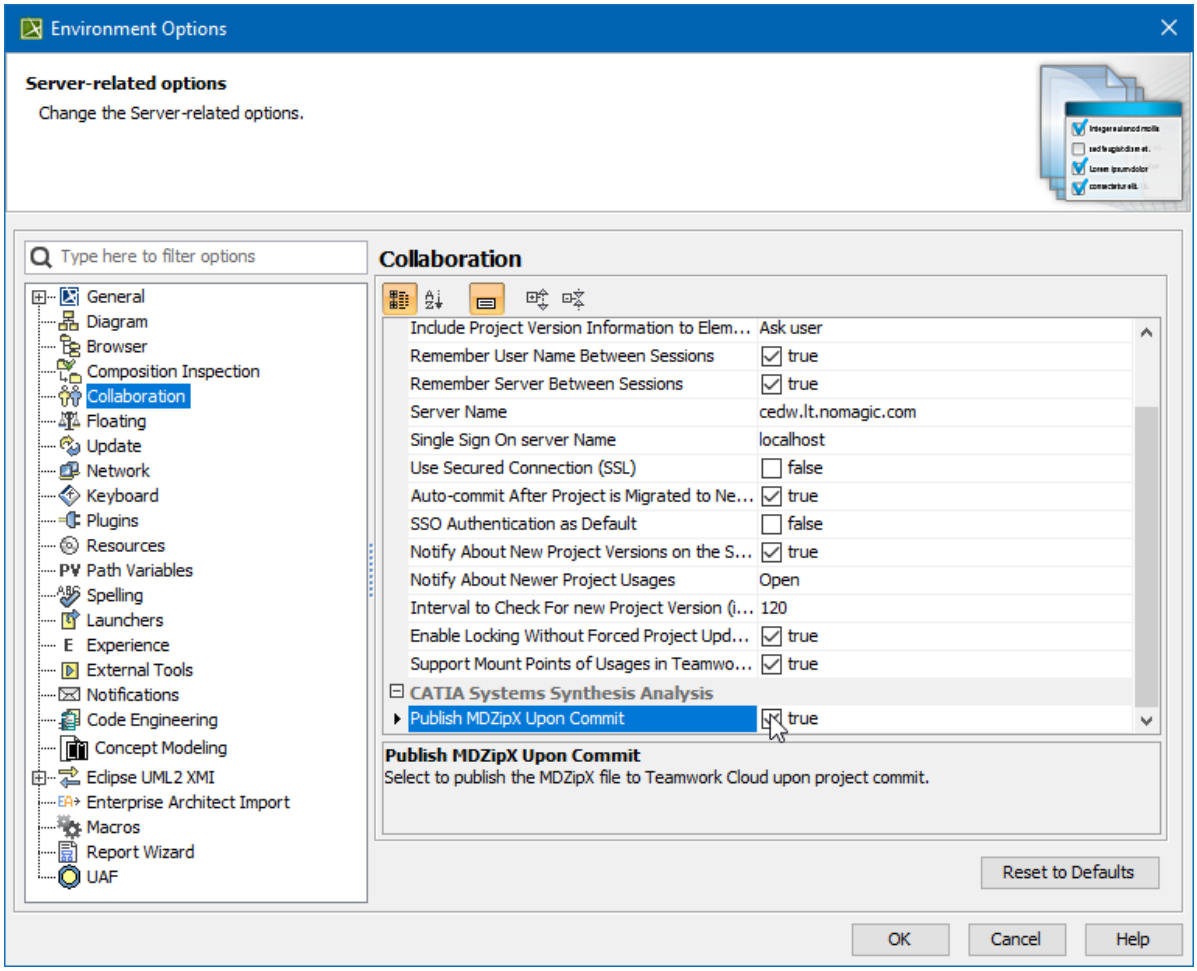
To enable the automatic MDzipX publishing

---

1. Click **Options** > **Environment**.
2. In the panel on the left, select **Collaboration**.
3. Under **CATIA Systems Synthesis**, select **Publish MDZipX Upon Commit**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

4. Click **OK**.



For more information, please refer to http://docservices/doc422/English/TraUserMap/smv-c-ad-NoMagicConfiguration.htm .

Diagrams

The following pages define the various types of diagrams and illustrate their uses in your projects.

UML diagrams

In software development, the diagram is the equivalent of a blueprint. To meet varying user requirements, we often need several different "blueprints" of the same system. Every system is described by many different aspects, such as:

- Functional (static structure and dynamic interactions)
- Nonfunctional (timing requirements, reliability, and deployment)
- Organizational (work organization and mapping to code modules)

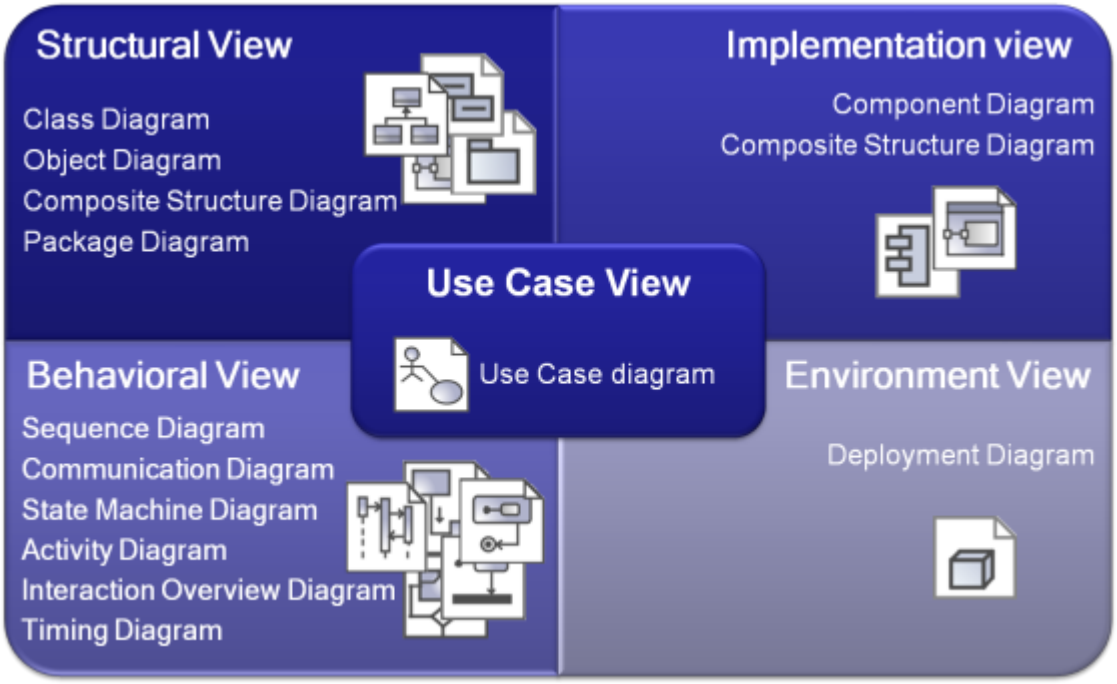
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

MagicDraw supports the following diagrams defined in UML 2:

(see page 297) Class diagram(see page 297)	(see page 300) Use Case diagram(see page 300)	(see page 304) Sequence diagram(see page 304)	(see page 311) Activity diagram(see page 311)
(see page 308) State Machine diagram(see page 308)	(see page 318) Component diagram(see page 318)	(see page 299) Object diagram(see page 299)	(see page 321) Package diagram(see page 321)
(see page 319) Deployment diagram(see page 319)	(see page 302) Communication diagram(see page 302)	(see page 310) Protocol State Machine diagram(see page 310)	(see page 299) Composite Structure diagram(see page 325)
	(see page 332) Interaction Overview diagram(see page 332)	(see page 324) Profile diagram(see page 324)	

Architectural views

UML defines 13 diagrams that describe 4+1 architectural views:



Several kinds of diagrams provide a visual notation for the concepts in each view.

Use Case view

The use case view represents the functionality and behavior of a system or subsystem as it is perceived by external users. This view is targeted mainly at customers, designers, developers, and testers.

The use case view usually is presented as a number of use cases(see page 876) and actors(see page 646) in Use Case diagrams(see page 300). Occasionally it is used in Activity(see page 311) and Sequence diagrams(see page 304).

The use case view is central because the contents drive the development of the other views. It is also used for project planning. Every single use case unit is deemed as a manageable unit during the project execution.

Structural view

The structural view represents structural elements for implementing a solution for defined requirements. It identifies all of the business entities and how these entities are related to each other. Usually entities are represented as classifiers and their instances in Class(see page 297) and Object diagrams(see page 299) in multiple abstraction levels. System decomposition to different layers can be displayed using Package diagrams(see page 321). A Composite Structure diagram(see page 325) can be used to represent the classifier inner structure. The system structural view artifacts are created by software architects and represent the system implementation design solutions.

Behavioral view

The dynamic behavior of the system is displayed on the Interaction (Sequence(see page 304) and Communication(see page 302)), State(see page 308), Activity(see page 311), Interaction Overview(see page 332), and Time(see page 403) diagrams. It focuses mainly on the interactions that occur between objects inside a system, activities and work performed by the various parts of a system, and state

changes within a particular object or collaboration. Rather than defining the participants of the system, it defines how particular use cases are executed, which provides value for the external user. The dynamic view is concerned about what is happening inside the system and how those actions impact other participants.

Implementation view

The implementation view describes the implementation artifacts of logical subsystems defined in the structural view. It can include the intermediate artifacts used in a system construction (code files, libraries, data files, etc.) This view defines dependencies between the implementation components and their connections by the required and provided interfaces. Components and their relationships are displayed on the Component diagram(see page 318). Inner parts of the component can be represented with the Composite Structure diagrams(see page 325). The implementation view helps analyze system parts and their dependencies in a higher component level.

Environment view

The environment view represents the physical arrangement of a system, such as computers and devices (nodes) and how they are connected to each other. In contrast to the component view, the deployment view is concerned with the physical structure of the system and the location of the software modules (components) manifested by artifacts within the system.

The environment view is displayed on the Deployment diagram(see page 319).

Class diagram

Overview

A Class diagram is a graphic representation of the static structural model. It shows classes(see page 666) and interfaces(see page 741), along with their internal structure and relationships. The classes represent types of objects that are handled in a system. The instances of those types (objects) are instantiated only on the runtime and are represented by an object and the interaction diagrams. A Class diagram does not show temporal information, it describes only the classification.

Classes can be related to each other in a number of ways: associated (connected to each other), dependent (one class depends/uses another class), specialized (one class is a subtype of another class), or packaged (grouped together as a unit – package). A Class diagram does not express anything specific about the relationships of a given object, but it does abstractly describe the potential relationships of one object with other objects.

A system typically has a number of Class diagrams – not all the classes are inserted into a single Class diagram. A class can have multiple levels of meaning and participate in several Class diagrams. A Class diagram is the logical map of an existing or future source code.

Purpose

A class diagram describes the types of objects (e.g., people, materials, information, and behaviors) in the system and various kinds of static relationships that exist among them. Class diagrams also show

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

the attributes and operations of a class and the constraints that apply to the way objects are connected.

Usage

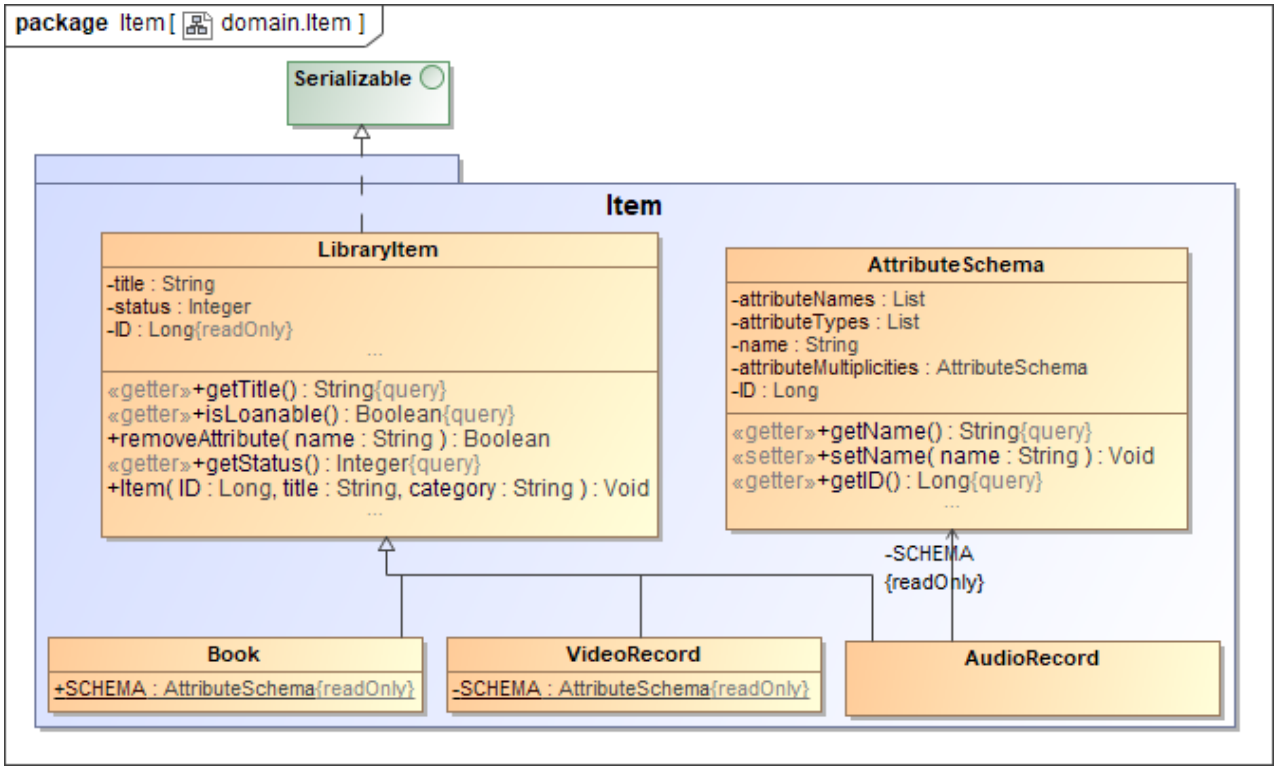
A class diagram can be used to:

- illustrate both simple and complex models for information systems
- understand the general overview of the schematics of an application
- visually express the specific needs of a system
- highlight the specific code that needs to be programmed and implemented to the structure
- provide an implementation-independent description of types used in a system
- share relevant information with stakeholders
- detect errors early in the lifecycle.

Summary:

Class diagrams are valuable because they:

- define the essential resources of the system
- define relationships between resources
- generate code
- model code (reverse engineering)
- provide a focus for all the other diagrams.



Example of a Class diagram

Relatged pages

- Creating diagrams(see page 483)
- Class(see page 666)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Object diagram

Overview
An Object diagram is an instance of a Class diagram that shows a complete or a partial view of the structure. This type of diagram displays instances of classifiers and links (instances of associations) between them.

Purpose

An Object diagram is usually used in the early phases of a project to model examples that show a possible snapshot of the system execution, for example, what the system can look like at some point in time. Additionally, it focuses on the attributes of a set of objects and helps to analyze how these objects relate to each other.

Usage

An Object diagram can be used to:

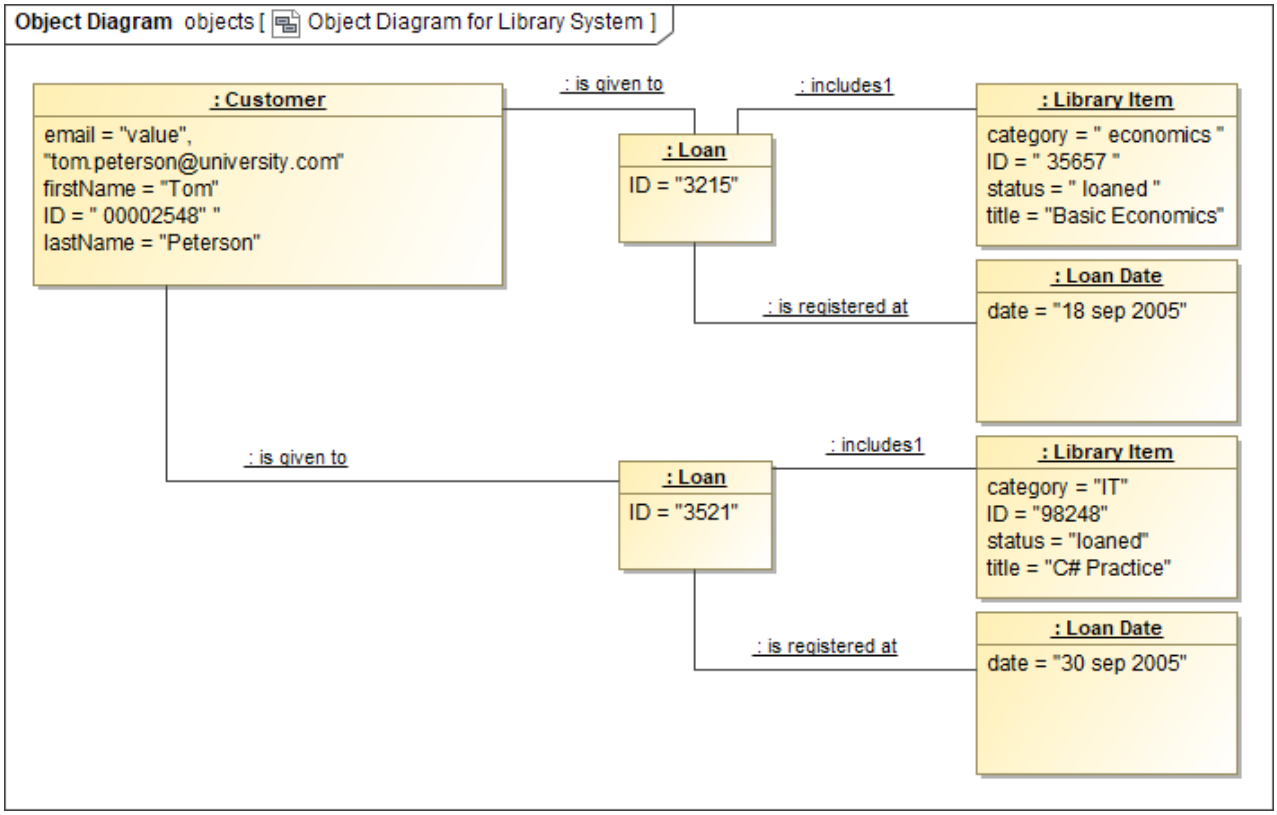
- represent a class instance
- describe the static aspect of the system
- understand the behavior of a particular object
- analyze object relations
- perform forward and reverse engineering on chosen systems.

Summary

Object diagrams are valuable because they:

- can be used to model test cases in order to validate the Class diagram
- support the investigation of requirements by modeling examples from the problem domain
- examine a specific iteration of a general system
- later in the project, model test cases to see whether the created Class diagram is correct and complete
- allows users to get a high-level overview of the system.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Example of an Object diagram

Related pages

- Creating diagrams(see page 483)

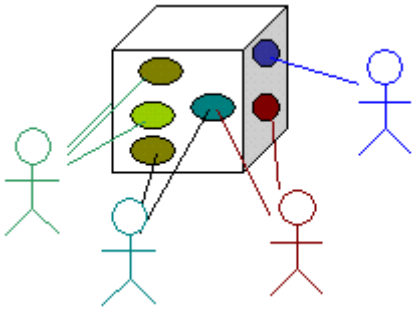
Use Case diagram

Overview

A Use Case diagram models the users' expectation for using the system. The people and systems that interact with the target system are called *actors*, whereas the features of the system used by actors are called *Use Cases*.

In general, a Use Case is a description of the functionality (a specific usage of a system) that a system provides. They can be written in many different ways, but the most common one is to represent a view of the system from outside the system. The Use Case descriptions can exist in a textual form (a simple table), where the Use Case diagram provides additional information about the relationship between the use cases and the external users. The Use Case diagram also allows a definition of the system's boundary.

The Use Cases(see page 876) are described only in terms of how they appear when viewed externally by the user (a system's behavior as the user perceives it), and do not describe how the functionality is provided inside the system. The Use Cases(see page 876) are not object-oriented, but they are included in the UML to simplify the approach of the project's lifecycle – from specification to implementation.



Purpose

A Use Case diagram defines the expectations of the external actors, for example, people, systems, or devices that need to interact with the system. Their interactions may be to provide input, to receive output, or to dialog with the system in order to cooperate in the completion of a task. All these interactions are focused using specific features of the system - *Use Cases*. It should be noted that Use Case diagrams do not provide any details about the implementation of these features. The goal of the Use Case approach is to focus the development effort on the essential objectives of the system without getting lost in, or driven by, particular implementations or practices.

Usage

Use Case diagrams, together with use case narratives and scenarios, define the goals of a system, or another classifier, such as an enterprise, subsystem, or component.

A Use Case diagram is ideal to:

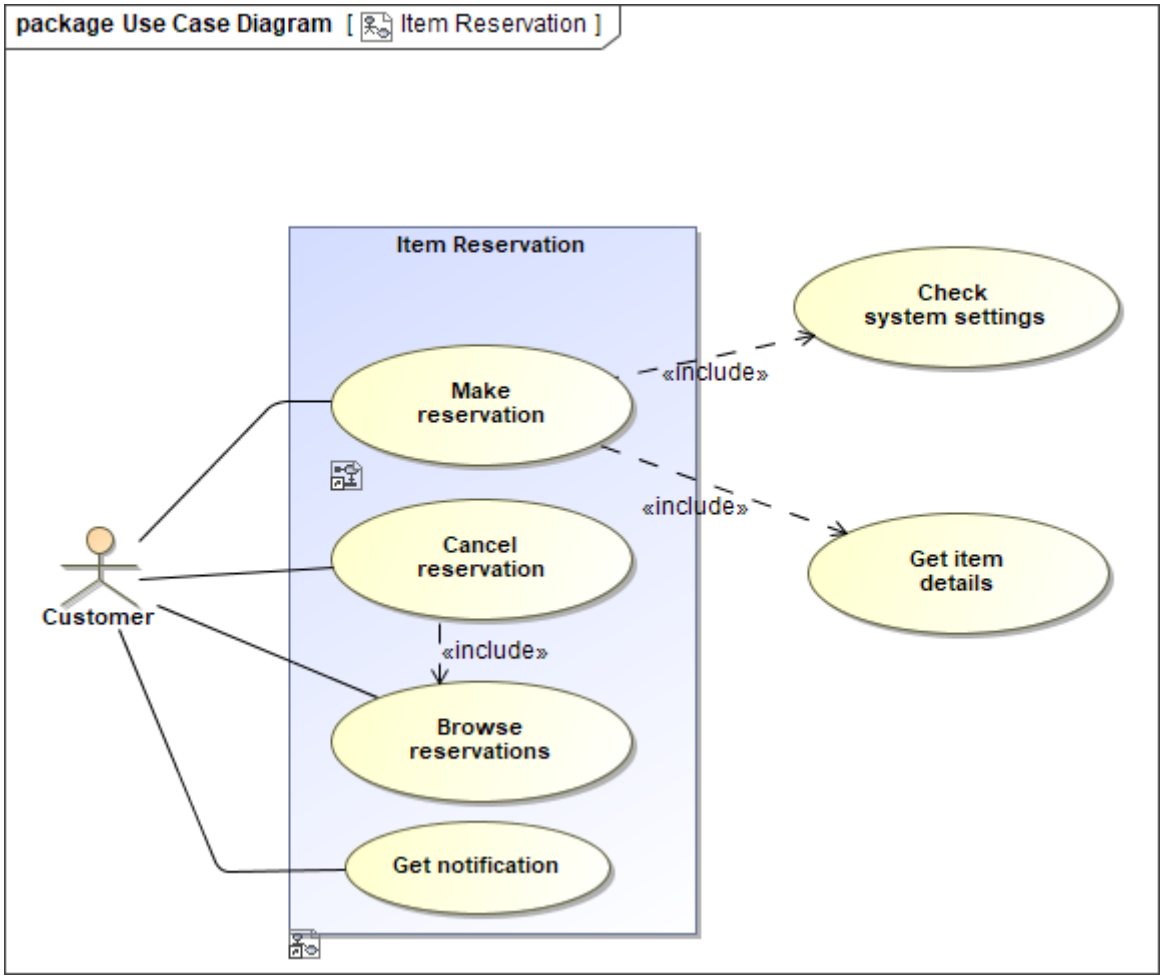
- represent the goals of system-user interactions
- define and organize functional requirements in a system
- identify internal and external factors that may influence the system and should be taken into consideration
- specify the context and requirements of a system
- model the basic flow of events in a Use Case.

Summary

Use Case diagrams are valuable because they:

- identify the client's expectations for the system
- identify specific features of the system
- identify shared behavior among system features
- allows clients to view their requirements in a simple and easy way.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Example of a Use Case diagram

Related pages

- Creating diagrams(see page 483)

Communication diagram

Overview

The Communication diagram is the UML 2.0 version of the Collaboration diagram. It illustrates various static connections between objects, models, and their interactions. It also presents a collaboration that contains a set of instances as well as their required relationships given in a particular context, and includes an interaction that defines a set of messages. These messages specify the interaction between the classifier roles within a collaboration that will serve to achieve the desired result.

A Communication diagram is given in two different forms: at the instance level or at the specification level.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Purpose

A Communication diagram can be used to model the interactions between objects or parts in terms of sequenced messages. It represents a combination of information taken from Class(see page 297), Sequence(see page 304), and Use Case(see page 300) diagrams describing both the static structure and dynamic behavior of a system.

Usage

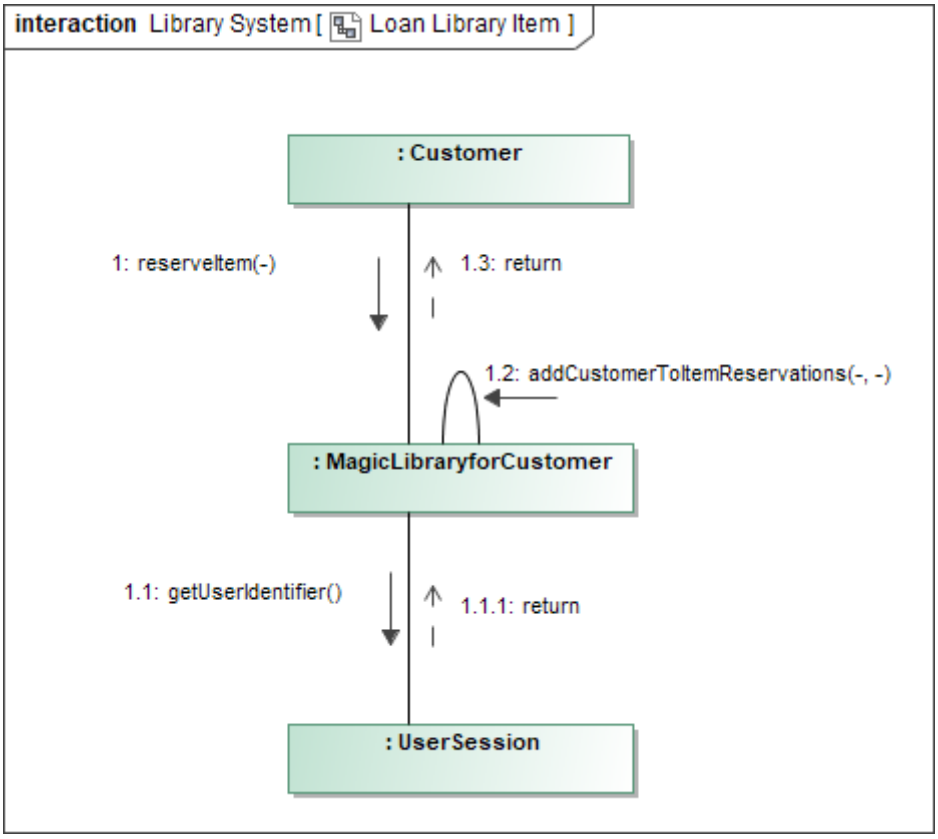
A Communication diagram can be used to:

- model message passing between objects or roles that deliver functionalities of Use Cases and operations
- capture interactions that show the messages passed between objects and roles
- model alternative scenarios within Use Cases or operations that involve the collaboration of different objects and interactions
- model mechanisms within the architectural design of the system
- identify objects, their attributes, and operations that participate in Use Cases.

Summary

Communication diagrams are valuable because they:

- reveal the structural requirements that are necessary to complete a task
- explicitly identify the objects that participate in an interaction
- reveal the interface requirements of the participating classes
- identify the structural changes required to satisfy an interaction
- identify data that passed as part of interactions.



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example of a Communication diagram

Related pages

- Creating diagrams(see page 483)

Sequence diagram

Overview

The Sequence diagram is a type of Interaction diagram that focuses on the Message (see page 756)interchange between various Lifelines(see page 746).

A Sequence diagram shows the interaction information with an emphasis on the time sequence. The diagram has two dimensions: the vertical axis representing time and the horizontal axis representing the participating objects. The time axis could be an actual reference point by placing the time labels as text boxes. The horizontal ordering of the objects is not significant to the operation, and you can rearrange them as necessary.

There are two Sequence Diagram modeling approaches:

- Activation bars as the lifelines.
- Activation bars as the Execution Specifications (new modeling approach).

Purpose

The purpose of the Sequence diagram is to represent some aspect of the communication between objects to perform a task. The main benefit of the diagram is that it helps to identify the messages exchanged between objects.

Usage

A Sequence diagram can be used to:

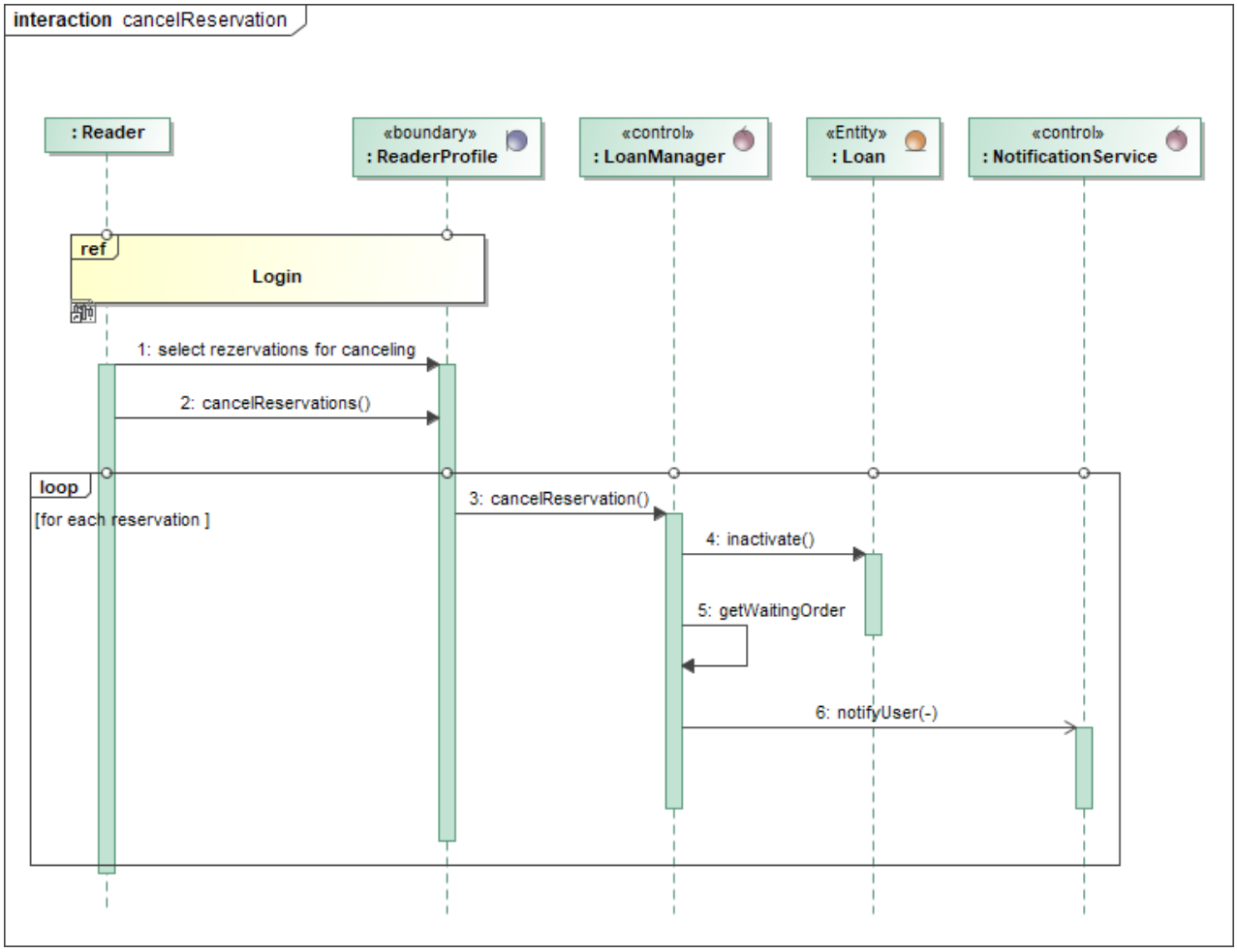
- represent the details of a UML Use Case
- model the logic of a sophisticated procedure, function, or operation
- analyze the interaction of objects and components to complete a process
- plan and understand the detailed functionality of an existing or future scenario.

Summary

Sequence diagrams are valuable because they:

- allow seeing the specific questions, commands, and data being communicated during the execution of a specific task
- identify the communication required to fulfill an interaction
- identify the objects that participate in an interaction
- identify the data that is passed as part of the interactions.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Example of a Sequence diagram

Related pages

- Working with Execution Specifications(see page 305)
- Creating diagrams(see page 483)
- Dragging in Sequence diagrams(see page 488)
- Combined Fragment(see page 717)

Working with Execution Specifications

On this page

- Setting behaviors or actions on the Execution Specifications(see page 306)
- Setting the Behavior Execution Specification behavior by adding an operation for the message(see page 307)
- Drawing the Time and Duration constraints(see page 307)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

The activation bars represent real model elements - Behavior and Action Execution Specifications. This benefits of using UML Execution Specifications are:

- Navigation. By double-clicking the activation bars (Behavior Execution Specifications) you can navigate from a sequence diagram to other behavior diagrams.
- Time and Duration Constraints can be attached to Behavior and Action Execution Specifications.

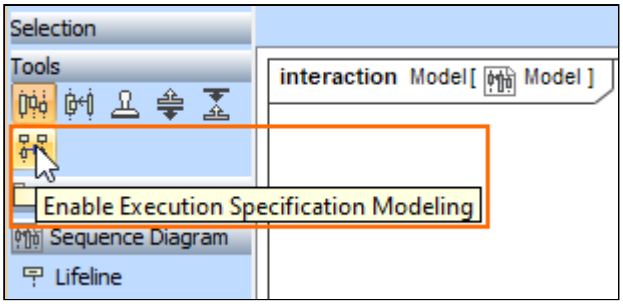
This method of modeling is optional. By default, the modeling approach where activation bars represent lifelines is used.

To switch to the new mode with activation bars representing UML Execution Specifications

---

- In the Sequence diagram palette, click the **Enable Execution Specification Modeling** button.

  This action enables the new modeling mode for each Sequence diagram individually.



- When you switch to this modeling approach, you **will not be able** to return to previous modeling method.
- If you open the Sequence Diagram with the activation bars representing UML Execution Specifications with the modeling tool version 19.0, such diagram will be read-only.

After you have enabled the Execution Specification Modeling, the activation bars in Sequence diagram represent:

- Behavior Execution Specification
- Action Execution Specification

You can refactor one Execution Specification to another at any time.

Setting behaviors or actions on the Execution Specifications

From the Containment tree, you can drag the Behavior or Action on the Execution Specification.

When you drag the behavior:

- If the activation bar represents Behavior Execution Specification, the dropped behavior is set for the **behavior** property of the Behavior Execution Specification.
- If the activation bar represents Action Execution Specification, the Action Execution Specification is converted to the Behavior Execution Specification and then the dropped behavior is set for the **behavior** property of the Behavior Execution Specification.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

When you drag the action:

- If the activation bar represents Action Execution Specification, the dropped action is set for the **action** property of the Action Execution Specification.
- If the activation bar represents Behavior Execution Specification, the Behavior Execution Specification is converted to the Action Execution Specification and then dropped action is set for the **action** property of the Action Execution Specification.

Once the behavior or action is set for the Execution Specification, you can easily navigate to the diagrams of the set behavior or action.

To navigate to the behavior diagram

- Double-click the Behavior Execution Specification.

To navigate to the activity diagram

- Double-click the Action Execution Specification.

Setting the Behavior Execution Specification behavior by adding an operation for the message

When you have an operation with a method, you can add that operation to the message and the method is set as a behavior of the Behavior Execution Specification at the end of the message.

To set an operation with a method

1. In the Sequence diagram, draw a message between Behavior Execution Specifications.
2. In the Containment tree, select an operation with a method and drag it to the message.
3. Open the Specification window of the Behavior Execution Specification which is at the end of the message. In the Behavior property cell, the operation method is set as a behavior.

- If you change the operation to a different one that has a different method, that method will overwrite the behavior of the Behavior Execution Specification.
- If you change the operation to a different one that has no method, the previously set method will remain as a behavior of the Behavior Execution Specification.

Drawing the Time and Duration constraints

When you draw new Time constraints in a Sequence diagram (from the diagram palette):

- If you click near the top of the activation bar, the constrainedElement will be set to the Execution Specification of that bar, and *firstEvent=true*.

- If you click near the bottom of the activation bar, the constrainedElement will be set to the Execution Specification of that bar, and *firstEvent=false*.

When you draw new Duration constraints in a Sequence diagram (from the diagram palette):

- If you click near the top of the activation bar, the constrainedElement[i] will be set to the Execution Specification of that bar, and *firstEvent[i]=true.*
- If you click near the bottom of the activation bar, the constrainedElement[i] will be set to the Execution Specification of that bar, and *firstEvent[i]=false*.

Related pages

- Creating diagrams(see page 483)

State Machine diagram

Overview

A State Machine diagram is a graph that represents a State Machine. States(see page 848) and various other types of vertices (pseudostates) in the State Machine graph are rendered by the appropriate State(see page 848) and Pseudo States(see page 843) symbols, while T(see page 872)ransitions(see page 872) are generally rendered by directed arcs that interconnect them. The states can also contain subdiagrams by a physical containment or tiling. Note that every State Machine has a top State containing all the other elements of the entire State Machine. The graphical rendering of this top state is optional.

State Machine. The State Machine is a specification of the sequence of states an object or an interaction undergoes in response to events during its life, combined with its responsive actions. The State Machine can represent the sequence of states of a particular collaboration, e.g. collection of objects, or even the whole system, which is also considered a collaboration. The abstraction of all possible states defined in a State Machine is similar to the way Class diagrams(see page 297) are abstracted: all possible object types (classes(see page 666)) of a particular system are described.

Purpose

A State Machine diagram (also called Statechart diagrams) represents the behavior of entities capable of dynamic behavior by specifying their response to the receipt of event instances. Typically, State Machine diagrams describe the behavior of Classes(see page 666), but the Statecharts can also describe the behavior of other model entities. such as Use Cases(see page 876), Actors(see page 646), Subsystems(see page 860), Operations(see page 677), or methods.

Usage

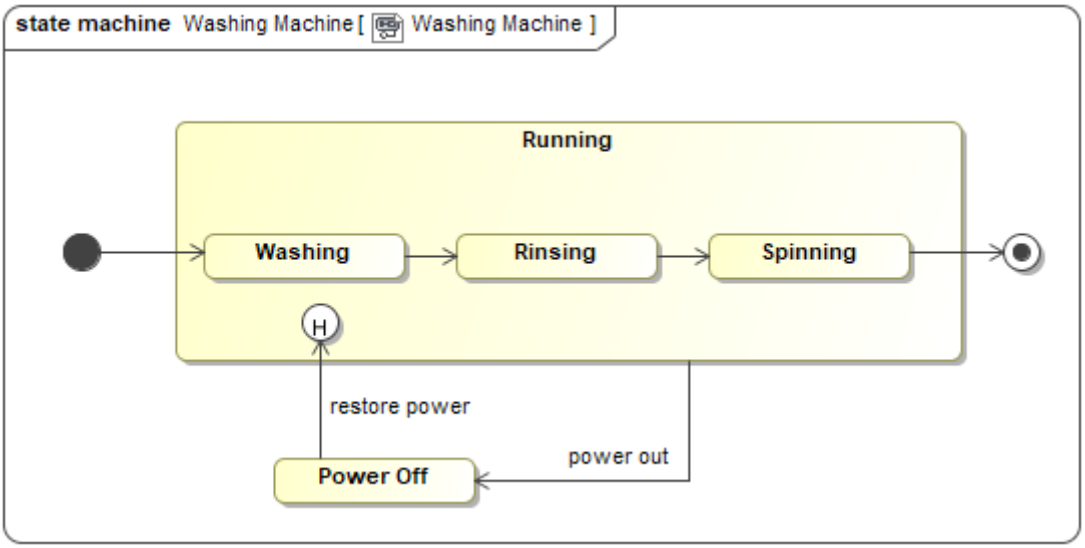
A State Machine diagram can be used to:

- describe how an object moves
- show the behavior of a State Machine or a related set of State Machines
- illustrate Use Case scenarios
- depict event-driven objects

Summary

State Machine diagrams are valuable because they:

- identify events an object will/will not respond to depending on the object's current state
- identify the specific responses of an object to everything that can happen to it
- discover and validate the data that is needed to define the state of the object and affected attributes
- help to discover the internal effects of behaviors.



Example of a State Machine diagram

Related pages

- Creating diagrams(see page 483)
- Dragging in State Machine diagrams(see page 487)

Working with State Machine diagram

Displaying inner elements

To display inner elements

- When creating a new State Machine diagram, right-click a State in the Containment tree, point to **Create Diagram**, and then click **State Machine Diagram**.

  > Inner elements are displayed on the State Machine diagram pane automatically upon creating the first diagram under the State element.

- When modifying a preexisting State Machine diagram, select a diagram pane or the State shape and perform one of the following steps:
  - From the shortcut menu, click **Display** > **Display Inner Elements**.
  - On the diagram toolbar, click

and select **Display Inner Elements**.

Working with behaviors

You can select Entry, Do Activity, Exit, deferrable triggers, and internal transitions directly in the compartment of the state shape. Thus, you can:

- Draw dependencies among particular behaviors on a state shape.
- Move behaviors(see page 487) from one state shape to another.
- Convert the selected behavior to another.

To convert a behavior to another behavior

---

1. On a state shape, select a behavior.
2. From the behavior's shortcut menu, select **Refactor** > **Convert To** > *<behavior>*

Protocol State Machine diagram

Overview

A Protocol State Machine diagram is, in general, a more narrowly defined State Machine diagram(see page 308). It is always defined in the context of a classifier. It specifies which Operations(see page 677) of the classifier can be called, in which State(see page 848), and under which condition, thus specifying the allowable call sequences on the classifier's operations.

The Protocol State Machine presents the possible and permitted Transitions(see page 872) on the instances of its context classifier, together with the Operations(see page 677) that carry the Transitions(see page 872).

In this manner, an instance lifecycle can be created for a classifier, by specifying the order in which the Operations(see page 677) can be activated and the States(see page 848) through which the instance progresses during its existence.

Purpose

The Protocol State Machine diagram is created for use with the Protocol State Machine and the Protocol Transitions. It limits the scope of the State Machine diagram to show only the transitions that trigger the state changes.

Usage

A Protocol State Machine diagram can be used to:

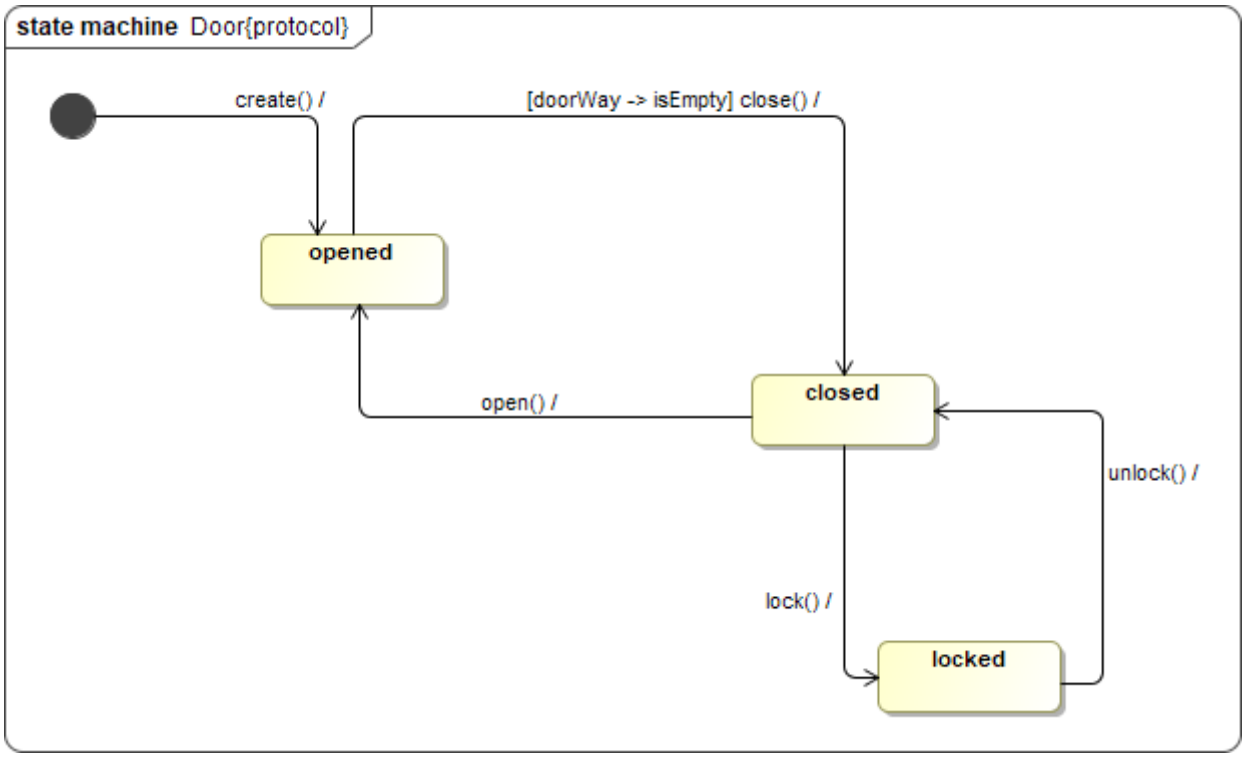
- show different stable class states of objects that could exist for some time
- explain how objects could change their states over time

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Summary

Protocol State Machine diagrams are valuable because they:

- place the focus on what it takes to legally change an object
- reveal conditions that ensure the integrity of the object while it is being manipulated by other objects in the system
- keep the designer from focusing on the implementation details before resolving the integrity issues.



Example of a Protocol State Machine diagram

Related pages

- Creating diagrams(see page 483)

Activity diagram

Overview

An activity diagram is a type of Behavior diagram that can be used to model any logic - from workflow to Use Cases to methods. It focuses on flows driven by the internal processing, e.g., the implementation of an Operation(see page 677), or a Use Case(see page 876), as opposed to external events. Activity diagrams are organized according to Actions(see page 629).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Although Activity diagrams are often associated with Interaction diagrams, they focus more on the work performed by a system, rather than an object interaction. Additionally, they do not make the links among actions and objects very clear. Instead, Activity diagrams capture Actions(see page 629) and display their results.

> You can create a new activity diagram under the following elements: Action(see page 629), Structured Activity Node(see page 782), Expansion Region(see page 788), Conditional Node(see page 789), Loop Node(see page 789), Sequence Node(see page 789).

Purpose

The purpose of this diagram is to focus on flows driven by the internal processing as opposed to external events. It provides a convenient way to describe complex algorithms, parallel operations, and business processes. Together with the Communication(see page 302) and Sequence diagrams(see page 304), they are used to relate Use Cases(see page 876). Use the Activity diagrams in situations in which all or most of the events represent the completion of internally-generated actions, i.e. procedural flow of control.

Usage

An Activity diagram can be used to:

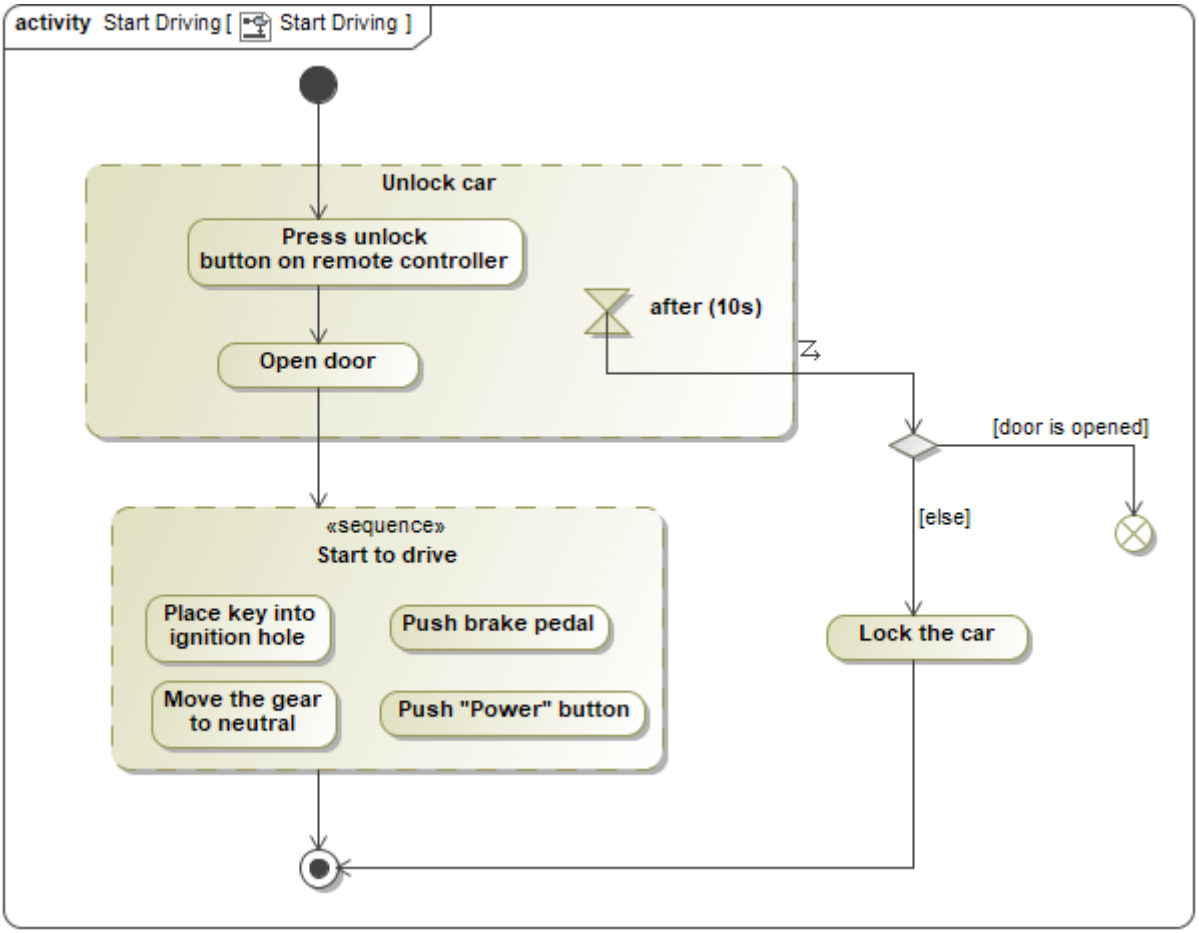
- illustrate a business process or workflow between users and the system
- simplify and improve processes by clarifying complicated Use Cases
- understand workflow across many Use Cases
- analyze and describe the steps performed in a UML Use Case
- demonstrate the logic of an algorithm
- model method, function, operation, and other software architecture elements.

Summary

Activity diagrams are valuable because they:

- represent the logic that is necessary to implement system behaviors
- represent logic at any level the design needs, from system workflow to individual method implementations
- can be learned quickly
- are relatively familiar to users since it borrows a portion of its notation from flowcharts.
- support and encourage parallel behavior, which makes them a great tool for workflow modeling and multi-threaded programming.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Example of an Activity Diagram

Related pages

- Creating diagrams(see page 483)
- Dragging in Activity diagrams(see page 489)

Working with Activity diagram

On this page

- Behavior Creation Mode(see page 314)
- Creating Actions by pasting the copied list(see page 314)
- Smart layout(see page 315)
    - Dynamic centerlines(see page 315)
    - Diagram orientation(see page 316)
- Deleting from Activity diagrams(see page 317)
- Displaying inner elements(see page 317)

Behavior Creation Mode

You can use the **Behavior Creation Mode** to automatically create an Activity after creating a Call Behavior Action(see page 633) in the Activity Diagram(see page 313). The newly created Activity will be set as the behavior of the Call Behavior Action.

To turn on the **Behavior Creation Mode** in an Activity diagram

---

- In the diagram palette(see page 24), click
  .

  > ⚠ The Behavior Creation Mode works separately for each project.

When the **Behavior Creation Mode** is on:

- A Behavior(see page 662) is created together with the Call Behavior Action(see page 633). It inherits the type of the Call Behavior Action owner.
- If a Call Behavior Action(see page 633) and its Behavior(see page 662) are not titled, type a name on the Call Behavior Action shape on the diagram, and the typed name will be set as the Behavior's name automatically.

Creating Actions by pasting the copied list

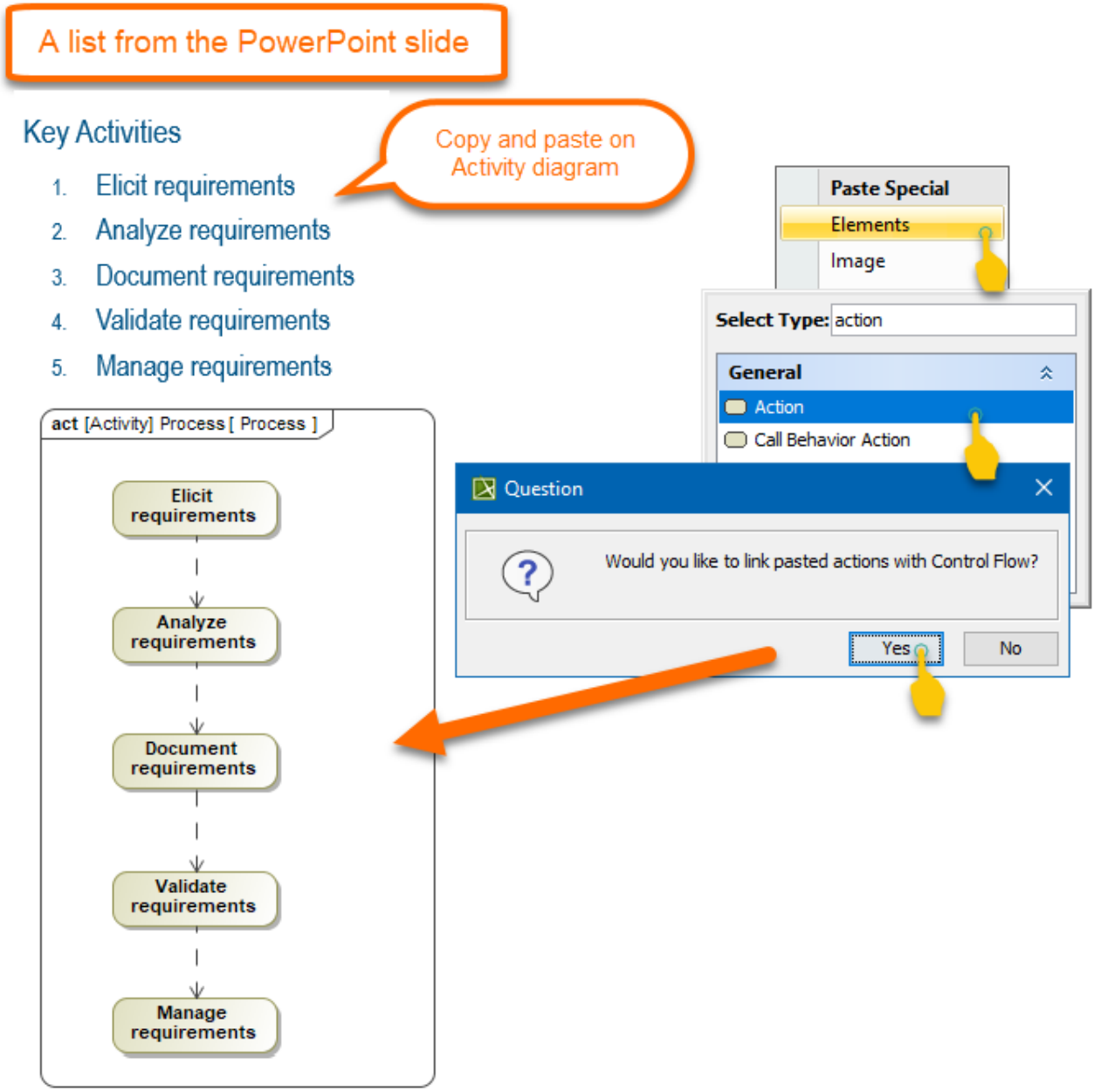
You can copy the item list from the text editor (Word, Notepad, etc.), web page, PDF file, PowerPoint slide and paste it in the Activity diagram to create actions.

To create Actions by pasting items from the list

---

1. Copy the list.
2. Go to the diagram and press Ctrl+V.
3. Select an Action.
4. In the question dialog click **Yes** if you need to connect the Actions with Control Flow relationship.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

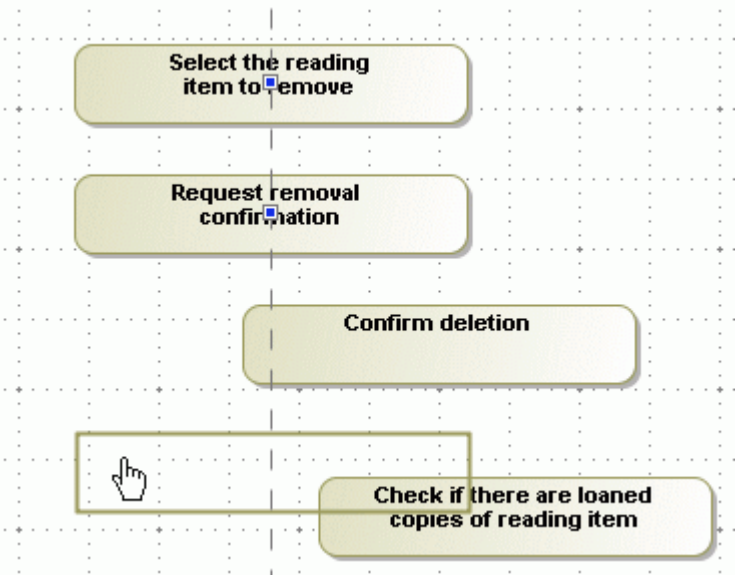


Smart layout

Dynamic centerlines
The centerlines are displayed only when the center of the shape that was moved or newly drawn is located near the center of a preexisting shape in the diagram. These lines help you to easily align shapes when drawing a diagram.

When the center of the shape that was moved coincides with a center of any shape that is placed to its right or left, a horizontal centerline appears. When the center of the shape is close to any center of a shape that is located above or below it, a vertical centerline appears.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



To switch off the dynamic centerlines

- Click the **Show Centerlines** button

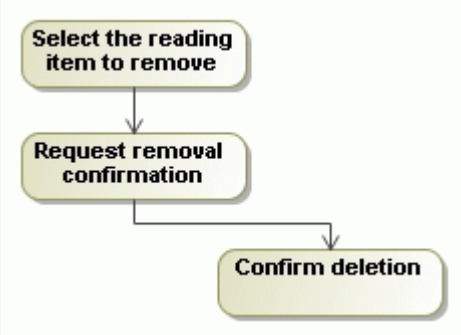
  in the diagram toolbar, or press **C**.
- From the **Options** main menu, select **Environment**. In the **Environment Options** dialog, click the **Diagram** category, and clear the **Show Centerlines in Flow Diagrams** check box in the **Display** properties group.

Diagram orientation

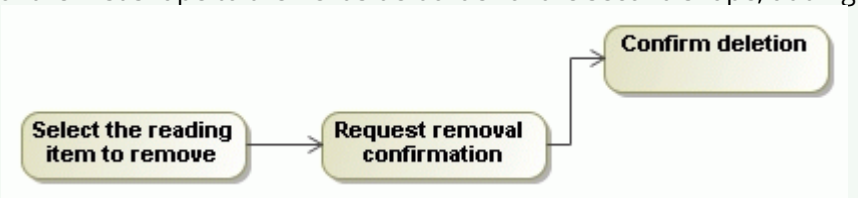
The diagram orientation is used to assign the correct rectilinear path breaks and draw paths between the Activity diagram shapes. The paths can be drawn from side to side, or from the lower to the upper shape borders.

Example

If you have a vertical diagram orientation and shapes are not in the same centerline, the paths will be connected from the lower border of the first shape to the upper border of the next shape, adding break points:



If you have a horizontal diagram orientation, the paths will be connected from the side border of the first shape to the next side border of the second shape, adding break points:



To change the diagram orientation

- From the **Options** main menu, select **Project**. In the **Project Options** dialog(see page 214), expand **Symbol Styles > Default > Diagram** group (if it is not expanded by default) and, in the options list, change the value for the **Diagram Orientation** property.
- From the diagram pane shortcut menu, select **Diagram Properties,** or press Shift+Enter to open the **Diagram Properties** dialog. Change the value for the **Diagram Orientation** property in that dialog.

Deleting from Activity diagrams

- Note that after deleting the last symbol of an element from the Activity diagram, the element will be automatically removed from the model. This does not affect Pins(see page 805) and Activity Parameter Nodes(see page 783).
- If a Call Behavior Action(see page 633) that has an assigned Behavior(see page 662) is deleted from the Activity diagram, then this Behavior is also deleted from the model.

> This is valid if the assigned Behavior is editable and does not have more usages, i.e. there is not a Call Behavior Action that has a reference to this Behavior.

Displaying inner elements

To display inner elements

- When creating a new Activity diagram, right-click an Activity in the Containment tree, point to **Cre ate Diagram**, and then click **Activity Diagram**.

Inner elements are displayed on the Activity diagram pane automatically upon creating the first diagram under the Activity element.

When modifying a previously created Activity diagram, select a diagram pane and do one of the following:
- From the shortcut menu, click **Display** > **Display Inner Elements**.
- On the diagram toolbar click

,
and select **Display Inner Elements**.

Related pages

- Creating diagrams(see page 483)
- Dragging in Activity diagrams(see page 489)
- Creating elements from other resources(see page 911)

Component diagram

Overview

A Component diagram represents pieces of software in the implementation environment in terms of code components.

A Component(see page 685) contains information about the logical Class(see page 666) or Classes that it implements, thus creating a mapping from a logical view to a component view. Dependencies between the Components make it easy to analyze how a change in one component affects the others. The Components may also be shown with any of the Interfaces(see page 741) that they expose. They, as with almost any other model elements, can be grouped into Packages(see page 800), much like Classes or Use Cases(see page 876).

Purpose

A Component diagram models the implementation view. It is used in the later phases of the software development when there is a need to divide up Classes among different Components. When working with the CASE facilities, the Components are used for file-class mapping during code generation, reverse engineering, and round-trip engineering operations.

Usage

A Component diagram can be used to:

- analyze system components and how they relate
- analyze system's physical structure
- understand the exact service behavior provided by software.

Summary

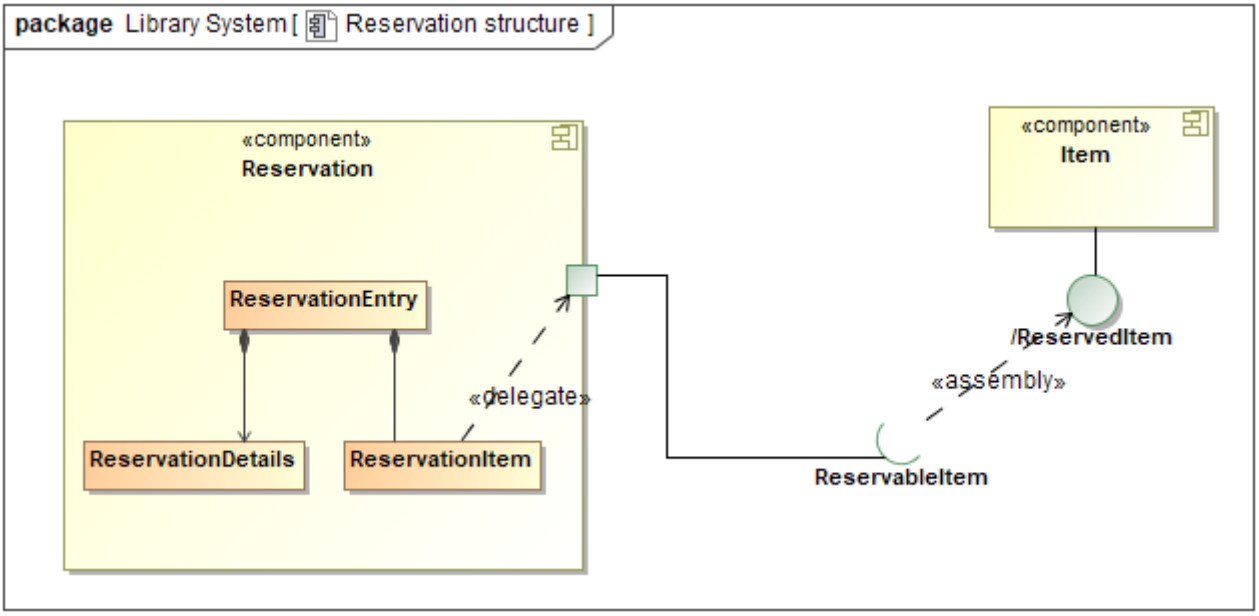
Component diagrams are valuable because they:

- reveal possible software configuration issues

- provide an accurate picture of existing systems prior to making changes/enhancements
- model the real software in the implementation environment
- help to reveal bottlenecks in an implementation.

As of MagicDraw 17.0.1, the Component diagram (or the Deployment diagram(see page 319)) replaces the Implementation diagram, which is no longer supported in UML standard. An Implementation diagram created with earlier versions of MagicDraw, now opens as
- Deployment diagram, if Nodes *were used* in the Implementation diagram
- Component diagram, if Nodes *were not used* in the Implementation diagram

Customized diagrams that were based on the Implementation diagram are now based on the Component diagram.



Example of a Component diagram

Related pages

- Creating diagrams(see page 483)

Deployment diagram

Overview

A Deployment diagram falls under the structural diagramming family. It focuses on showing the physical layout of various hardware components (Nodes(see page 782)), e.g. CPU, a database server, and an application server that compose a system as well as the distribution of executable programs (software components) on hardware. For instance, to describe a website, the Deployment diagram would show what hardware components exist, what software components run, and how the different pieces are interconnected. Deployment diagrams are crucial when dealing with distributed systems.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

As of MagicDraw 17.0.1, the Deployment diagram (or the Component diagram(see page 318)) replaces the Implementation diagram, which is no longer supported in UML standard. An Implementation diagram created with earlier versions of MagicDraw, now opens as

- Deployment diagram, if Nodes were used in the Implementation diagram
- Component diagram, if Nodes were not used in the Implementation diagram

Customized diagrams that were based on the Implementation diagram are now based on the Component diagram.

Purpose

Using a Deployment diagram, you can show the actual computers and devices (Nodes), along with the connections they have to each other, thus specifying a system topology. Inside the Nodes, executable Components(see page 685) and objects are located in a way that it shows where the software units are residing and on which Nodes they are executed. You may also show dependencies between Components.

Usage

A Deployment diagram can be used to:

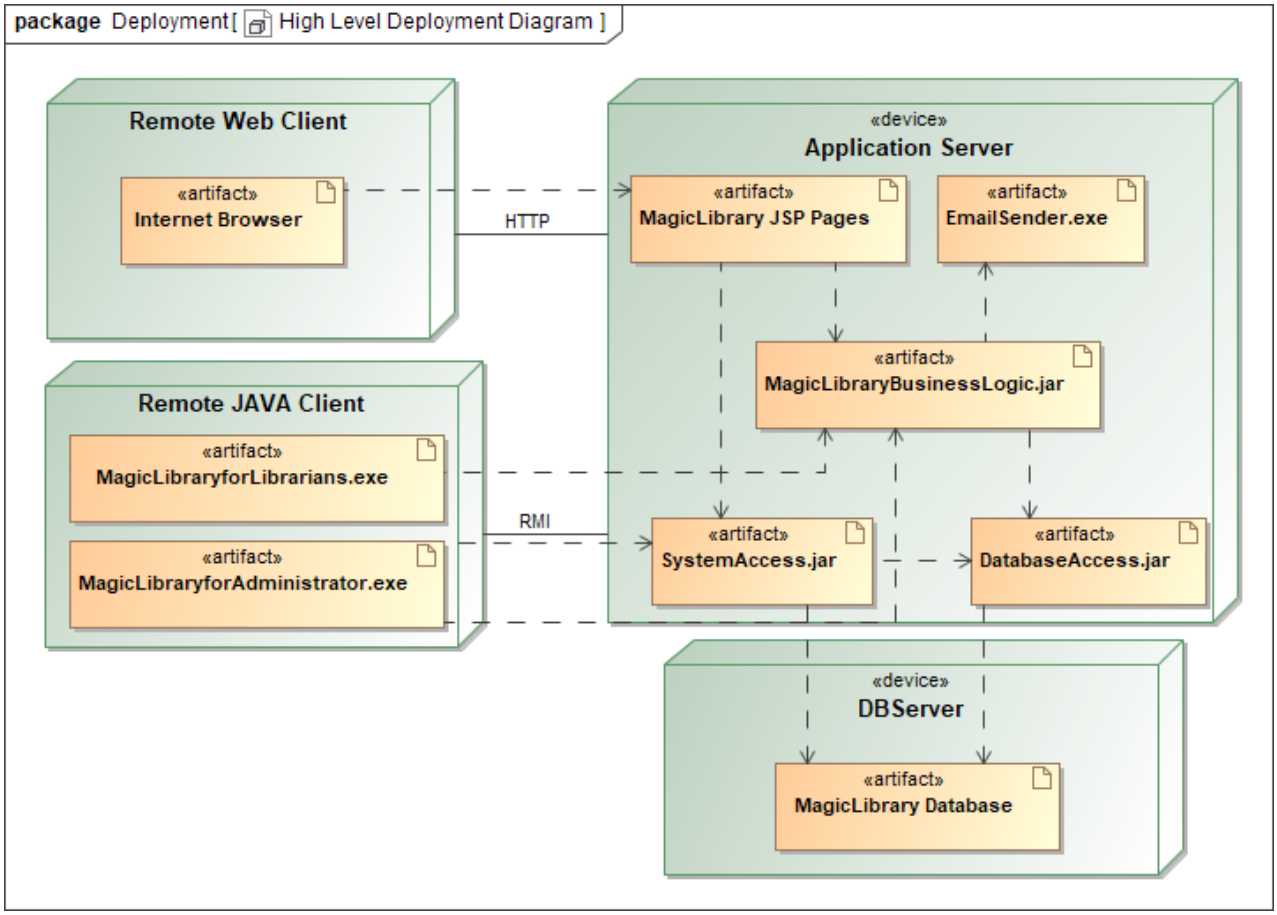
- show the structure of the run-time system
- capture the hardware that is going to be used to implement the system and links between different hardware items
- model physical hardware elements and the communication paths between them
- plan the system architecture
- document the deployment of software components or nodes.

Summary

Deployment diagrams are valuable because they can be:

- used to model the hardware platform for a system
- used to identify hardware capabilities that have an impact on the performance planning and software configuration.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Example of a Deployment diagram

Related pages

- Creating diagrams(see page 483)

Package diagram

Overview

A Package diagram shows the structure of the system at the level of Packages(see page 800). Generally, it arranges and organizes packages and Dependencies(see page 699) between the Packages. Package is the major element used in the Package diagram.

Package. A Package diagram uses Packages(see page 800) to group UML elements. They are also used to provide a namespace for the grouped elements. Just like classes(see page 666) can be grouped into packages, packages can be nested within other packages. A Package, as an entity, may have all the relationships that can be drawn for a class. Those relationships are derived from the classes or packages that are nested within two particular packages (i.e., the relationship between packages reflects a set of relationships between classes placed in those packages).

Purpose

A Package diagram shows the arrangement and organization of model elements in a project. It shows both structure and dependencies between sub-systems or modules.

Usage

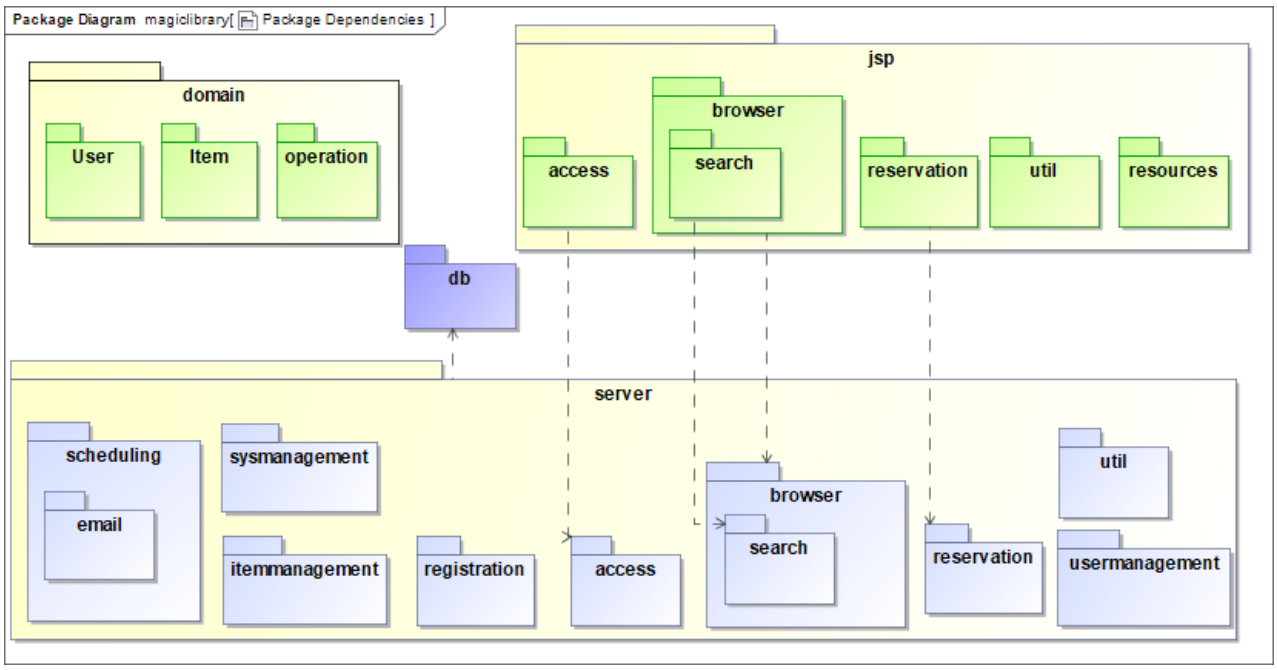
A Package diagram can be used to:

- organize large systems containing diagrams, documents, and other deliverables
- simplify complex Class diagrams.

Summary

Package diagrams are valuable because they:

- can be used to structure high level model elements
- can be used to create an overview of a large set of models
- help to organize a large model
- group related elements

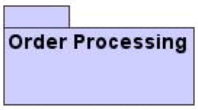
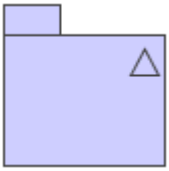


Example of Package diagram

Related pages

- Creating diagrams(see page 483)

Package diagram elements

Element	Button (hot key)	Notation
Package A group of classes and other model elements.	 (P)	
Model A model is an abstraction of a physical system from a particular point of view. A model contains a hierarchy of packages/ subsystems and other model elements that describe the system.	 (M)	
Profile A Profile is a kind of Package that extends a reference metamodel.		
Package Merge A package merge is a directed relationship between two packages that indicates that the contents of the two packages are to be combined.		
Package Import A package import (see page 704)is defined as a directed relationship between an importing namespace and a package, the members of which are imported into that namespace.		
Element Import An element import is defined as a directed relationship between an importing namespace and a packageable element.		

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Profile diagram

Overview

In general, a Profile diagram has been introduced in UML 2.0 to display the usage of profiles. It is an extensibility mechanism that allows extending and customizing UML. To be more specific, the Profiles package contains mechanisms that allow metaclasses from existing metamodels to be extended to adapt them for different purposes.

Purpose

A Profile Diagram extends UML by adding new building blocks, creating new properties, and specifying new semantics so that the language would be suitable to the specific domain, for example, medicine, financial services, or engineering.

Usage

A Profile diagram extends UML to allow the use of:

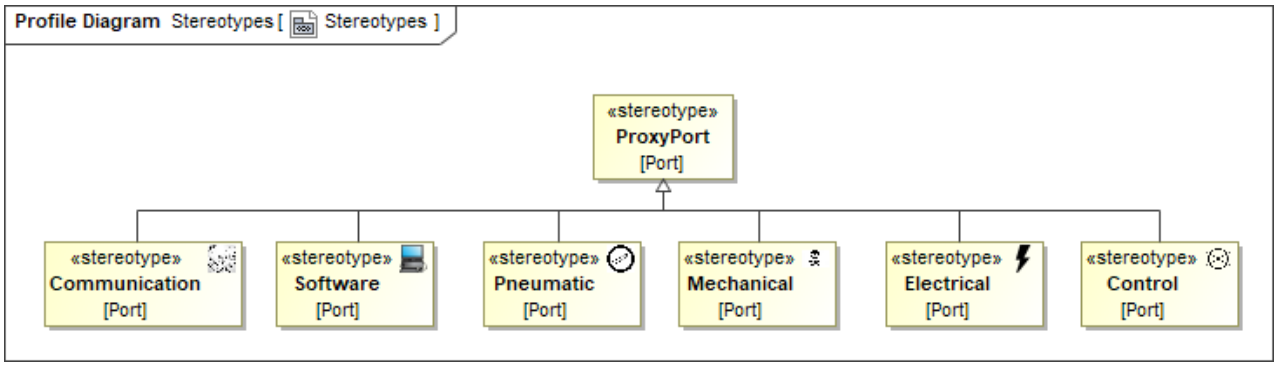
- user-defined stereotypes
- meta attributes
- constraints.

Summary

Profile diagrams are valuable because they allow for adaptation of the UML metamodel for:

- different platforms (e.g., Java Platform, Microsoft .NET Framework, Enterprise Edition (Java EE), etc.)
- domains (e.g., business process modeling, medical applications, service-oriented architecture)

> If the Class(see page 666), Data Type(see page 696), Primitive Type(see page 698), Enumeration(see page 697), Association(see page 651), Direct Association, Generalization(see page 722) are not available on the diagram palette, make sure you are working in the Expert mode.



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example of Profile diagram

Related pages

- Creating diagrams(see page 483)

Composite Structure diagram

Overview

The Composite Structure diagram allows a decomposition and modeling of the internal structure of the classifiers, such as classes(see page 666), components(see page 685), nodes(see page 782), and other elements that are able to own other elements.

Purpose

A Composite Structure diagram can be used to visually represent the parts of a class, component, or collaboration, including the interaction points (called ports) that are used to access features of the structure. Additionally, Composite Structure diagrams model collaborations that describe behavior, resources used to perform the behavior, and the roles the participating resources take during the behavior.

Usage

Composite Structure diagram can be used to:

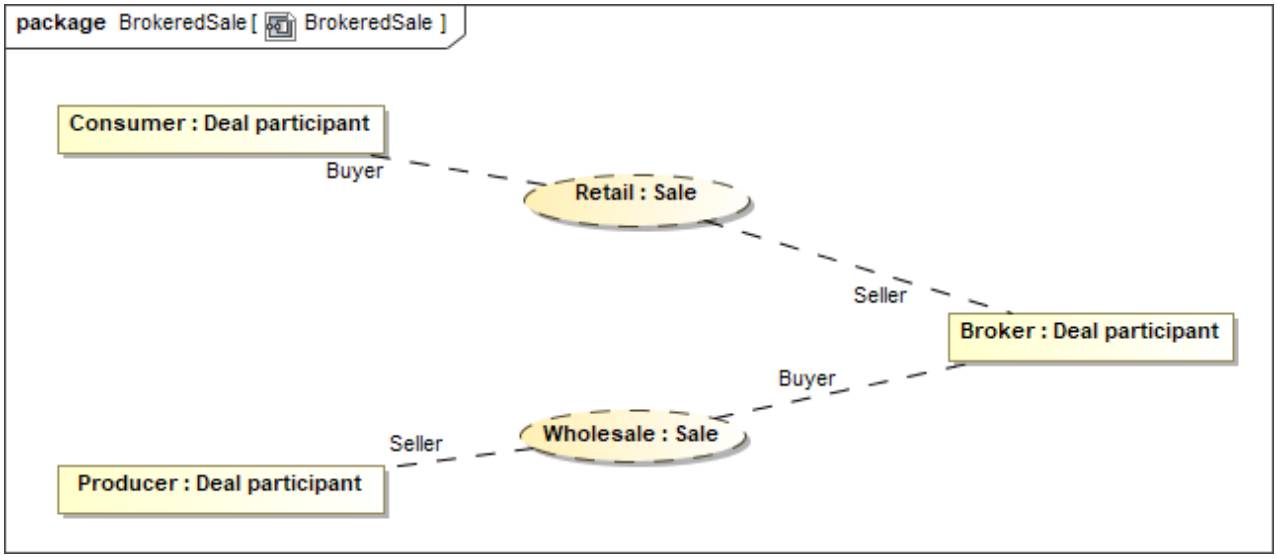
- analyze an object to see exactly what it is composed of
- detail internal actions of a class, including the relationships of nested classes
- show objects that need to be defined as a composition of other classified objects.

Summary

Composite Structure diagrams are valuable because they:

- show complex components design
- show components interfaces separately from their structures
- describe the roles played by elements in the structure.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Example of a Composite Structure diagram

« Profile diagram(see page 324) Composite Structure Diagram context »(see page 326)

Realted pages

- Creating diagrams(see page 483)
- Composite Structure Diagram context(see page 326)
- Nesting parts that are association ends(see page 327)
- Mapping ports on a Part type change(see page 328)
- Creating nested parts by pasting items from the list(see page 330)

Composite Structure Diagram context

When creating a new Composite Structure Diagram(see page 325) for the element that cannot be the context of this diagram (e.g. Package(see page 800)), the new context element (which is a Class(see page 666)) is created automatically.

Turning off an automatic creation of diagram context element

To turn off an automatic creation of diagram context element

1. On the main menu(see page 24), select **Project** > **Options**.
2. In the **Project Options** dialog, find the **Diagram Context** option.
3. Select the **Do Not Create** value.
4. Click **OK**.
   The automatic creation of diagram context element is turned off.

This means that when you create a new diagram for the element which cannot be the context of this diagram (e.g. Package(see page 800)), the **Select Diagram Context** dialog will open automatically. It allows you to select an existing element as a context, create a new context element by enabling the **Creation Mode**, or click the **Cancel** button if you do not want to specify the diagram context. The diagram will be created in the Package.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

If you want to enable the automatic creation of a diagram context element, select the **Create Automatically** value for the **Diagram Context** option.

Changing the diagram context element

After the diagram is created, you can change its context element.

To change the diagram context element

1. Right-click the diagram and select **Specification**.
2. In the Specification window(see page 913), find the **Context** property.
3. Click on its value cell, and select ... .
4. In the **Select Diagram Context** dialog, do one of the following:
   - Select an existing element as a context in the **Tree** or **List** tab
   - Create a new context element by enabling the **Creation Mode** > **Create** and select it.
5. Click **OK**.
   The diagram context element is changed.

« Composite Structure diagram(see page 325)

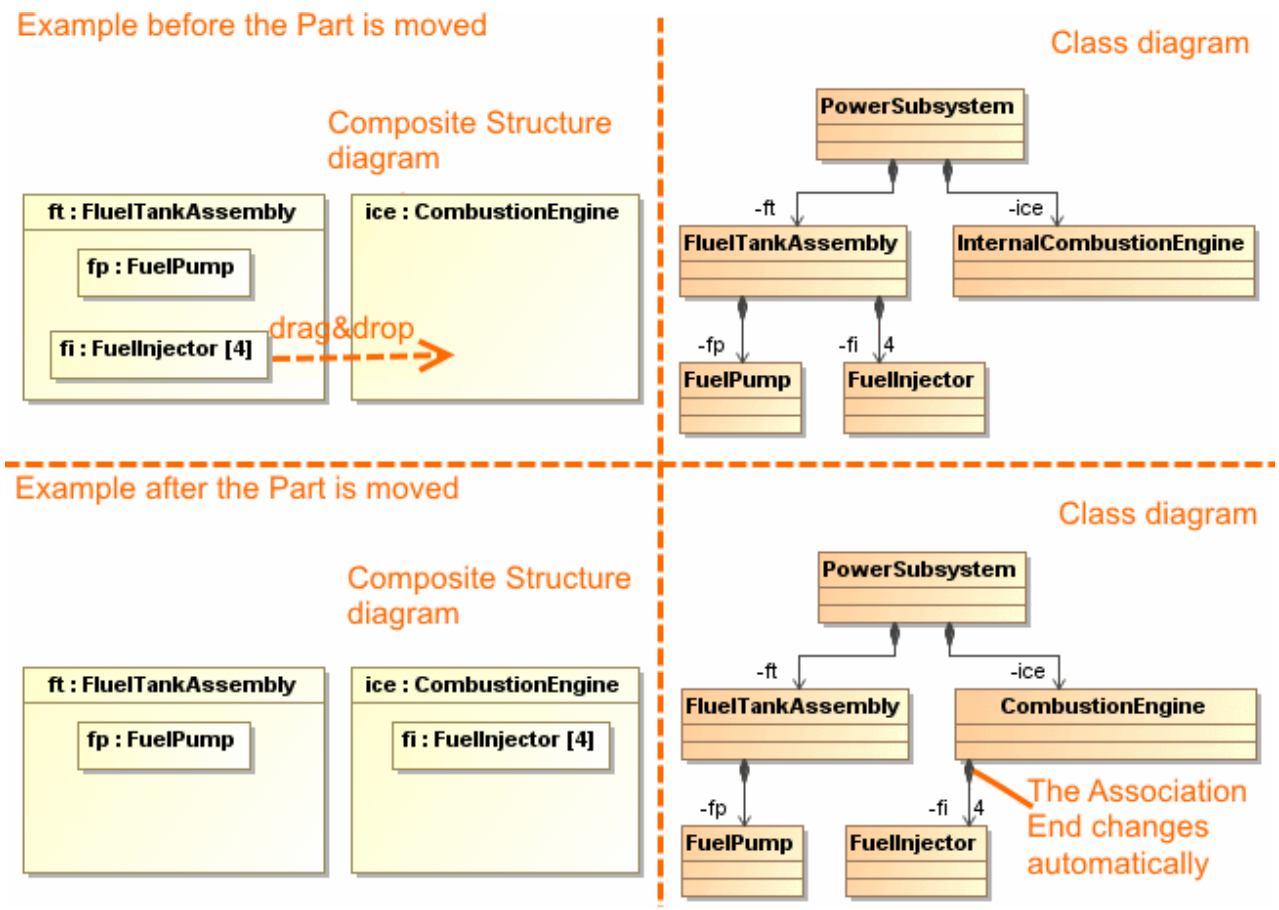
Nesting parts that are association ends »(see page 327)

Nesting parts that are association ends

In this section you will find the description on how to nest a Part to another Part in the Composite Structure diagram. Until MagicDraw version 17.0.2, nesting of parts was not available if a Part was a property of an Association(see page 651), that is, if the Part represented the Association End(see page 654).

If a classifier has an association connection with another classifier, the Association End in the classifier is referenced as a property. In a Composite Structure diagram, a Part is representation of a property. So the Part can represent the property that is an Association end too. If the Association End is composite, parts are represented in a nested structure.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



See an example in the preceding figure. In the Class diagram(see page 297), the Class *FuelTankAssembly* is connected with the Class *FuelInjector* with the composite Association. In the Composite Structure diagram, the same Association is represented in the following way: the Part with the type *FuelInjector* is nested into the Part with the type *FuelTankAssembly*. By dragging and dropping the Part with *FuelInjector* to the Part with the *CombustionEngine*, changes the Association in the Class diagram - the *FuelInjector* will be connected with the *CombustionEngine*.

« Composite Structure Diagram context(see page 326)

Mapping ports on a Part type change »(see page 328)

Mapping ports on a Part type change

On a Part(see page 804) type change, ports(see page 809) of the original Part type are automatically replaced with ports of a new type. Connectors are reconnected accordingly too.

Ports are mapped automatically if particular rules are met. Otherwise, the **Port Mapping** dialog appears.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

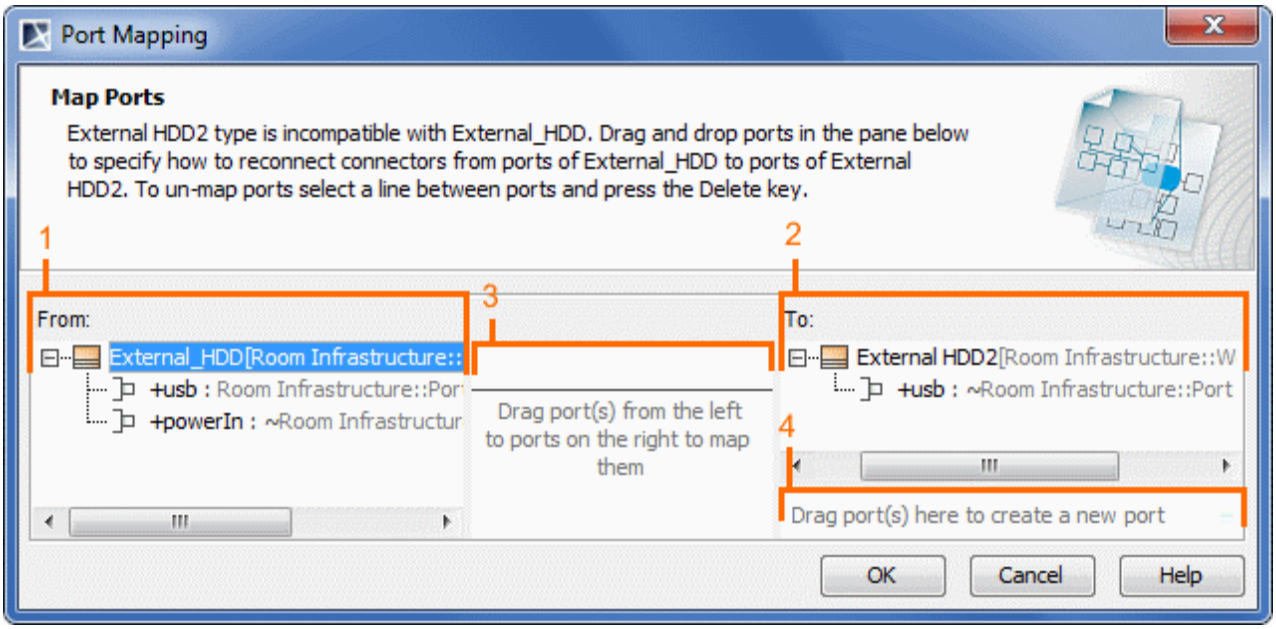
Rules of automatic port mapping

Ports are mapped automatically:

- When the following properties of the Port of the original Part type are the same as the properties of the Port of the new part type:
  - **Port type and name**
  - **Port type** (note that only one Port of the new Part type has the same type as Port(s) of the original Part type)
  - **Port name** (valid if the type of the original Port is **not** specified)
- Ports of the original Part type and the Port of the new Part type has the same Interface(see page 741).
- When the new Part type is subtype of the original Part type (inherited ports are mapped automatically with ports which they are inherited).
- When the ports of a new type redefines ports of original Part type.

The Port Mapping dialog

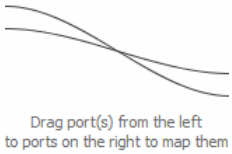
The **Port Mapping** dialog appears on changing a Part type when ports of the type cannot be mapped automatically.



See the elements of the **Port Mapping** dialog described in the following table.

Element number	Element	Description
1	From:	Lists the type of the Part from which you want to make a replacement. Ports of the type are listed under the type.
2	To:	Lists the type of the Part to which you want to make a replacement. Ports of the type are listed under the type.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Element number	Element	Description
3		Displays lines between the mapped ports.
4	Drag port(s) here to create a new port	Creates a new port in the **To** list. In the **From** list, click a port and then drag it to the "Drag port(s) here to create a new port" box. The port for the new type will be created and mapped.

To map ports in the **Port Mapping** dialog

1. Select a Port in the **From** box.
2. Drag the selected Port onto a Port in the **To** box. The ports are mapped and the line between them is displayed.

   > You can also select multiple number of ports in the **From** box and drag onto a port in the **To** box. The selected ports will be mapped with the Port.

To create a new Port in the **To** box

1. Select a Port in the **From** box.
2. Drag the selected Port onto the following box: "*Dragport(s)heretocreateanewport*". The new Port is created in the **To** box and ports are mapped.

   > You can also select multiple number of ports in the **From** box and drag onto the box: "*Drag port(s) here to create a new port*". The selected number of ports will be created.

To un-map ports in the **Port Mapping** dialog

1. Select the line between the mapped ports.
2. Press the **Delete** key. The line is deleted and ports unmapped.

« Nesting parts that are association ends(see page 327)

Creating nested parts by pasting items from the list »(see page 330)

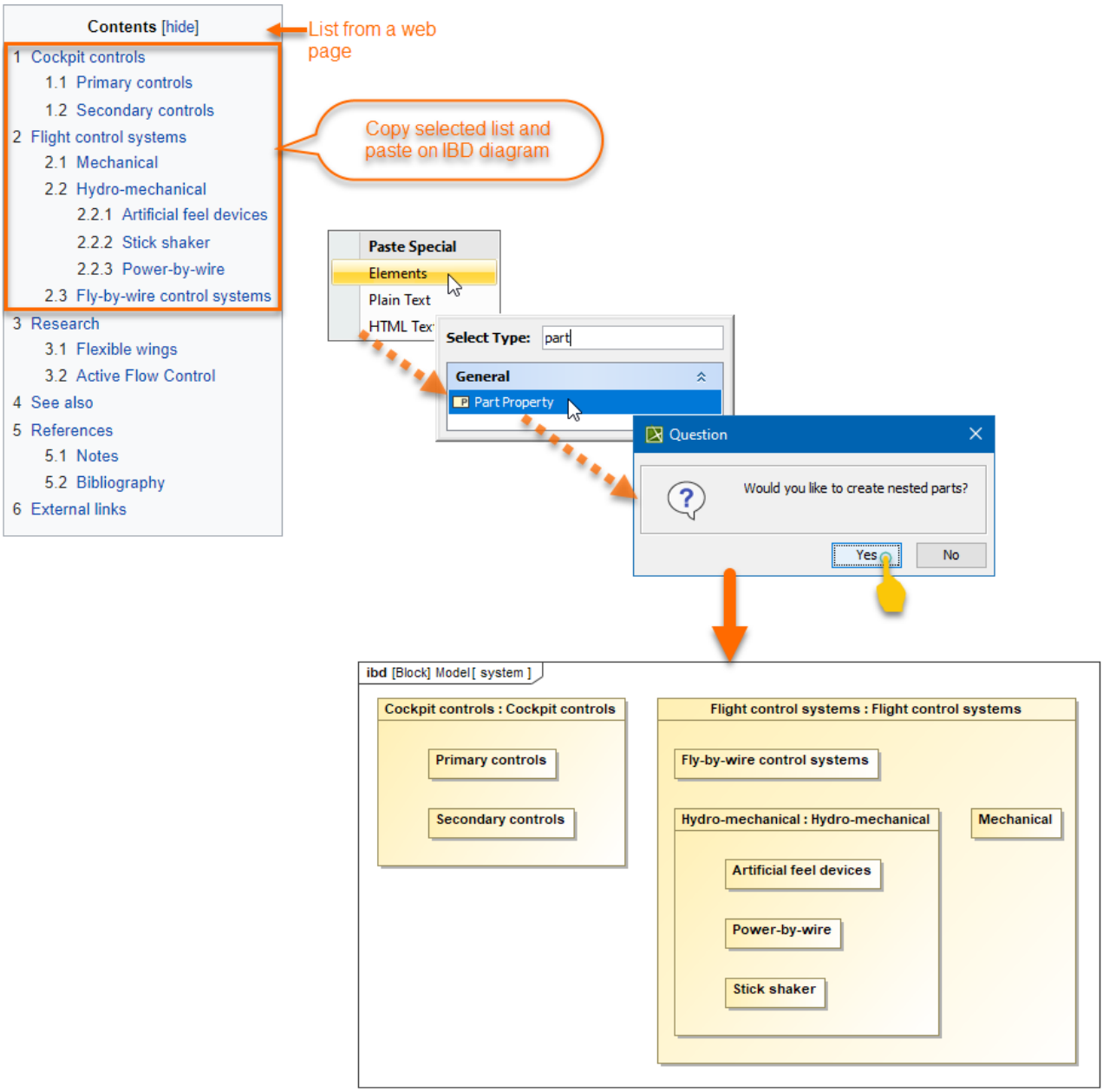
Creating nested parts by pasting items from the list

When you have an indented list of the items you want to create as nested parts in the diagram, you can copy the list and past it in the Composite Structure Diagram diagram pane.

You can copy the list from the text editor (Word, Notepad, etc.), web page, PDF file, PowerPoint slide, etc.

To create parts by pasting items from the list

1. Copy the list.
2. Go to the diagram and press Ctrl+V.
3. In the open menu, select **Element**.
4. Select the Part.
5. In the question dialog click **Yes** if you need to create nested parts.



The nested parts are created according to the copied list

Related pages

- Creating elements from other resources(see page 911)

Interaction Overview diagram

Overview

An Interaction Overview diagram focuses on the flow of control between interactions. It is based on the Activity diagram(see page 311) notation. Interactions in the Interaction Overview diagram are represented using the Interaction Use(see page 740) element. All elements in the diagram, except for the Interaction Use, are the same as in the Activity diagram.

Purpose

An Interaction Overview diagram is intended to be used to model the flow of logic in a series of interactions. To put it another way, the Interaction Overview diagram uses blocks of interactions and individual interactions instead of modeling a series of discrete behaviors (activities).

Usage

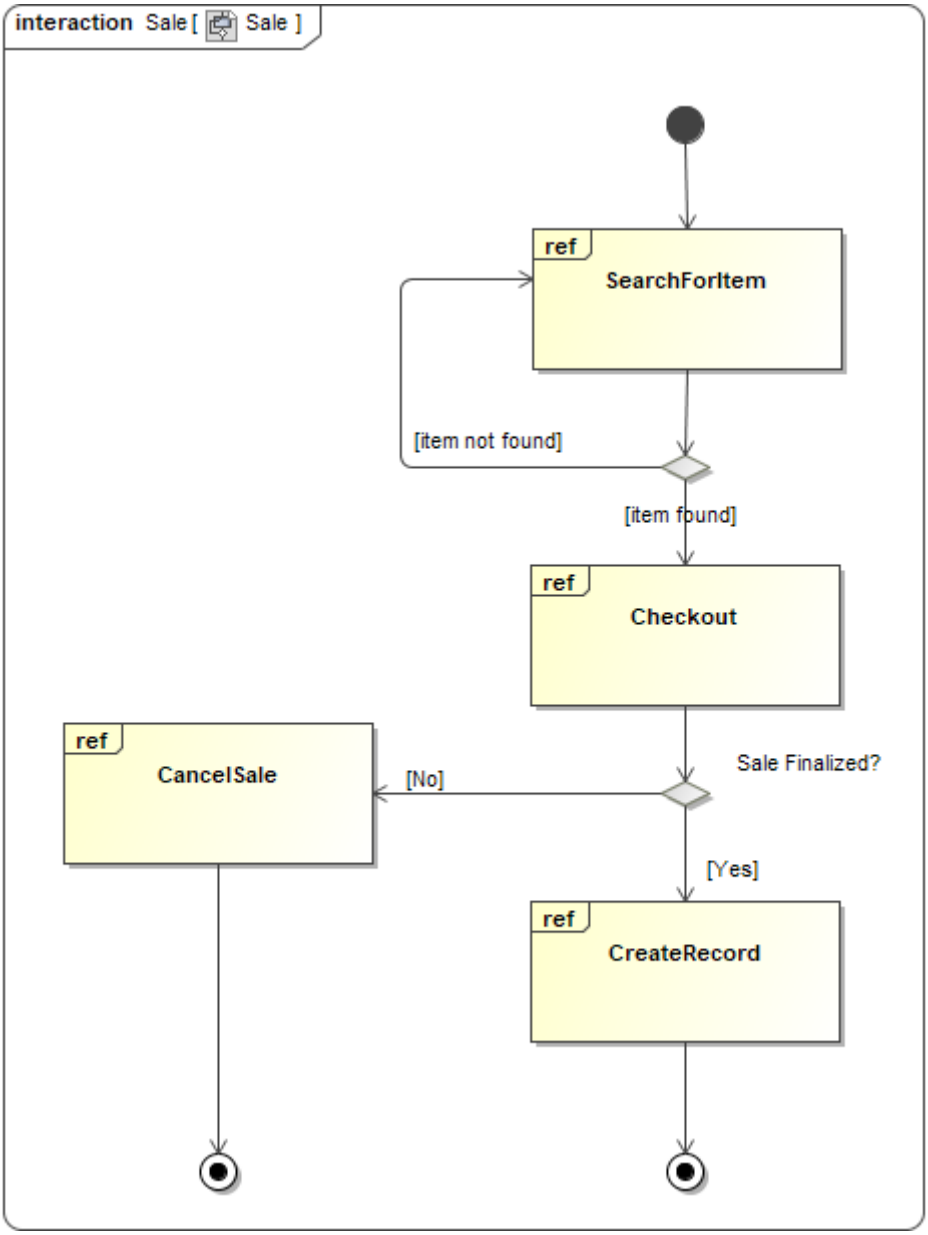
Interaction Overview diagrams visualize a sequence of activities just like Activity diagrams do; however, they picture each individual entry as a frame which can contain a nested interaction diagram.

Summary

Interaction Overview diagrams are valuable because they:

- provide a high-level view of the logical progression through a set of interactions
- keep the focus on the "big picture" and away from implementation details
- encourage the reuse of interactions

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Example of an Interaction Overview diagram

Related pages

- Creating diagrams(see page 483)

Analysis diagrams

The pages below describe dependency matrices, relation maps and robustness diagrams, including their purpose and how to create them.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related pages

- Dependency Matrix(see page 334)
- Relation Map(see page 364)
- Robustness diagram(see page 375)

Dependency Matrix

Demo introducing Dependency Matix.

Sorry, the widget is not supported in this export.
But you can reach it using the following URL:

https://www.youtube.com/watch?v=7Utc3D2vU1I

This functionality is available in Standard, Professional, Architect, and Enterprise editions only.

The Dependency Matrix is a quick method for representing dependencies compactly. Diagrams, UML elements, and extended UML elements serve as row and column entries. The cells in the matrix show where these elements are associated (related).

Dependency matrices can show dependency criteria, such as

- UML relationships
- BPML, SysML, UPDM, and other DSL relationships
- Semantic dependencies (dependencies through properties)
- Relationships through tags (tags allow for relating UML elements that cannot be represented on the same diagram)
- Transitive relationships (through the OCL or meta chain)

Dependency matrices are useful for

- Quickly visualizing and creating dependency criteria.
- Compactly visualizing relations of a large system whose system relations cannot be represented by a diagram on a single sheet of paper due to the size of the system.
- Studying relations from a particular scope and type of element by filtering the unimportant data. Hierarchical columns allow for browsing directly in the created matrix through the model scope and visualize the required area.
- Showing relations that cannot be represented on diagrams, such as a representation (a class by the lifeline), a behavior representation in other diagrams, an operation representation by Call Behavior Action, Use Case relations with describing activities through the property Owned Behavior, etc. The semantic dependency matrix is needed for deeper model analysis. The Dependency Matrix allows the representation of any kind of relations through element properties.

To learn about Dependency Matrix properties, please click here(see page 2054).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related pages

- Creating Dependency Matrix(see page 335)
- Using Dependency Matrix(see page 335)
- Dependency Matrix environment(see page 358)

Creating Dependency Matrix

The matrix element in the model is similar to the diagram element. After creating a new matrix, it appears in the Model Browser(see page 29) as a model element.

To create a Dependency Matrix

---

1. In the Containment tree, select an element that can be the owner of the generic table.
2. Do one of the following:
   - From the main menu, select **Diagrams** > **Create Diagram**. Type “depe” and press **Enter**.
   - On the main toolbars, click the **Create Diagram** button. Type “depe” and press **Enter**.
   - Press **Ctrl**+**N**. Type “depe” and press **Enter**.
   - Right-click the element and from the shortcut menu select **Create Diagram** > **Analysis Diagrams** > **Dependency Matrix**.

   The newly dependency matrix opens on the right side of the application window.
3. Type the name of the matrix.
4. Select criteria(see page 335) for the matrix.
5. Click the **Refresh** button. The matrix is created.

You can also create dependency matrices using a custom dependency matrix type(see page 2052).

Using Dependency Matrix

You can modify a Dependency Matrix after it is created. The Dependency Matrix usage and modification features are described in

- Selecting criteria(see page 335)
- Understanding cell content(see page 346)
- Modifying relationships in Dependency Matrix(see page 349)
- Navigation from cell(see page 354)
- Rearranging rows and columns(see page 355)
- Validation in Dependency Matrix(see page 357)

Selecting criteria

On this page

- Specifying element type(see page 336)
- Specifying dependency criterion(see page 337)
- Specifying relationship direction(see page 339)

- Specifying what elements are visible on the matrix(see page 339)

To create a Dependency Matrix, first of all you need to define which data you wish to display on its pane. Select row and column element types, row and column scope, direction of relationships as well as dependency criteria for this. Be advised that the easiest way to define all the above mentioned criteria, except the type of relationships direction is using the drag-and-drop operation.

A lot of properties of the Dependency Matrix can be specified in the the matrix Specification window(see page 913).

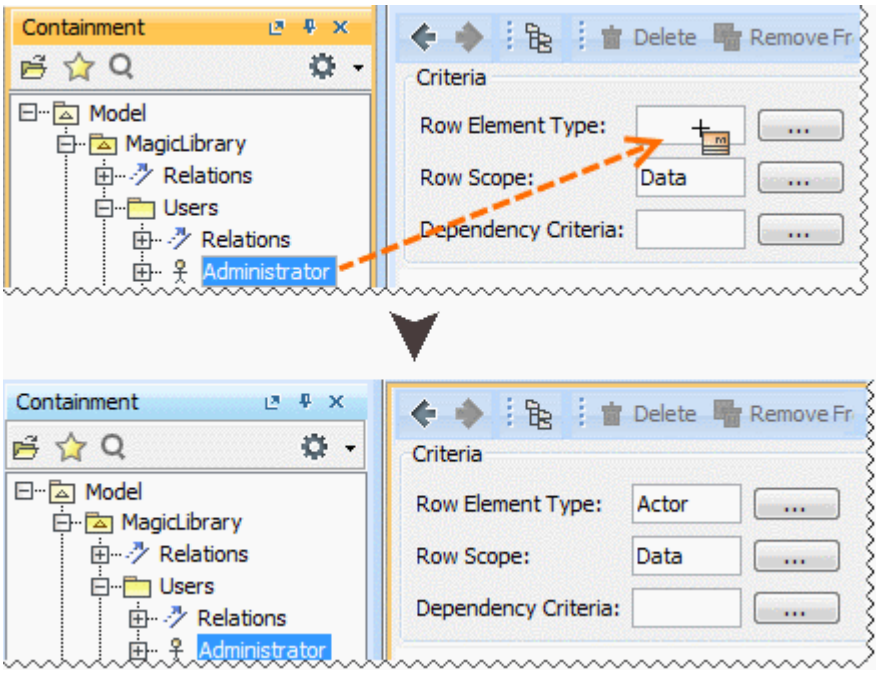
> You can open the matrix Specification window(see page 913) in one of the following ways:
> - From the matrix shortcut menu(see page 361)
> - From the matrix toolbar(see page 358)
> - Press Enter in the active matrix.

Specifying element type
To specify a row/column element type

Do either:

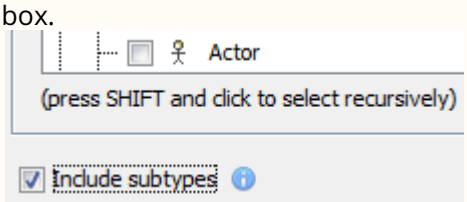
- In the Model Browser(see page 29), select one or more elements which types you wish to see on your matrix and drag them to the **Row Element Type**/**Column Element Type** box in the **Criteria** area.

> - For the multiple selection Press and hold Shift or Ctrl.
> - If you want to add additional elements, press and hold Ctrl while dragging the elements to the **Row Element Type**/**Column Element Type** box.



- Click the button next to the **Row Element Type/Column Element Type** box and in the opened dialog select what element types you wish to see on your matrix. Click **OK**.
- Open the matrix Specification window(see page 913), click the **Row Element Type/Column Element Type** property value cell, then click the button, and in the opened dialog select what element types you wish to see on your matrix. Click **Close**.

To display subtypes of selected element types, click to select the **Include subtypes** check box.



Specifying dependency criterion

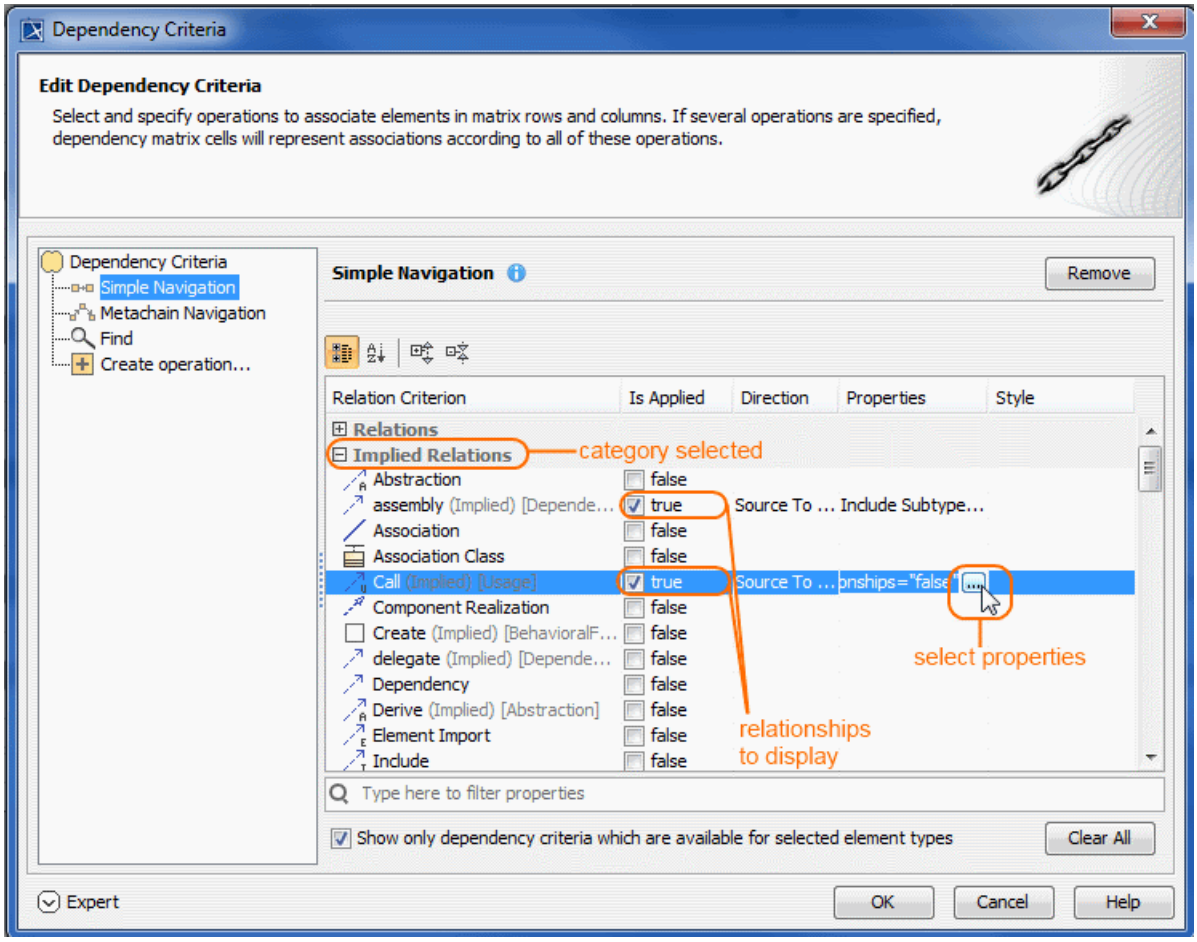
Implied relationships are logical relationships, calculated from inherited or inner elements of the model element, in order to provide valuable information to the user.

You can select to show implied relationships that represent relationships between model elements caused by aggregation and composition. When supertypes, aggregates or compositions have relationships to dependent elements, the corresponding relationships are displayed for subtypes or leaves.

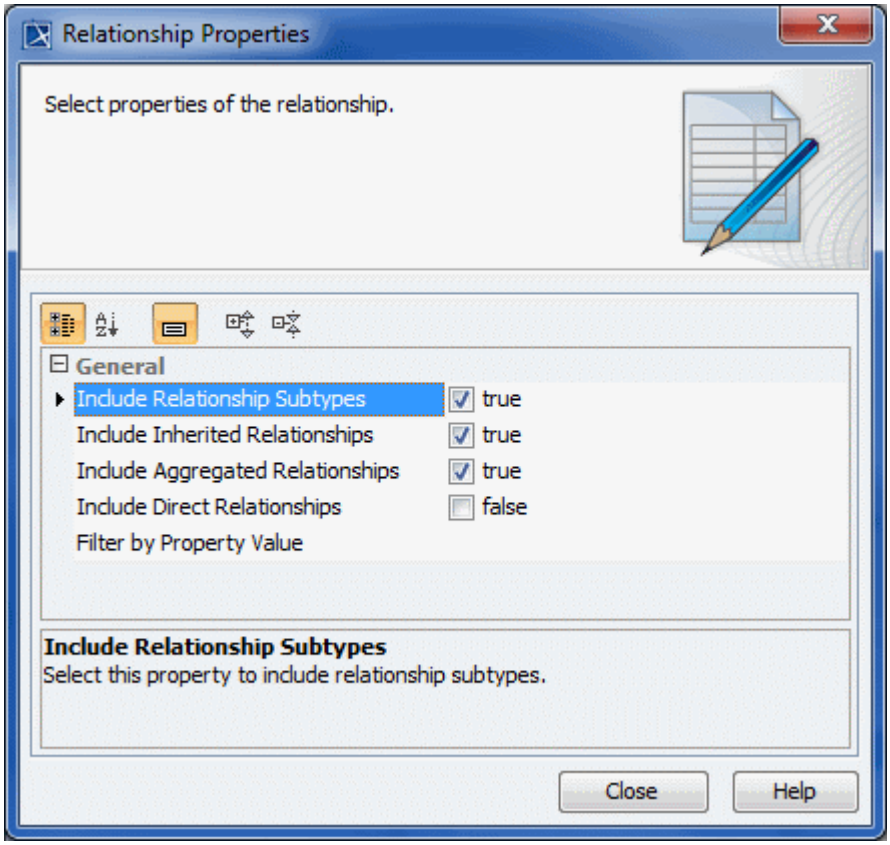
To specify which implied relationships you want to display on the matrix

1. Click the button next to **Dependency Criteria**.
2. Under **Relation Criterion**, find the **Implied Relations** category and select the relationship you want to display.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



3. In the relationship's **Properties** column, click the button to select properties of the relationship. When you're done, click **Close**.



4. In the relationship's **Style** column, click the button to select how the relationship is represented in the matrix(see page 343) .
5. When you're done, click **Close**, then **OK**.

Specifying relationship direction
To specify a relationship direction

Do either:

- Select a direction from the **Direction** drop-down list in the **Criteria** area.
- Open the matrix Specification window(see page 913), click the **Direction** property value cell and select what direction relationships you wish to see on your matrix. Click **Close**.

Specifying what elements are visible on the matrix
To specify what elements you wish to see on the matrix: related, non-related, or all

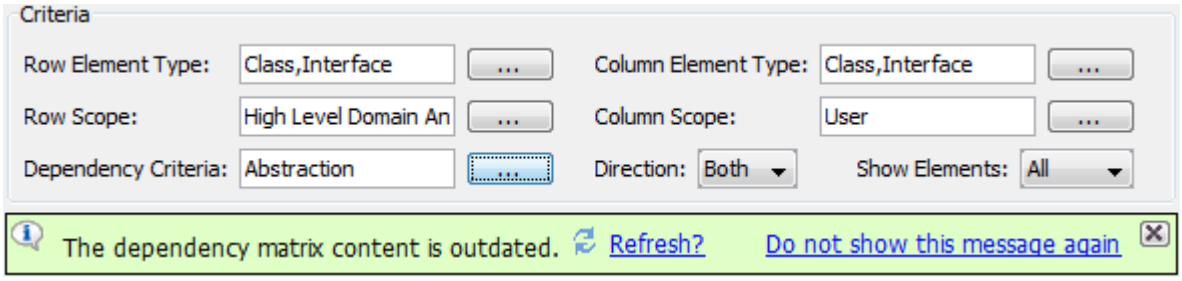
---

Do either:

- From the **Show Elements** drop-down list in the **Criteria** area, select
  - **With relations** if you need to see only related elements from the selected scope.
  - **Without relations** if you need to see only non-related elements from the selected scope.
  - **All** if you need to see both related and non-related elements from the selected scope.
- Open the matrix Specification window(see page 913), click the **Show Elements** property value cell and choose one of the preceding described values. Click **Close**.

After the **Show Elements** value is changed, you should always refresh a matrix.

Once criteria are specified, you can create the matrix. Click on the Dependency Matrix toolbar(see page 358).

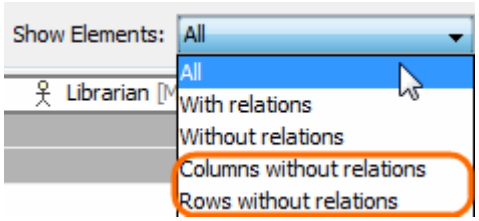


When a Dependency Matrix contains a large number of rows and/or columns, it may become difficult to work with, as you need to scroll in order to see certain parts of it. In this case, you can select to exclude cells (either rows or columns) that contain relationships and make your Dependency Matrix view more compact.

To specify what cells to display on the matrix

1. From the **Show Elements** drop-down list in the **Criteria** area, select either

   - **Columns without relations** – to show columns without relationships and all rows.
   - **Rows without relations** – to show rows without relationships and all columns.



2. Click to refresh the matrix.

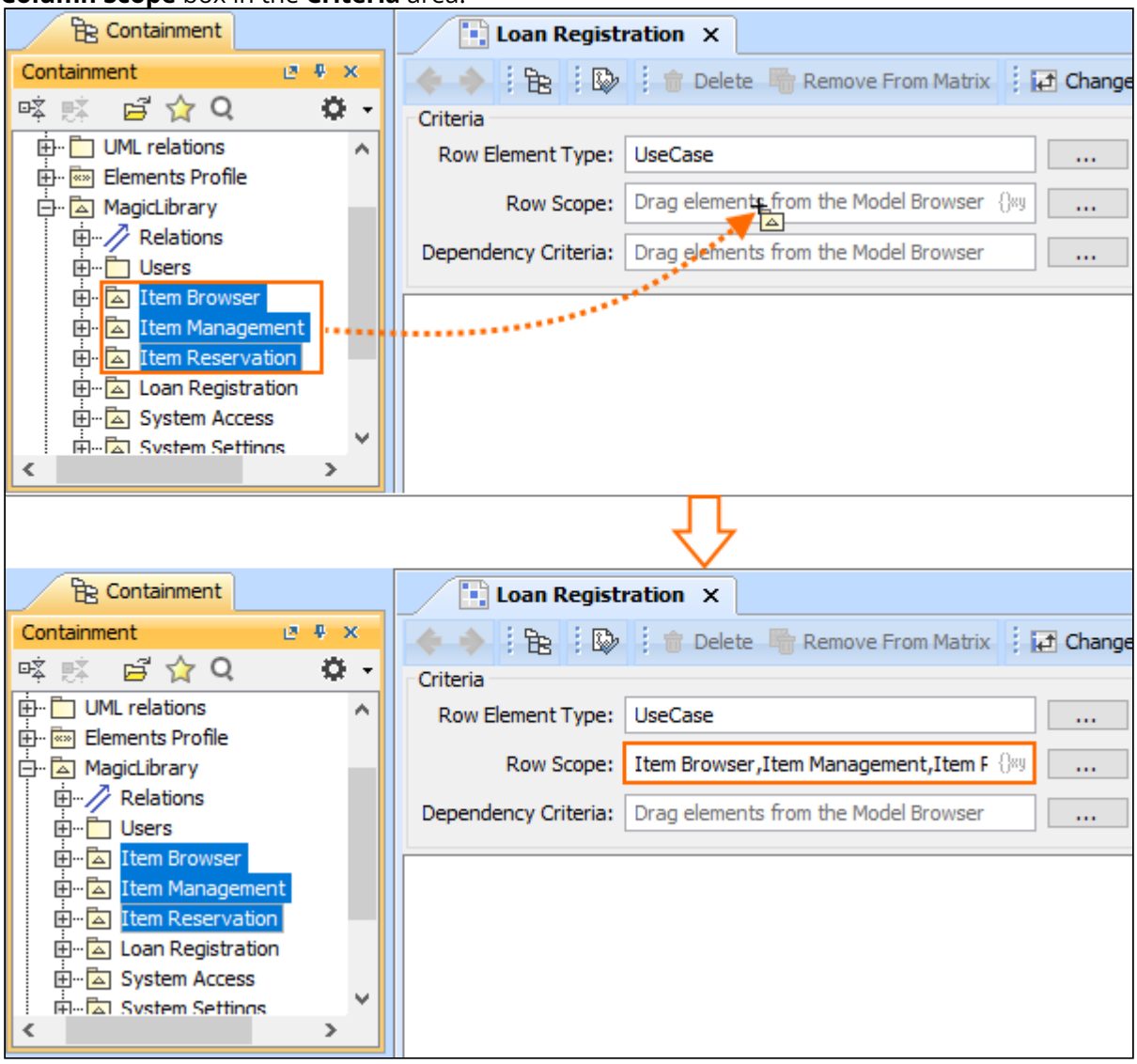
Specifying scope
When you define a scope for the row or a column, you can:

- Simply drag selected packages on the scope box.
- Define expressions for building the personalized hierarchy trees.
- Build a list of elements related through the defined criteria.

To specify the row/column scope

Do one of the following:

- In the Model Browser(see page 29), select one or more packages and drag them to the **Row/Column Scope** box in the **Criteria** area.



> - For the multiple selection Press and hold Shift or Ctrl.
> - If you want to add additional elements, press and hold Ctrl while dragging the elements to the **Row/Column Scope** box.

- Click the ... button next to the **Row/Column Scope** box and in the open dialog select the needed packages. Click **OK**.
- Open the matrix Specification window(see page 913), click the **Row/Column Scope** property value cell, then click the ... button, and in the open dialog select the needed packages. Click **Close**.

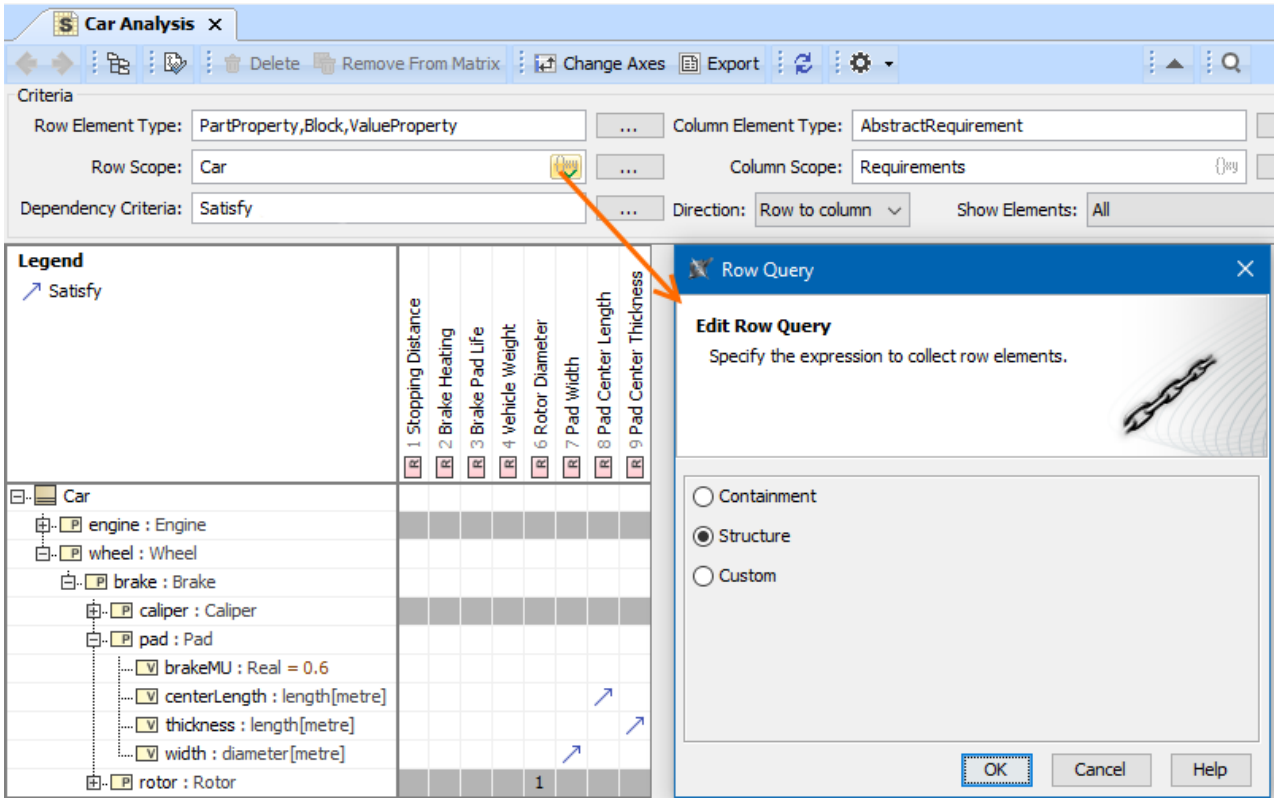
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

You can specify the criterion for building a personalized Row/Column hierarchy tree in matrices by choosing from the following tree-view options:

- Select **Containment** to display the decomposition of elements related through the Containment relationship.
- Select **Structure** to display the decomposition of Part Properties related through the Composition relationship.
- Select **Custom** to display the decomposition of elements related through any criteria of your choice.

By selecting Structure, you can build a custom tree-like view for the rows and columns of the matrix. Currently it is possible to build a tree hierarchy by specifying your own criteria. You can select the predefined Structure criteria which builds the same hierarchy as in the Structure Tree in Model Browser.

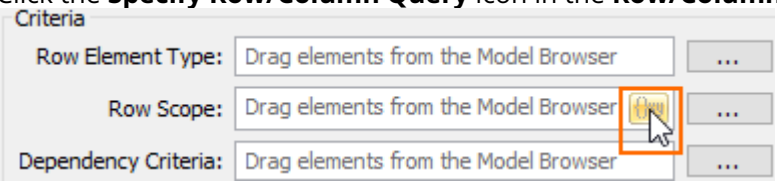
Select Custom, if you need to collect elements that are not necessarily owned by the same package, you can specify the query in the **Row/Column Query** dialog, and see the results added to it immediately.



Specifying criteria for building a personalized tree-view in a matrix

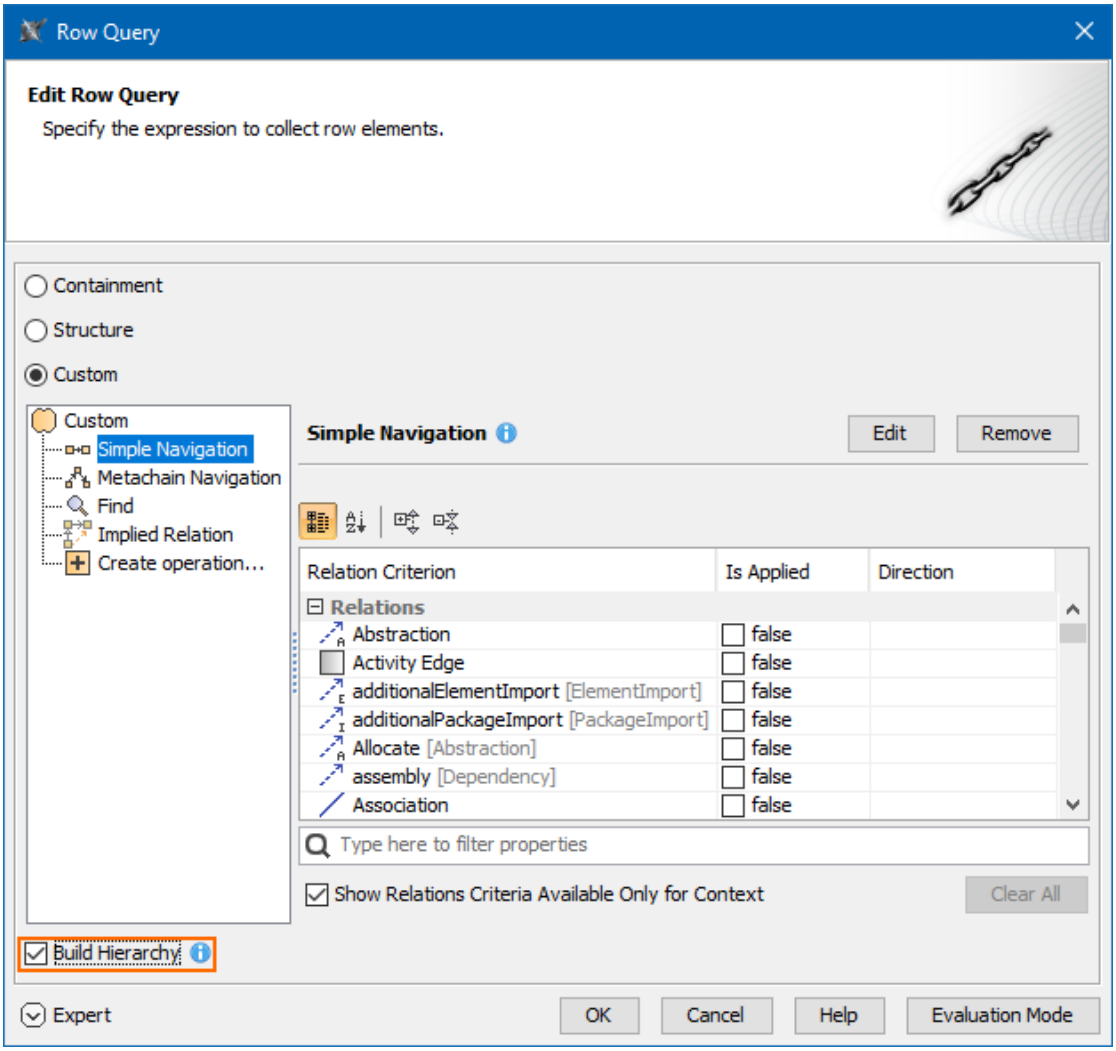
To specify the row/column scope query

1. Click the **Specify Row/Column Query** icon in the **Row/Column Scope** field.



2. In the Row/Column Query dialog, select one of the following:
   - **Containment**. Select if you want that the row/column elements would be displayed as in the Containment tree. Owned elements will be collected recursively starting from the elements specified as scope values.
   - **Structure**. Select if you want that the row/column elements would be displayed as in the Structure tree[67]. This option is available only when the SysML Plugin is installed.
   - **Custom**. Select if you want to make your own adjustments. Select the Build Hierarchy check box to recursively execute the specified expression for building multilevel hierarchy starting from the scope as the root.

   > If you have selected **Custom**, specify the expression(see page 1227).



3. Click **OK**.

Specifying relationship line styles

67 https://docs.nomagic.com/display/SYSMLP190SP4/Structure+tab

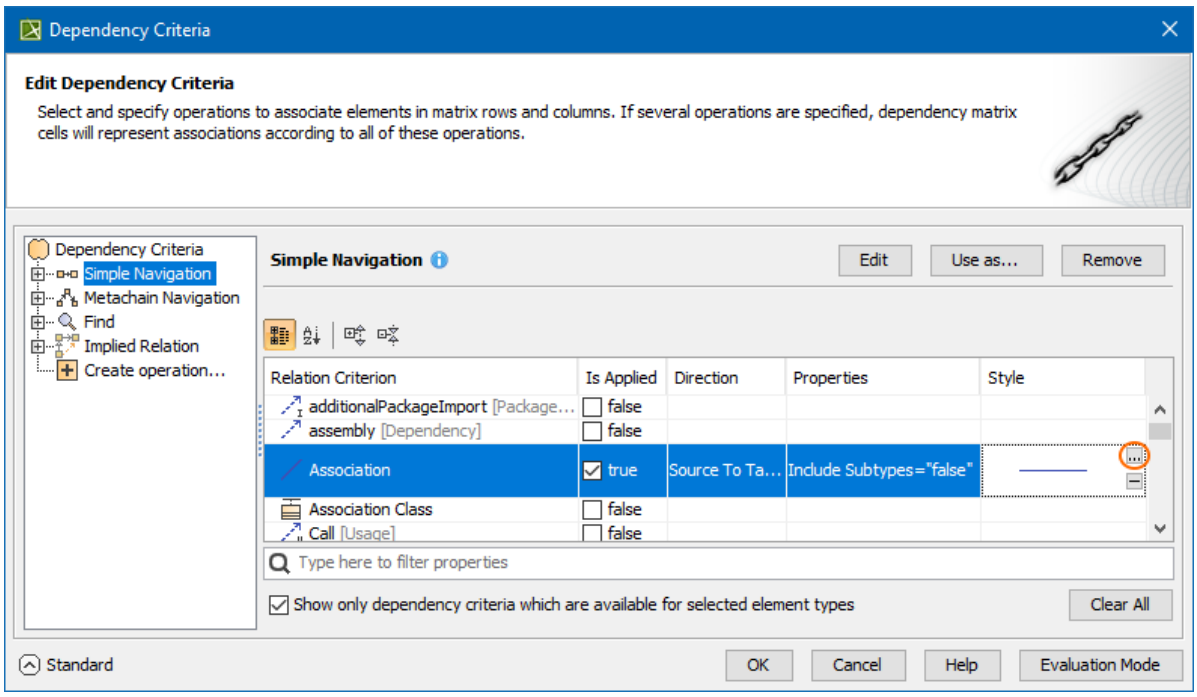
You can specify the colors, line styles, or, alternatively, set an icon for the representation of your relationships in **Dependency Matrices**. To be able to do this, you need to access the **Style** dialog in the first place.

To open the **Style** dialog

1. Open the **Dependency Criteria** dialog by
   - double-clicking the **Legend**.
   - clicking the

     button next to **Dependency Criteria** in the **Criteria** area(see page 360).
2. Open the **Style** dialog by clicking the

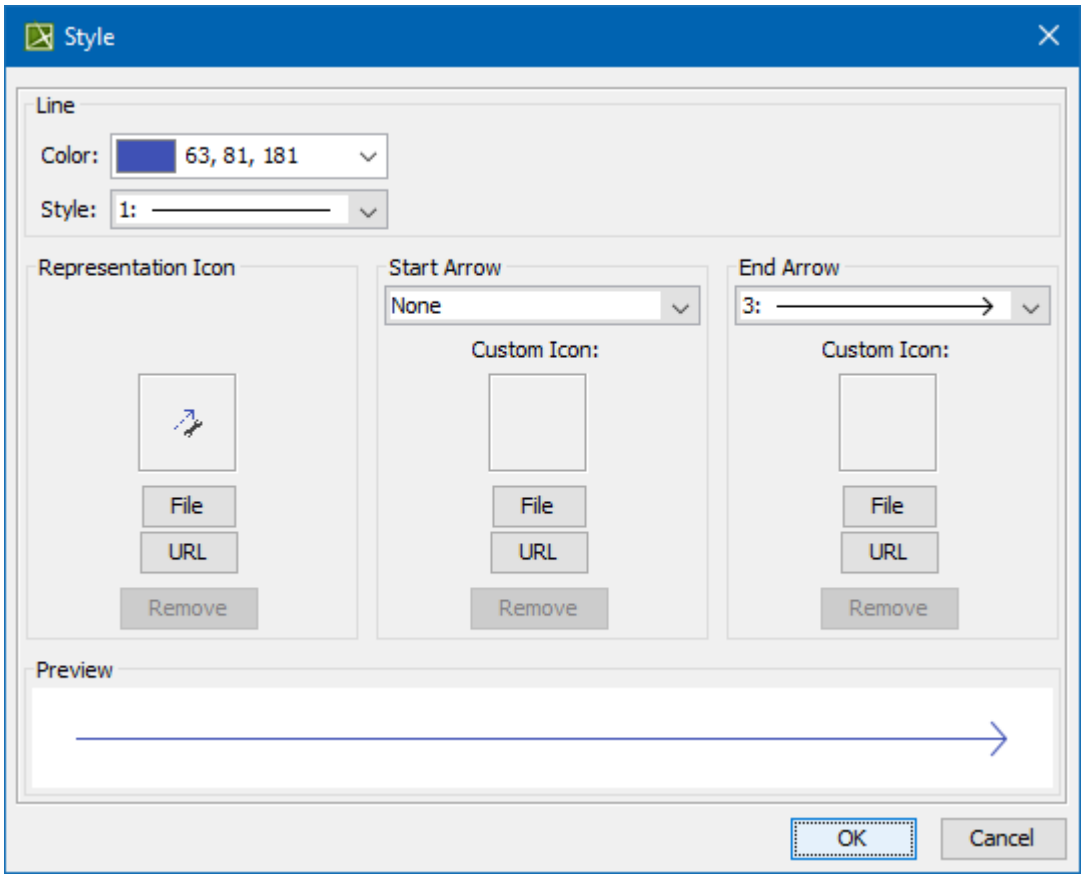
   button in the **Style** column of your selected relation criterion.



3. In the **Style** dialog, customize **Color**, **Style**, **Start Arrow**, **End Arrow** or set **Representation Icon** (see the table(see page 345) below for more information).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

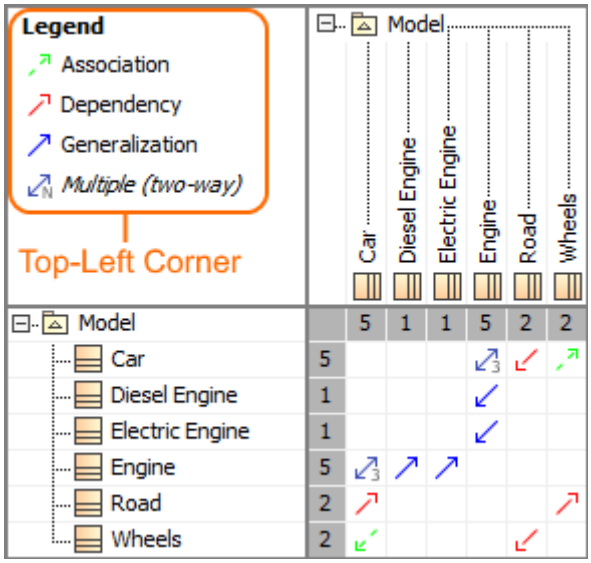
4. When you are done, click **OK** to close the dialog.



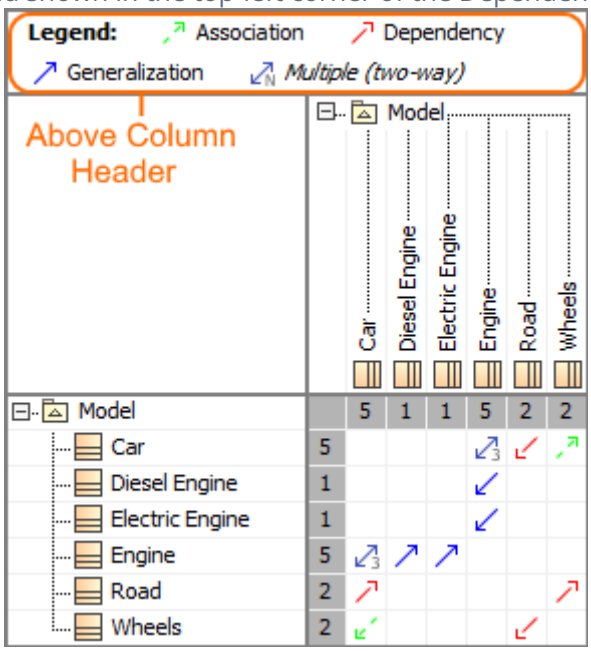
Style Options

Color	Specify the line color.
Style	Specify the line style.
Start Arrow	Specify the arrowhead style.
End Arrow	Specify the arrow end style.
Representation Icon	Represent relationships using an icon. • **File** - upload the icon image from your file system. • **URL** - add a path to the icon available on a shared location. • **Remove** - remove the uploaded icon image or the specified URL.

Your selection is shown in the customized legend in the top-left corner or above the matrix. The legend is displayed by default in all new projects, but if you don't want to see it all the time, simply click the ⚙ button in the Dependency Matrix toolbar(see page 358) and then click **Legend Location** > **Do Not Display**.



The legend shown in the top-left corner of the Dependency matrix.



The legend shown above column header in the Dependency matrix.

Understanding cell content

This page explains the content of the cell in the Dependency matrix. The following sections describe the Dependency matrix areas and cell content:

- Rows, columns, and cells in the Dependency matrix(see page 347)
- Representation of the dependencies in a cell(see page 347)
- Numbers in the Dependency matrix(see page 348)

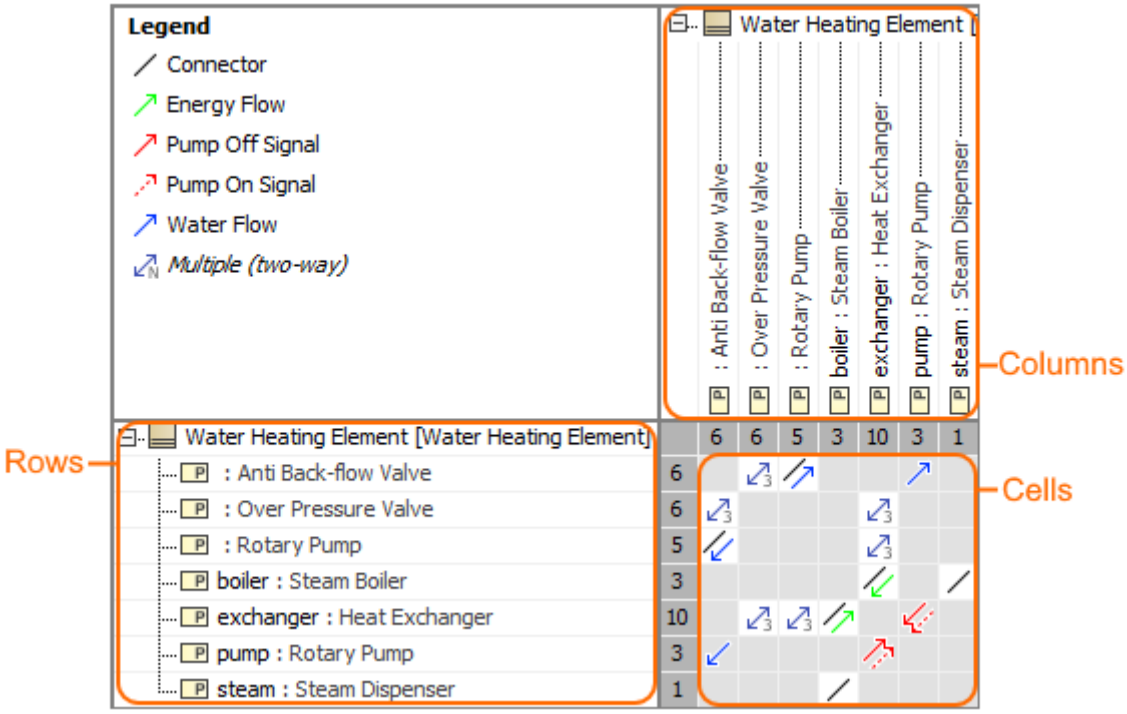
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- The details of a cell(see page 348)

Rows, columns, and cells in the Dependency matrix
The Dependency matrix contains the following areas:

- *Rows* display elements specified in the **Row Element Type** box and **Row Scope** box of the criteria area(see page 360).
- *Columns* display elements specified in the **Column Element Type** box and **Column Scope** box of the criteria area(see page 360).
- *Cells* display dependencies between elements specified in the rows and columns. The content of the cells depends on selected criteria in the **Dependency Criteria** box and the **Direction** box of the criteria area(see page 360).

> **Tip**
> How to specify boxes in the criteria area of Dependency matrix (see page 335)



The main areas of the Dependency matrix. The illustration displays concepts from the SysML Plugin.

Representation of the dependencies in a cell
The default representation of the dependencies in the cells includes the following:

- **Arrowed diagonal**. The direction of the arrow is from the client element to the supplier element.

> **Arrowed diagonal representation**
> The representation of the arrowed diagonal is different when the same elements are connected through:
> - One dependency , e.g., .
> - Two dependencies, e.g., and .
> - More than two dependencies, e.g., , .

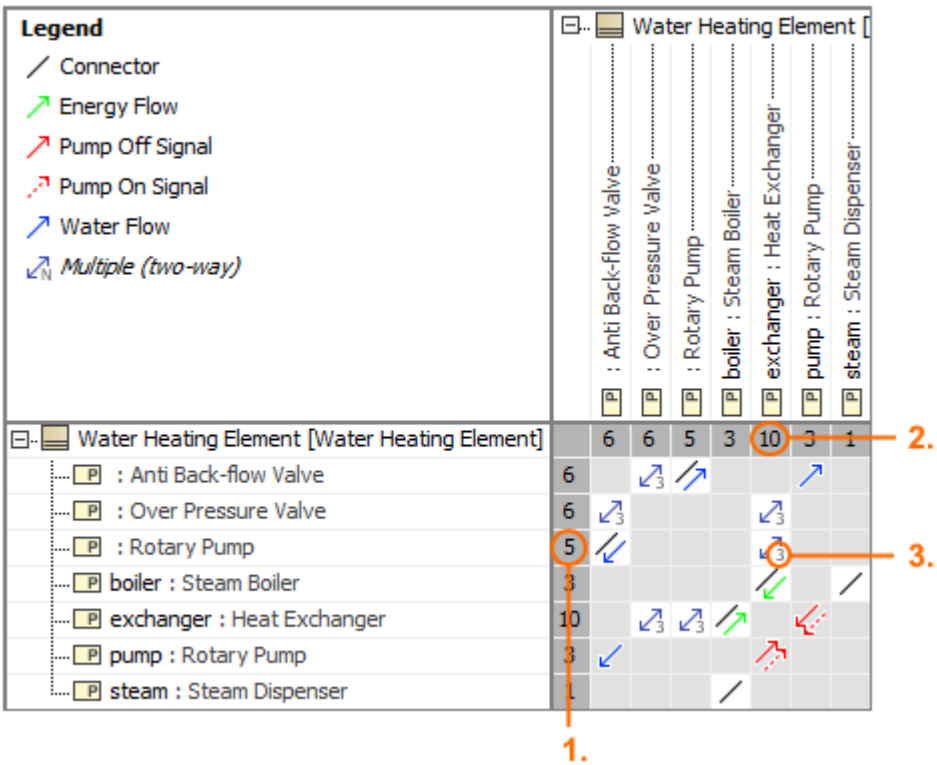
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- **Diagonal**. Elements depend on each other.

> **Tip**
> You can specify the colors and line styles used to represent the relationships in your dependency matrices. How to specify relationship line styles >(see page 343)

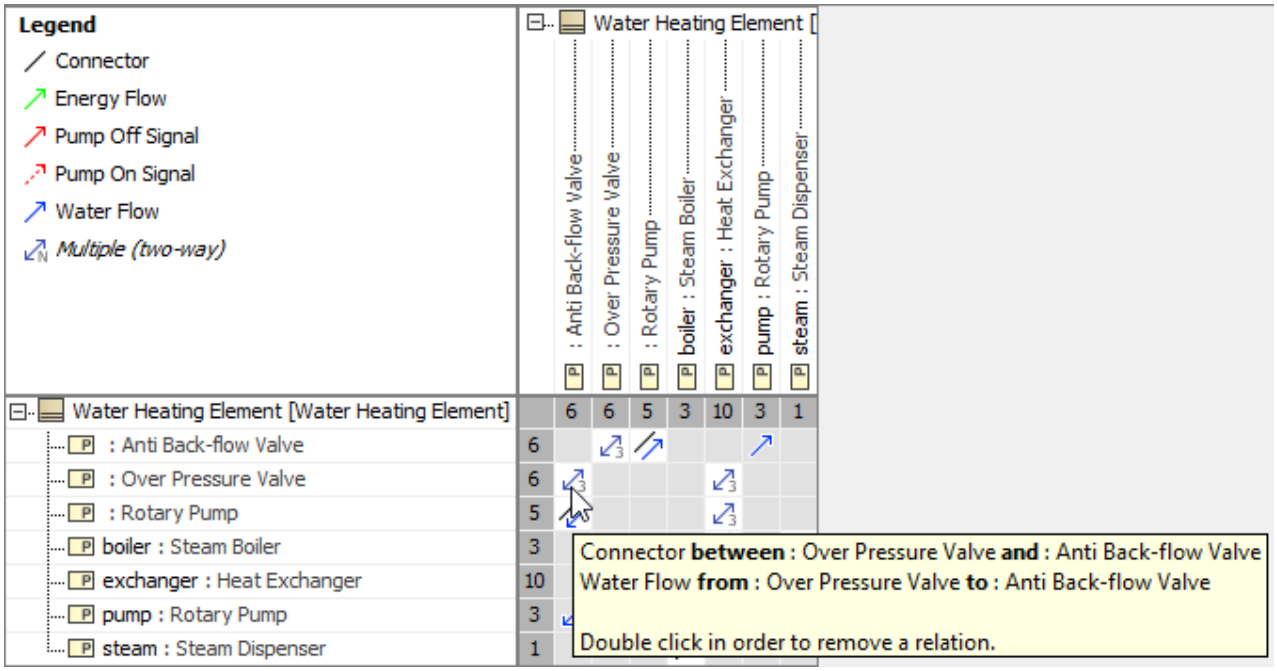
Numbers in the Dependency matrix
There are three types of numbers directly in the matrix (see the following figure):

1. The number of dependencies in the row.
2. The number of dependencies in the column.
3. The number of criteria in the cell.



The details of a cell
Move your mouse over a cell to view details of the dependency (relationship) represented in the cell. The ToolTip displays information about the relationships.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The ToolTip displays information about existing relations between Over Pressure Valve and Anto Black-flow Valve Blocks. The illustration displays concepts from SysML Plugin.

Related pages

- Creating Dependency Matrix(see page 335)
- Using Dependency Matrix(see page 335)
- Dependency Matrix environment(see page 358)

Sample model

The model used in the figures of this page is the **dependencyMatrixSample** model. Download dependencyMatrixSample.mdzip[68].

Modifying relationships in Dependency Matrix

You can edit relationships and element properties directly in matrices. Clicking the selected cell allows for

- Creating or deleting relationships between elements.
- Defining or removing element properties or tags.

This feature not only facilitates management of ordinary relationships between elements, but also allows for a faster creation of traceability links between elements, for example, between requirements and the architecture or requirements and test cases. Such an improvement saves huge amounts of time in comparison to linking elements in diagrams. It significantly increases applicability and usability of matrices.

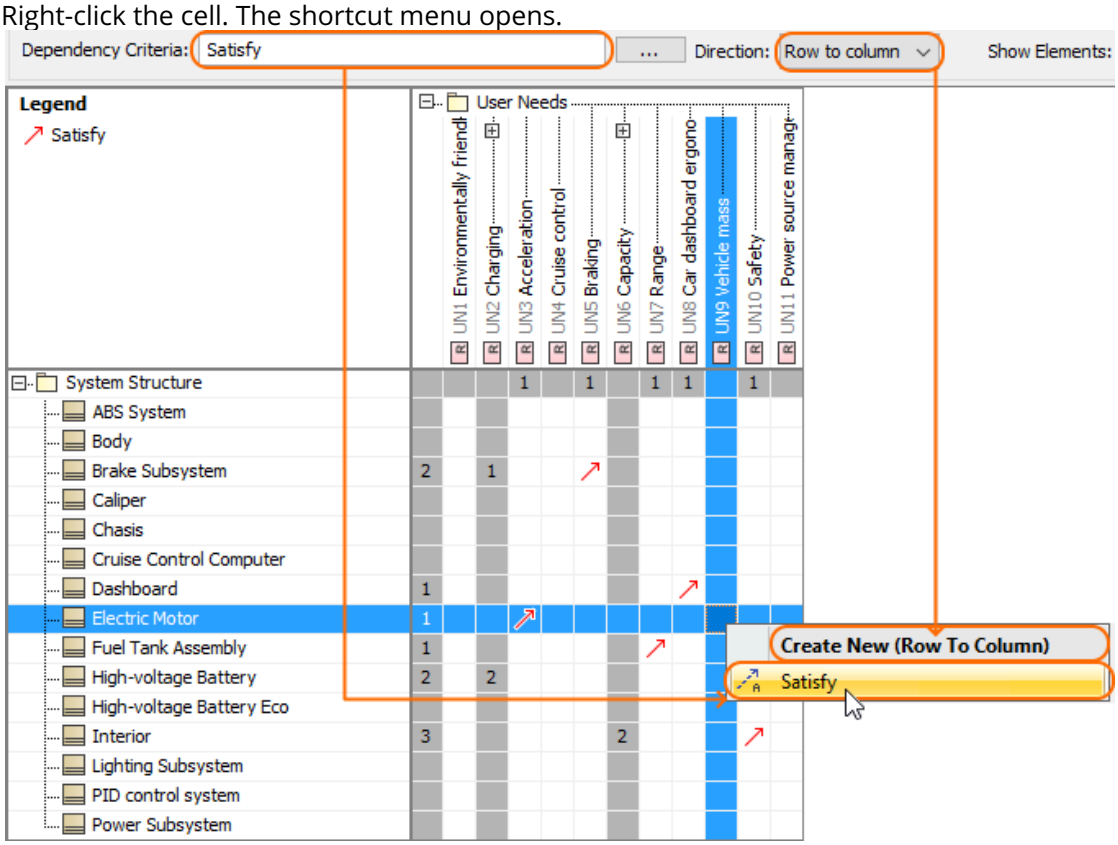
To create a new relationship between model elements in the dependency matrix

68 https://docs.nomagic.com/download/attachments/55847477/dependencyMatrixSample.mdzip?api=v2&modificationDate=1494945196318&version=1

Commands on the shortcut menu of the cell, for creating a new relationship, depend on criteria specified in the **Criteria** area of the matrix:

- Possible relationship types correspond to the types specified as dependency criteria (see solid-line boxes in the following figure).
- Possible directions correspond to the directions specified in the **Direction** drop-down list (see dashed-line boxes in the following figure).

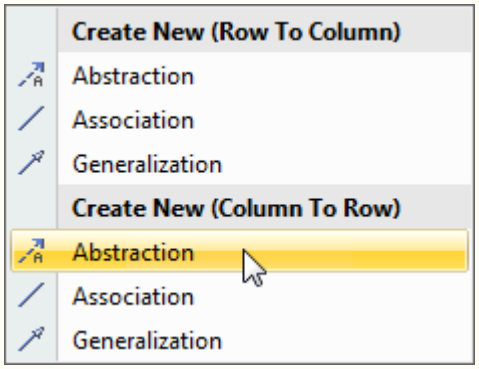
1. Do one of the following:
   - Double-click the cell. If only one relationship type and direction is available, the relationship is created instantly. If several relationship types or directions are available, the shortcut menu opens.
   - Right-click the cell. The shortcut menu opens.



2. From the shortcut menu, under **Create New (Row To Column)** or **Create New (Column To Row)**, select a relationship type you need to create.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

If you have specified to display in the matrix, relationships with both directions, the shortcut menu of the cell will look like the one in the following figure.

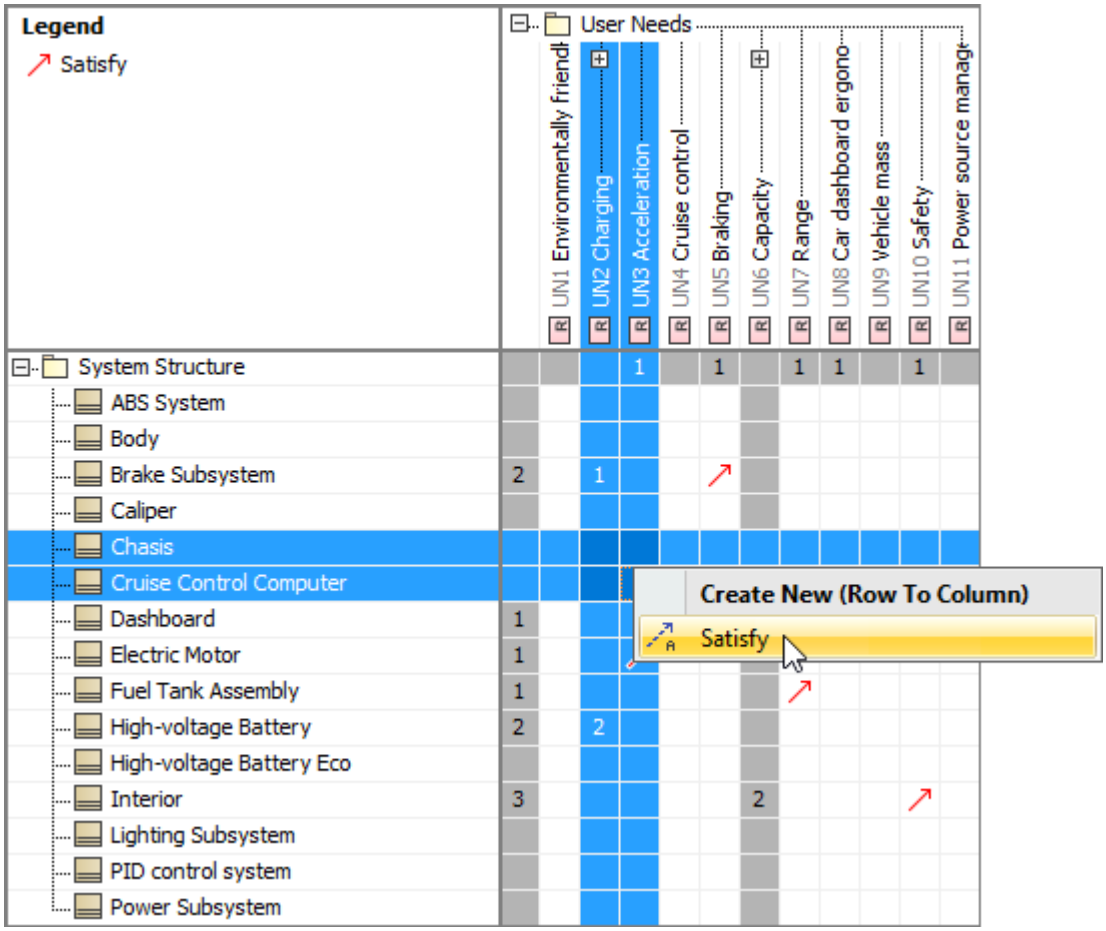


Thus make sure you have selected a relationship type with relevant direction.

To create multiple relationships between model elements simultaneously in the dependency matrix

1. Press and hold Ctrl on your keyboard to be able to select multiple row and column elements.
2. Do one of the following:
   - Double-click the cell while holding Ctrl. If only one relationship type and direction is available, the relationship is created instantly. If several relationship types or directions are available, the shortcut menu opens.
   - Right-click the cell. The shortcut menu opens.
3. From the shortcut menu, select the relationship type to create.

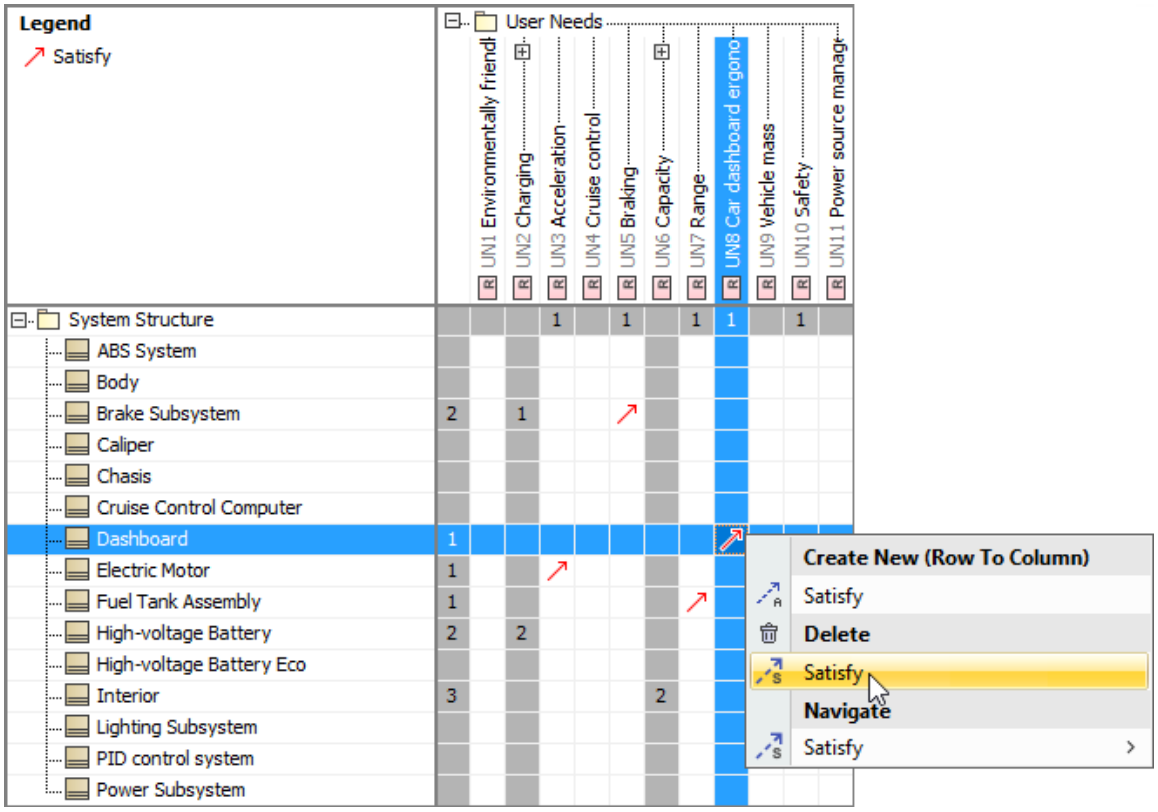
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



To remove a relationship between model elements from the dependency matrix

1. Do one of the following:
   - Double-click the cell. If only one relationship type and direction is available, the relationship is removed instantly. If several relationship types or directions are available, the shortcut menu opens.
   - Right-click the cell. The shortcut menu opens.
2. From the shortcut menu, under **Delete**, select the relationship you need to remove. The relationship is removed both from the matrix and the model.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



To remove all relationships of a cell

1. Double-click or right-click the cell. The shortcut menu opens.
2. From the shortcut menu, under **Delete**, select **Delete All**. The relationships are removed both from the matrix and the model.

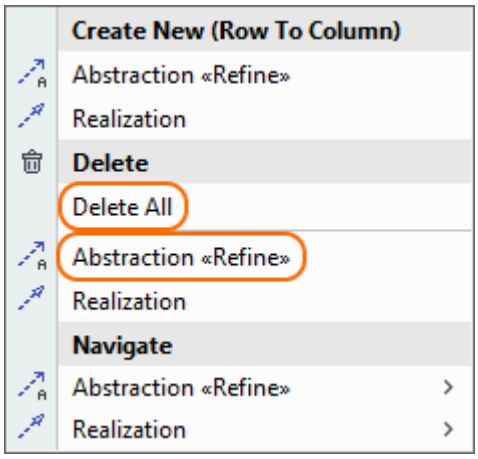
To remove relationship(s) of multiple cells

1. Press Ctrl and select multiple row and column elements.
2. Double-click or right-click the last active cell. The shortcut menu opens.
3. In the shortcut menu, under **Delete**, do one of the following:

- Select the specific relationship type to remove.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

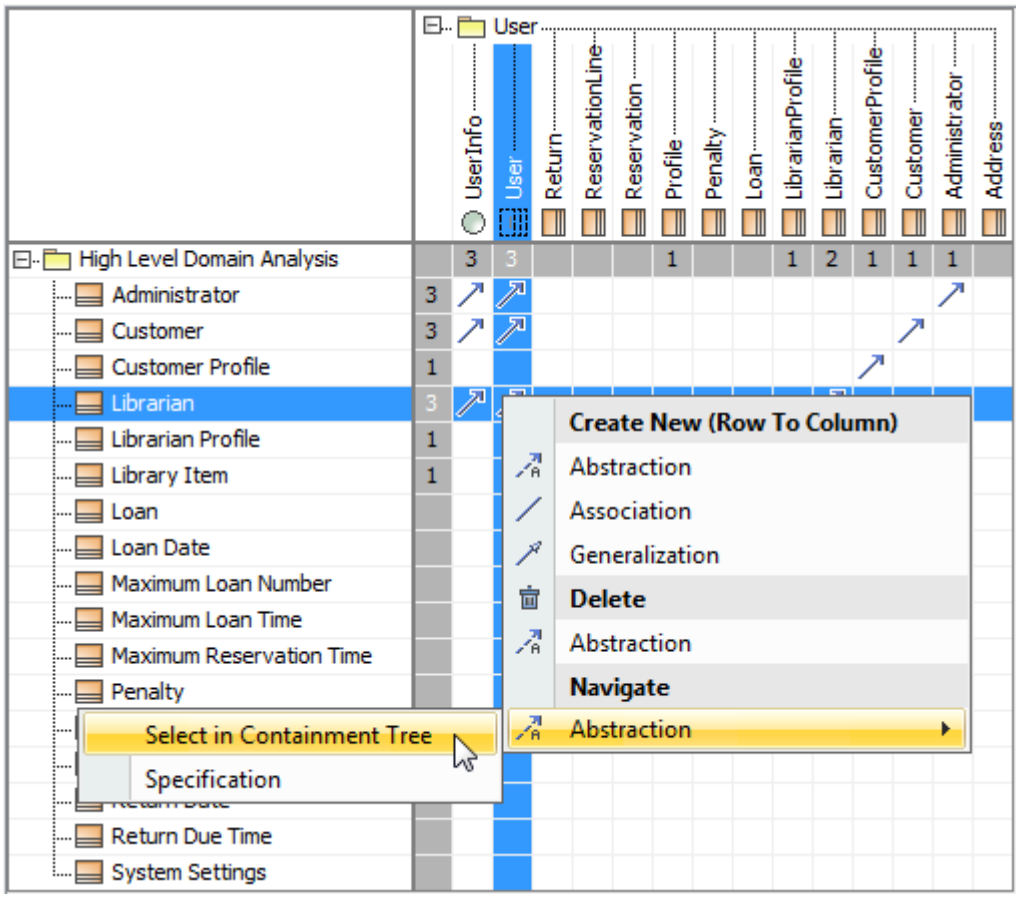
- Click **Delete All** to remove all relationships at once.



Navigation from cell

To select a relationship in the Containment tree(see page 32)

1. Right-click the cell. The shortcut menu opens.
2. On the shortcut menu, under **Navigate**, point to the relationship, and click **Select in Containment Tree**.



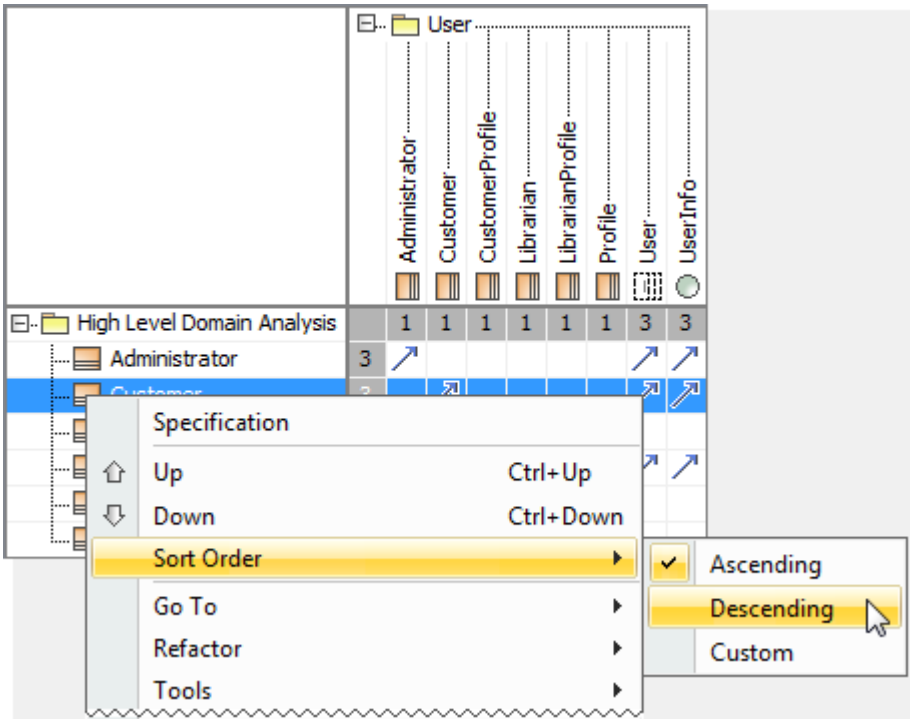
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To open the Specification window(see page 913) of a relationship in the cell

1. Right-click the cell. The shortcut menu opens.
2. On the shortcut menu, under **Navigate**, point to the relationship, and click **Specification**.

Rearranging rows and columns

You can sort both rows and columns either in ascending or descending order (see the following figure) or create your own order by shifting selected items (either grouped or non-grouped) from one place to another. The sort order persists after changing the row or column owner's display mode, which can be defined by the **Row Owner Display Mode** or **Column Owner Display Mode** property in the matrix Specification window(see page 913).



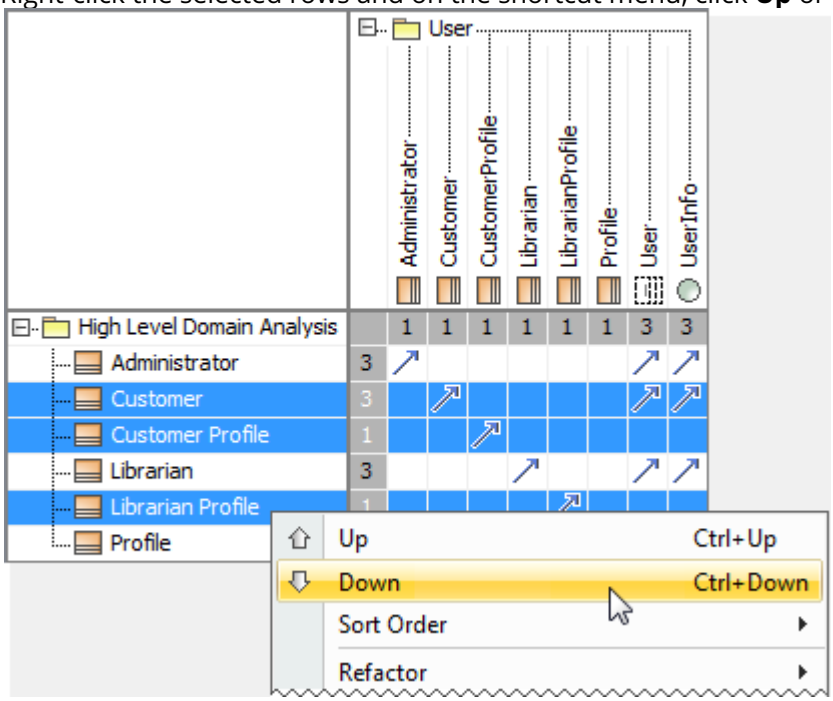
To shift rows up or down

1. Select rows you need to move.

   - To select *nonadjacent rows*, select a single row, and then hold down the Ctrl key while you click other rows that you want to select.
   - To select *adjacent rows*, select a single row, and then hold down the Shift key while you click other rows that you want to select

2. Do either:

- Right-click the selected rows and on the shortcut menu, click **Up** or **Down**.



- Press Ctrl+Up Arrow or Ctrl+Down Arrow.

> After the manual rearrangement of rows, the sort order of the rows automatically changes to **Custom**.

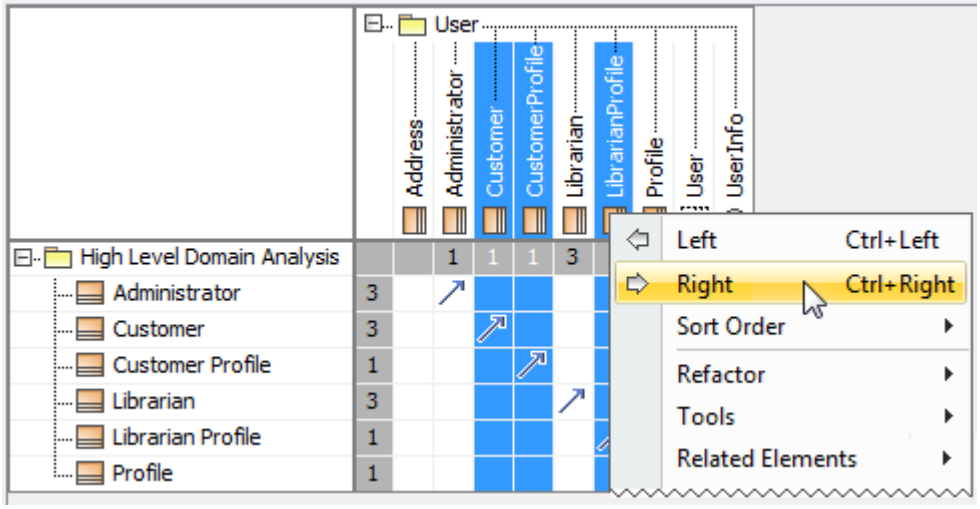
To shift columns left or right

1. Select columns you need to move.

   > - To select *nonadjacent columns*, select a single column, and then hold down the Ctrl key while you click other columns that you want to select.
   > - To select *adjacent columns*, select a single column, and then hold down the Shift key while you click other columns that you want to select

2. Do either:

- Right-click the selected columns and on the shortcut menu, click **Left** or **Right**.



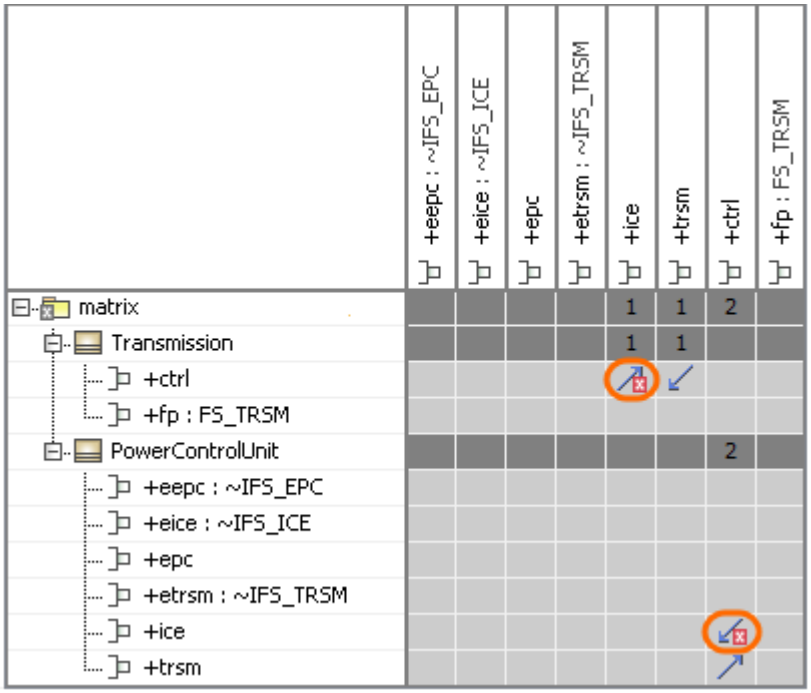
- Press Ctrl+Left Arrow or Ctrl+Right Arrow.

> ⚠ After the manual rearrangement of columns, the sort order of the columns automatically changes to **Custom**.

Validation in Dependency Matrix

The validation(see page 1306) results for relationships are visualized in matrices. Matrix cells are highlighted whenever relationships violate validation rules. This feature is useful, for example, to check which test cases for requirements verification fail, to check if dependencies are valid or up-to-date when doing gap analysis.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



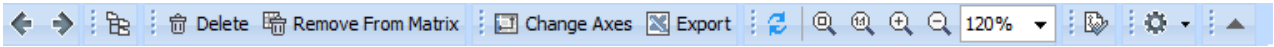
Related pages

- Validation(see page 1306)

Dependency Matrix environment

The dependency matrix environment elements, such as toolbar, **Criteria** area, and shortcut menus are described in the following pages:

Dependency Matrix toolbar



See the detailed descriptions of the dependency matrix toolbar buttons in the following table.

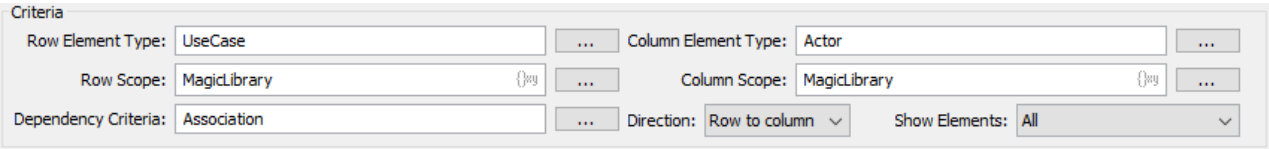
Button	Button name	Description
	Back (Alt+Left)	Navigates back to the previous diagram that was open in the current diagram tab.
	Forward (Atl+Right)	Navigates forward to the diagram in the current diagram tab.
	Select in Containment Tree (Alt+B)	Opens the Containment tab(see page 32) in the Model Browser(see page 29) and select the current diagram or symbol in the Containment tree.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Button	Button name	Description
	Delete (Ctrl+D)	Deletes the selected element from the matrix and model. If you undo the deletion, you should refresh the matrix to make the deleted elements appear in the matrix again.
	Remove From Matrix (Delete)	Removes the selected element from the matrix. The selected element is not removed from the model.
	Change Axes	Transposes rows and columns.
	Export	Exports a matrix as a Comma Separated Values (.csv) file. The file can be opened with Microsoft Excel.
	Refresh	Refreshes the matrix after updating the criteria or data.
	Fit in Window (Ctrl+W)	Fits the matrix in the active window size.
	Zoom 1:1 (Ctrl+Slash, Ctrl+NumPad /)	Shows the actual size of the table.
	Zoom In (Ctrl+Plus, Ctrl+NumPad +, Ctrl+Equals)	Enlarges the matrix maintaining aspect ratio.
	Zoom Out (Ctrl+Minus, Ctrl+NumPad -)	Reduces the matrix maintaining aspect ratio.
120%	Percentage drop-down box	Shows the ratio of size. Click to select the size ratio from the percentage list or enter the value.
	Validate Diagram	Validates the matrix against all validation rules stored in the model, except those that are ignored.
	Options	Expands the menu with other options.
	• Specification • Legend Location	Opens the matrix Specification window wherein you can specify matrix properties. Lists legend location options and selection to not show it.

Button	Button name	Description
	Suppress/Expand Criteria Area	Suppresses or expands the matrix **Criteria** area.

Dependency Matrix Criteria area



See the descriptions of the dependency matrix Criteria area in the following table.

Element Name	Element Type	Description
Row Element Type	Text box with	Click the ... button to select an element type or multiple element types to show in rows of the dependency matrix.
Column Element Type	Text box with	Click the ... button to select an element or multiple element types to show in columns of the dependency matrix.
Row Scope	Text box with	Click the ... button to define a scope of the model (packages/profiles) from which elements should be displayed in rows of the dependency matrix. Click to to open the **Query** dialog and specify the criteria that define row scope.
Column Scope	Text box with	Click the ... button to define a scope of the model (packages/profiles) from which elements should be displayed in columns of the dependency matrix. Click to to open the **Query** dialog and specify the criteria that define column scope.
Dependency Criteria	Text box with	Click the ... button to define what relationships between row and column elements you wish to display in the dependency matrix cells.

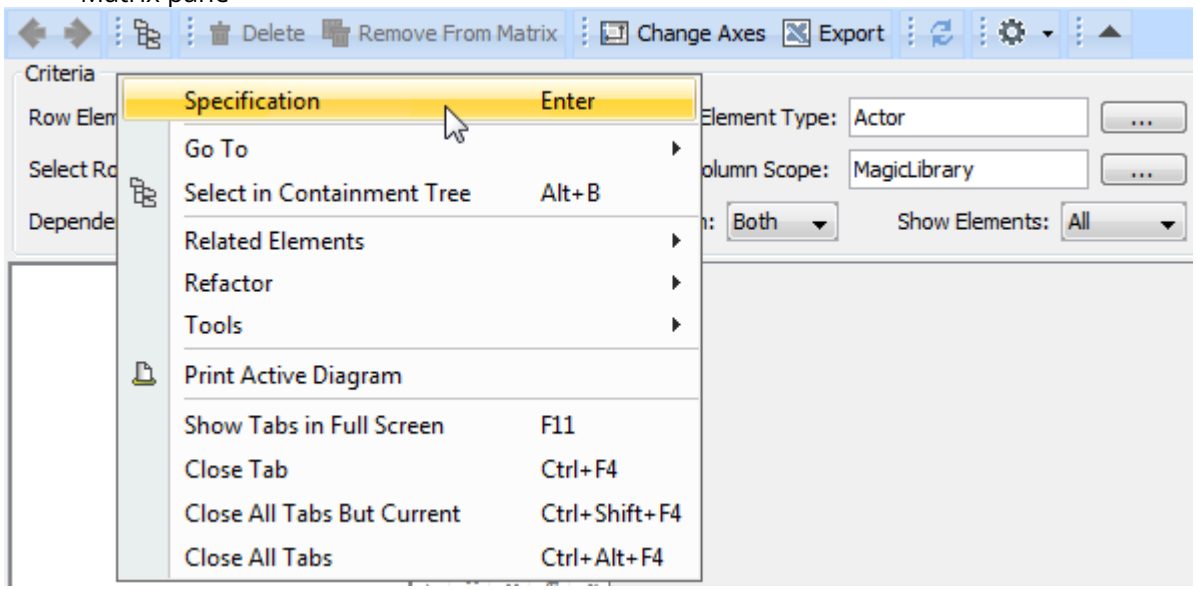
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Element Name	Element Type	Description
Direction	Drop-down list	Select a direction of relationships for the relationship analysis and representation in the dependency matrix. Be aware that new relationships will be created with the selected direction.
Show Elements	Drop-down list	Select to show only related (by a selected dependency criteria), only non-related, or all elements.

Dependency Matrix shortcut menu

To open the dependency matrix shortcut menu

- Right-click an empty space on one of the following
  - Dependency Matrix Criteria area(see page 360)
  - Matrix pane



Command	Description
Specification (Enter)	Opens the Specification window(see page 913) of a dependency matrix.
Go To	Navigates back or forward to the diagram that was open in the current diagram tab, finds and navigates to diagrams wherein the matrix is used, or to the matrix owner in the Containment tree(see page 32), or to a hyperlinked object, or to an element to which the selected element is related through a traceability relationship.

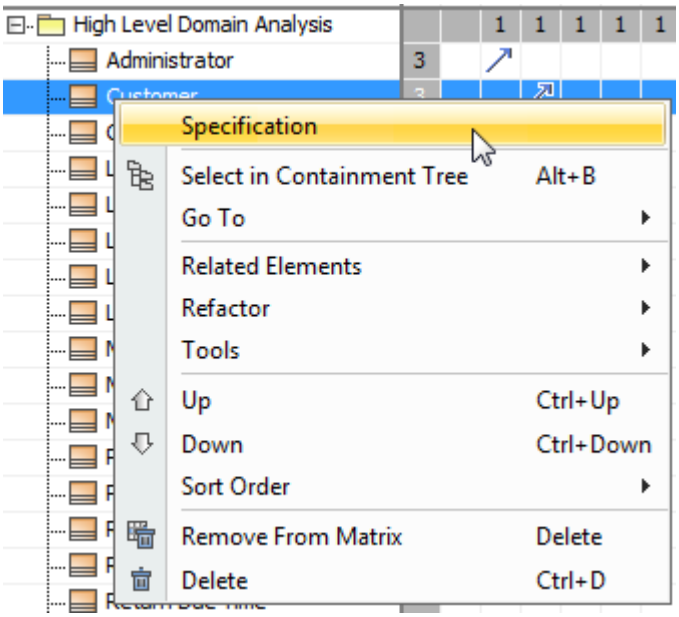
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Command	Description
Select in Containment Tree (Alt+B)	Selects the dependency matrix in the Containment tree(see page 32) of the Model Browser(see page 29).
Related Elements	Allows selecting whether to create a Relation Map(see page 364) for the matrix or search for model elements which depends on the matrix or are used by the matrix.
Refactor	Converts or replaces(see page 936) the matrix to a diagram indicated by the user.
Tools	Opens a list of tools (see page 1083)that are available for the matrix.
Print Active Diagram	Prints a matrix.
Show Tabs in Full Screen (F11)	Shows a matrix in the full screen mode. Click **Close Full Screen** to return to the previous view.
Close Tab (Ctrl+F4)	Closes an active matrix.
Close All Tabs But Current (Ctrl+Shift+F4)	Closes all diagrams except an active one.
Close All Tabs (Ctrl+Alt+F4)	Closes all opened diagrams.

Row and column element shortcut menu

To open a row/column element shortcut menu

- Right-click the appropriate row/column.



Command (Shortcut key)	Description
Specification	Opens the element's Specification window(see page 913).
Select in Containment Tree	Selects the element in the Containment tree(see page 32) of the Model Browser(see page 29).
Go To	Finds and navigates to a diagram which is owned by the selected element, or to the one, wherein the element shape is drawn, or to a hyperlinked object, or to an element to which the selected element is related through a traceability relationship. For more information, see • Element usage in diagrams(see page 1278) • Inserting hyperlinks(see page 1008), • Traceability properties in Go To submenu(see page 1209)
Related Elements	Opens the dialog for defining options to search for specified usages or dependencies of the selected element. For more information, see Analyzing usages and dependencies(see page 1192).
Refactor	Converts or replaces(see page 936) the selected element to an element indicated by the user.
Tools	Opens a list of tools(see page 1083) that are available for the selected element.

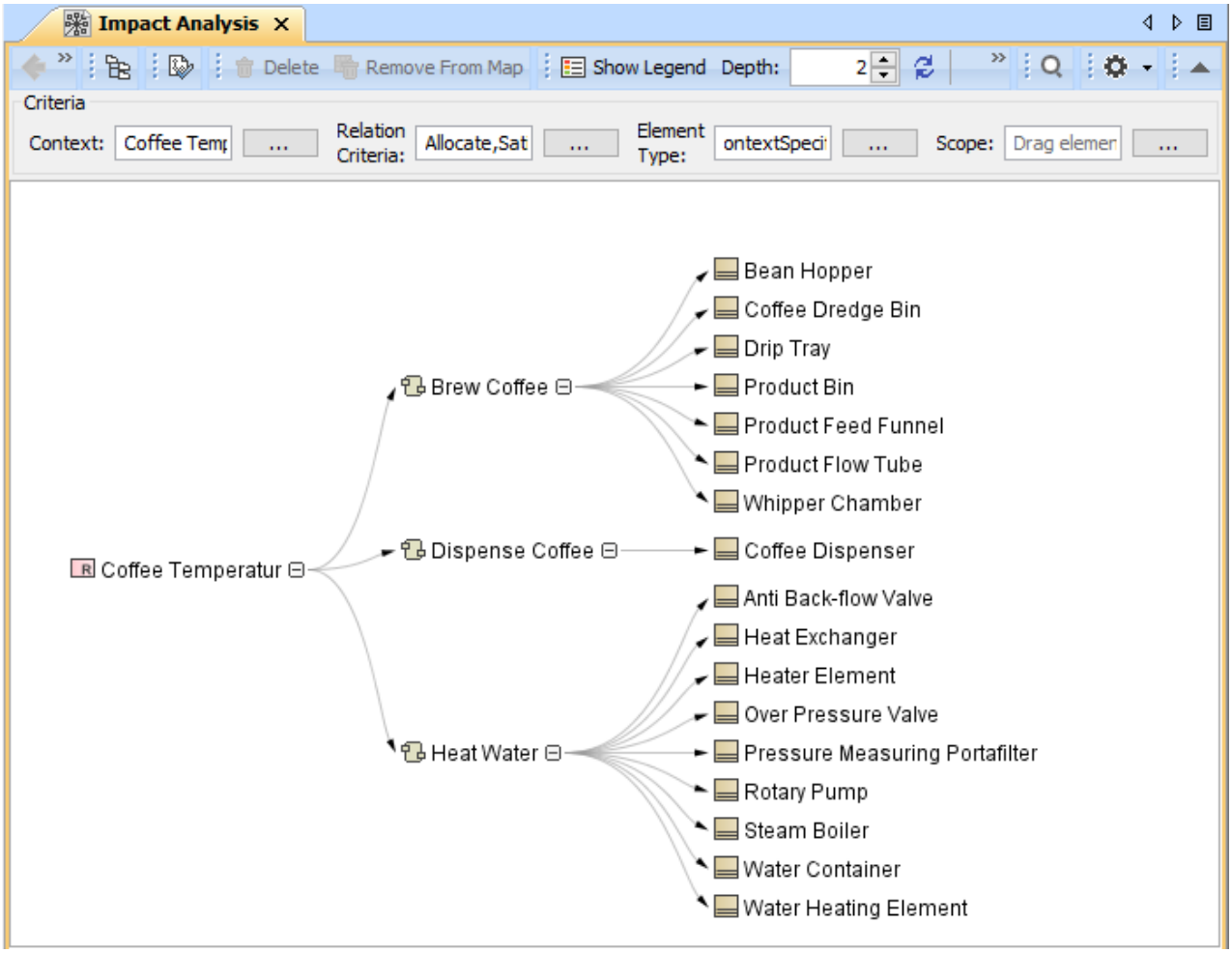
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Command (Shortcut key)	Description
Up (Ctrl+Up Arrow)	Moves selected rows (either grouped or non-grouped) up. The command is available on the shortcut menu of one or more rows.
Down (Ctrl+Down Arrow)	Moves selected rows (either grouped or non-grouped) down. The command is available on the shortcut menu of one or more rows.
Left (Ctrl+Left Arrow)	Moves selected columns (either grouped or non-grouped) left. The command is available on the shortcut menu of one or more columns.
Right (Ctrl+Right Arrow)	Moves selected columns (either grouped or non-grouped) right. The command is available on the shortcut menu of one or more columns.
Sort Order	Point to the command and select the order for sorting rows or columns. It can be either Ascending or Descending. Selecting Custom does not affect the order of rows/columns. The Custom order is automatically selected after shifting at least one row or column to from one place to another. The sort order persists after changing the row or column owner's display mode, which can be defined by the **Row Owner Display Mode** or **Column Owner Display Mode** property in the matrix Specification window(see page 913).
Remove from Matrix	Removes the selected element from the matrix.
Delete	Deletes the selected element from both the matrix and the model. If you undo the deletion, you should refresh the matrix to make the deleted elements appear in the matrix again.

Relation Map

The main purpose of all relation maps is to review and analyze relations among the elements and create new elements directly in the relation map. All modeling tools allow you to create the Relation Map Diagram, which provides you with a rapid review, analysis, and creation of relationships among the elements of the entire model. See the following figure.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The example of Relation Map Diagram. It is created by using the *CoffeeMachine sample model* that comes only with SysML Plugin.

> You can create other predefined relation maps (Activity Decomposition, Structure Decomposition, Instance, Requirement Containment, Requirement Derivation Maps) only if you have the SysML Plugin[69] installed. Learn more about predefined relation maps in SysML Plugin >>[70]

The following procedures explain how to work with relation maps.

Creating relation map

To create a relation map

1. Select an element, for which you need to create the relation map. This element is the context of the relation map.

69 https://docs.nomagic.com/display/SYSMLP190SP4/SysML+Plugin+Documentation
70 https://docs.nomagic.com/display/SYSMLP190SP4/Predefined+Relation+Maps

2. Do one of the following:
    - From the main menu, select **Diagrams** > **Create Diagram**. Type “rel” and press **Enter**.
    - On the main toolbars, click the **Create Diagram** button. Type “rel” and press **Enter**.
    - Press Ctrl+N. Type “rel” and press **Enter**.
    - On the **Analyze** menu, click **Create Relation Map**.
    - Right-click the element and from the shortcut menu select **Create Diagram** >**Analysis Diagrams** > **Relation Map**.
    - Right-click the element and from the shortcut menu select **Related Elements** >**Create Relation Map**.

The newly created relation map opens on the right side of the application window.

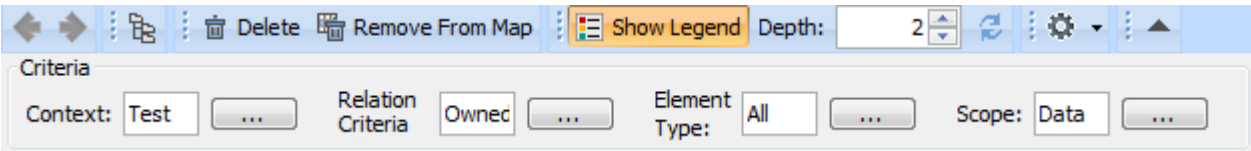
- The context of the relation map automatically becomes the owner of it when using any of the last two approaches.
- When using other approaches, you can specify another element as the owner.
- The actions relevant for diagrams are valid for relation maps as well.

Specifying criteria, layout, and depth

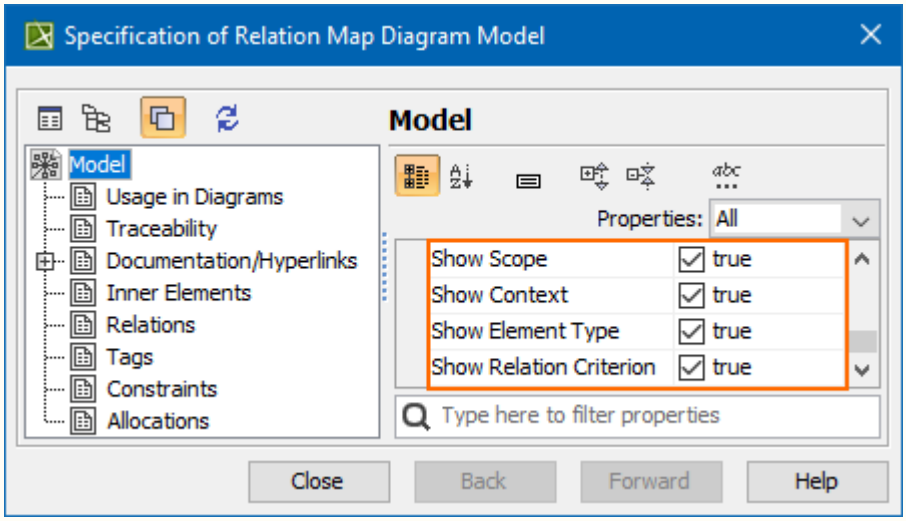
After creating a relation map, you need to specify criteria, layout, and depth for it.

To change the relation map properties

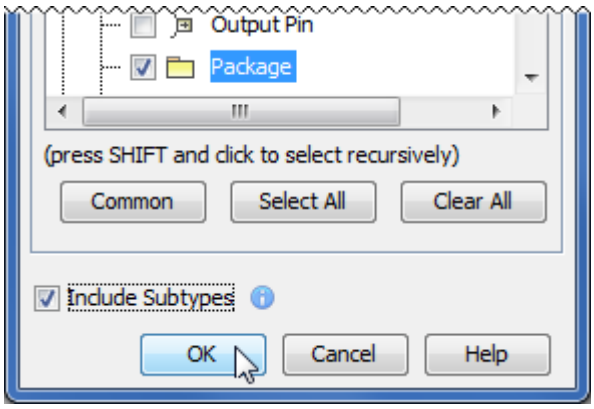
- Right-click the diagram pane and from shortcut menu, select **Specification**, or click on the Options toolbar and then select **Specification**. Change the property values in the Specification window of the relation map.
- Specify the most popular property values by using the relation map environment capabilities, such as toolbars and the **Criteria** area.

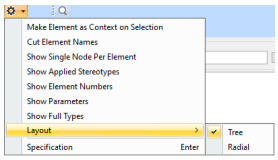
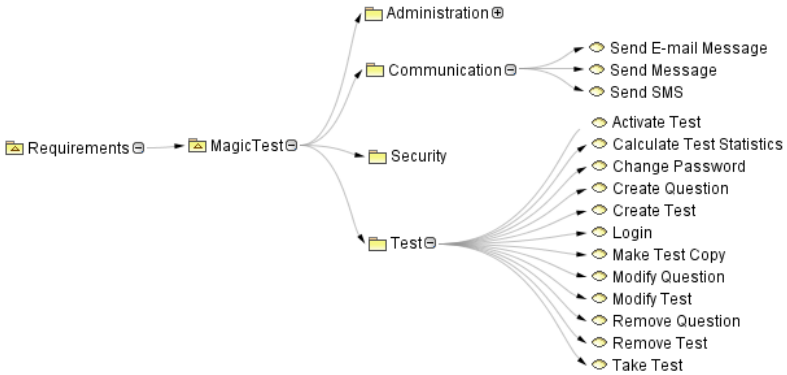
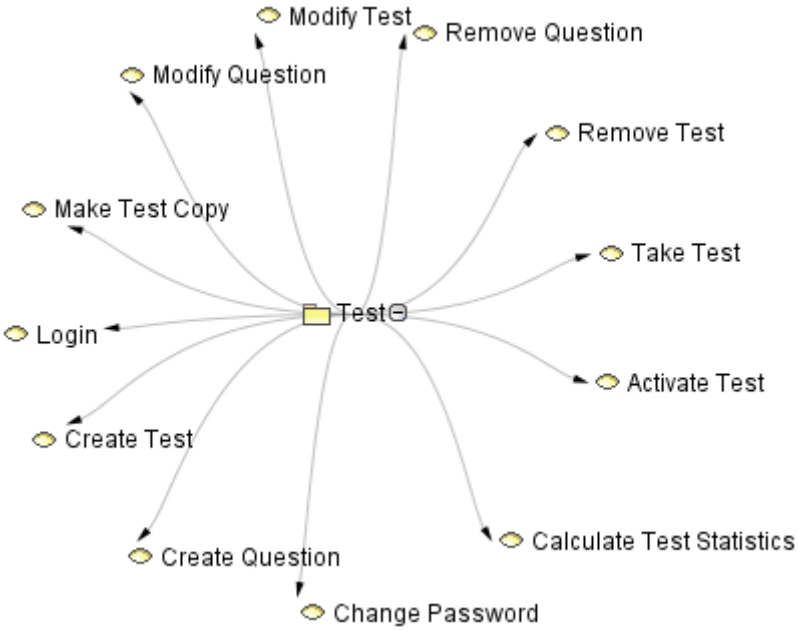
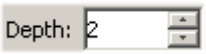


You can choose whether you want to show or hide Context, Relation Criteria, Element Type, and/or Scope. In the Relation Map **Specification** window, set the needed check-box to true or false.



Element name	Element type	Description
Context	Text box with ...	Click to select the main element from which the structure is started, that is, the context of the structure. You can change the context(see page 370).
Relation Criteria	Text box with ...	Click to select relation criteria that will be displayed on the relation map. You can select to show implied relationships(see page 335) – logical relationships calculated from inherited or inner elements of the model element in order to provide additional information.

Element name	Element type	Description
Element Type	Text box with ...	Click to select element types that will be displayed on the relation map. Standard and custom subtypes of the selected element types can be optionally included when displaying them on the relation map. For example, if you want to display packages as well as profiles, models, smart packages, and other custom subtypes of the Package, select this type and then click to select **Include Subtypes** check box. 
Scope	Text box with ...	Click to select packages from which the relation map structure will be shown.

Element name	Element type	Description
	Command on the Options button menu	Select a the relation map layout: • Tree.  • Radial. 
Depth 	Spin box	Type or select the level of the relation map branches that will be automatically expanded.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Changing the context

Context is the element, from which a relation map is drawn and the analysis is started. You can set any element as the context.

To set an element as the context of the relation map

- Drag the selected element from the Model Browser to the relation map.
- In the Criteria area, click the button next to the Context box and then select the element in the **Select Context Element** dialog.
- If the **Make Element as Context on Selection** option is turned on, click any node to make it the new context of the of the structure. To turn this option on, click on the toolbars and select **Make Element as Context on Selection**.

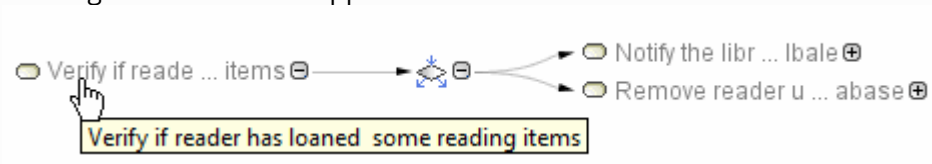
Cutting element names

The element names can be cut if they are too long, which makes the relation map difficult to read.

To cut element names

1. Click in the Diagram toolbar.
2. Select the **Cut Element Names** option.

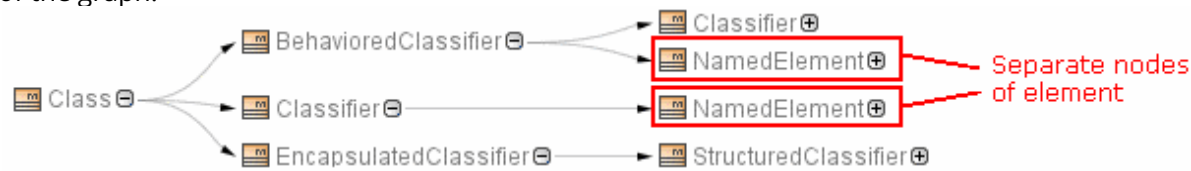
To see the entire name of an element, move the pointer over the element name. A ToolTip showing the entire name appears.



Showing single node per element

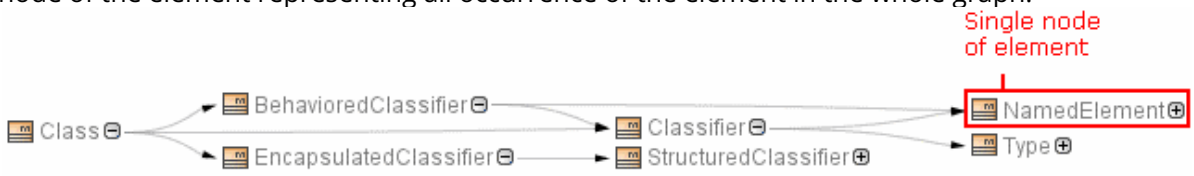
To represent element node in relation map

- Turn the **Show Single Node Per Element** option off in the , if you need to see separate nodes of the same element representing every occurrence of the element in different branches of the graph.



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Turn the **Show Single Node Per Element** option on in the , if you need to see only one node of the element representing all occurrence of the element in the whole graph.



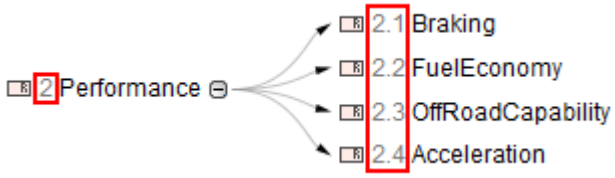
Displaying element numbering

To show element numbering

---

1. Click in the Diagram toolbar.
2. Select the **Show Element Numbers** option.

The element numbering appears before its name.



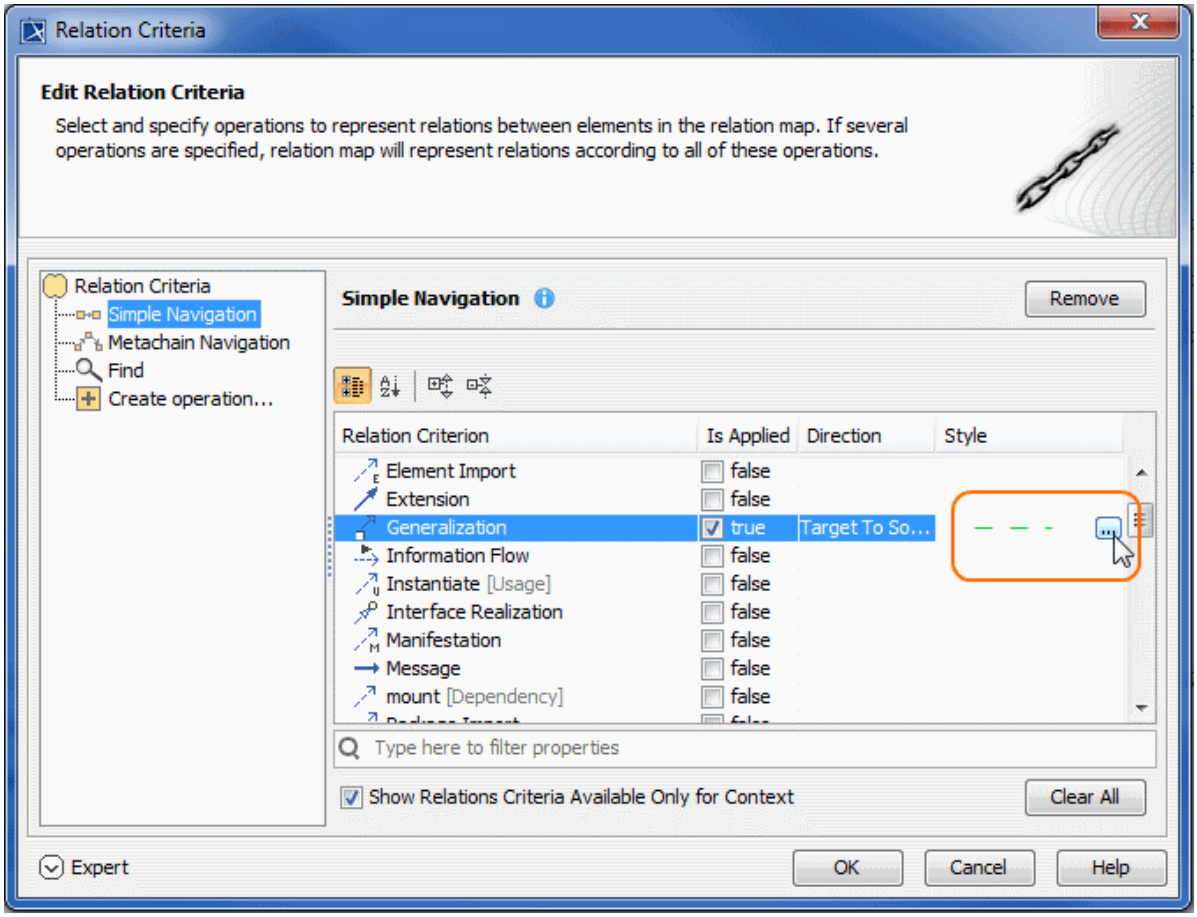
Specifying colors and legend

As of version 18.3, you can specify the colors and line styles to be used for representing the relationships in your relation maps. Your selection is displayed in the legend in the top right corner of the diagram pane.

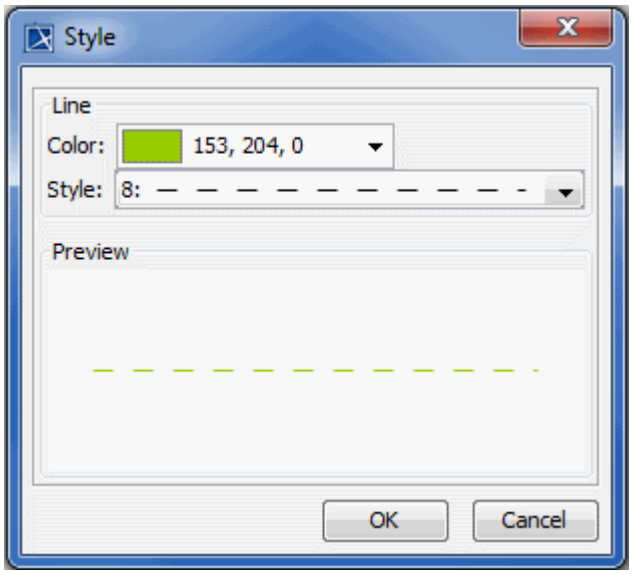
To specify colors and line styles

---

1. Open the **Relation Criteria** dialog by
   - double-clicking the legend header (**Legend**).
   - clicking the button next to **Relation Criteria** in the **Criteria** area(see page 1227).
2. Open the **Style** dialog by clicking the button in the **Style** column of your selected relation criterion.

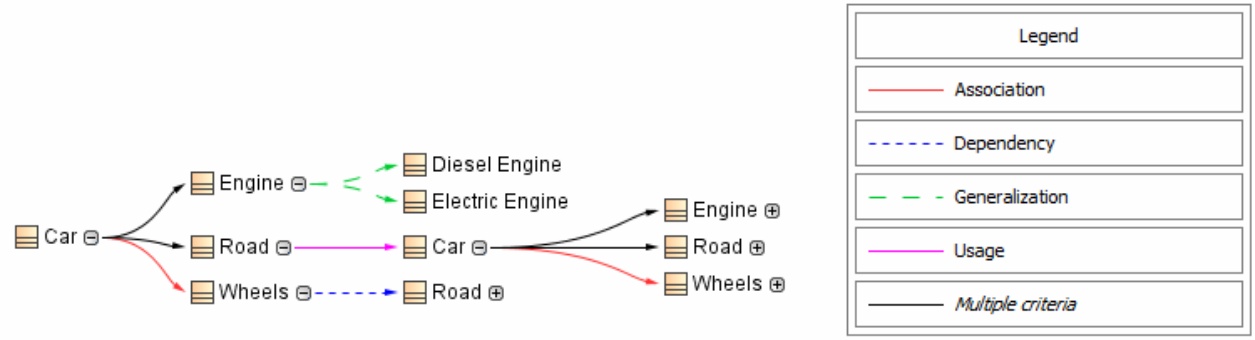


3. Change the line color and style to what you like and click **OK** when you're done.



4. When you've changed line colors and styles for all the relation criteria you needed to, click **OK** to close the **Relation Criteria** dialog.

The customized legend helps you identify relation criteria. When working with relation maps, the Show Legend button is clicked to select by default. If you don't want to see the legend all the time, simply click the button to cancel the selection.



Manipulations in the relation map

You can manipulate and analyze your relation map in the following ways:

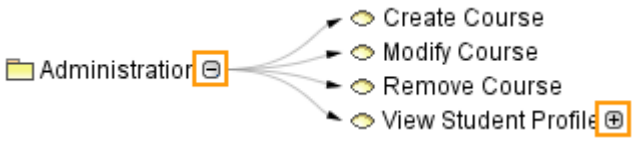
- Suppress(see page 373)/ expand (see page 373)branches
- Move the whole structure(see page 373)/ the selected Node(see page 373)
- Zoom in(see page 374)/ zoom out(see page 374)/ Fit in Window(see page 374)/ Zoom 1:1(see page 374)
- Hide(see page 374)/ remove(see page 374) an element
- Restore the layout(see page 374)
- See invalid elements(see page 375)
- Select an element in the Containment tree(see page 375)
- Decompose an element(see page 375)
- Opening an existing element list for the unnamed element(see page 375)
- Create element of the same type(see page 375)

To suppress branches

- Click near the Node.

To expand branches

- Click near the Node.



To move the relation map structure

- Click on an empty place in the Relation Map and drag it.
  The whole structure will be moved appropriately.

To move the selected Node

- Click a Node and drag it.
  The selected Node will be moved appropriately.

To zoom in the relation map structure

Do one of the following:

- Hold down Ctrl while scrolling up.
- Click the Zoom In button on the toolbar.
- Press Ctrl+NumPad +.

To zoom out the relation map structure

Do one of the following:

- Hold down Ctrl while scrolling down.
- Click the Zoom Out button on the toolbar.
- Press Ctrl+NumPad -.

To fit in window

Do one of the following:

- Click the Fit in Window button on the toolbar.
- Press Ctrl+W.

To zoom 1:1

- Click the Zoom 1:1 button on the toolbar.
- Press Ctrl+NumPad /.

To hide an existing element in the diagram

- Press Delete.

To remove an existing element from the model

- Press Ctrl+D.

To restore the layout of the manually suppressed/expanded branches, moved, removed and hidden symbols

---

- Click on the View toolbar.

> If the **Preserve manually suppressed/expanded branches and hidden elements** option in the **Environment Options** dialog > **Diagrams** > **Relation Map**, is selected, the layout of moved elements is restored.

To start the validation (see page 1306)of the relation map

- Click the Validate Diagram button

  on the toolbar of that diagram, table, matrix, or map. The diagram is validated against all validation rules stored in the model, except those that are ignored.

To select an element in the Containment tree

- Select the element and click on the Navigation toolbar.
- On the selected element shortcut menu, click **Select in Containment Tree**.

To decompose an element

1. Select an element.
2. Do one of the following:
   - From the selected element shortcut menu, select **Create Related Element**.
   - Click or press Insert (or ⌘ + I).
3. Select element type and relationship. If selected element type exists in the model, select an appropriate element from the list to represent it in the relation map. How to open an existing element list for the unnamed last element >>(see page 375)
   The element is decomposed.

To open an existing element list for the unnamed element

1. Select unnamed element.
2. Do one of the following:
   - Click the left mouse button.
   - Press Space.
   An existing element list opens, if such elements already exist in the model.

To create element of the same type

1. Type element name.
2. Press Ctrl+Enter while in the name edit mode.

Robustness diagram

Functionality Availability
This feature is available in Standard, Professional, Architect, and Enterprise editions.

The Robustness Analysis involves analyzing the narrative text of use cases, identifying a first-guess set of objects that participate in those use cases and classifying the objects based on the roles they play.

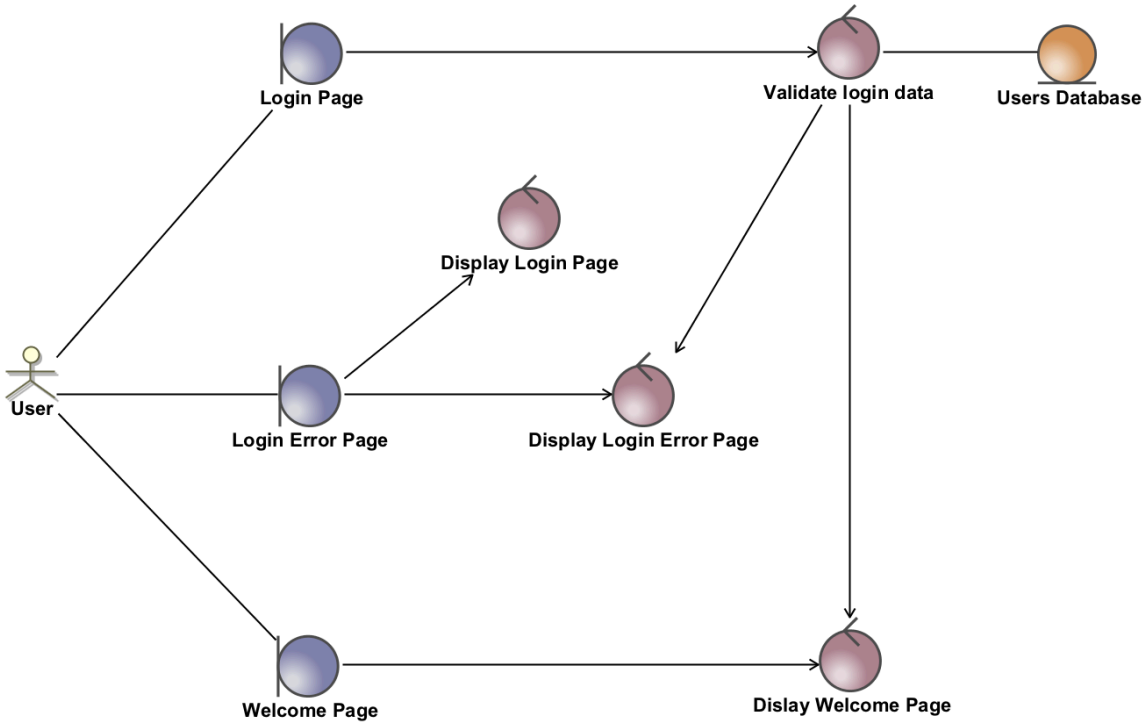
- Boundary or Interface objects are what actors use in communicating with the system.
- Entity objects are mostly objects from the domain model.
- Control objects (known as controllers because they often are not real objects) serve as the "glue" between boundary objects and entity objects.

The Robustness Analysis acts as a preliminary design within the project lifecycle, providing the missing link between an analysis and a detailed design.

Four basic rules apply:

1. Actors can only talk to the boundary objects.
2. The boundary objects can only talk to the controllers and actors.
3. The entity objects can only talk to the controllers.
4. The controllers can talk to the boundary objects and entity objects, and to other controllers, but not to the actors.

Both the boundary objects and entity objects are nouns; the controllers are verbs. Nouns cannot talk to other nouns, but verbs can talk to either nouns or verbs.



Example of the Robustness diagram *Logging In*

Related pages

- UML diagrams(see page 294)
- Extension diagram(see page 376)

Extension diagrams

Our modeling tool supports these extensions of UML diagrams, as well as various other types of diagrams:

	User Interface Modeling diagram(see page 378)	Content diagram(see page 399)	Free Form diagram(see page 404)	Networking diagram(see page 400)
	WSDL diagram(see page 401)	CORBA IDL diagram(see page 402)	Time diagram(see page 403)	

Working with Diagrams

For general information about working with diagrams, read Diagramming(see page 482). If needed, you can either enable or disable a plugin that allows you to create the appropriate extension diagram in the **Plugins** options group of the Envir(see page 2310)onment(see page 2310) Options(see page 2310) dialog.

Patterns

You can create various types of classes in every class diagram using a Pattern Wizard(see page 1162). It contains GOF, Java, Junit, CORBA IDL, XML Schema, and WSDL design patterns. You can also create new patterns and modifications of existing ones using Java code or JPython scripts.

To open the Pattern Wizard

1. In the Class diagram(see page 297), click the **Class by Pattern** button.
2. From the Class(see page 666)'s shortcut menu(see page 376), select **Tools** and then select the **Apply Pattern** subcommand.
3. Select a Class and select **Apply Pattern** from the **Tools** main menu.

Related Pages

Related pages

- UML diagrams(see page 294)
- Diagramming(see page 482)

User Interface Modeling diagram

A User Interface (UI) Modeling diagram makes it possible to build prototypes of user interfaces, connect UI mockups with the whole Architectural model, export them as images, and create browsable reports for presentations. In short, they help gather information faster and thus save time and money.

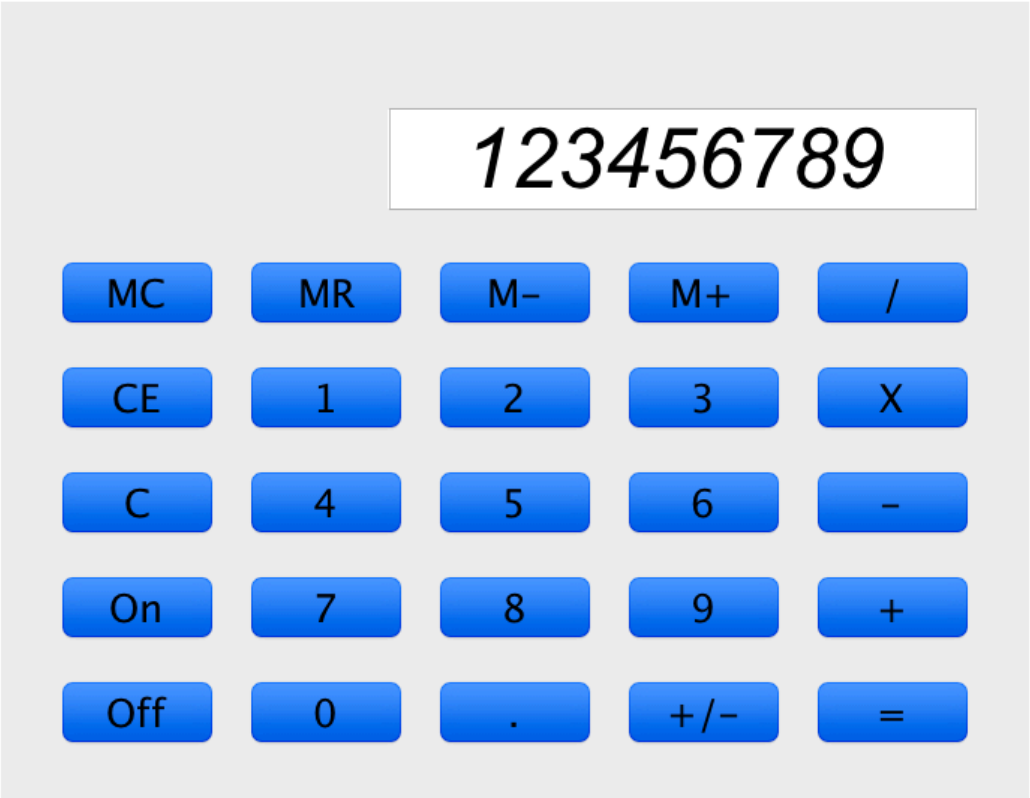
The advantage of User Interface Modeling lies in its ability to:

- Create WYSIWYG User Interface prototypes rapidly.
- Integrate User Interface development with UML specifications.
- Get immediate feedback from prospective users on real situations and reuse it for future designs.
- Create browsable reports with the Report Wizard(see page 1547).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

User Interface Diagram Model for Calculator

This diagram shows a Calculator user interface diagram.



Example of User Interface diagram Model for *Calculator*

Related pages

- UML diagrams(see page 294)
- Extension diagram(see page 376)
- Why is prototyping so important?(see page 380)
- Working with User Interface Modeling diagrams(see page 380)
- User Interface Modeling(see page 381)
- Case Studies for User Interface Modeling(see page 392)

Sample model

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

A Web UI sample project, representing the UI Modeling Diagram usage for web-based interface modeling, is available at <*MagicDraw installation directory*>*samples**diagrams**User* Interface Modeling.

Why is prototyping so important?

Various versions of User Interfaces are tested to see how clients respond. Client response is important for main dialogs, such as a sign-up screen for a website or an e-commerce application. Working with User Interface prototypes instead of "real" User Interfaces enables you to work with small details (such as color or the position of a button), as well as a substantially reduced designer and programmer overhead in the design phase. By building mock-ups or prototypes, you receive fast feedback from prospective users for future designs. User Interface prototyping makes this possible. The feedback loop makes for quicker sophisticated designs that work for everybody, which is what matters.

You can build mock-ups or prototypes to meet the following objectives:

- Integrate GUI development with UML specifications. Since this is a new field with increased design capability, it is easier to see the missing parts in UML or User Interface modeling.
- Enable business analysts to quickly gather requirements via permanent feedback from prospective users, making it easier to obtain all the information needed.
- Create browsable User Interface prototypes/ GUI simulation/ Story-boarding. Several prototypes can be connected via hyperlinks with one another and then be presented together in a report to simulate the workflow of an application.
- Resolve flow issues. It is easier to think through a problem when a User-related Interface can be changed quickly.
- Get "buy-in" from stakeholders. A visual representation of a particular User Interface can be shown more rapidly.
- Run a usability test before full production so that potential errors in usability, like overfilled screens or a usage too complicated, can be avoided.
- Test a series of interaction widgets. Consider, for example, that another text field or button would be good on a screen. Since modifying a prototype is easy, it is much simpler to present these suggestions to others.

Working with User Interface Modeling diagrams

To work with a User Interface Modeling diagram

Do one of the following:

- Click the **User Interface Diagram** button in the Diagram toolbar(see page 24).
- In the Model Browser(see page 29), click either the Package or the Model. From their shortcut menu select **Create Diagram** > **Other Diagrams** > **User Interface Modeling Diagram**.

After loading a project and creating a diagram, you can add UI modeling elements to the diagram pane.

To add new elements to the User Interface Modeling diagram pane

---

- Simply drag and drop the elements from the Text, Containers, Button, or Other toolbar.

To load a sample with an already created User Interface model

---

1. Open either the *UI Modeling Samples.mdzip* or *UI Modeling in System Development.mdzip* projects, found in *<modeling tool installation directory>\samples\diagrams\User Interface Modeling*.
2. After loading the sample, select the diagrams from the *Index* or *Overview d*iagrams by clicking the hyperlinks.

Related pages

- UML diagrams(see page 294)
- Extension diagram(see page 376)

User Interface Modeling

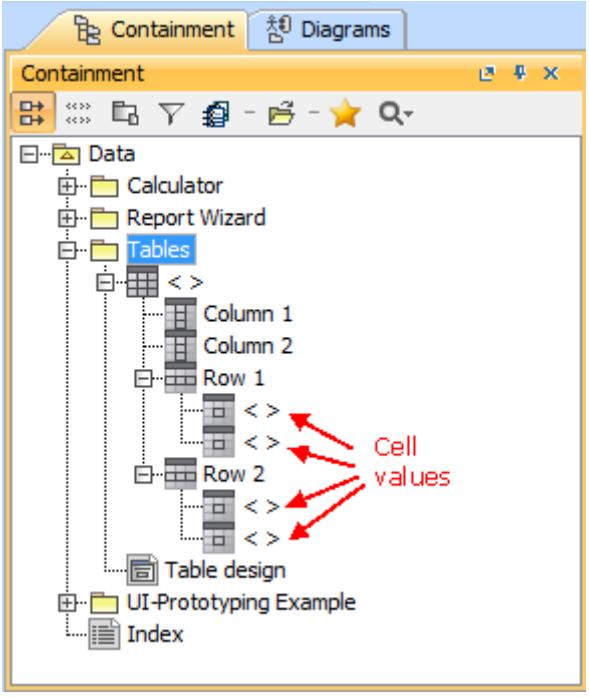
On this Page:

- Modifying a table(see page 381)
- Modifying a tree(see page 385)
- Nesting(see page 386)
- Reusability(see page 386)
- Specifying elements(see page 386)
- Icon usage(see page 388)
- Using symbol properties(see page 389)
- User interface prototyping(see page 390)

This section presents basic information about User Interface Modeling in a modeling tool.

Modifying a table

As the following figure shows, the newly created table has two columns and two rows already added to the table. Each row contains a number of cells equal to the number of columns.



Structure of newly created table

To modify the contents of the table, you can do the following:

- Change the titles of the existing columns.
- Add a new column.
- Change the order of the columns.
- Add a new row.
- Change the order of the rows.
- Add cell values.

To add a new column to a table

---

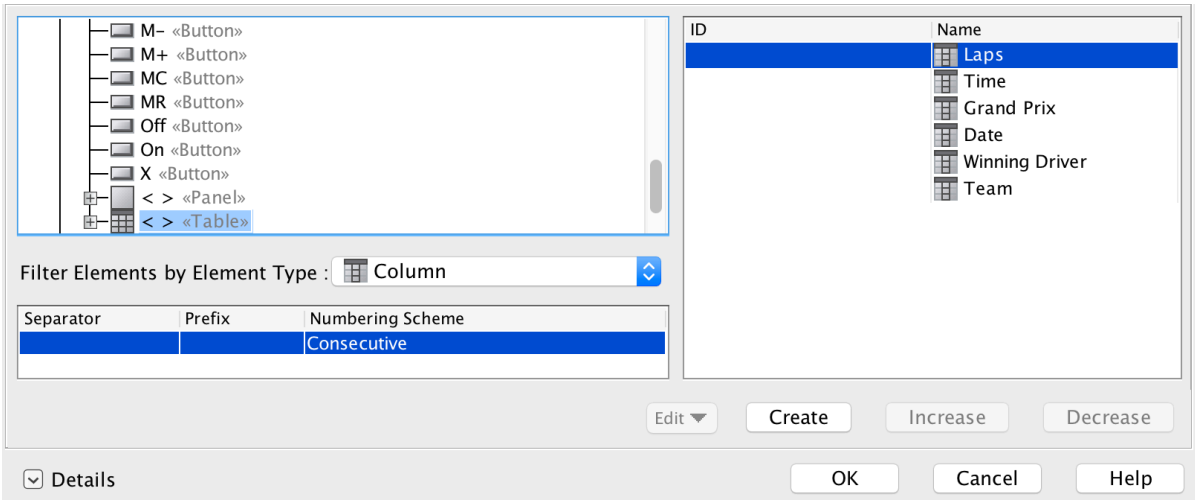
1. Right-click the table in the Containment tree. From the shortcut menu, select **Create Element** > **Column**. A new column is added to the table.
2. Type the title for the new column. With the new column, a new cell is added to each row of the table.

To change the order of the columns in a table

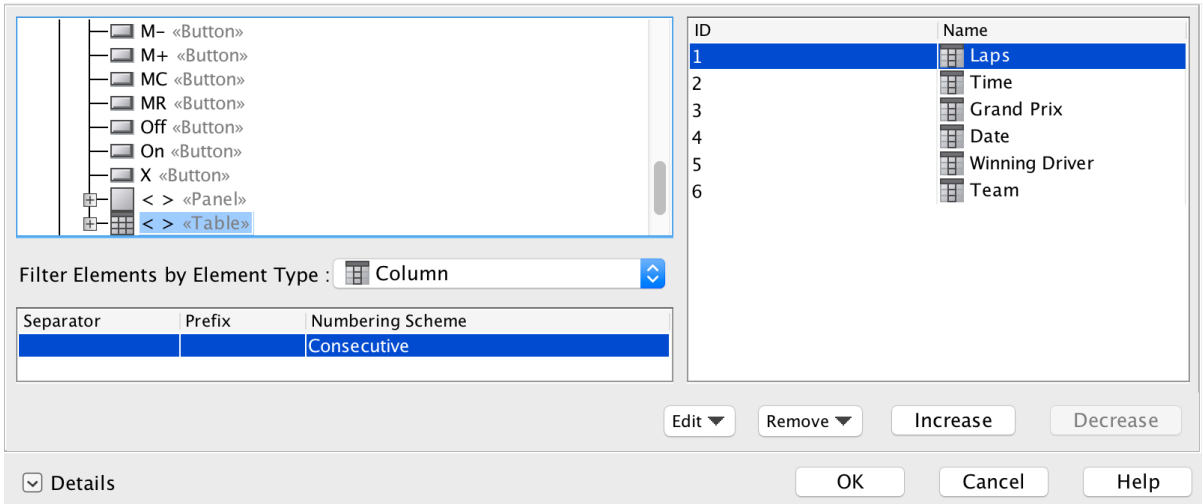
---

1. Right-click a column of the table in the Containment tree. From the shortcut menu, select **Element Numbering**. The **Element Numbering** dialog opens.
2. In the list on the right side of the dialog, select all columns and click **Create**. To select all columns, click the first column in the list and then hold down the Shift key when selecting the last one.

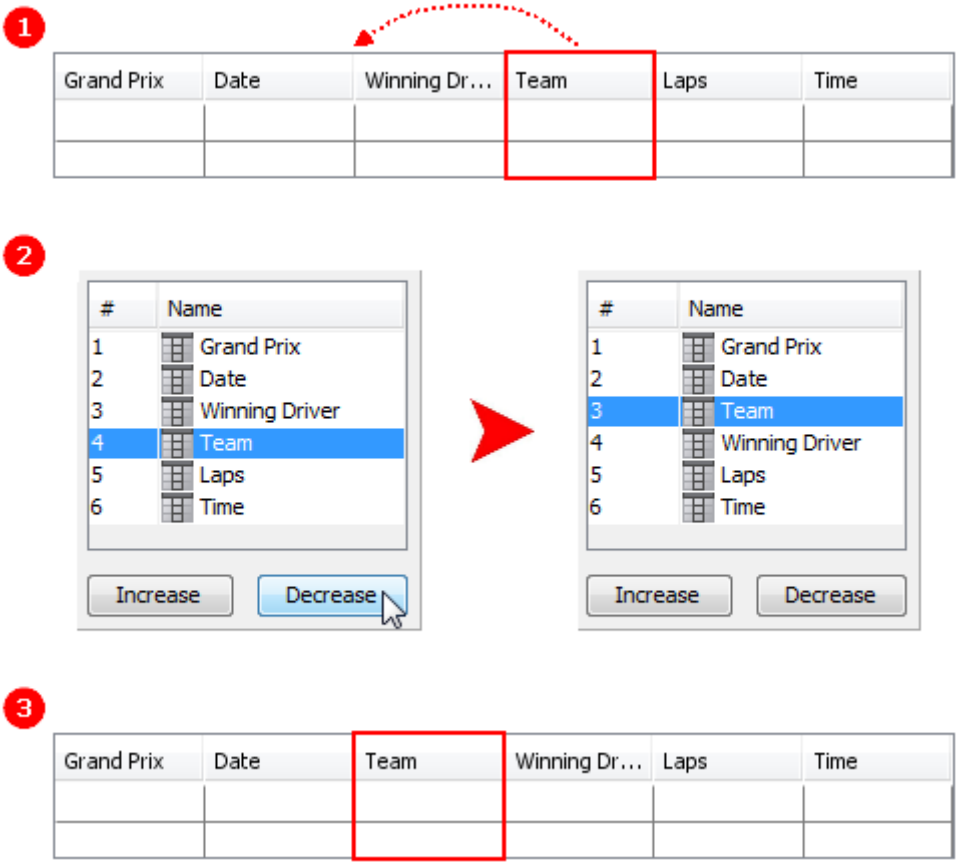
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



3. All the columns become numbered.



4. Change the order of columns in the table by changing their positions in the list (see the following figure). The Element Numbering (see page 1046)feature is to order the selected items.
   - To move a column to the left, select the column and click the **Decrease** button.
   - To move a column to the right, select the column and click the **Increase** button.



Moving column "Team" to left

To add a row to a table

1. Right-click the table in the Containment tree. From the shortcut menu, select **Create Element** > **Row**. A new row is added to the table.
2. Type the title for the new row.

To change the order of the rows in a table

1. Right-click a row of the table in the Containment tree. From the shortcut menu, select **Element Numbering**. The **Element Numbering** dialog(see page 1046) opens.
2. In the list on the right side of the dialog, select all rows and click **Create**.
   - To select all rows, click the first row in the list and then hold down the Shift key when selecting the last one. All the rows become numbered.
3. Change the order of rows in the table by changing their positions in the list.
   - To move a row up, select the row and click the **Decrease** button.
   - To move a row down, select the row and click the **Increase** button.

To change a cell value

1. Select the cell of the table row in the Containment tree.

2. Change the cell value in one of the following ways:
    - Press the F2 key. Type a new value.
    - Click the cell value again and wait while it becomes editable. Type a new value. Press Enter when you are done.
    - Right-click the cell to open the Specification window of the cell. Type a new value in the cell of the **Text** property value.

Modifying a tree

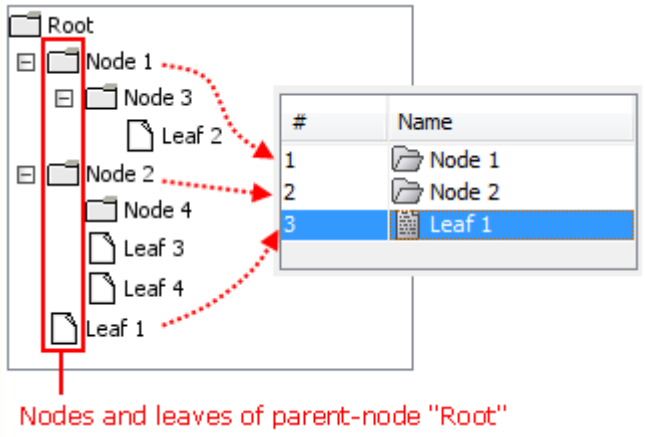
You can add as many nodes and leaves to a tree as you need. You can also change the order of the elements in the tree.

To add a node or a leaf to a tree or to another node

1. Right-click the tree or a node in the Containment tree. From the shortcut menu, select **Create Element** > **Node** or **Leaf**.
2. Type a name for the new node or leaf.

To change the order of the elements in a tree

At a time, you can rearrange the order only of the same parent- node leaves and/or nodes.

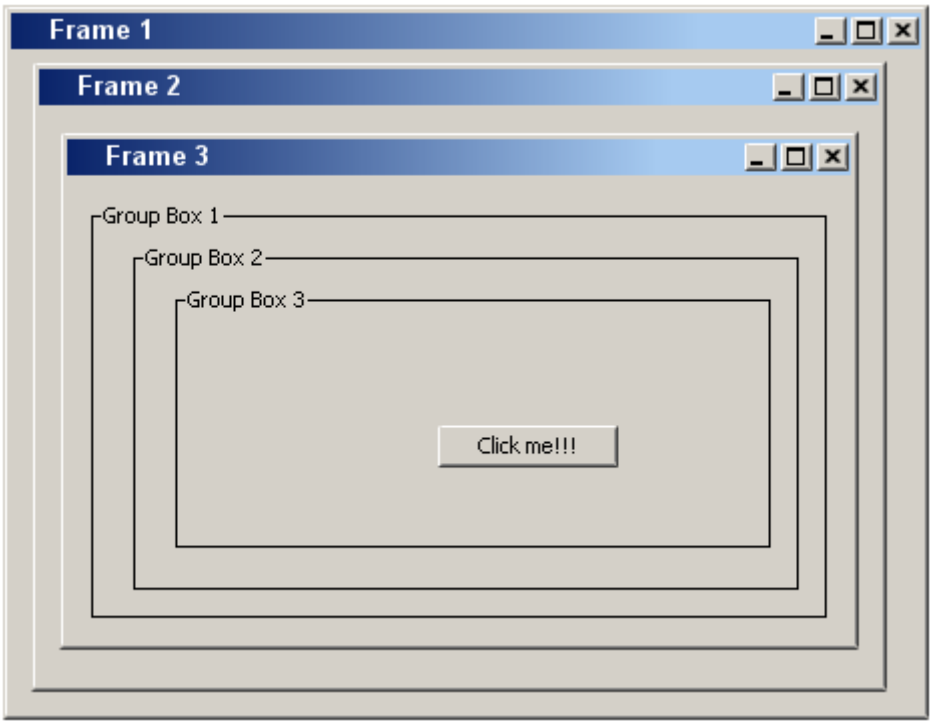


1. Right-click a leaf or a node of the tree. From the shortcut menu select **ElementNumbering**. The **Element Numbering** dialog(see page 1046) opens.
2. In the list on the right side of the dialog, select all leaves and/or nodes, and then click **Create**.
    - To select all leaves and/or nodes, click the first leaf or node in the list and then hold down the Shift key when selecting the last leaf or node in the list.
3. All the leaves and/or nodes become numbered.
4. Change the order of leaves and/or nodes in the tree by changing their positions in the list appropriately.
    - To move a leaf or a node/leaf up, select the row and click the **Decrease** button.
    - To move a leaf or a node/leaf down, select the row and click the **Increase** button.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Nesting

You can move and arrange User Interface Modeling elements since they nest each other, and create deep structured User Interface models. However, not all elements have the nesting ability: only **Contai ner** elements can nest other elements. All frames nest in each other. As a result, if you were to move the element with title *Frame 1*, all other elements would stay in position within this element.



Example of a deep structure that was created using containers

Reusability

If you have created a complex model and need to use it again, you do not have to create a new one. All you need to do is select the elements, copy them and paste them into the same diagram or in any other diagram. You can also reuse a single element by copying and pasting it or by dragging it from the Browser onto the diagram pane.

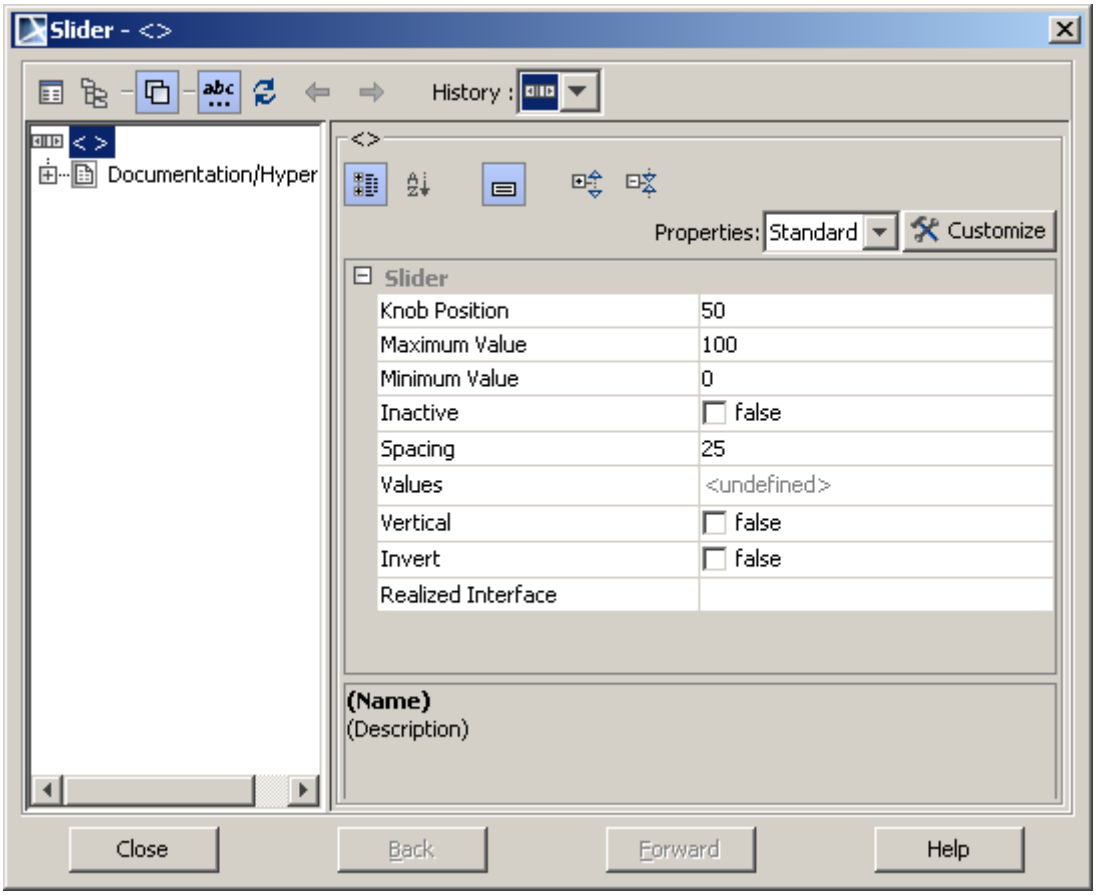
Specifying elements

You can modify UI modeling elements by editing their properties in the following three ways:

- Via the Specification window(see page 913).
- Via the shortcut menu.
- On a diagram pane.

To edit properties via the Specification window

1. Select an element on the diagram.
2. Right-click on it and select **Specification** on the shortcut menu.
3. Edit one of the properties to modify the element.



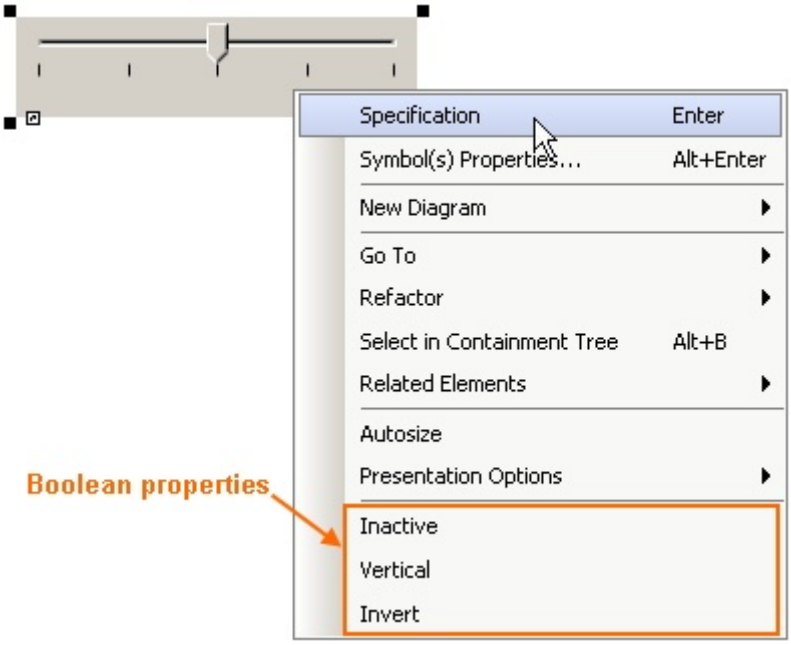
Specification dialog box of the Slider element

The so-called boolean properties that have only two values can also be edited via the shortcut menu.

> ⓘ This following instructions are for boolean properties ONLY.

To edit properties via the shortcut menu

1. Select an element on the dia gram.
2. Right-click on it. The shortcut menu will appear and the editable properties which will be displayed at the bottom.
3. Select the property you want to change.



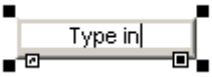
Shortcut menu of the Slider element with boolean properties

For the elements that own a text or a title (such as a button or a frame) it is also possible to edit the appropriate properties straight on a diagram.

> ⓘ This following instructions are for the text or title properties ONLY.

To edit the element properties on a diagram

1. Select an element on the diagram.
2. Click the selected element.
3. Type a text on the element (see the following figure).
4. Press ENTER. The text you have typed will appear on the element and in the **Text**/**Title** property of the element as well.



Editing title of element

Icon usage

Some elements are capable of owning an icon. User Interface Modeling provides a number of frequently-used icons, such as cut, delete, undo/redo, etc. Of course, it is also possible to use images from your computer as icons.

To add an icon

1. Select an element which can own an icon, such as a button.
2. Right click on the element and select **Specification** in the shortcut menu.
3. In the **Icon** list, select the icon you want to appear on the element.
    - If you want to use an image from your computer as an icon, click **custom**.

Setting an icon for a button

Using symbol properties

User Interface Modeling also allows you to modify User Interface elements via symbol properties(see page 975).

To edit an element via symbol properties

1. Select an element on the diagram.
2. Right-click and select **Symbol Properties** on the shortcut menu.
3. Change one of the properties listed in the table below. For example, if you want to set the same background color to more than one UI modeling element, you can select all these elements in the diagram and then change the required property for all of them in the same way.

The following table lists the supported symbol properties:

Symbol Property	Effect
Fill Color	Sets the background color of the component to the chosen color; property **Use Fill Color** must be marked.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

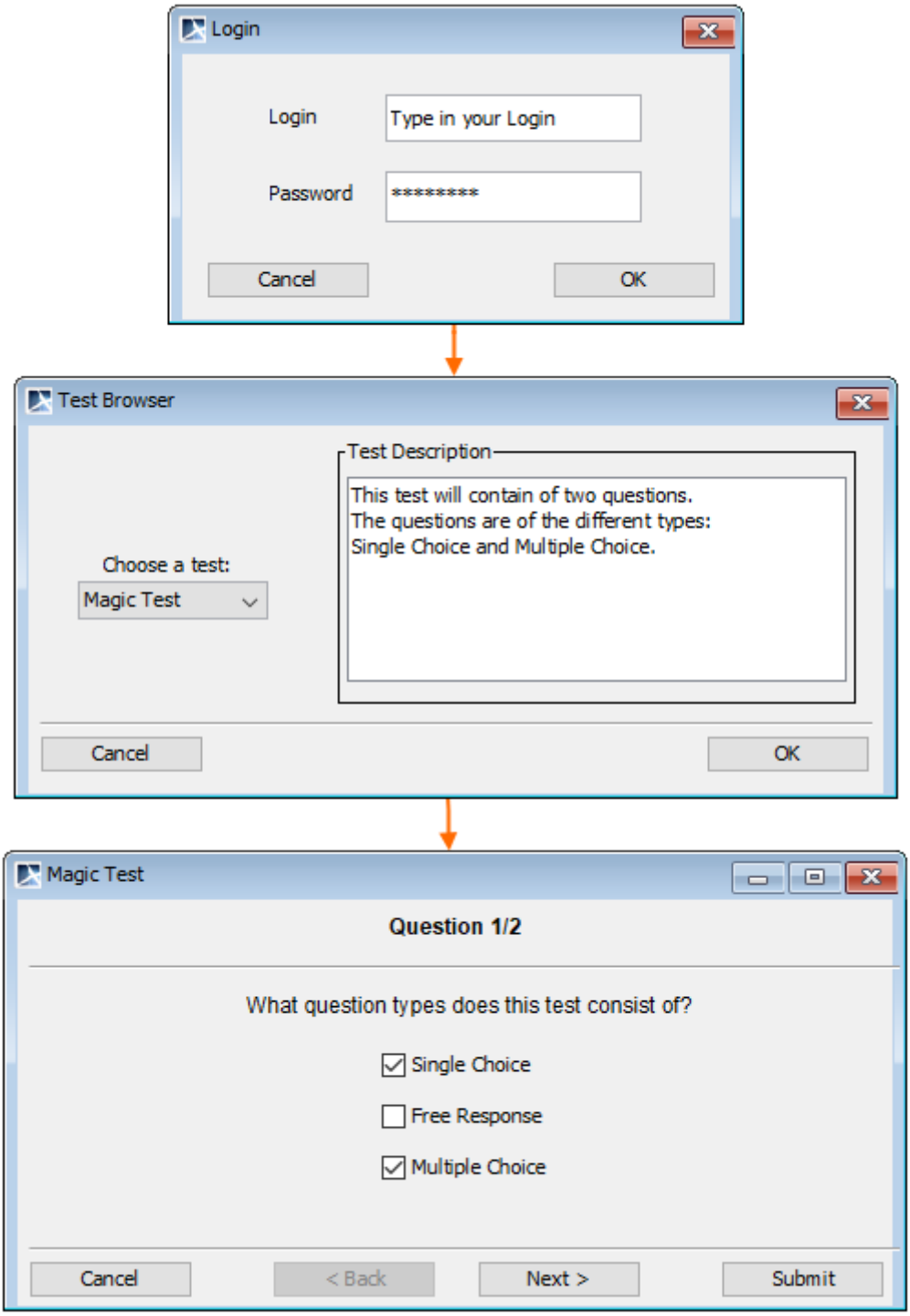
Font	Sets the font if there is any text in the component.
Text Color	Sets the text color of the component to the chosen color.
Pen Color	Sets the color of the border for components that have a color.

User interface prototyping

User Interface Modeling allows you to create browsable reports. You can create a hyperlink from the UI model element to the report. Once a report of these models has been created, click on any element in the report. You will then be directed to the other diagram. When you click **OK** in the upper left-hand frame you will be directed to the next frame. After clicking **OK** in that frame, you can get to a couple of frames from which you can step forward and backward as you want, as they are all linked to one another.

Case Studies for User Interface Modeling(see page 392) provides an example of how this feature works and explains how to add hyperlinks and create browsable reports.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Example of User Interface Models and their Workflow

Related References:

- UML diagrams(see page 294)
- Extension diagram(see page 376)
- User Interface Modeling diagram(see page 378)
  - Why is prototyping so important?(see page 380)
  - Working with User Interface Modeling diagrams(see page 380)
  -

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

User Interface Modeling(see page 381)
- Case Studies for User Interface Modeling(see page 392)

Case Studies for User Interface Modeling

On this Page:

- Case Study 1 - Modeling user interface for the Report Wizard dialog(see page 392)
- Case Study 2 - Slider Example(see page 395)
- Case Study 3 - User Interface Prototyping Example(see page 397)

Sample Model:

<MagicDraw installation folder>\samples\diagrams\User InterfaceModeling\UI Modeling Samples.mdzip.

Case Study 1 - Modeling user interface for the Report Wizard dialog

This case study provides step-by-step instructions for modeling the **Report Wizard** dialog. If you are already familiar with this case study, please review Case Study 2 - Slider Example(see page 395), or Case Study 3 - User Interface Prototyping Example(see page 397)

Step #1. Create a new Project

1. Choose **New Project** from the **File** menu.
2. Name it *Report Wizard*.

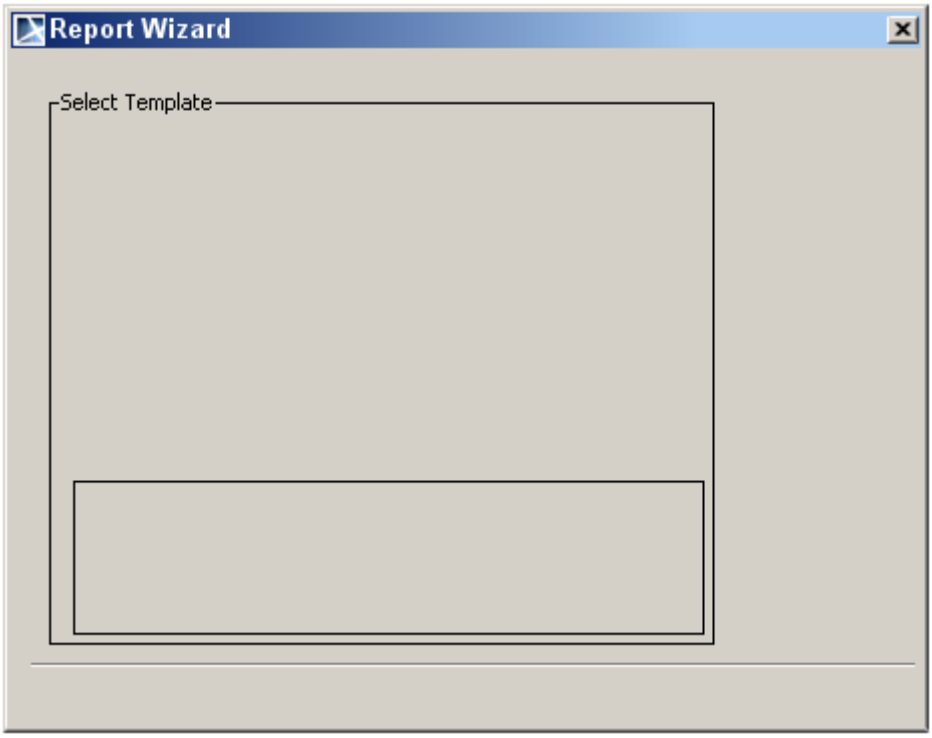
Step #2. Create a new Diagram

1. Click **Create Diagram** and select **User Interface Diagram**.
2. Name the new diagram *Report Wizard*.

Step #3. Create Container components

1. Click the **Frame** button under the **Containers** toolbar and drag-drop it on the diagram pane. A Frame component is created.
2. Name the Frame component by doing one of the following:
   - Click the Frame shape on the diagram pane and type *Report Wizard*
   - Open the Frame's Specification window(see page 913).
     - Double-click the Frame shape on diagram pane.
     - In the opened specification window, type *Report Wizard* in the **Title** field.
3. Clear the check box properties **Maximize** and **Minimize** in the Frame's Specification window.
4. Define an Icon for the Frame symbol:
   a. In the Frame's Specification window, select the **Custom** option in the **Icon** property. The Open dialog box opens.
   b. Open *<MagicDraw installation>\plugins\com.nomagic.magicdraw.uimodeling*. Select the *nomagic.png* image as the frame icon.
5. Under the **Containers** toolbar, drag and drop two **Group Box** elements in the Frame symbol. Under the **Other** toolbar, drag and drop one **Separator** element in the Frame symbol.
   a. Place one Group Box into the other Group Box. Double-click on the larger Group Box and ensure **Titled** is set to *true*. In the **Title** field type *Select Template.*
   b. Open the specification window of the smaller Group Box and clear the check box property **T itled**.

c. Arrange everything appropriately.

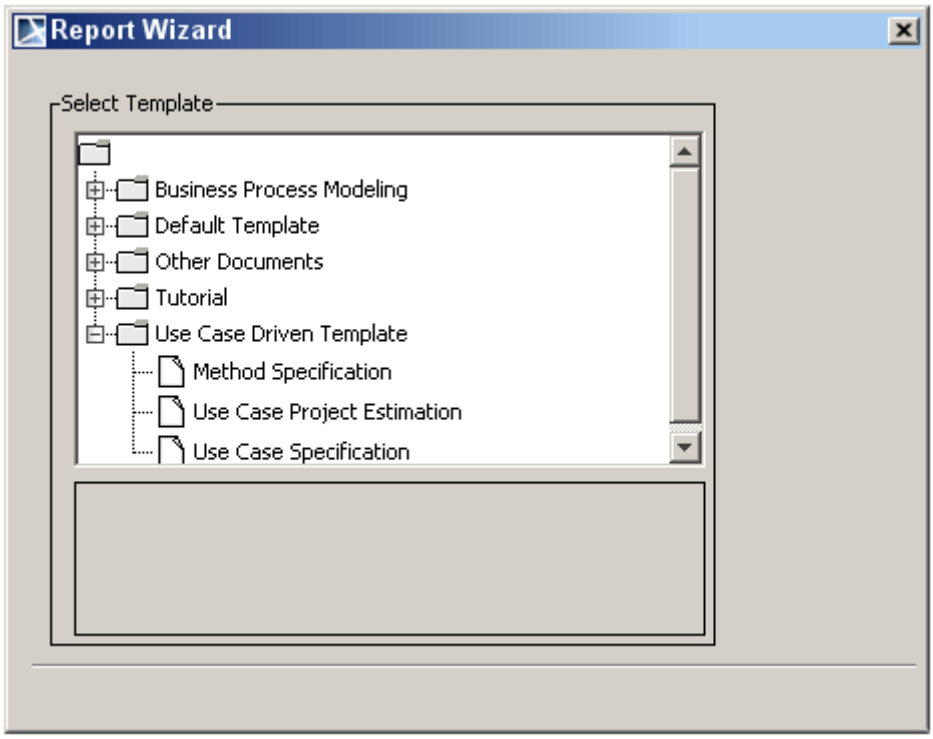


Report Wizard Frame with two Group Boxes and one Separator

Step #4. Create a Tree

1. Create a new Tree component:
    a. Under the **Other** toolbar, drag and drop the Tree element onto the *Select Template* Group Box.
2. Delete old nodes from the Tree component.
3. Create new nodes in the Tree component.
    a. In the Containment tree, right-click on the Tree to invoke its shortcut menu and choose **Create Element** > **Node**.
    b. Create five new Nodes in this way and name them as shown below.
4. Add Leaves to Nodes.
    a. You can add a Leaf by right-clicking on a Node in the containment tree and then choosing **Create Element** > **Leaf**.
    b. Add at least one Leaf to every Node to indicate that the nodes have internal elements.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

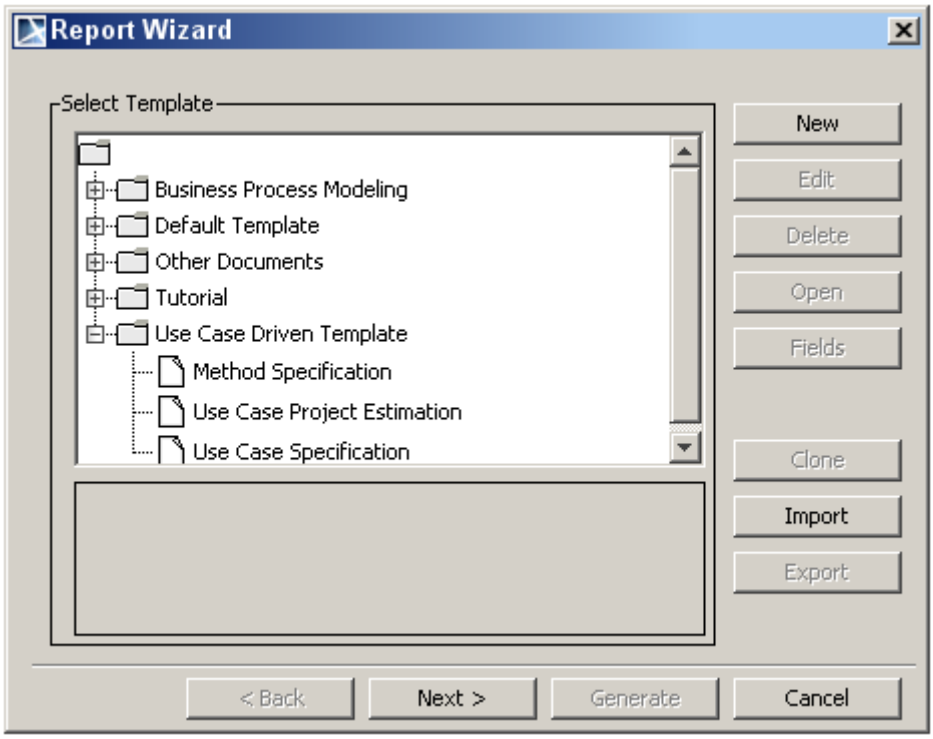


The Report Wizard Window with Added Tree

Step #5. Add Buttons

1. Create predefined text buttons.
    a. Under the **Buttons** toolbar, expand the **OK Button** group.
    b. Right-click the Back button to expand the text buttons group. Create *Back*, *Next* and *Cancel* buttons. Guide yourself according to the figure below.
2. Create regular buttons.
    a. Under the **Buttons** toolbar, drag and drop the **Button** symbol onto the diagram pane inside the Frame, next to the *Select Template* Group Box.
    b. Select the created button in the diagram pane and type in *New.*
    c. Create the remaining buttons by repeating steps 2.a and 2.b. Refer to the figure below.
3. Double-click on the other buttons to invoke their specification window, excluding the *Next >*, *Cancel*, *New* and *Import* buttons. Clear the check box **Inactive** to *true.*
4. It should look like the figure shown below.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Sample of the Report Wizard window

Step #6. Using the Report Wizard window

This User Interface model could now be exported as an image. The steps to do this are as follows:

1. Select **Save As Image** from the **File** menu.
2. In the Save As Image dialog, select **Active Diagram**.
3. In **Image File** define the location where the image should be placed.
4. Select *Joint Photographic Experts Group (.jpg)* (or any format you would like) in **Image Format** and then click Save.

Case Study 2 - Slider Example

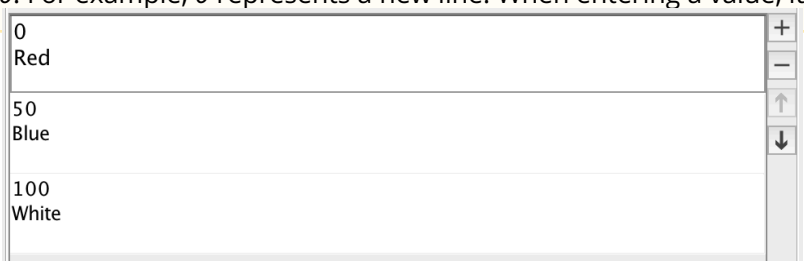
This case study contains step-by-step instructions for creating a User Interface model with Sliders. It also shows how to customize the symbol properties of User Interface components. However, it does **not** explain how to create a new project and a new diagram. Please refer to "Case Study 1 - Modeling user interface for the Report Wizard dialog(see page 392)" for those instructions. If you are already familiar with these case studies, please review Case Study 3 - User Interface Prototyping Example(see page 397)

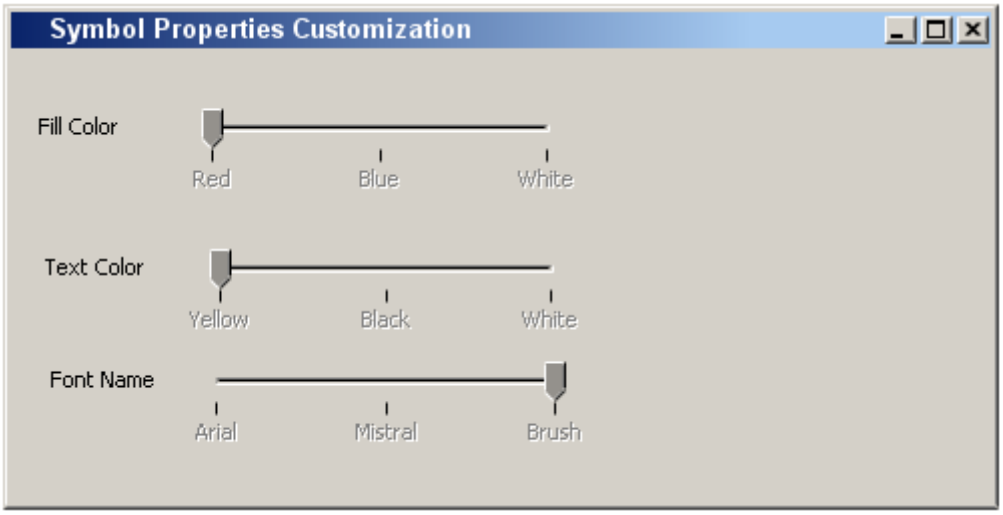
Step #1 Create a Frame, Labels and Sliders

1. Create a new **Frame**.
    a. Under the **Containers** toolbar, click the **Frame** button.
    b. Drop it on the diagram pane.
    c. Click on your newly created frame and title *Symbol Properties Customization.*
2. Add a new **Label** to the Frame.
    a. Under the **Tools** toolbar, click the **Label** button. Place the label onto the frame.
    b. Click on the label until you can edit the name and type in *Fill Color.*
    c. Create the remaining two labels and name them *Text Color* and *Font Name*.
3. Add **Sliders** to the Frame.

a. Under the **Other** toolbar, click the **Slider** button. Place the slider onto the frame.
b. Double-click on the Slider to invoke its specification window.
c. In the opened window, set the **Spacing** property value to 50.
d. Set the **Knob Position** property value to 0 in order to move the knob to the left position.
e. In the Slider's specification window, select the **Values** property and click the + symbol to the right of the field. A Values dialog window opens. If there are values already defined in the window, click the - symbol to delete them.
f. To add new values, click the + symbol to the right of the window and then type *0 Red*. *0 Red* must be written in a specific way. First, type in *0* and press **Enter**. Type *Red* on the next line. Rename the other two given values to *50 Blue* and *100 White* using the same procedure.
g. Draw two more sliders and repeat steps 3a - 3f until your frame looks like the one shown in the figure below.

Notation Formality
When setting the values of a slider it is important to note that there is no empty space between *0*. For example, *0* represents a new line. When entering a value, it should look like this:

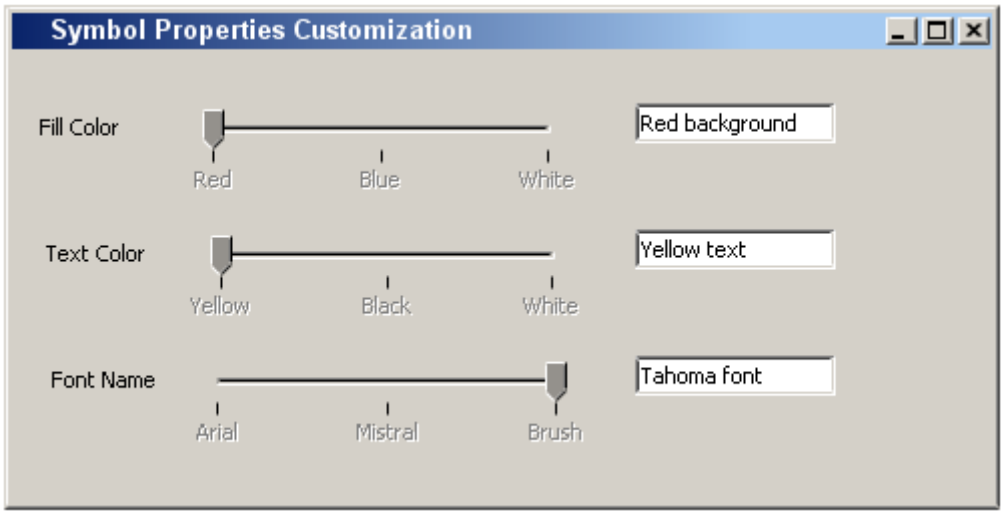




Symbol Properties Customization Frame with Added Labels and Sliders

Step #2 Add Text Fields

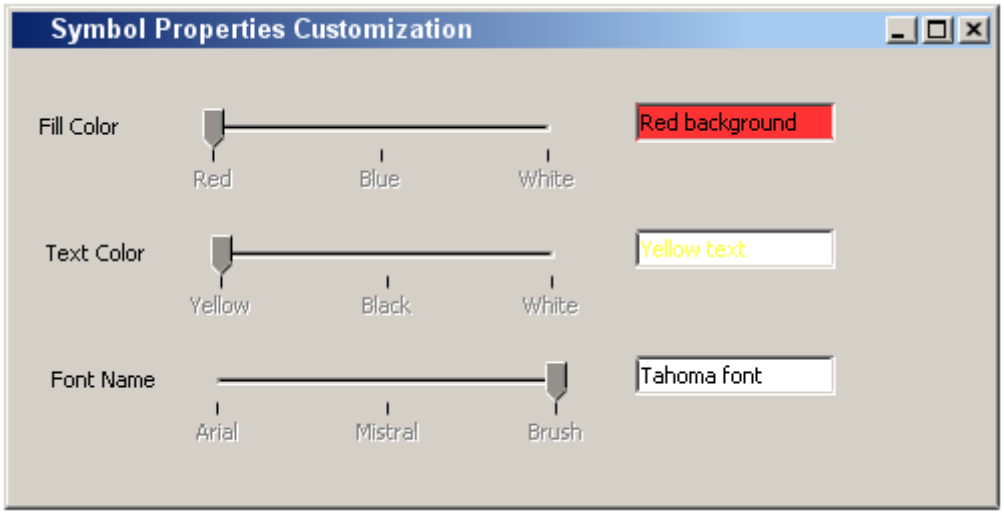
1. Create Text Fields.
   a. Under the **Text** toolbar. click the **Text Field** button. Drop it onto the frame.
   b. Name the text field *Red Background*.
   c. Repeat the previous two steps for the remaining two text fields, as shown in the figure below.



Added Text Fields to the Frame

Step #3 Edit Symbol Properties for the Text Fields

1. Edit **Symbol Properties** for the first text field.
    a. Select the first text field on the Frame shape.
    b. Right-click on it and select **Symbol Properties** from its shortcut menu.
    c. Check off the **Use Fill Color** property and change the **Fill Color** property value to red. Change the fill color by clicking the ... to the right of the field, and in the Color dialog, click **RGB** and type in *FF0000* in the **Color Code** field.
2. Edit **Symbol Properties** for the remaining fields.
    a. Follow steps 1a - 1b for the next steps.
    b. For the second text field, select yellow in **Text Color**.
    c. For the third one, select the font name Tahoma in **Font**, as shown in the figure below



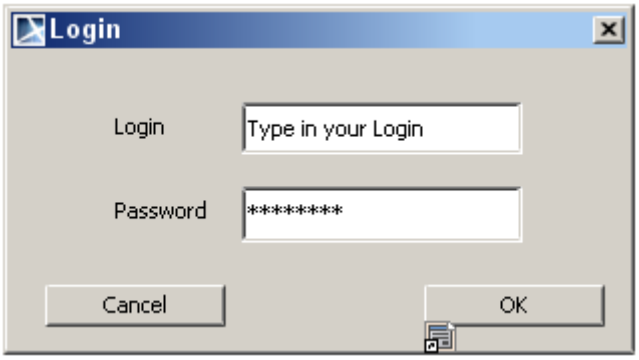
Finalized Slider Example

Case Study 3 - User Interface Prototyping Example

This case study demonstrates how to connect several user interface models and create a browsable report. The models should represent a test application with a Login Dialog, a Test Browser and a test with several questions. Here are step-by-step instructions on how to add hyperlinks and create browsable reports. If you are already familiar with this case study, please review Case Study 2 - Slider Example(see page 395), or Case Study 1 - Modeling user interface for the Report Wizard dialog(see page 392)

Step #1 Create first Model

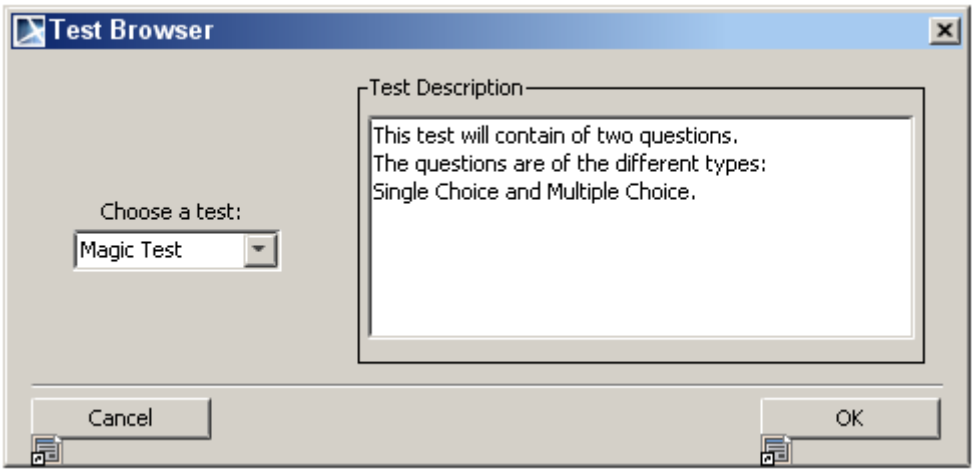
1. Create a new Package for the model.
2. Create a model similar to the one shown below.



First Model of Prototyping Example - Login Dialog

Step #2 Create second Model

1. Create another new Package for this model.
2. Create a model similar to the one depicted below.



Second Model of Prototyping Example - Test Browser

Step #3 Create remaining Models

1. Create a separate Package for each model.
2. You can build models similar to the ones shown in the detailed samples mentioned in the beginning of this chapter, or create your own in a similar manner.

Step #4 Add Hyperlinks

1. Add a hyperlink.
   a. Open the model with the Login Dialog.
   b. Select the **OK** button.
   c. Click on the smart manipulator **Hyperlinks/Go To** and select **Add Hyperlink** in the popup menu.
   d. In the Edit Hyperlink dialog, select **Element/Symbol** and click the "..." button. Browse to the Package **Test Browser**, select the User Interface diagram(see page 378) in it and confirm two times with OK - a diagram symbol will appear next to the OK button. Double clicking on the OK button will lead to the other diagram.
2. Now connect all the buttons in the other diagrams with hyperlinks. The diagram symbol appears next to a component with a hyperlink.

Step #5 Create a browsable Report

1. Create a new report.
    a. Select **Report Wizard** from the **Tools** menu.
    b. Expand the **Default Template** in the tree and select **Web Publisher 2.0**.
    c. Confirm three times by clicking Next until the dialog appears allowing you to add data to the report. Add all Packages that contain the created diagrams and click Next.
    d. Give the **Report file** a name to the output file. Clear the check box **Display in viewer after generating report**.
    e. Click Generate. The report is built and shown in your default browser.

Related References:

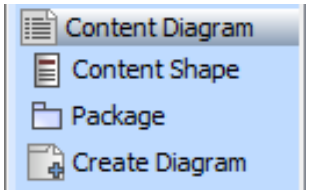
- UML diagrams(see page 294)
- Extension diagram(see page 376)
- User Interface Modeling diagram(see page 378)
    - Why is prototyping so important?(see page 380)
    - Working with User Interface Modeling diagrams(see page 380)
    - User Interface Modeling(see page 381)
    - Case Studies for User Interface Modeling(see page 392)

Content diagram

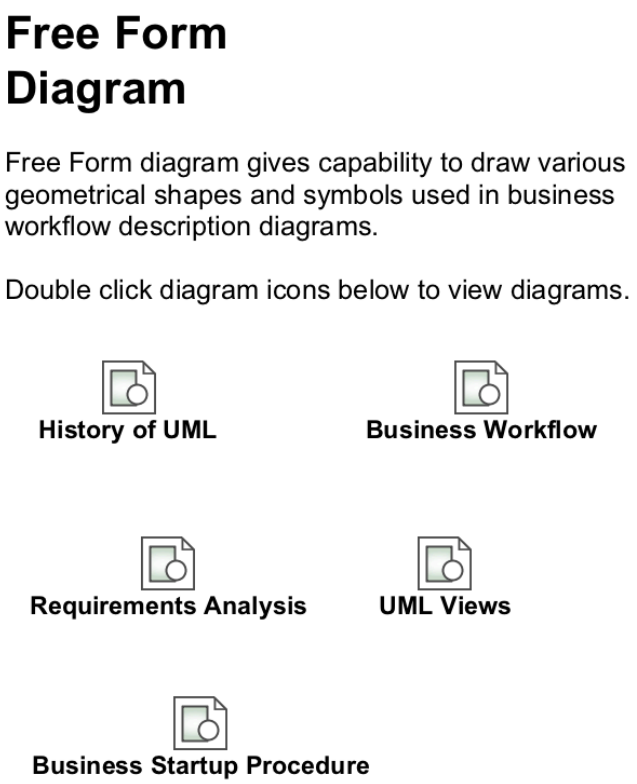
Functionality Availability
This feature is available in Standard, Professional, Architect, and Enterprise editions.

The Content diagram is an extension of UML notation. The purpose of the Content diagram is to generate or represent a project structure (diagrams) and relations between them. The Content table works as a table of contents for a project. The Content Shape creates a table of contents of all diagrams of the project.

All Content diagrams have their own specifications. You can specify their name, documentation, and view the relationships in which they participate. You can also add stereotypes(see page 856), tagged values(see page 865), and constraints(see page 689). For more information about diagrams creation, read Creating Diagrams(see page 483).



Content diagram palette



Example a Content diagram for the Free Form diagram

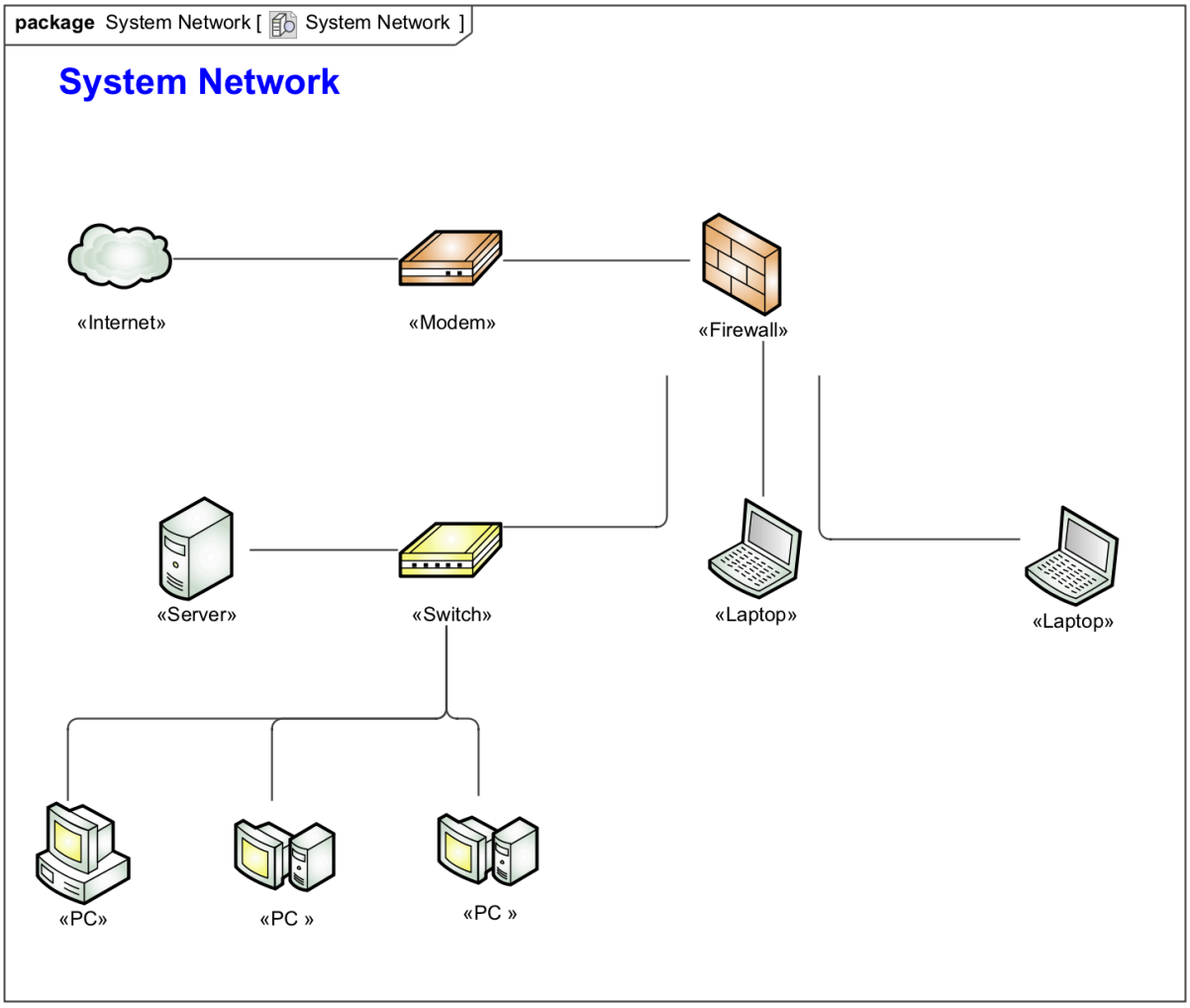
Related References:

- UML diagrams(see page 294)
- Extension diagram(see page 376)
- Diagramming(see page 482)

Networking diagram

> **Functionality Availability**
> This feature is available in Standard, Professional, Architect, and Enterprise editions.

The Networking diagram provides a visual display of the network topology. The *Networking Profile* contains stereotypes(see page 856) for the network description. Elements with icons can be drawn on the diagram pane. The Networking diagram is commonly used to depict hardware nodes as well as the connections between them.



Example of the Network diagram *System Network*

Related Pages:

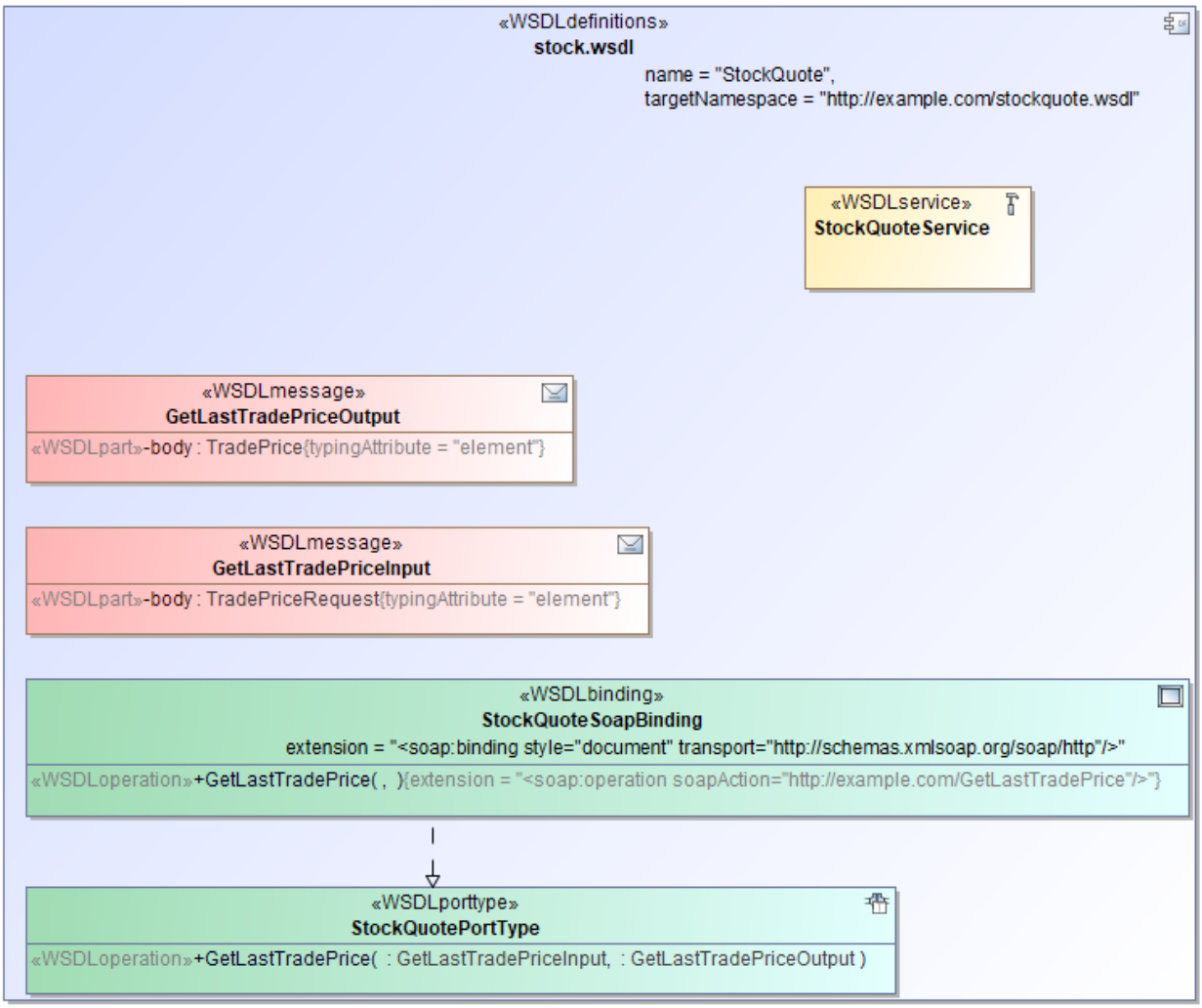
- UML diagrams(see page 294)
- Extension diagram(see page 376)
- Diagramming(see page 482)

WSDL diagram

> **Functionalities**
> This feature is available in Architect and Enterprise editions.

Use the WSDL diagram to draw WSDL elements. It allows the creation of all elements used in the wsdl file, except schema. The schema elements can be created using the XMLSchema diagram. WSDL plugin provides patterns to create binding elements. The WSDL plugin requires the XMLSchema plugin.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Example of the WSDL diagram *Stock*

Related Pages:

- UML diagrams(see page 294)
- Extension diagram(see page 376)
- Diagramming(see page 482)

CORBA IDL diagram

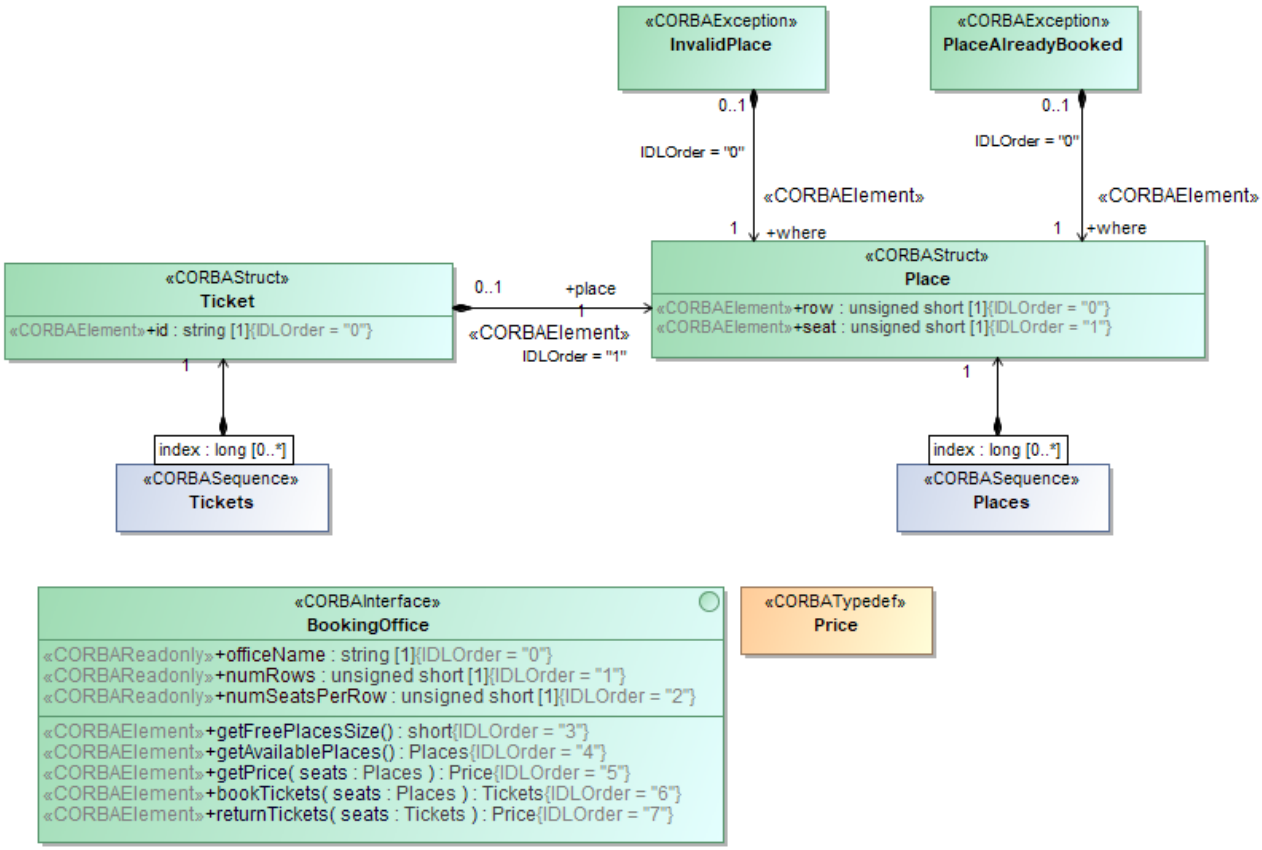
> **Functionalities**
> This feature is available in Architect and Enterprise editions.

The CORBA IDL diagram facilitates the creation of CORBA IDL elements. The following patterns are also available for CORBA IDL: Interface, Value Type, Type Definition, Sequence, Array, Fixed, Union,

Enumeration, Struct, and Exception. For more information about CORBA IDL usage in MagicDraw, see MagicDraw Code Engineering UserGuide.pdf, chapter "CORBA IDL Mapping to UML".

You can select either the UML Interface or the UML Class as a base element for the CORBA Interface. For more information about the CORBA IDL Interface implementation, see CORBA Interface Implementation.

Reference: UML™ Profile for CORBA™ Specification, Version 1.0, April 2002. http://www.omg.org/technology/documents/formal/profile_corba.htm



Example of the CORBA IDL diagram *Tickets Purchasing*

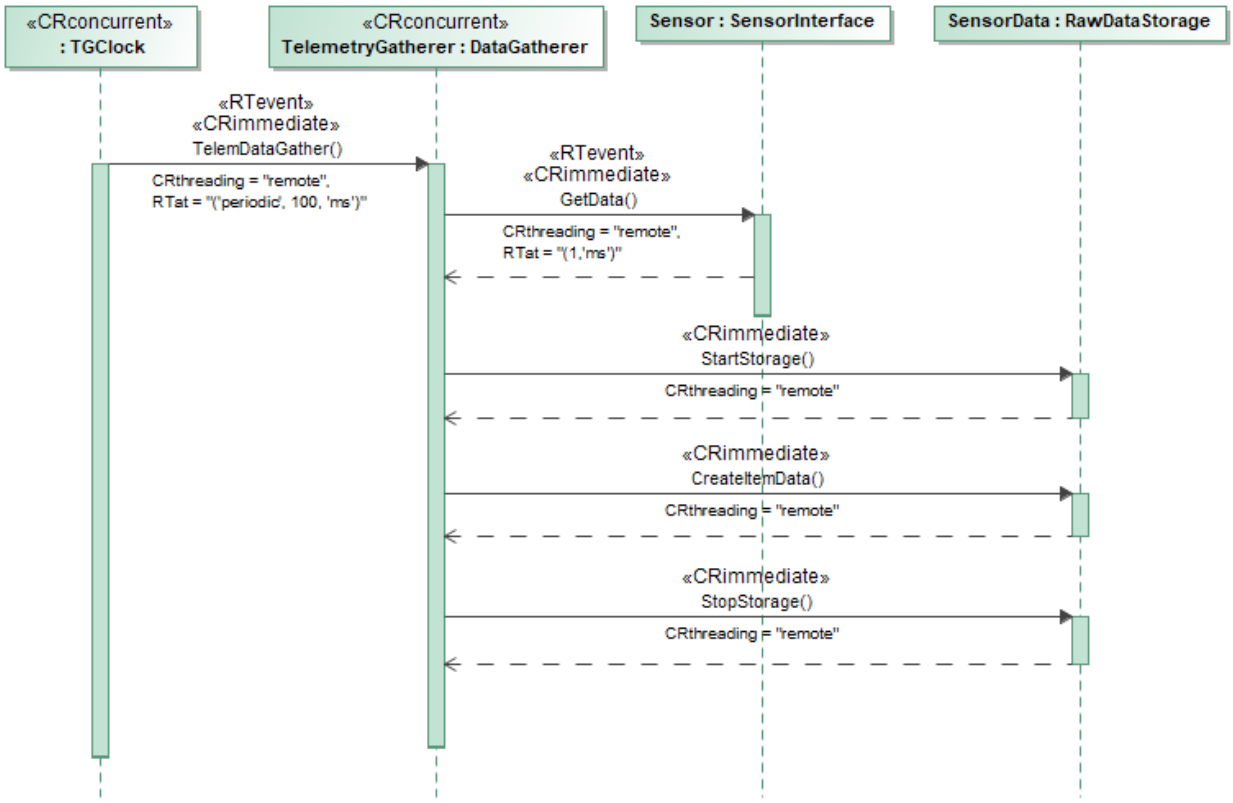
Related Pages:

- UML diagrams(see page 294)
- Extension diagram(see page 376)
- Diagramming(see page 482)

Time diagram

Functionalities
This feature is available in Standard, Professional, Architect, and Enterprise editions.

A Time Diagram is an extension of UML notation. The time diagram is similar to a sequence diagram, but the model elements of the time diagram have predefined stereotypes.



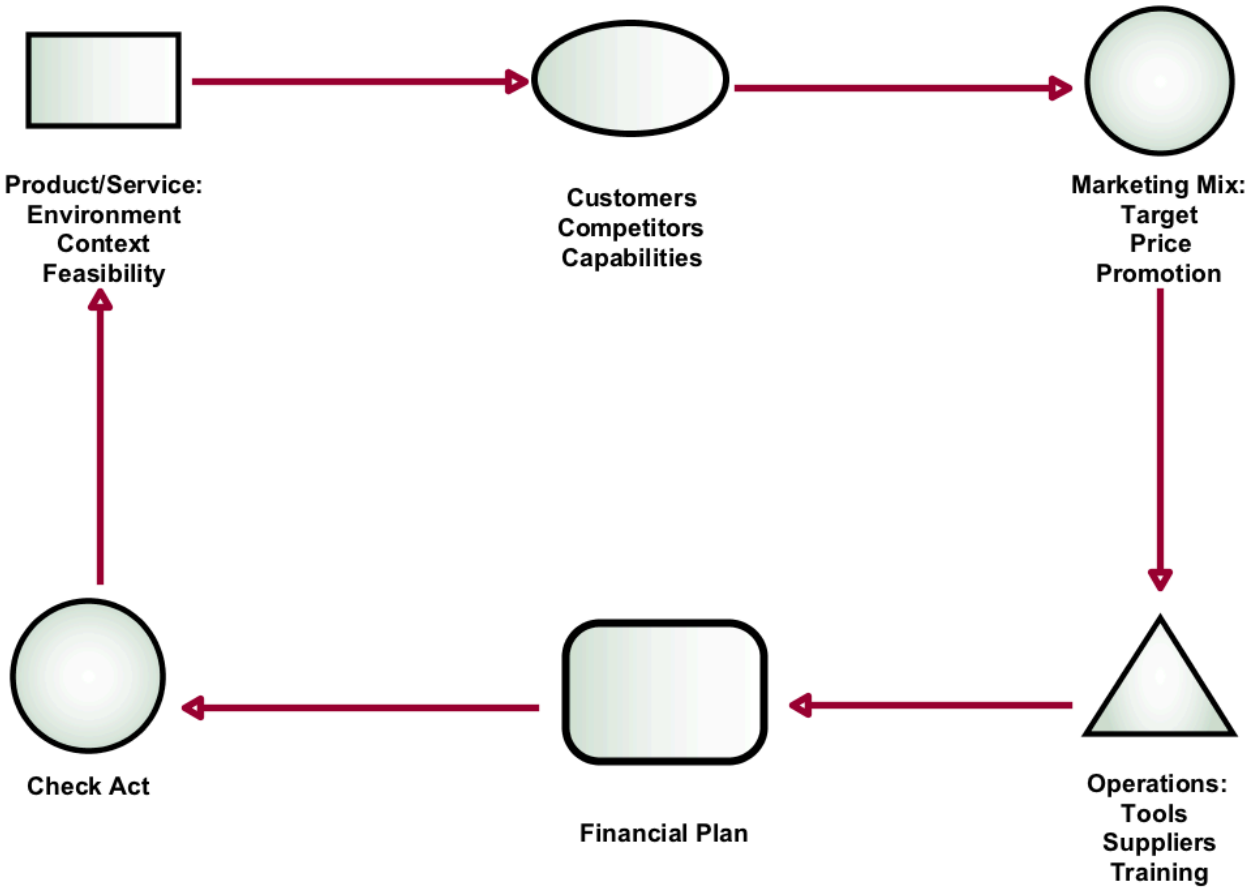
Example of a Time diagram

Related Pages:

- UML diagrams(see page 294)
- Extension diagram(see page 376)
- Diagramming(see page 482)

Free Form diagram

The Free Form diagram allows you to draw various geometrical shapes. The diagram also includes shapes for drawing business flowcharts.



Example of the Free Form diagram Business Startup Procedure

Related pages:

- UML diagrams(see page 294)
- Extension diagrams(see page 376)
- Diagramming(see page 482)

Tables

On this page

- Introduction(see page 406)
- Table areas(see page 407)

Introduction

You can use modeling tool tables to represent and modify information about the properties of model elements and/or relationships between them. You can create four kind of tables: Generic Table, Glossary Table, Metric Table, and Instance Table. The different purposes for each table are illustrated below:

- Generic Table(see page 441) allows you to manage selected elements and their properties in a single place. The example below is from a generic table.mdzip[71] sample model.

#	Name	Visibility	Documentation
1	Student	public	A person attended to study or studying the course. Student is supervised and provided with the support of a teacher or a group of teachers.
2	Teacher	public	A person who designs the course and/or prepares the learning material for the course. He/she supervises and provides with a support a student or a group of students during the study process.
3	Class	public	A physical place where students and a teacher or teachers meet face-to-face for learning purposes.
4	Course	public	A unit of teaching provided in various formats (lectures, meetings, seminars, self-learning, etc.). Usually the one course covers the one subject. The course is designed and prepared by a teacher or a group of teachers. The course is designed for students.

- Glossary Table(see page 452) allows you to define specific term(s) and gives you the ability to reuse the term(s) anywhere in your model. The example below is created only for this image and is not saved in any of the sample models.

#	Term	Description
1	Author	A user who has committed a new project version.
2	Comment	Optional description of changes in the committed version.
3	Project	A physical working unit that consists of model, model visualizations (diagrams, tables, matrices, etc.), configuration data, and references to other elements residing in modules.
4	Project category	Project category concept which enables visual grouping of projects in Teamwork Server repository.
5	Repository	A storage place for projects and their versions managed by the Teamwork Server.
6	Version	A unique number assigned to the committed project. Project version numbers begin at zero (for the initial version) and increase with every new project version.

- Instance Table(see page 459) allows you to manage an Instance Specification of your model in spreadsheet form. The example is from SpacecraftMassRollup.mdzip[72] sample model.

#	Name	ma (kg)	me (kg)	mr (kg)	margin (kg)
1	spacecraft	130.0	95.0	15.0	35.0
2	telecom	35.0	27.0	5.0	8.0
3	amplifier	10.0	8.0	10.0	2.0
4	antenna	20.0	19.0	20.0	1.0
5	propulsion	80.0	68.0	7.0	12.0
6	tank	44.0	38.0	44.0	6.0
7	thruster	29.0	30.0	29.0	-1.0

71 https://docs.nomagic.com/download/attachments/55847477/generic%20table.mdzip?api=v2&modificationDate=1508506693854&version=1

72 https://docs.nomagic.com/download/attachments/55847477/SpacecraftMassRollup.mdzip?api=v2&modificationDate=1508506748322&version=1

- Metric Table(see page 468) allows you to collect various metrics used to evaluate the current state of your model. The example below is from MagicLibrary requirements.mdzip[73] sample model.

#	Name	Metric Suite	Requirements	Covered By Design	Covered By Design Percentage
1	2014 12 15	Requirement Coverage (Treat Owner As Grouping Element)	38	35	92,11

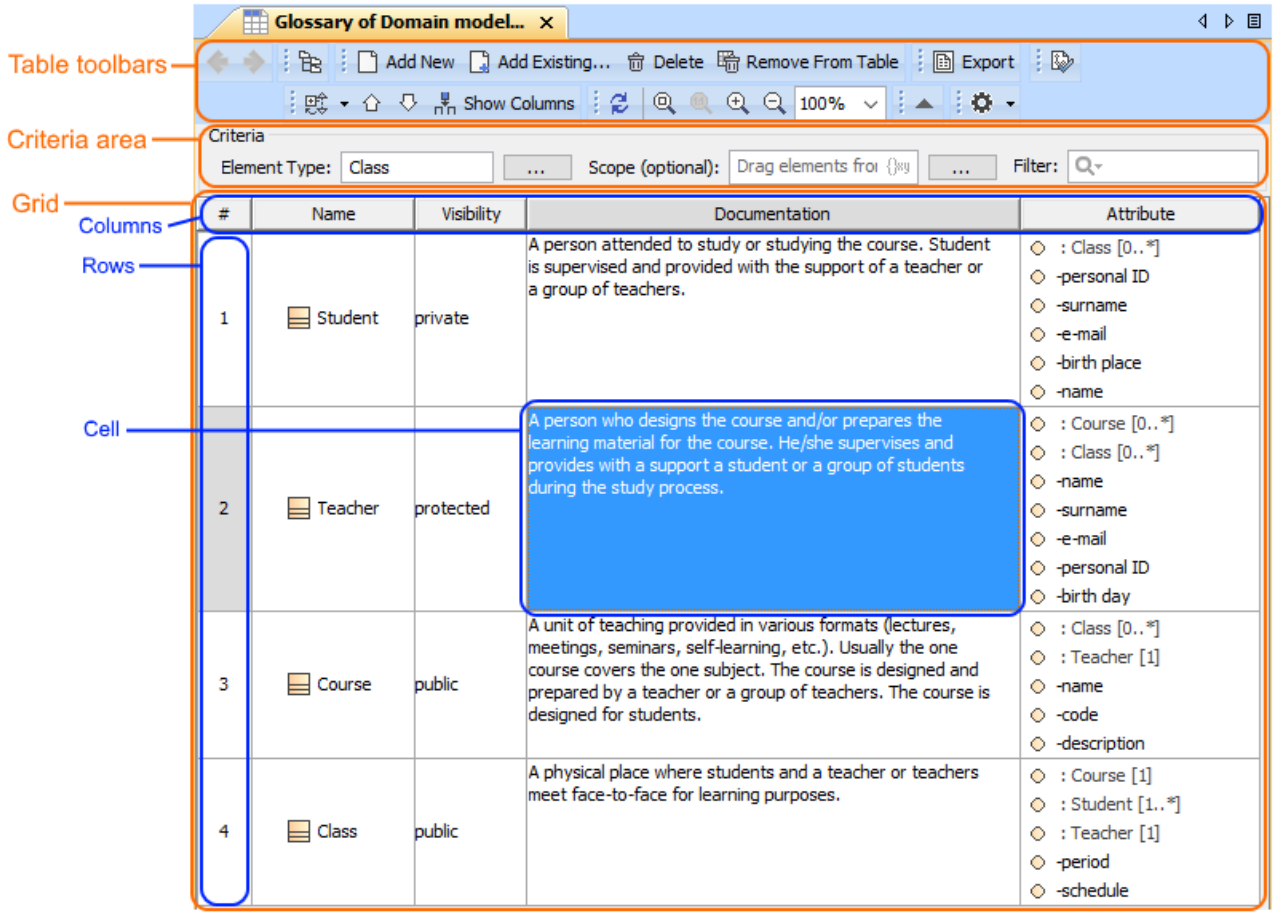
Table areas

Before working with tables, you should understand the structure of all tables supported by MagicDraw. All tables contain the following areas:

- *Table toolbars* allow you to quickly access the options for managing data in the table. Learn more about table toolbars >>(see page 408)
- *Criteria area* allows you to define the elements that you want to display in the table, specify the scope of the table, and filter necessary information in tables which contain large amount of data. Learn more about Criteria area >>(see page 415)
- *Grid* displays data according to the defined criteria area(see page 415). You can modify the displayed data in the grid by editing cells, removing, hiding, creating new, or selecting other options from the table toolbar(see page 408). The grid consists of:

  - *Rows* which appear along the side of the table, and are labeled using numbers starting with 1. Each table row represents an element.
  - *Columns* which appear along the top of the table, and are labeled using element property names. Each table column represents element properties.
  - *Cells* are the intersections of every row and column which represent element property values.

73https://docs.nomagic.com/download/attachments/55847477/MagicLibrary%20requirements.mdzip?api=v2&modificationDate=1508506778567&version=1



Basic features of tables

- Basic table toolbar buttons(see page 408)
- Table Criteria area(see page 415)
- Basic tasks in tables(see page 417)

Specific features of tables

- Specific table toolbar buttons(see page 408)
- Table Criteria area(see page 415)
- Instance table(see page 459)
- Generic table(see page 441)
- Glossary table(see page 452)
- Metric table(see page 468)

Table toolbars

On this page

- Basic toolbar buttons(see page 409)

- Specific toolbar buttons(see page 414)

> **Important**
> If the table toolbars are not available in a server project, lock the table for editing. Make sure you have the right to edit the model of this project.

Basic toolbar buttons

The following table describes the toolbar buttons found in tables.

Button	Shortcut keys	Description
Navigation toolbar		
	Alt+B	Click the Containment tree(see page 32) to select the element of the selected row.
Edit toolbar		
Add New > Create	Insert	Click to create a new element in the table. This element will also be added to the model. If several element types are available, a shortcut menu with available types will open. You will be able to select an element of the type you need. **Note** • In Instance Table, expand and select Create to add a new instance. If this button is not active, the classifier is not specified for the instance table. • In Metric Table, click Calculate Metrics and select Add New Metric with Different Parameters to create a new row with copied parameters.
Add Existing	Ctrl+Insert	Click to add an element from a model. The **Select Element** dialog(see page 1031) opens. Only elements of the type defined in a specific table will be listed. **Note** In Instance Table, expand to find this button.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Button	Shortcut keys	Description
Delete	Ctrl+D	Click to remove selected elements both from the table and the model. **Note** In Instance Table, expand to find this button.
Remove From Table	Ctrl+Delete	Click to remove only selected elements from the table. **Note** In Instance Table, expand to find this button.
Layout toolbar		
		Click to expand or collapse rows when the **Display Mode** under the **Options** button is selected as **Compact tree** or **Complete tree**. You may expand the command and select: • **Expand Selected Recursively** - to expand the selected row recursively. • **Expand All** - to expand all rows. • **Collapse Selected Recursively** - to collapse the selected row recursively. • **Collapse All** - to collapse all rows.
	Ctrl+Up Arrow	Click to shift selected elements (either grouped or non-grouped) up a row. The elements are automatically renumbered after moving.
	Ctrl+Down Arrow	Click to shift selected elements (either grouped or non-grouped) down a row. The elements are automatically renumbered after moving.
Show Columns / Columns	N/A	Click to change the set of columns to show on the table. **Note** In Instance and Metric tables, this button is called **Columns**. In Generic and Glossary tables, this button is called **Show Columns**.

Button	Shortcut keys	Description
Publish toolbar		
Export	N/A	Click to export the contents of the table to an **.html*, **.csv*, or **.xlsx* file. **Note** In the Metric Table this button also allows you export to *.xlsm file.
Excel/ CSV Sync	N/A	Click to open the Excel/CSV Sync menu: • **Excel/CSV File** - contains three commands: - **Select File** - opens the **Excel/CSV Sync Options** dialog(see page 479) and allows you to select a file from you file system or from your model to link it with modeling tool table and enable data syncing. - **Clear Selection** - removes the reference between modeling tool table and linked file and syncing is turned off. - **Open File** - opens the linked file. • **Read from File** - imports data from Excel/CSV file to modeling tool table. • **Write to File** - exports data from modeling tool table to Excel/CSV file. • **Sync Options** - opens the **Excel/CSV Sync Options** dialog(see page 479) and allows you to customize sync and mapping options before syncing data between Excel/CSV file and modeling tool table.
Find in Diagram		
Find in Diagram	Ctrl+F	Click to open the search bar allowing to search for textual information in all types of diagrams, including tables, matrices and maps. Use ▲/▼ to navigate the search results.
View toolbar		
	N/A	Click to update the contents of the table after specifying the scope. **Note** In the Instance Table, this button updates data after specifying the smart package as the scope criteria.

Button	Shortcut keys	Description
	Ctrl+W	Click to fit the table in the active window size.
	Ctrl+Slash Ctrl+NumPad/	Click to show the actual size of the table.
	Ctrl+Plus Ctrl+NumPad+ Ctrl+Equals	Click to enlarge the table maintaining aspect ratio.
	Ctrl+Minus Ctrl+NumPad-	Click to reduce the table maintaining aspect ratio.
120%	N/A	Click to show the size ratio. Click to select the size ratio from the percentage list or enter the value.
Validation toolbar		
	N/A	Click to validate the table against all validation rules stored in the model, except those that are ignored.
Options toolbar		

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Button	Shortcut keys	Description
	N/A	Click to open the Options menu. Each table contains different options under this button. Below are the meanings of all available options: • **Show Full Paths** – displays the full paths of elements (where owner information can be displayed). • **Save Filter Criteria** – keeps the filter criteria(see page 918) unchanged after reloading the project or after restarting MagicDraw. • **Show Detailed Column Name** – in a generic table, shows property group and/or stereotype names in the table header. This is helpful when the table displays several columns with the same name. In a metric table, it shows the names of metric suites in column headers but it works when the metric table represents the metrics of two or more metric suites. In an instance table, it shows classifier names in column headers. • **Show Column Type** – in a metric table, shows types of metric and parameter values in column headers. For example, *Blocks Count : Integer*. In an instance table, it shows types of slot values in column headers. For example, *Year : Integer* or *Boat.Beam : String*. • **Show Column Icons** - shows icons in the column headers. • **Show Abbreviation in Column Type** – in a metric table, shows abbreviations of metric or parameter names in column headers, if they are specified. • **Show Units on Values** - in an instance table, shows unit symbols next to values. • **Show Units on Columns** - in an instance table, it shows unit symbols in column headers. • **Wrap Text** - wraps long text in table cells. The option is disabled by default. • **Limit Row Height** - strips cell entries that cover more than 10 element values or plain text lines. Three dots are added at the end of the stripped cell. The complete text is available in the tooltip or in the cell edit mode. • **Display Mode** - allows selection of one of the following data visualization options: - **Complete tree** - displays elements hierarchically with all their owners in the column header. - **Compact tree** - displays elements hierarchically with their direct and common owners in the column header. - **List** - displays elements in plain list. • **Use Acronyms** - recognizes acronyms everywhere in the text. Only capitalized terms are recognized as acronyms. Lowercase words are not interpreted as acronyms. • **Specification**– opens the Specification window(see page 913) of an appropriate table.
Suppress/Expand Criteria Area toolbar		

Button	Shortcut keys	Description
▲	N/A	Click to hide the **Criteria** area.
▼	N/A	Click to show the **Criteria** area.
Other buttons		
	Alt+Left Arrow	Click to navigate to the previously opened diagram, table, dependency matrix, or relation map.
	Alt+Right Arrow	Click to navigate to the previously opened diagram, table, dependency matrix, or relation map.

Specific toolbar buttons

The following table describes the toolbar buttons found only in specific tables.

Table name where button can be found	Button	Shortcut keys	Description
	Edit toolbar		
Instance Table	> Create with Parts	Alt+Insert	Click to create a new instance with parts.
Instance Table	> Delete with Parts	Alt+D	Click to remove selected elements both from the table and the model.
Metric Table	Calculate Metrics	N/A	Click to open the Calculate Metrics menu and then select the needed command(see page 468).
Requirement Table	Add Nested	Alt+Insert	Click to add a nested Requirement in the model.
	Layout toolbar		

Table name where button can be found	Button	Shortcut keys	Description
Metric Table	Show Metric Suites	N/A	Click to select the metric suites whose columns you need to show on the Metric table.
	Other buttons		
Glossary Table		N/A	Click to turn the glossary on/off in the project.

Related pages

- Legends(see page 530)

Table Criteria area

The **Criteria** area appears below the table toolbars. If it is not displayed, click in the table toolbar to expand it.

The following table describes the contents of the Criteria area according to different tables and provides the references to procedures when trying to specify any box of this area.

Table name where the box can be found	Box Name	Description	How to...
Generic Table Glossary Table	Element Type	Click ... to open the **Select Element Type** dialog and select one or more types of elements you need to display in the Generic table.	• Create a Generic Table without help of a wizard(see page 441) • Add or remove element types(see page 441)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Table name where the box can be found	Box Name	Description	How to...
Metric Table	Metric Suite	Click ... to open the **Select Metric Suite** dialog and select the metric suites you want to use for calculating the metrics of your model. If the Metric Table is created following the procedure described in Create a Metric table(see page 0), the Metric Suite box has one or more default values. Otherwise, you must specify the metric suite criteria to start working with the Metric Table.	• Create a Metric Table(see page 468)
Instance Table	Classifier	Click ... to open the **Select Classifier** dialog and select one or more classifiers whose instances you need to display in the Instance Table.	• Specify the classifier (see page 459)
All tables	Scope (optional)	Click ... to select one or more packages, which contain the elements you need to display in the table. Click to open the **Query** dialog and specify the criteria that define your scope.	• Specify the scope for the table(see page 417) • Specify the Scope Query(see page 415)
All tables	Filter	Type the phrase by which the element should be filtered. To open the menu with more filter options, click in the **Filter** box. To reset filter options, click in the **Filter** box.	• Filter table data(see page 438) • Save the filter criteria(see page 440)

Related pages

Basic tasks in tables

Tables are diagrams which allow you to manage any amount of model data in a spreadsheet-like form. It is much easier to analyze a group of data in one place, especially when you can create new elements, add existing elements, edit their properties directly from the table, and paste data from the clipboard. This allows you to sort and filter data, show, hide, reorder columns, and clear or edit individual cells. Exporting tables into more commonly used file formats, such as *.html*, *.csv*, or *.xlsx*, makes it simple to share them with stakeholders.

> **Important**
> Each table row represents an element. Each column represents element properties. The column header and row number header are highlighted in grey when selecting single or multiple cells.

The following procedures provide the basic tasks to get you started working with all modeling tool tables:

- Creating an empty table(see page 417)
- Specifying the scope for the table(see page 417)
- Working with cells(see page 419)
- Working with columns(see page 423)
- Working with rows(see page 427)
- Copying and pasting data(see page 435)
- Filtering table data(see page 438)
- Saving the filter criteria(see page 440)
- Exporting table(see page 440)

Related pages

- Legends(see page 530)
- Creating diagrams(see page 483)
- Editing property values(see page 921)

Creating an empty table

You can create an empty table in the same way you create other diagrams: Creating diagrams.(see page 483)

Specifying the scope for the table

You can specify the scope in the table Criteria area(see page 415) for every table.

To specify the scope

---

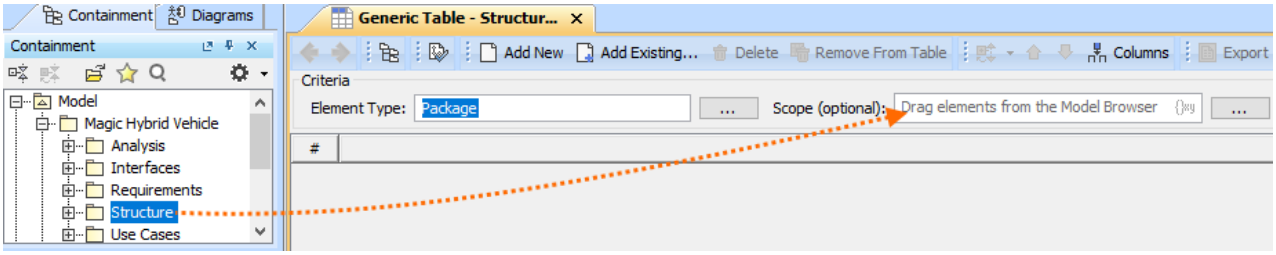
1. In the Criteria area of the table, click the ... button next to the **Scope (optional)** box.
2. In the **Select Scope** dialog, select one or more containers, such as packages or smart packages, and add them to the **Selected Elements** list.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

> The **Select Scope** dialog is a modification of the element **Selection** dialog(see page 1032).

3. Click **OK**.
   The scope is defined, and the contents of the table are updated with the elements from the selected scope. In the future, all elements created in that scope will automatically be added to the table.

Alternatively, you can select one or more elements directly in the Model Browser(see page 29) and drag them to the **Scope (optional)** box, as shown in the image below:



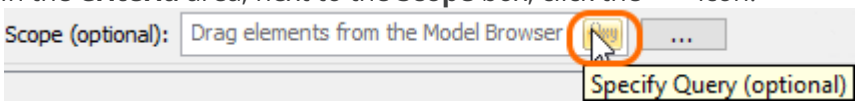
> If the contents of the table have not been updated, try the following:
> - Click on the **View** toolbar, in case a smart package is specified as the scope criteria.
> - Make sure the specified scope contains elements specified in the **Element Type** box, **Metric Suite** box, or **Classifier** box.
> - Make sure at least one element is specified for the table.

In addition to specifying the scope for the table yourself, you can set the expression to collect table elements automatically.

To specify the expression to collect row elements

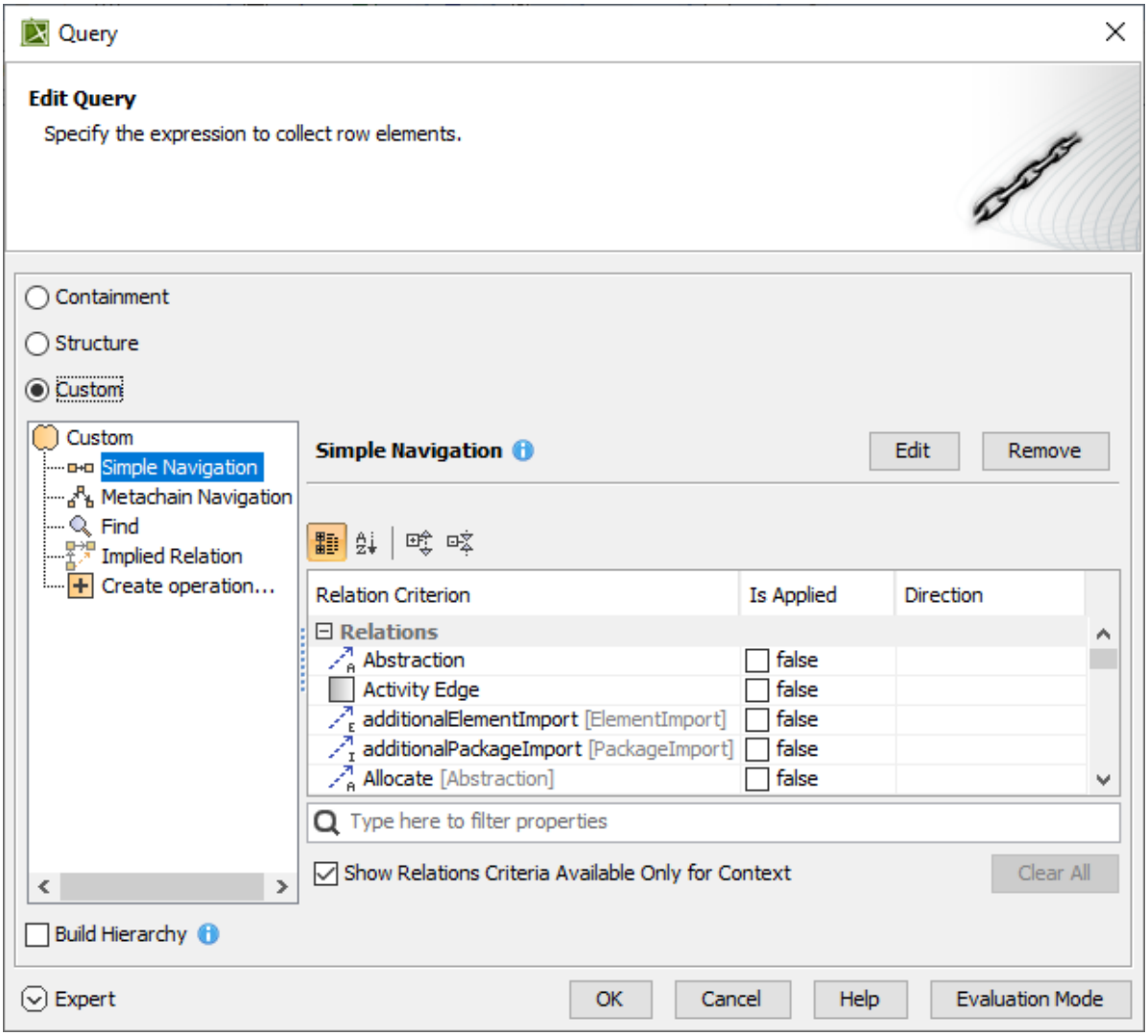
---

1. In the **Criteria** area, next to the **Scope** box, click the icon.



2. Once the **Query** dialog opens (see an image below), select **Custom** from the list of available query criteria.
3. Specify the desired expression to collect elements for your table automatically.
4. Click **OK**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Specifying the expression in the Query dialog

Working with cells

On this page

- Selecting cells(see page 420)
- Entering values into a cell(see page 420)
- Opening the Specification window of an element from a cell(see page 421)
- Clearing values from cells(see page 422)
- Navigation between cells(see page 423)

This section explains how to work with cells in the table: selection, entering and clearing values, opening an element's Specification window, and navigation.

Selecting cells

In all tables, you can select:

- Single cell.(see page 420)
- Multiple cells.(see page 420)
- Entire table.(see page 420)

To select a single cell

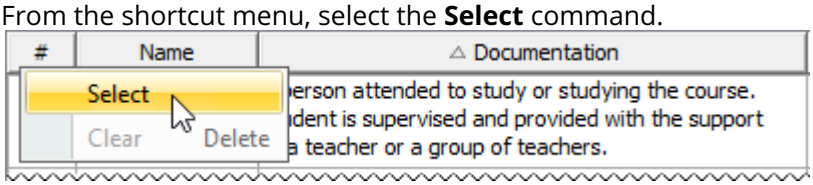
1. Press the left mouse button on the table.

To select multiple cells, do one of the following

- Press the left mouse button and drag a mouse pointer.
- Press Shift and use the left mouse button or any of arrow key from the keyboard.
- Press Ctrl and use the right mouse button.

To select an entire table

1. Right-click the column # header.
2. From the shortcut menu, select the **Select** command.



Entering values into a cell

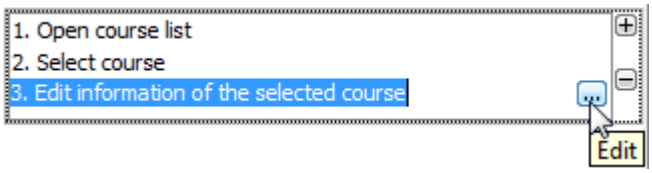
Individual cells in the table can not be edited if the text is highlighted in grey. This means the cell contains a predefined value and is non-editable.

All changes made in the table are saved in the model. The procedure below describes how to enter a value into an editable cell.

To enter a value into a cell

1. Double-click a cell you want to edit.
2. One of the following actions is available, depending on the property type of the column in which the selected cell is found:
   - Type the value directly in the selected cell.
   - Click the **Edit** button. The property value editor(see page 921) opens.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



- Click the **Add** (+) to specify a value.
- Select the value from the drop down list.

3. In the opened property value editor specify required properties.
   The value is saved in the model.

Opening the Specification window of an element from a cell

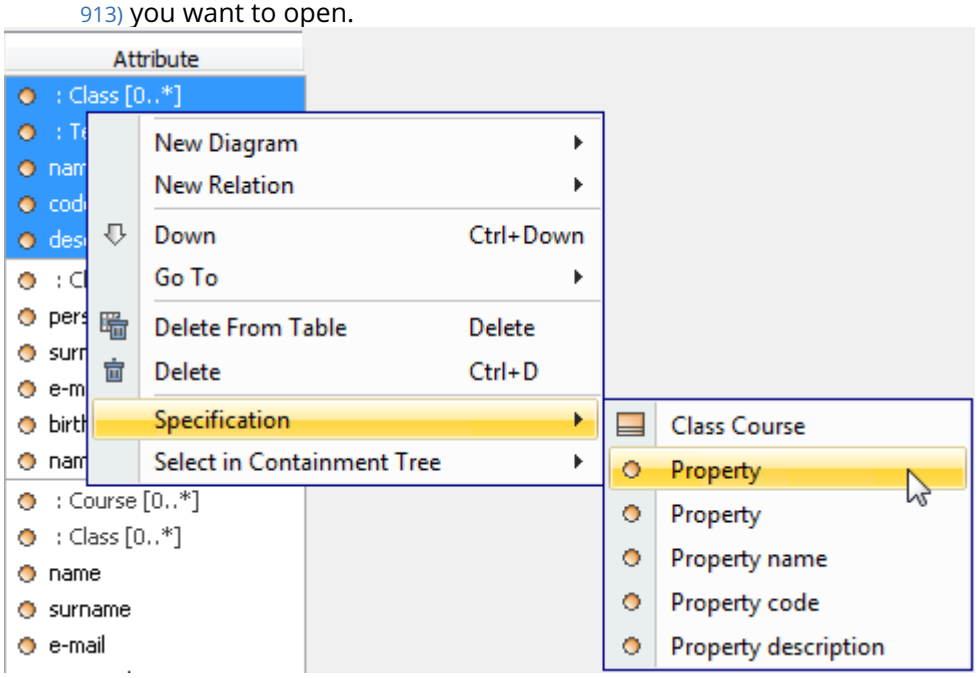
The property can be edited in the element's Specification window(see page 913) if it is not locked.

To open the element's Specification window(see page 913) from a cell

---

Do either:

- If cell is non-editable, double-click it.
- If cell is editable, right-click on a cell and from the shortcut menu select Specification. Then:
  - If the selected cell does not refer to any element, the Specification window of the element in the row will open.
  - If the selected cell refers to more than one element, the submenu with the list of appropriate elements will open. Choose the element whose Specification window(see page 913) you want to open.



Clearing values from cells

You can clear values from a single or multiple cells in cells containing the following data types: String, Boolean, Number, or Enumeration. You can clear the whole table, column, or row.

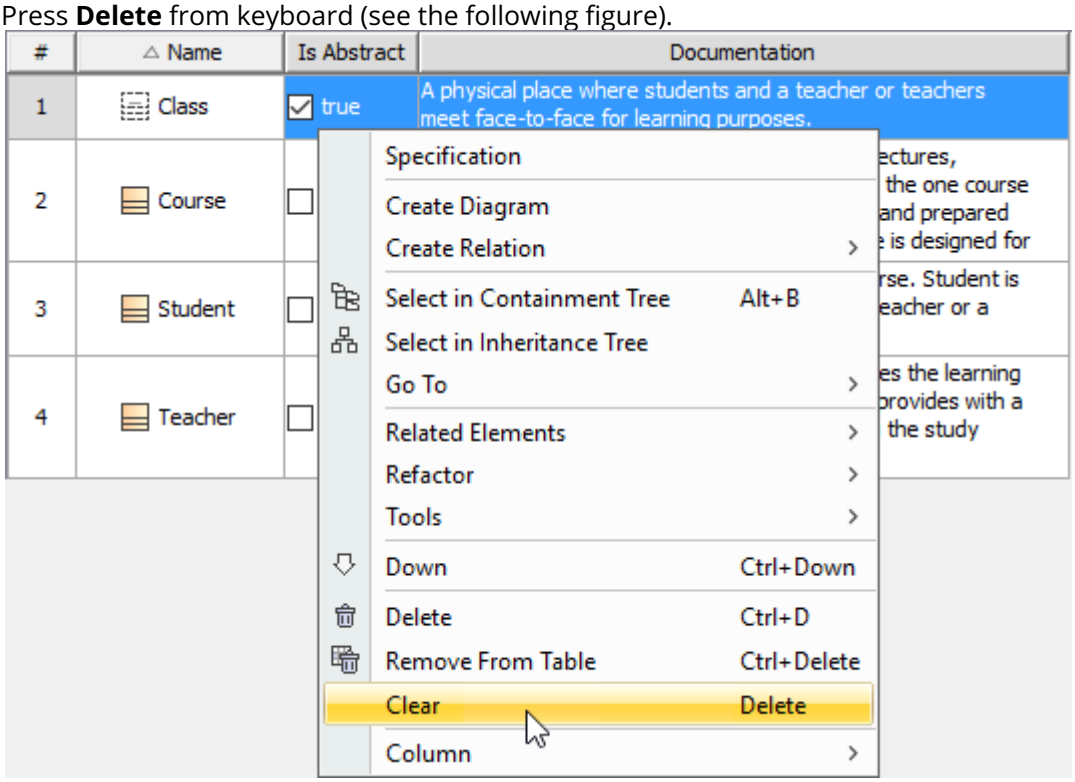
> **Data types**
> Only String, Boolean, Number, or Enumeration data types can be cleared from a cell.

The procedures below describes how to clear values from individual cells, from a block of cells, or from entire table:

- How to clear values from cells(see page 422)
- How to values from entire table(see page 423)

To clear values from cells

1. Select single or multiple cells in the table. How to select cells in the table (see page 420)
2. Do one of the following:
   - From the shortcut menu, select the **Clear** command (see the following figure).
   - Press **Delete** from keyboard (see the following figure).



Selected cells of String, Boolean, Number, or Enumeration data types become empty. The following example demonstrates that the *Documentation* cell is empty after clearing, because its

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

data type is a String. The *Is Abstract* cell value also is cleared because the value type is Boolean.

#	Name	Is Abstract	Documentation
1	Class	false	
2	Course	false	A unit of teaching provided in various formats (lectures, meetings, seminars, self-learning, etc.). Usually the one course covers the one subject. The course is designed and prepared by a teacher or a group of teachers. The course is designed for

To clear values from an entire table

1. Select the entire table. How to select cells in table >(see page 420)
2. Do one of the following:
   - From the table shortcut menu, click the **Clear** command.
   - Press Delete on the keyboard.

Navigation between cells

Navigate between cells using the arrow keys from keyboard and keyboard shortcuts.

Press	To move...
Right Arrow key	One cell to the right.
Left Arrow key	One cell to the left.
Down Arrow key	One cell down.
Up Arrow key	One cell up.
Tab	One cell to the right.
Shift+Tab	To the previous cell.
Enter	To the cell below.
Shift+Enter	To the cell above.

Working with columns

On this page

- Selecting an entire column(see page 424)
- Clearing values from an entire column(see page 424)
- Hiding a column(see page 424)
- Reordering columns(see page 425)

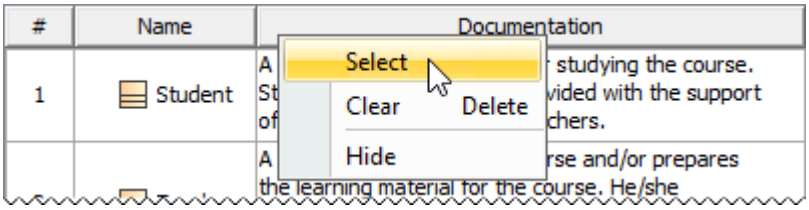
- Adding and removing columns(see page 425)
- Adding and removing custom columns(see page 425)
- Setting detailed column names(see page 426)

This section shows how to work with columns in the table: selection, clearing values, hiding, reordering, adding, and removing columns.

Selecting an entire column

To select an entire column

1. Right-click the column header.
2. From the shortcut menu, click the **Select** command.

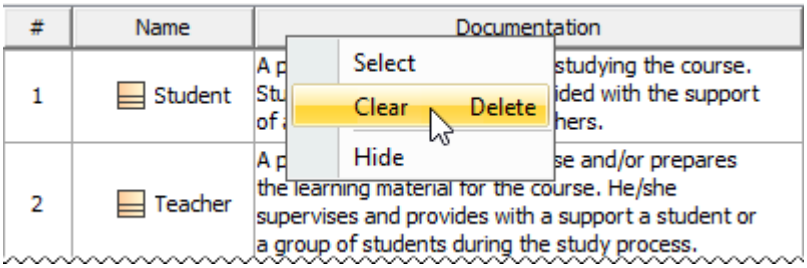


Clearing values from an entire column

> **Data types**
> Only String, Boolean, Number, or Enumeration data types can be cleared from a cell.

To clear values from an entire column

1. Right-click the column header.
2. Do one of the following:
   - From the shortcut menu, click the **Clear** command.
   - Press Delete from the keyboard.

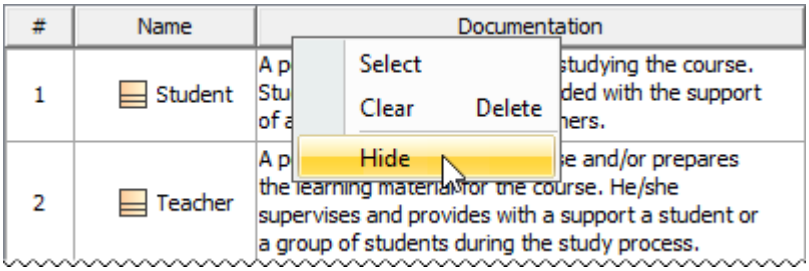


Hiding a column

To hide a column

1. Select single or multiple cells, and from the shortcut menu, select **Column** > **Hide**.
2. Right-click the column header to open the shortcut menu, and select the **Hide** command.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Reordering columns

To reorder columns

- Click the column header and drag it to a desired place. Only the first column cannot be moved.

Adding and removing columns

You can add or remove columns from table by using:

- The **Show Columns** button in Generic and Glossary tables.
- The **Columns** button in in Instance and Metric tables.

To add or remove columns

1. To open the available columns list:
   - On the table toolbar(see page 408), click the **Show Columns**/**Columns** to open the menu with common properties corresponding to element types that have been selected for the table.
   - On the table toolbar(see page 408), click **Show Columns**/**Columns** > **Select Columns** to open the dialog with all properties, including tags of stereotypes corresponding to selected element types.

   Using the **Select Columns** dialog, you can choose to display the properties of the specified element type, as well as the properties that are subtypes of the specified element type.

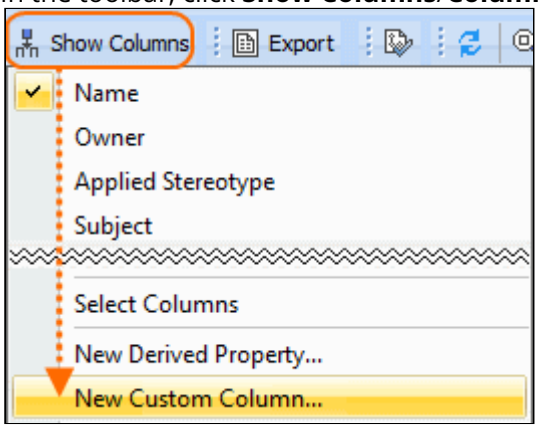
   The **Select Columns** command is not supported in the Glossary table.
2. Do one of the following:
   - Select the properties you want to see as columns in the table.
   - Unselect the properties you do not want to see as columns in the table.

Adding and removing custom columns

Custom columns are not supported in the Glossary table.

To add custom columns

1. In the toolbar, click **Show Columns/Columns** > **New Custom Column**.

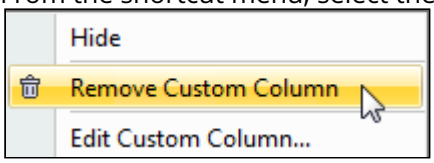


2. In the **Expression** dialog, specify criteria for the custom column, and click **OK**.

Additional information
- You can read more about it in Defining expressions(see page 2152), as the procedure for custom columns and for derived properties is the same.
- You can also select to show implied relationships in the table. The procedure for specifying which relationships to display in the table is the same as described in Specifying dependency criterion(see page 335).

To remove custom columns

1. Right-click the custom column header you want to remove.
2. From the shortcut menu, select the **Remove Custom Column** command.



Setting detailed column names

Column names in the table header are set automatically. They are element property names. You cannot change column names. If an element has some properties with the same name (for example, in associations or if a tag of stereotypes is selected as a column), detailed column names could provide more precise information.

To see detailed column names

- On the Options toolbar, click ⚙ ▾ and then select **Show Detailed Column Name**.
  The group name (between brackets) to which the selected property belongs, and/or the stereotype name (just before the property name), will be displayed in the column heading.

The following diagram shows three different columns with the same name for the association element: **Name**, **Name** (Role of A), and **Name** (Role of B) and a column for the class element with the stereotype «Teacher». **Name** (Tags). If the command **Show Detailed Column Name** were not selected, there would be four columns with the same headings, i.e., **Name**.

Criteria

Element Type: Association, Class ...

Name with a group name

Name with a tag name

#	Name	Documentation	Name (Role A)	Name (Role B)	«Teacher».Name (Tags)
1	instantiates		hasCourses	takePlace	
2	Course	The unit of the teaching information.			Brigit Smith
3	Class	The room where courses take place.			

Working with rows

This section outlines how to work with rows in the table: selection, clearing values, adding and removing rows.

- Adding and removing rows(see page 427)
- Selecting an entire row(see page 428)
- Organizing rows(see page 429)
- Clearing values from rows(see page 430)
- Displaying/hiding element numbers in a row(see page 430)
- Displaying row elements hierarchically(see page 431)
- Nesting additional elements in a hierarchical table(see page 434)

Back to Top(see page 0)

« Working with columns(see page 423) Adding and removing rows »(see page 427)

Adding and removing rows

Adding a new row
You can create a new element in the model by adding a new row in the table. Each type of table requires a different procedure to add new row(s).

To add a new row in the Generic and Glossary tables

1. On the table toolbar, click the Add New button.

> **More information**
> Learn more about adding terms in Glossary Table >(see page 452)
> Learn more about adding new elements in Generic Table >(see page 441)

To add a new row in the Instance Table

1. On the table toolbar, click > Create

> **More information**
> Learn more about filling the table with instances >(see page 459)

To add a new row in the Metric Table

1. On the table toolbar, click Calculate Metrics > Add New Metric with Different Parameters

> **More information**
> Learn more about calculating new metrics with different parameter values >(see page 468)

To add an existing element as a row in the Generic, Instance, Glossary, Requirement, and Metric tables

1. In the table toolbar, click the Add Existing... button.
2. Select the element existing in the model and click **OK**.

Removing a row from a table
To remove a row from a table

1. Select the whole row or select one cell of the row you want to remove.
2. Do one of the following:
   - Click the **Remove From Table** button on the table toolbar.
   - Press **Ctrl+Delete** from keyboard.

Removing an element from a model
You can delete an element from the model when removing the row from a table.

To remove an element from a table

1. Select the whole row(see page 428) or select one cell of the row you want to remove.
2. Do one of the following:
   - Click the **Delete** button on the table toolbar.
   - Press **Ctrl+D** from keyboard.

Selecting an entire row

To select the entire row

- Click the number cell of the row you want to select.

#	Name	Documentation
1	Student	A person attended to study or studying the course. Student is supervised and provided with the support of a teacher or a group of teachers.
2	Teacher	A person who designs the course and/or prepares the learning material for the course. He/she supervises and provides with a support a student or a group of students during the study process.

Organizing rows

Sorting rows
You can change the order of rows in the following ways:

- Sort the rows by one column.(see page 429)
- Sort rows by multiple columns.(see page 429)

To sort rows by a selected column

- Click the header of the column by which you want to sort the rows in the table. A small arrow showing the automatic sort order appears on the column header. Clicking the header once sorts the rows in ascending order. Clicking the header twice sorts the rows in descending order. All the rows in the table are automatically renumbered after sorting.

Ascending

#	Name	Make : String	Year : Integer
1	WN123AB	Hanter	1977
2	WN234CD	Calbaria	1999

Descending

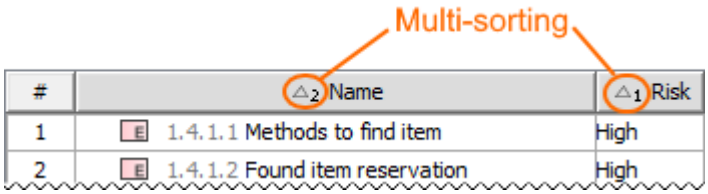
#	Name	Make : String	Year : Integer
1	WN234CD	Calbaria	1999
2	WN123AB	Hanter	1977

To sort rows by multiple columns

- Press Ctrl + click on the first column header you want to sort the rows in the table. Repeat this action for the next column header you want to sort the rows in the table. A minimum of two column headers should be selected to multi-sort rows. A small arrow with a number showing the automatic multi-sort order appears on the column header. All the rows in the table are

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

automatically renumbered after multi-sorting.



Back to Top(see page 0)

Moving rows up and down
To move a row up or down

1. Select the row.
2. Do either:
   - On the table toolbar, click the **Up** or **Down** button appropriately.
   - Right-click the row and select the **Up** or **Down** command appropriately.
   - Press Ctrl+Up Arrow or Ctrl+Down Arrow appropriately.

   The selected row as well as associated rows are automatically renumbered after moving.

Clearing values from rows

To clear values from an entire row

1. Right-click the number cell to select the entire row.
2. From the shortcut menu, click **Clear**.
   Alternatively, you can press **Delete** on your keyboard after selecting the entire row.

Data types
Only String, Boolean, Number, or Enumeration data types can be cleared from a cell.

To clear a value from a cell

1. Right-click the specific cell of the row.
2. From the shortcut menu, click **Clear**.
   Alternatively, you can press **Delete** on your keyboard after selecting the cell.

Displaying/hiding element numbers in a row

When creating a new table, elements represented as rows are automatically numbered in the name column. The following procedure is important for those who modify existing tables created before Version 18.4.

To display/hide element numbers in the table

1. Open the **Specification** window of the table.
2. Set the **Show Element Number** property value to *true* to display element numbers, or *false* to hide them.
   The element numbers are displayed in the name columns before the element name (see the following example).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

#	Name
1	44 MagicLibrary
2	44.4 Item reservation
3	44.4.4 Items purchase
4	44.4.4.4 Item request management
5	44.4.4.4.4 Requests status tracking
6	44.4.4.4.3 Requests confirmation
7	44.4.4.4.2 Requests deny
8	44.4.4.4.1 New items requests evaluation
9	44.4.4.3 External global repository

Displaying row elements hierarchically

Hierarchical tables are very useful for managing and reviewing data. Hierarchy in a table is a tree-like structure where all the rows are listed according to the elements containment.

> **Important**
> The hierarchy in the Instance table(see page 459) is displayed according to an instance value that is a slot value of another Instance Specification. Composite instances are displayed.

Displaying elements hierarchically in a table
To display elements hierarchically

---

1. In the table toolbar, click
.
2. Under the **Display Mode** option choose:
   - **Complete tree** to display row elements in a hierarchy where all the owners are represented in separate rows.

#	Name
1	Parts
2	SUV parts
3	Main structure
4	Brake
5	Caliper
6	Engine
7	Transmission
8	System
9	Pad
10	Wheel

- **Compact tree** to display row elements in a hierarchy where the common owners are grouped together in a separate row.

#	Name
1	Main structure [Parts::SUV parts]
2	Brake
3	Caliper
4	Engine
5	Transmission
6	System
7	Pad
8	Wheel

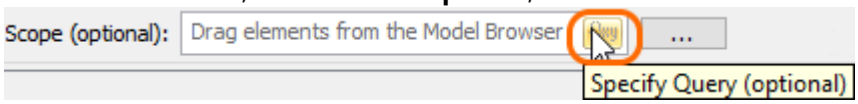
The hierarchy is created in the table by listing all the elements as they are shown in the Containment tree.

Displaying row elements in custom hierarchy

To display elements in custom hierarchy

---

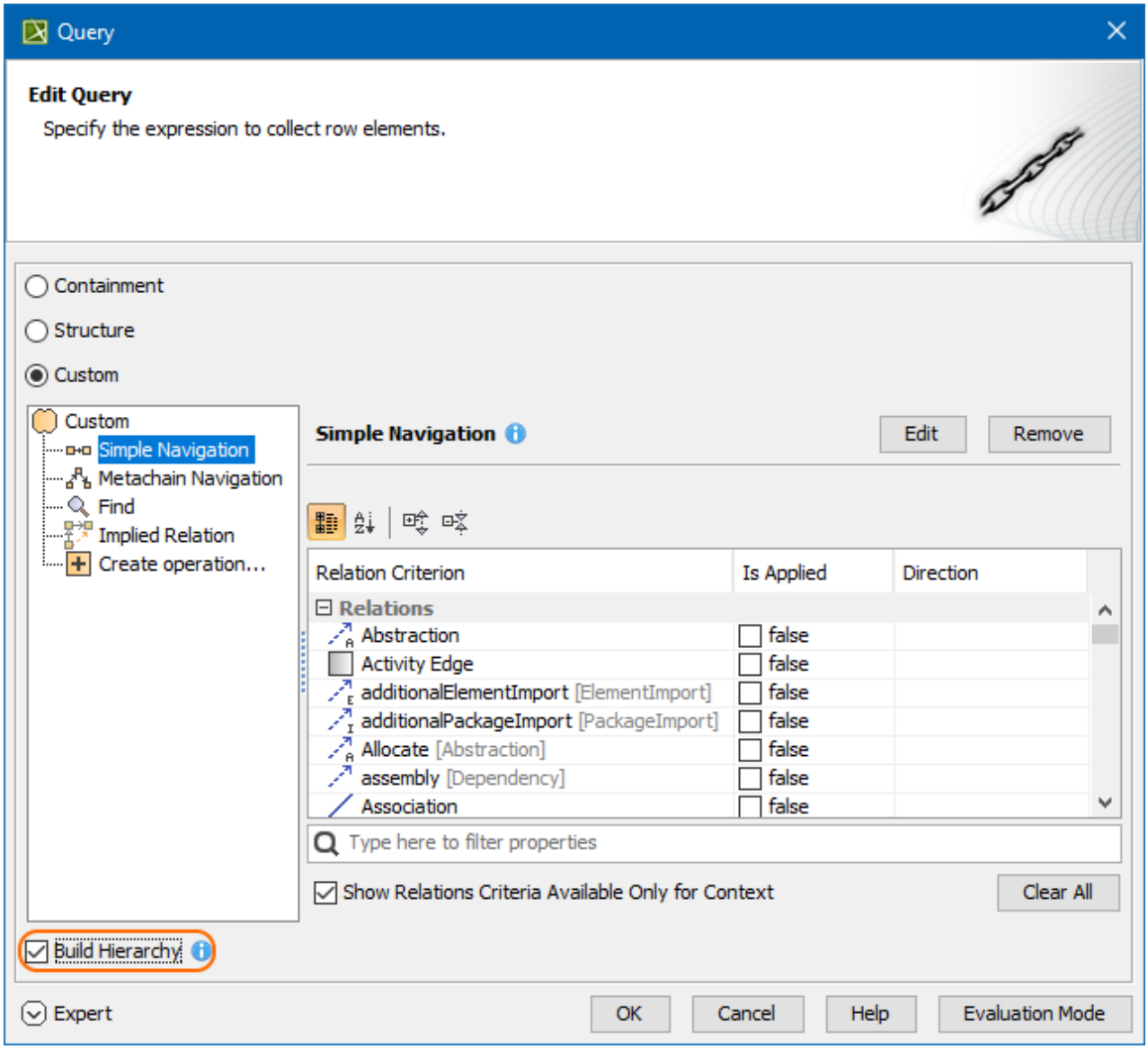
1. In the **Criteria** area, next to the **Scope** box, click the icon.



2. Specify the expression to collect table elements:

- Select **Containment** to display the decomposition of elements related through the Containment relationship.
- Select **Structure** to display the decomposition of Part Properties related through the Composition relationship.
- Select **Custom** to display the decomposition of elements related through any criteria of your choice.

> If you have selected Custom, please specify the expression(see page 1227).

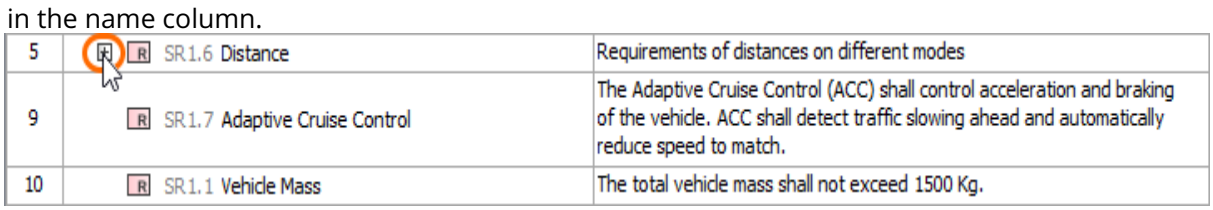


Specifying expression in the Query dialog

Expanding/collapsing nodes in a hierarchical table

To expand/collapse nodes in a hierarchical table

- Click ⊞ / ⊟ in the name column.



5	SR1.6 Distance	Requirements of distances on different modes
9	SR1.7 Adaptive Cruise Control	The Adaptive Cruise Control (ACC) shall control acceleration and braking of the vehicle. ACC shall detect traffic slowing ahead and automatically reduce speed to match.
10	SR1.1 Vehicle Mass	The total vehicle mass shall not exceed 1500 Kg.

- Select the specific row, hold Ctrl, and then press the right arrow key/left arrow key on the keyboard.

« Displaying/hiding element numbers in a row(see page 430)

Nesting additional elements in a hierarchical table »(see page 434)

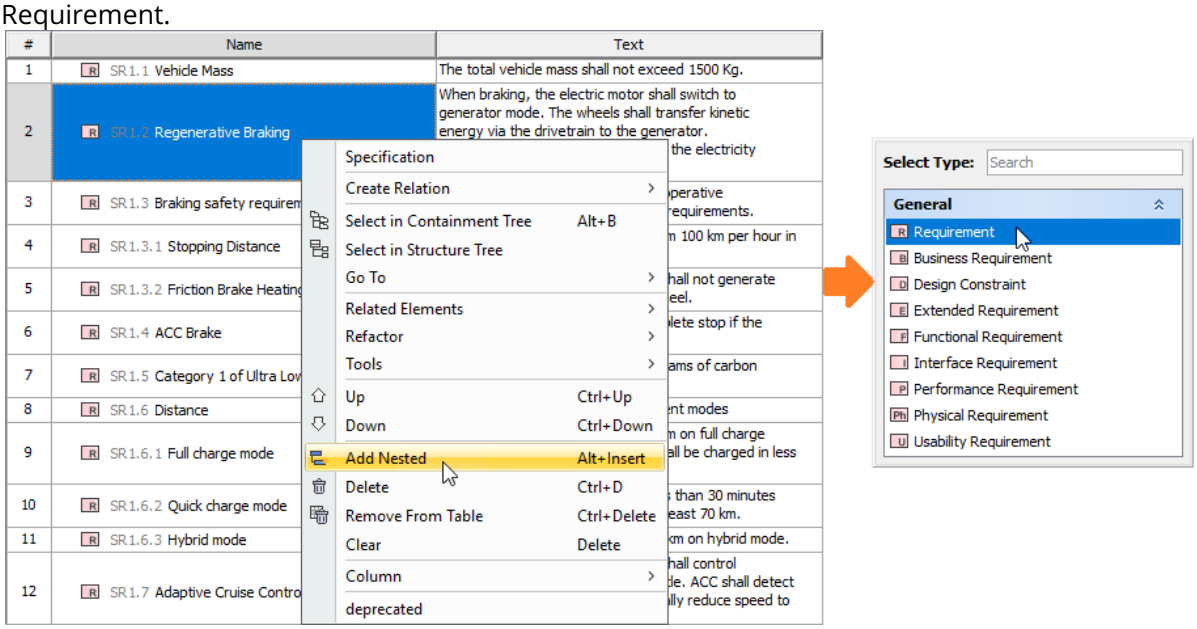
Nesting additional elements in a hierarchical table

Apart from displaying elements hierarchically, you can also build a hierarchy by nesting additional elements within the selected element directly in the table. The **Complete** tree view is enabled in the table by default.

To build a custom hierarchy

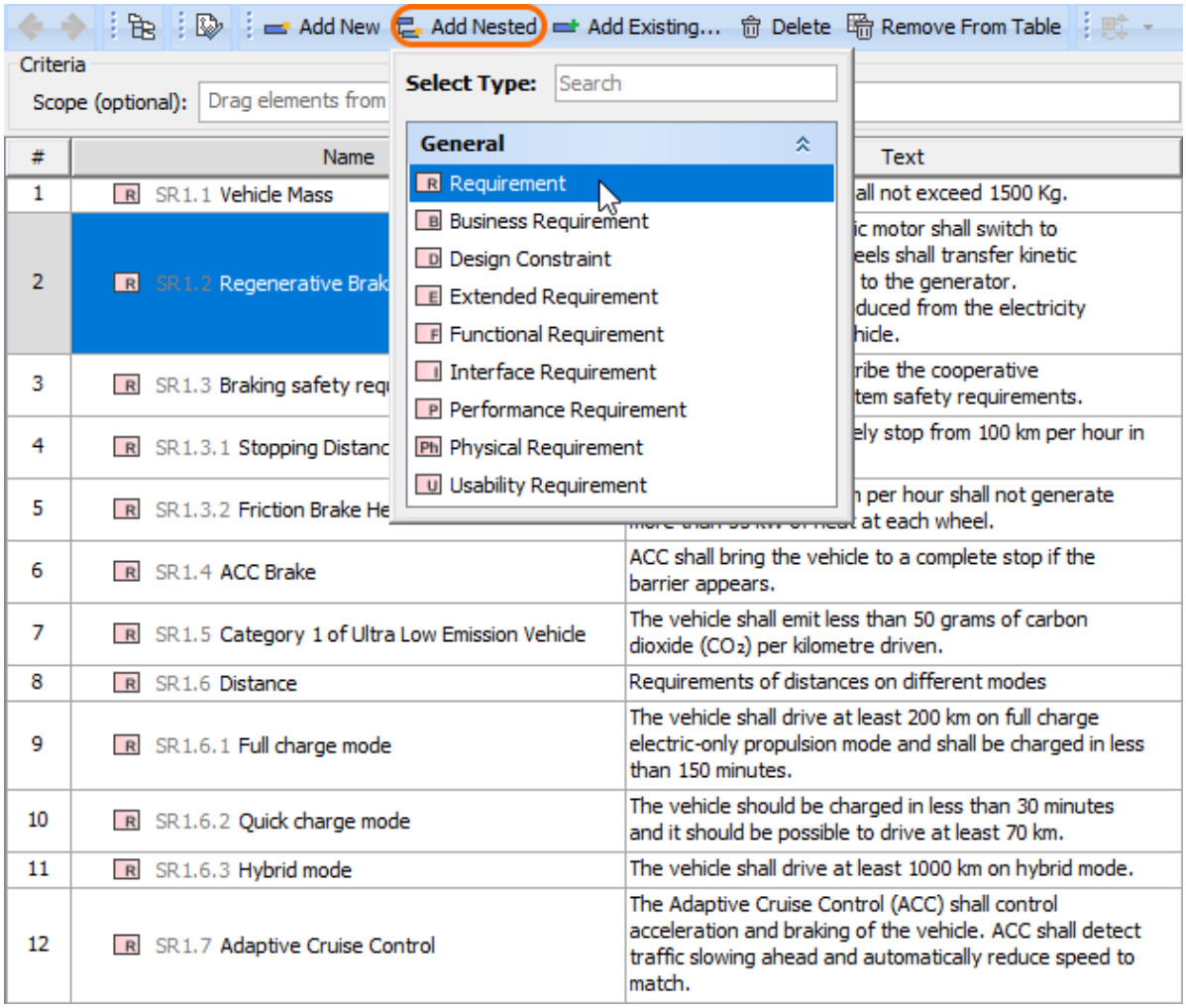
Do one of the following:

- In the table, right-click the specific element to nest an additional element within, select **Add Nested** in the drop-down menu, and specify the type of the element to create, e.g., Requirement.



- Select the specific table element to nest an additional element within, click the **Add Nested** button in the table toolbar(see page 27), and specify the type of the element to create, e.g., Requirement.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Please note that the **Add Nested** button is available in the table toolbar of the Requirement table[74] only.

Copying and pasting data

You can transfer information by copying and pasting data between tables and other resources, such as Excel spreadsheets, documents, web pages, email messages, presentations. You are able to copy and paste data:

- From other resources to modeling tool tables(see page 435)
- From modeling tool tables to other resources(see page 436)
- Between cells in the modeling tool table(see page 437)

From other resources to modeling tool tables

You can copy information from other resources (e.g. Excel spreadsheets, documents, web pages, email messages, presentations) and paste it to a modeling tool table. After pasting the data to the modeling

74 https://docs.nomagic.com/display/SYSMLP190SP4/Requirement+Table

tool table, the new elements are created in the model. If elements already exist in the model, the information updates. The ID prefixes are set automatically after pasting data in modeling tool tables.

The following rules should be satisfied for a successful copy-and-paste operation to modeling tool tables:

- The cells must be editable.
- The number of columns in the other resource should be the same as the number of columns in the modeling tool table.
- The data types of columns in the modeling tool table must be compatible with the copied information from the other resource.

How to copy data from another resource and paste it to modeling tool :

- Copy and paste to Instance table >(see page 459)
- Copy and paste to Generic table >(see page 441)
- Copy and paste to Glossary table >(see page 452)

To copy data from another resource and paste it to the modeling tool table

---

1. Copy the table from another resource.

   - The number of columns in that table should be the same as the number of columns in the table.
   - The data types of columns in the table must be compatible with the copied information.

2. Open the table where you want to paste the copied data.
3. In the Criteria area, click

   to specify the Element Type (in Generic table), Classifier (in Instance table), or Metric Suite (in Metric table).
4. In the table toolbar, click the **Show Columns/Columns** button and choose the columns that you want to fill with the copied information.

   The cells must be editable.

5. Do one of the following:
   - Press Ctrl+V (Cmd+V on OS X).
   - In the main toolbar, select **Edit** > **Paste.**

   The new elements are created in the model. If the element already exists in the model, the information updates.

   If elements have an ID prefix in another resource, they are set automatically after the copy-and-paste command only for pasted elements in the table. Manage ID prefixes in the **Element Numbering** dialog(see page 1046).

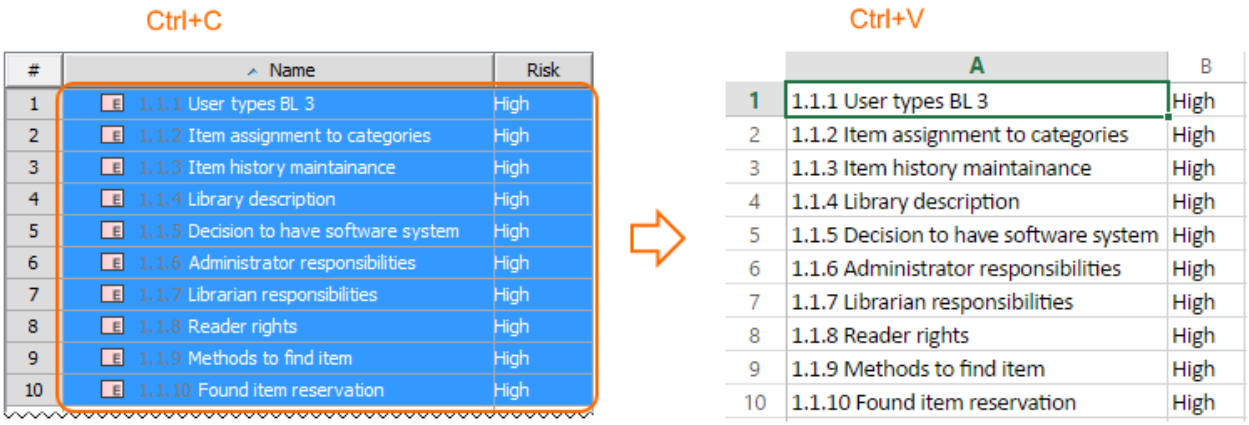
Back to Top(see page 0)

From modeling tool tables to other resources

You can transfer information by copying and pasting data from modeling tool tables to other resources (e.g. Excel spreadsheets, documents, web pages, email messages, presentations).

To copy data from modeling tool tables and paste it to other resources

1. Select the required cells in the modeling tool table. How to select cells >(see page 0)
2. Do one of the following:
   - Press Ctrl+C (Cmd+C on OS X).
   - In the main toolbar, select **Edit** > **Copy**.
   The data is stored in the clipboard.
3. Open the other resource where you want to paste the data and select the first cell in the area you want to paste what you copied (e.g. Excel spreadsheet).
4. Press Ctrl+V.
   The contents of the cells are transferred to other resource.



Between cells in the modeling tool table

You can fill the table cells by copying the content from the modeling tool table cell(s) and pasting it to another cell(s) in the same or another modeling tool table(s). After pasting the data to a modeling tool table the new elements are created in the model. If elements already exist in the model, the information updates. The ID prefixes are set automatically after pasting data in modeling tool tables.

The following rules should be satisfied for a successful copy-and-paste operation to modeling tool tables:

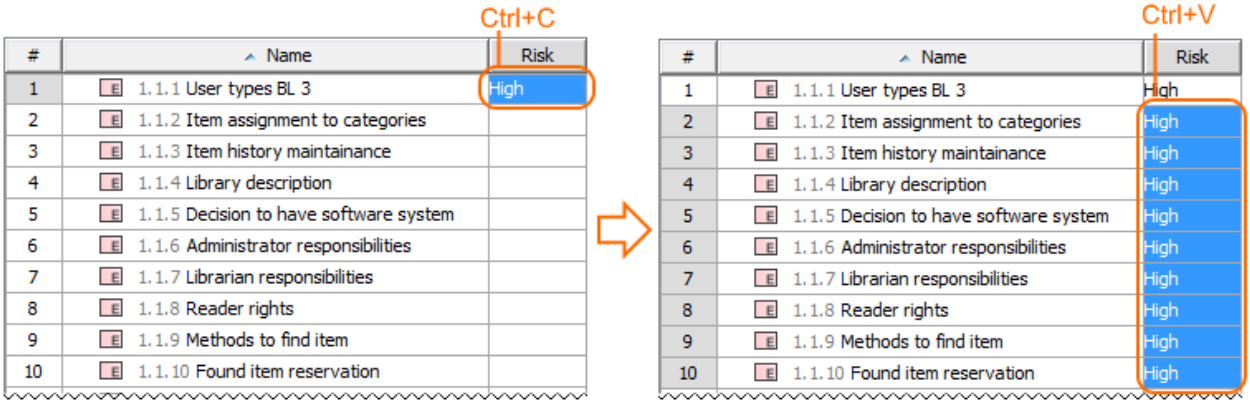
- The cells must be editable.
- The data types of columns in the modeling tool table must be compatible with copied information.
- The cells contain the following data types: String, Boolean, Number, or Enumeration.

Copy and paste cell contents in a modeling tool table

1. Select the cells that contain the data you want to copy. How to select cells >(see page 0)
2. Do one of the following:
   - Press Ctrl+C (Cmd+C on OS X).
   - In the main toolbar, select **Edit** > **Copy**.
   The data is stored in the clipboard.
3. Select a single or multiple cells where you want to paste what you copied.
4. Do one of the following:
   - Press Ctrl+V (Cmd+V on OS X).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- In the main toolbar, select **Edit** > **Paste.**
The content of the cell(s) is duplicated in another cell(s).



Filtering table data

By filtering data in the table you can quickly find the elements you need. This is especially handy when you are working with a large table containing many rows and columns. Elements can be filtered by a phrase specified in the Filter box of the table Criteria area(see page 415). The following figure shows the Generic table before and after filtering.

Before filter

Criteria

Element Type: Use Case ... Scope (optional): Drag elements from the Mode ... Filter:

#	Name	Pre Condition	Goal	Basic Flow of Events
1	Create User	The user has permmisions to create/modify information	Create a new user	1.Type the information of the user 2. Select the role type 3. Specify the permissions for the new user
2	Create Course	The user has permmisions to create/modify information	Create a new course	1. Type the course information 2. Select the class where the course take place 3. Specify the schedule of the course
3	Modify Class	The user has permmisions to create/modify information	Modify selected class	1. Open class list 2. Select class 3. Edit information of the selected class
4	Create Class	The user has permmisions to create/modify information	Create a new class	1. New class creation form is opened 2. Enter all class information (class name, schedule)
5	Modify User	The user has permmisions to create/modify information	Modify selected user	1. Open the user information form 2. Modify the information of the selected user
6	Modify Course	The user has permmisions to create/modify information	Modify information of the selected course	1. Open course list 2. Select course 3. Edit information of the selected course

After filter

Criteria

Element Type: Use Case ... Scope (optional): Drag elements from the Mod ... Filter: course

#	Name	Pre Condition	Goal	Basic Flow of Events
2	Create Course	The user has permmisions to create/modify information	Create a new course	1. Type the course information 2. Select the class where the course take place 3. Specify the schedule of the course
6	Modify Course	The user has permmisions to create/modify information	Modify information of the selected course	1. Open course list 2. Select course 3. Edit information of the selected course

Filtered rows and columns in the Generic table.

The search for the phrase can be performed either in the whole table or in selected columns. You can specify filter settings by clicking in the Filter box. The list with the following main areas opens:

- Specify the columns to search (1).

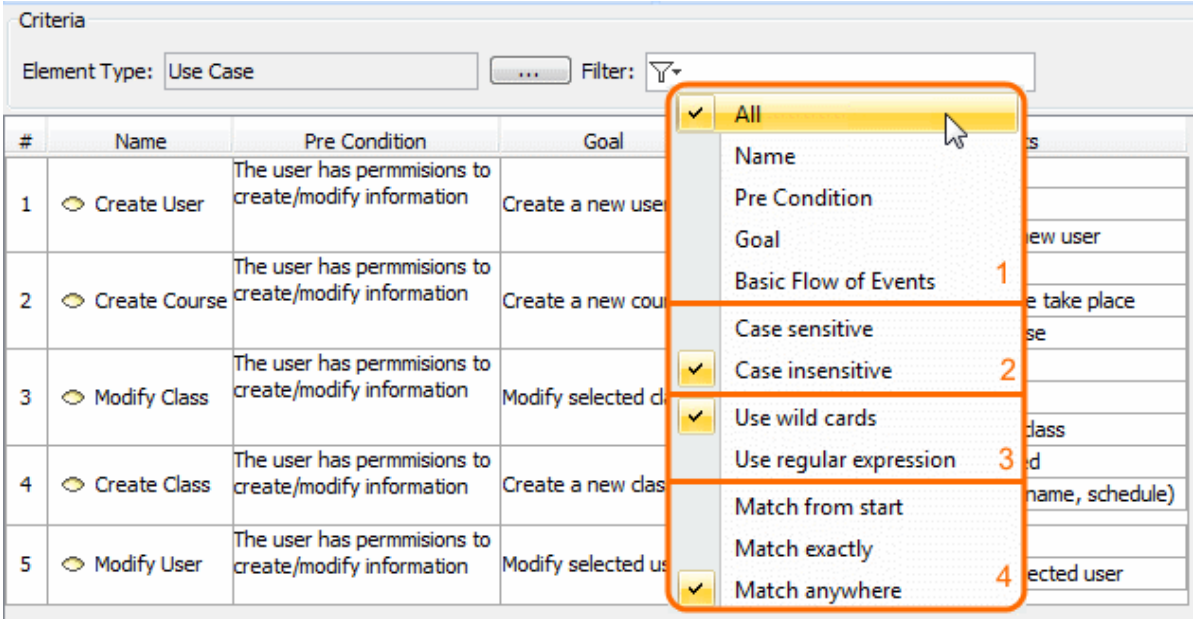
  You can search in the entire table by selecting the **All** option or select one or several particular columns.

- Specify case sensitivity (2).
- Specify wild card or regular expression (3).

  Using wild card
  In the **Filter** box, type a phrase using the following wildcard characters:
  - "*" substitutes any range of characters. If your search phrase is "*dd", all the elements containing "dd" in their names will be found. For example, "Address" and "ThreadDeath".
  - "?" substitutes only one character. If your search phrase is "?dd", only the elements with "dd" in their names starting from the second character can be found. For example, "Address". "ThreadDeath" cannot be found in this case, because it contains more than one character before the search phrase "dd".

  Be sure the **Match Anywhere** command is unchecked when using wildcard characters!

- Specify how the results should match your key words (4).



The main areas in the filter settings.

You can save the filter criteria for a particular table. The filter criteria will be saved even after reloading the project or restarting modeling tool.

Saving the filter criteria

To save the filter criteria

- On the **Options** toolbar, click ⚙ ▾ and then select **Save Filter Criteria**.

> ⚠ When exporting the table, only the filtered out columns and rows will be included in the result.

Exporting table

You can export the table to an *.html, *.csv, or *.xlsx file.

Before exporting a table, you can:

- Customize the template.html according to your needs.

  > ⚠ - The style of *.html tables is specified in template.html which can be found in <MagicDraw installation directory>\data\table.
  > - The file stores cascading style sheets (CSS) that define the appearance of an exported table.

- Prepare a table by showing only particular data.

To export a table

---

1. On the Publish toolbar, click the **Export** button.
2. In the opened dialog, do the following:
   a. Browse for a location to save the table.
   b. Type the exported table name.
   c. Select the exported table format.
3. Click **Save** when you are done.
   The table is exported to the location you have specified.

Generic table

On this page

- Description(see page 441)
- Purpose(see page 441)
- Example(see page 442)
- Specific tasks in Generic Table(see page 442)

Description

This feature is available in Standard, Professional, Architect, and Enterprise editions.

The Generic Table in MagicDraw gives you the ability to manage selected elements and their properties in a single place.

Purpose

A Generic Table displays a list of selected elements and their properties in a single place. Using Generic Tables enables you to:

- Edit property values of listed model elements.
- Create selected type elements.
- Create new derived properties or custom columns.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example

#	Name	Pre Condition	Goal	Basic Flow of Events
1	Create User	The user has permissions to create/modify information	Create a new user	1.Type the information of the user 2. Select the role type 3. Specify the permissions for the new user
2	Create Course	The user has permissions to create/modify information	Create a new course	1. Type the course information 2. Select the class where the course take place 3. Specify the schedule of the course
3	Modify Class	The user has permissions to create/modify information	Modify selected class	1. Open class list 2. Select class 3. Edit information of the selected class
4	Create Class	The user has permissions to create/modify information	Create a new class	1. New class creation form is opened 2. Enter all class information (class name, schedule)
5	Modify User	The user has permissions to create/modify information	Modify selected user	1. Open the user information form 2. Modify the information of the selected user
6	Modify Course	The user has permissions to create/modify information	Modify information of the selected course	1. Open course list 2. Select course 3. Edit information of the selected course

The Generic Table project sample allows you to read about the Generic Table feature and analyze examples within that project sample.
To open the sample, do either:

- On the Welcome screen, select **Samples** > **Diagrams** > **Generic table**.
- Go to *<MagicDraw installation directory>\samples\diagrams* and open the *generic table.mdzip* file.

All the examples given in this section are based on the data from this sample.

Specific tasks in Generic Table

All procedures dedicated only for Generic Table are as follows:

- Creating Generic Table(see page 442)
- Adding or removing element types(see page 447)
- Filling the table with elements(see page 448)

Related pages

- Legends(see page 530)

Creating Generic Table

You can create a generic table by any of the methods described in creating any diagram in MagicDraw(see page 483).

When creating a Generic Table you should specify the following:

1. Denote the name of the table and the owner package (MANDATORY).

2. Define element types that will be used in the table (MANDATORY).
3. Select elements corresponding to the element types (OPTIONAL).
4. Specify table columns (OPTIONAL). Columns represent element properties, so the table creation could be finished without specifying columns.

You can modify the table after it is created.

This section describes several of the most popular ways of creating a generic table:

- Creating a generic table without using a wizard(see page 443)
- Creating a Generic Table using the diagram creation wizard(see page 445)
- Creating a Generic Table for a set of selected elements using the diagram creation wizard(see page 447)

Creating a generic table without using a wizard

Once created, the table can be filled with data using the generic table environment capabilities.

To create a Generic Table without using a wizard

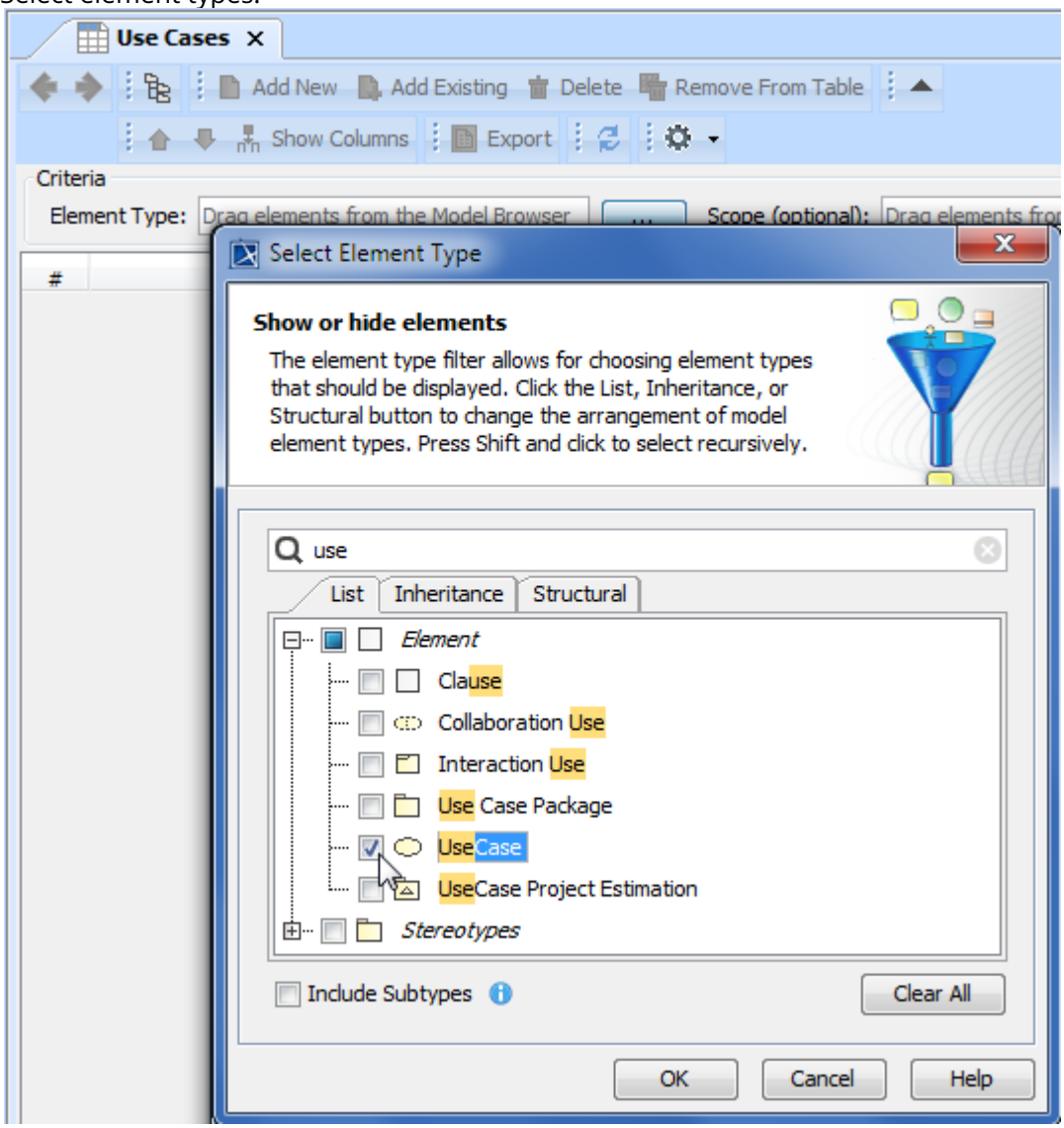
---

1. In the Containment tree(see page 32), select an element to be the owner of the Generic Table.
2. Do one of the following:
   - From the main menu, select **Diagrams** > **Create Diagram**. Type “gen” and press **Enter**.
   - On the main toolbars, click the **Create Diagram** button. Type “gen” and press **Enter**.
   - Press **Ctrl+N**. Type “gen” and press **Enter**.
   - Right-click the element and from the shortcut menu select **Create Diagram** > **Other Diagrams** > **Generic Table**.

   The newly created Generic Table opens on the right side of the application window.
3. Type the name of the table.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

4. Select element types.



Select desired element types and/or stereotypes in the **Select Element Types** dialog.
The **Element Type** box in the **Criteria** area of the Generic Table will display the selected items.
This action is optional, so you can continue the creation without selecting any type or stereotype.
You will be able to modify selected element types when the Generic Table is created.

> ⓘ If you drag an element from the Containment tree(see page 32) to the table, the type of this element will be automatically set as the **Element Type** box value.

5. Fill the table with elements:
   - Specify the scope, from which the elements should be added to the table. All the elements created in that particular scope will be automatically added to the table.

If the contents of the table have not been updated, try the following:

- Click on the **View** toolbar, in case a smart package is specified as the scope criteria.
- Make sure the specified scope contains elements that are specified in the **Element Type** box. Element types must correspond to the selected element types!
- Make sure at least one element type is specified for the table.

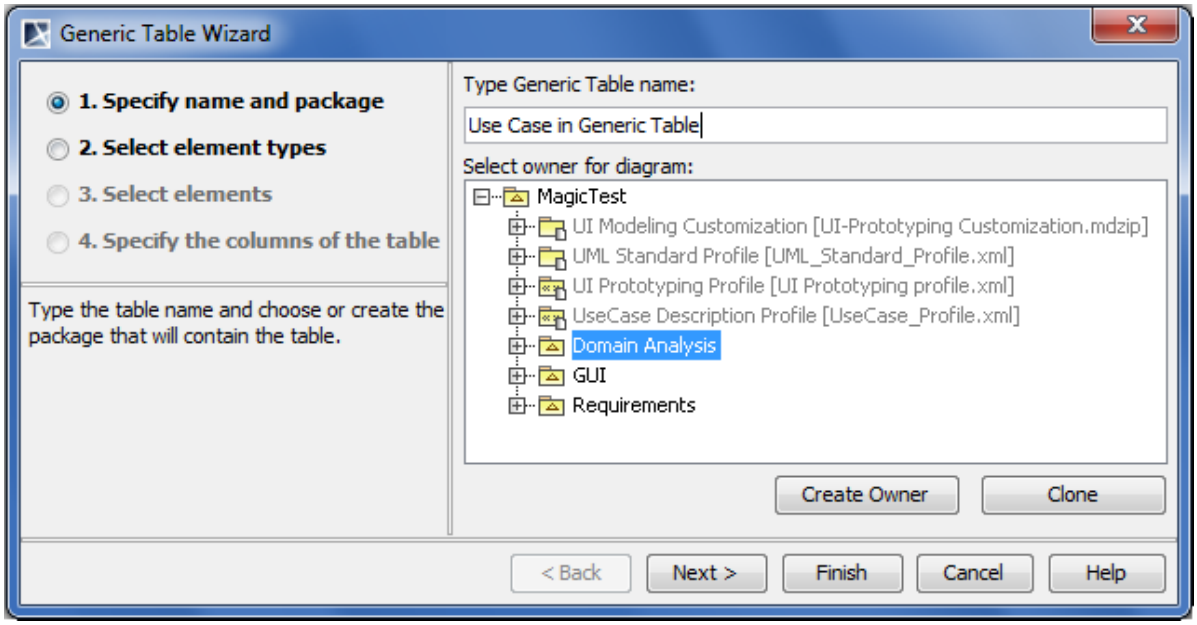
- Create new elements directly from the table. How to craete new elements >>(see page 449)
- Copy and paste elements from another resource. How to paste elements >>(see page 451)
- Add existing elements to the table individually. How to add existing elements >>(see page 449)

Creating a Generic Table using the diagram creation wizard

Once created the table is already filled with data.

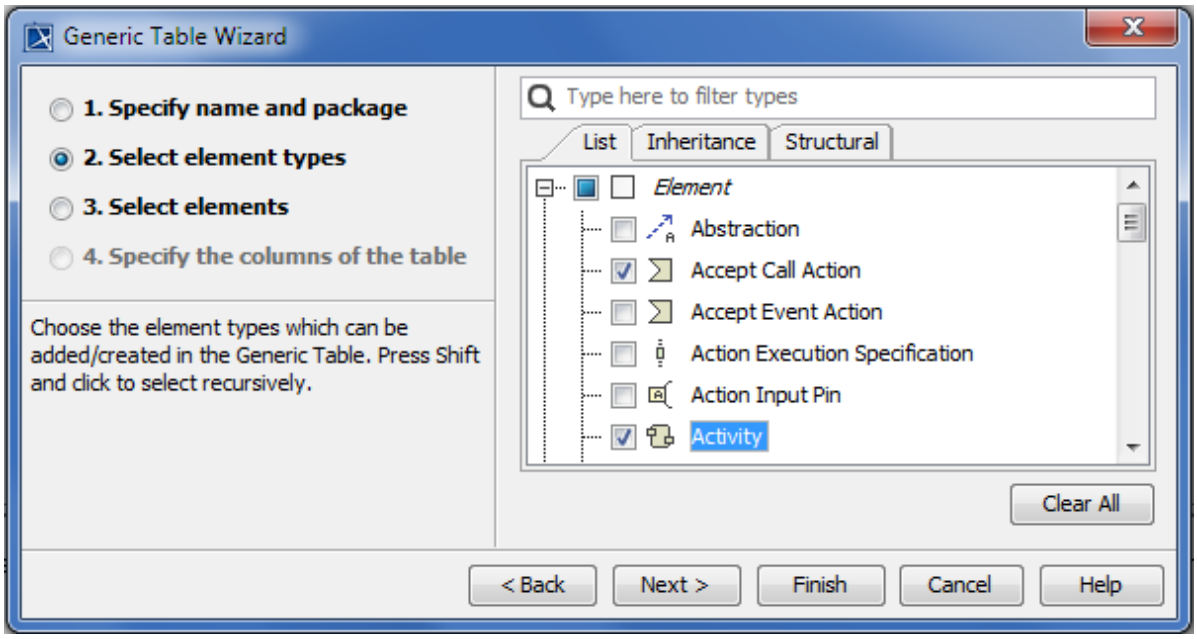
To create a Generic Table using the diagram creation wizard

1. Open the **Generic Table Wizard** dialog in one of the following ways:
   - On the **Diagrams** menu, click **Diagram Wizards** > **Generic Table Wizard**.
   - On the **Analyze** menu, click **Model Visualizer**.
2. Type the name of the table and select the owner for it.

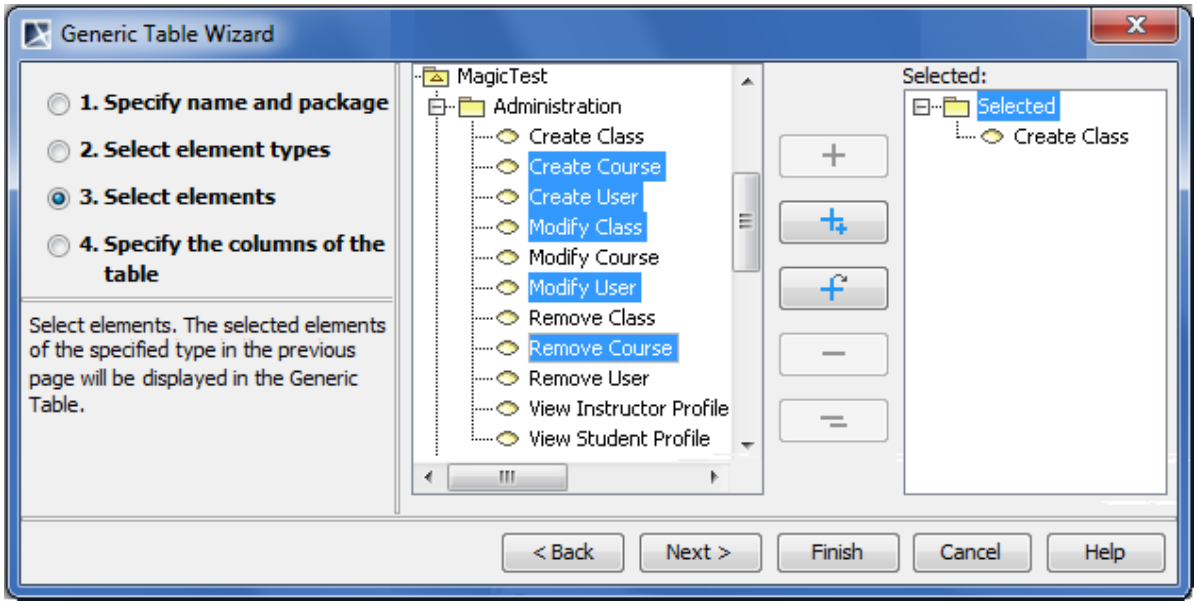


Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

3. Select element types (see details in the preceding procedure(see page 444)).



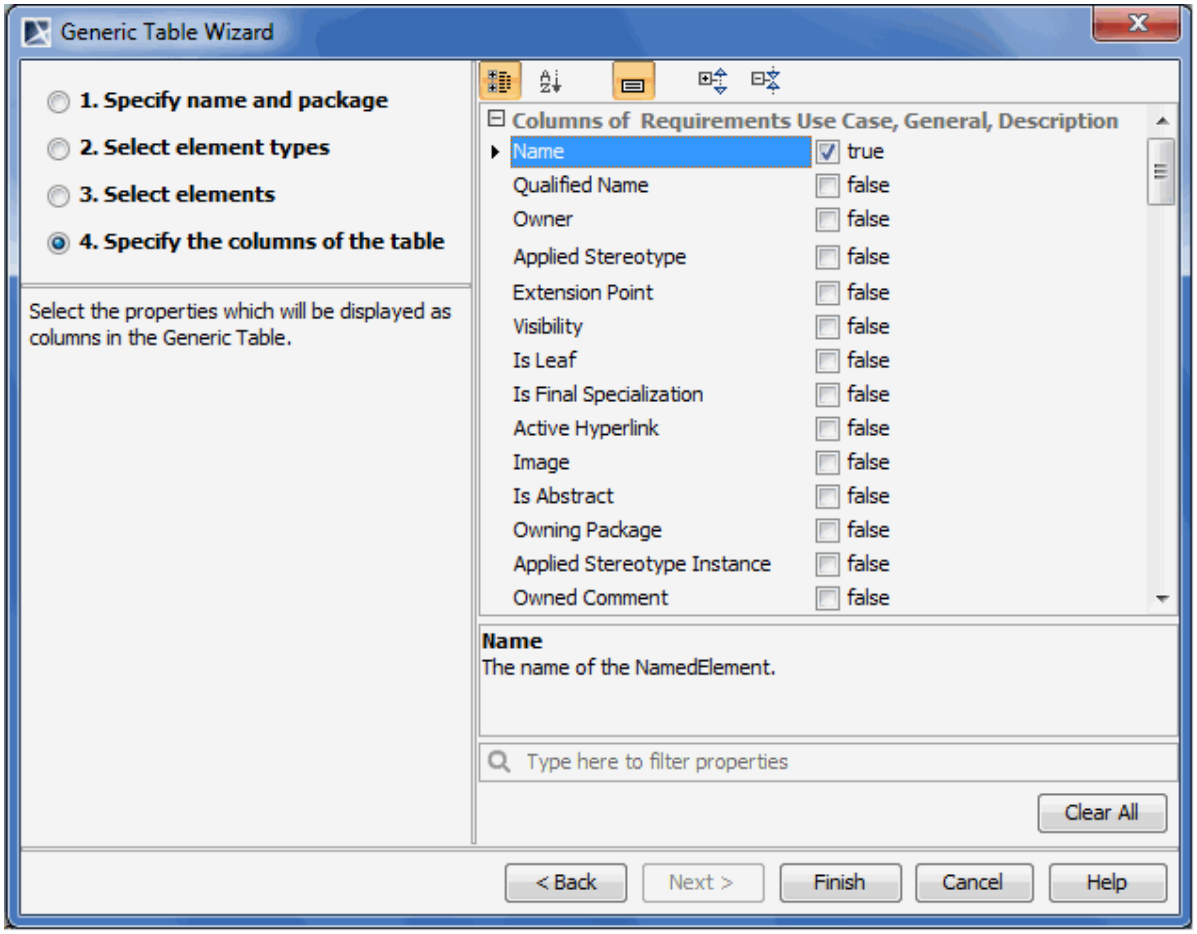
4. Select elements.



Select elements from the model tree and add them into the **Selected** list.

> ⚠ Though you can select any element from the Model Browser(see page 29), only the elements of selected types will be added into the Generic Table.
> If the Generic Table creation wizard is opened from one or more elements' shortcut menu, the element or the set of elements will be added to the **Selected** list automatically.
> For more information about the element selection and dialog buttons, see Elements multiple selection(see page 1032).

5. Specify the set of columns to show on the table.



The list of all available properties corresponding to selected element types is presented in the dialog. If selected element types have tags, they are also displayed in the list. All selected properties will be displayed as columns in the Generic Table. The **Name** property is selected automatically; all other properties should be selected by the user.
Select properties you need to see as columns to finish the table.

Creating a Generic Table for a set of selected elements using the diagram creation wizard

Note that element types, elements, and columns of the Generic Table will be automatically selected according to the set of selected elements.

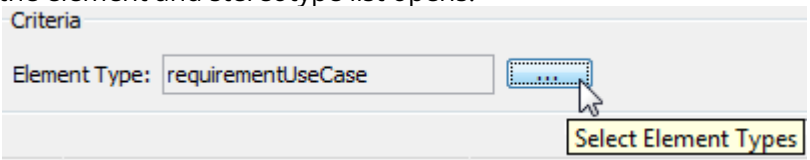
To create a new Generic Table for a set of selected elements

1. Select a set of elements you want to add to the Generic Table.
2. From the shortcut menu select **Tools** > **Generic Table Wizard**.
3. Follow the steps described in Creating a Generic Table with help of the diagram creation wizard(see page 445) but note that element types, elements, and columns will be automatically selected according to the set of selected elements.

Adding or removing element types

To add or remove element types

1. In the **Criteria** area, click the ... button next to the **Element Type** box. The dialog with the element and stereotype list opens.



2. Do one of the following:

- To add element types select the appropriate check boxes in the list.

  Properties corresponding to the selected types will be added to the list of available columns.

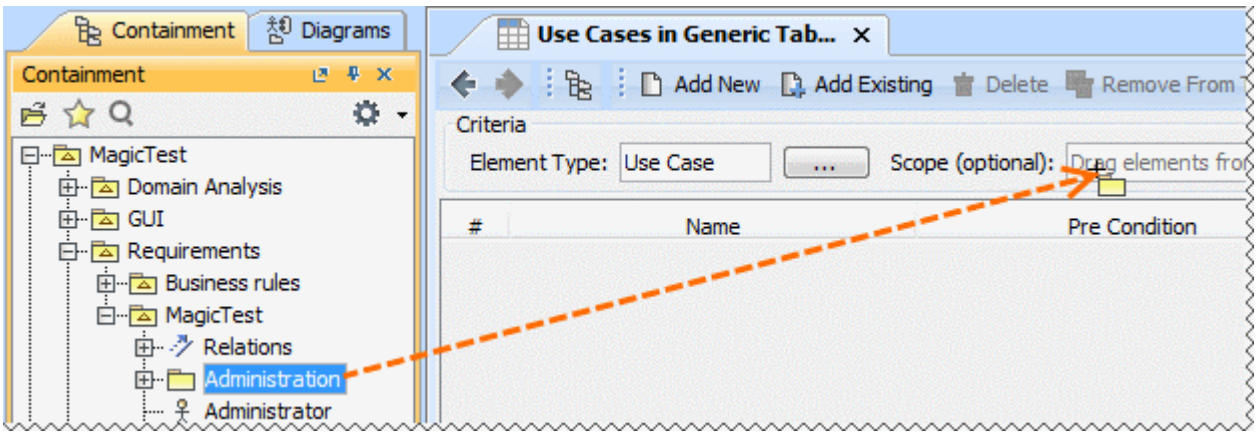
- To remove element types click to clear appropriate check boxes in the list.

  Properties corresponding to the deselected types will be removed from the list of available columns.

To add an element type, you can also drag a selected element to the **Element Type** box directly from the Model Browser(see page 29).

- To select *adjacent* elements, select a single element and hold down the Shift key while you click other elements.
- To select *nonadjacent* elements, select a single element and hold down the Ctrl key while you click other elements.

The new value of the **Element Type** box replaces the old value.



Filling the table with elements

You can fill the Generic Table for the specified Element Type in the following ways:

- Creating new elements(see page 449)
- Adding existing elements from the model(see page 449)
- Copying and pasting elements from another resources(see page 451)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Creating new elements

At least one element type must be selected in the **Element Type** box before adding a new element.

To create a new element

1. Click the **Add New** button on the Edit toolbar. If there is more than one element type selected, a submenu with the list of available element types opens.

   Only element types available to create in a possible diagram owner which contains the Generic Table will be displayed on the submenu.

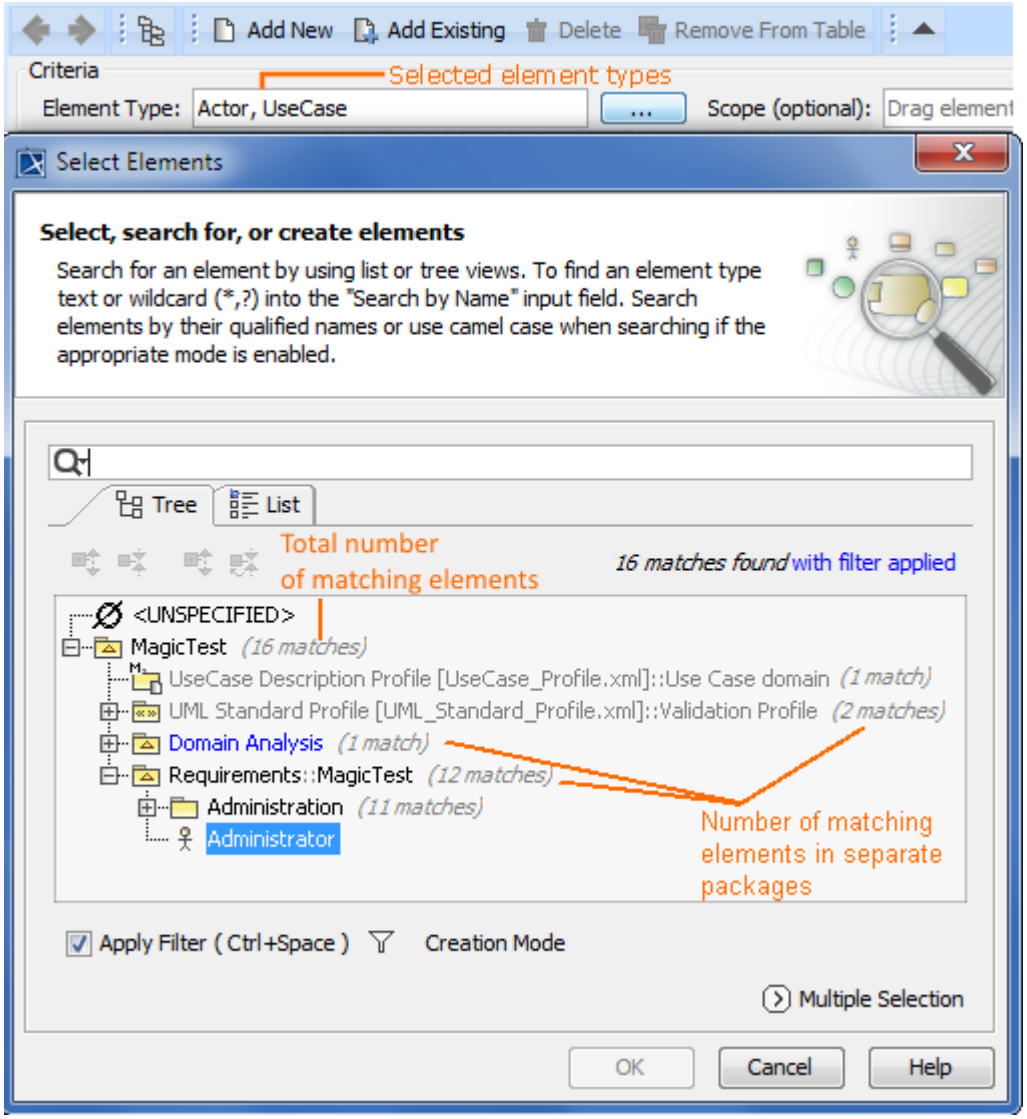
2. Select an element type. The element of the selected type will be added to the last row of the table and to the model.
3. Name the new element in the table.

Adding existing elements from the model

Elements you want to add should correspond to selected element types.

To add an existing element from the model

1. Click the **Add Existing** button on the Edit toolbar. The **Select Element** dialog(see page 1031) opens.



2. Select the element you want to add to the Generic Table. This element and its name will be added to the table as the last row.

> ⓘ Use the **Multiple Selection**mode to add more than one element at a time.

> ⚠ If you have not defined the element type before starting to add elements to the table, it is automatically set after the first element is added to the table. The type of this element becomes the value of the **Element Type** box.

> ⓘ To add an existing element, you can drag a selected element to the contents of the Generic Table directly from the Model Browser(see page 29).
> If you drag a container, for example, a package or a smart package, the Generic Table will display its contents, but not the container itself.

You can copy the data from other sources such as HTML, Excel, Word, and paste it to the Generic Table. The main rules for successful copy-and-paste operations in Generic Tables are described in the Copying and pasting data(see page 435) page. The copy-and-paste operations create new elements or update already existing elements in the Generic Table.

To create elements in the table by pasting data from another resource

1. Copy a table from another resource.

   The number of columns in that table should be the same as the number of columns in the Generic Table.
   The data types of columns in the Generic Table must be compatible with copied information.

2. Open the Generic Table.
3. In the Criteria area, click ... to specify the Element Type.
4. In the table toolbar, click the **Show Columns** button and choose the columns that you want to fill with the copied information.

   - The cells must be editable in the Generic Table.

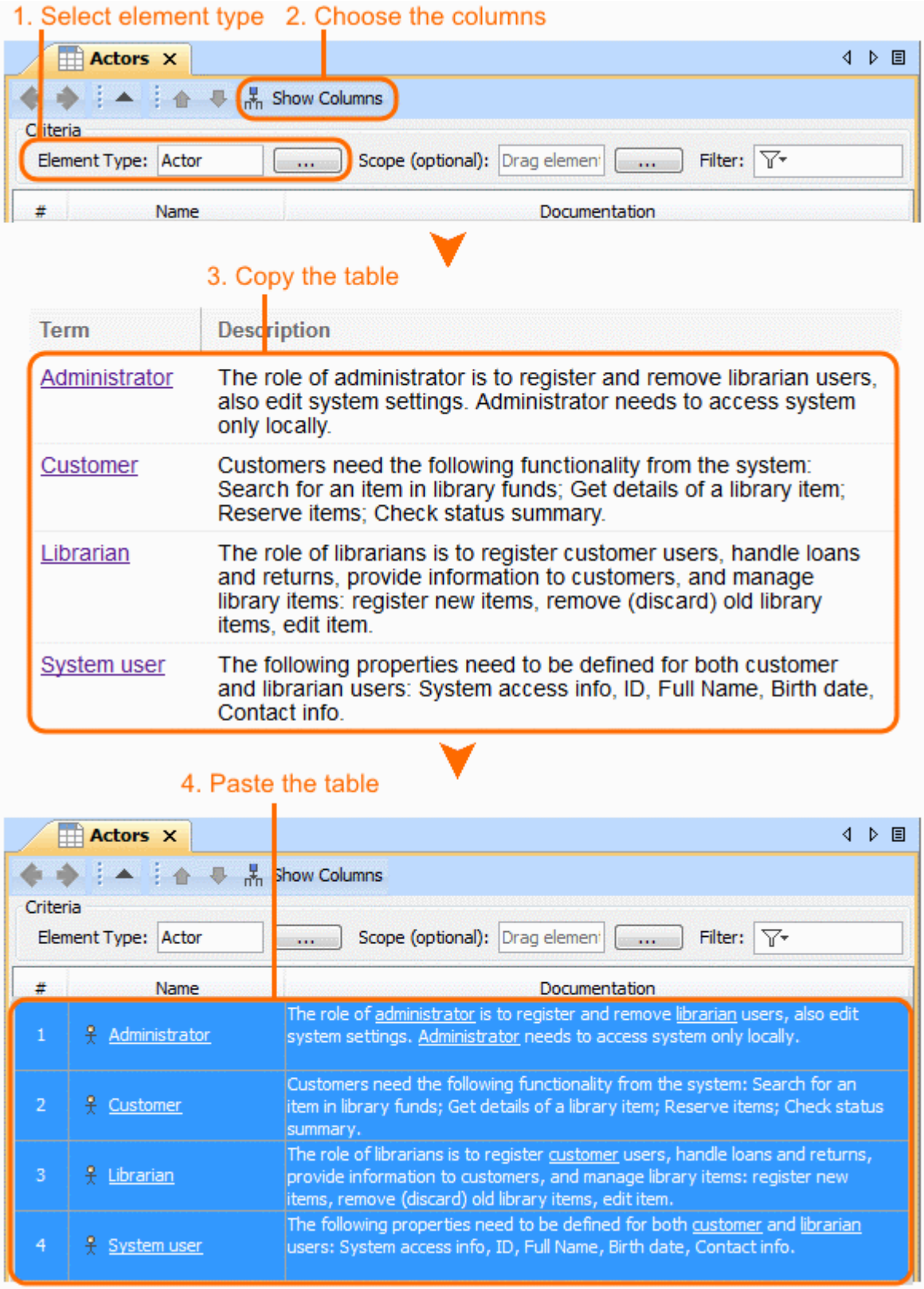
5. Do one of the following:
   - Press Ctrl+V (Cmd+V on OS X).
   - In the main toolbar, select **Edit** > **Paste.**

   Each row displays one element with its properties.

   - If the element already exists in the model, the information updates.
   - If elements have an ID prefixes in another resource, they are automatically set only for pasted elements in the Generic Table after the copy-and-paste command. Manage ID prefixes in the **Element Numbering** dialog(see page 1046).

   The following example demonstrates the main steps for pasting data in a Generic Table.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Glossary table

On this page

- Description(see page 453)
- Purpose(see page 453)
- Usage(see page 453)
- Example(see page 453)
- Specific tasks in the Glossary Table(see page 454)

Description

The Glossary saves time by ensuring consistent usage of terminology in the organization. It improves communication between team members, as terms are understood to have the same definitions and usages everywhere they appear.

Each row in the table represents a term. A word, phrase, or any element of the model can be a term. Words and phrases defined as terms are underlined. This enables you to identify terms in your model.

> The word or phrase is underlined only if it completely matches the defined term. For example, if you defined the word *Project* in your glossary, the word *Projects* would not be underlined because it is plural.

Purpose

The Glossary Table allows you to:

- Create a list of all the specific terms used in the project.
- Define your terms in one place.
- Use them everywhere in your project.
- Export your glossary to a new project and use it in other projects.
- Export your glossary into .html, .csv, or .xlsx files.
- Have several glossaries in one project.
- Customize the representation of the glossary in the table.

Usage

The video below demonstrates how to use the glossary in MagicDraw when working with Requirements.

Sorry, the widget is not supported in this export.
But you can reach it using the following URL:

https://www.youtube.com/watch?v=l6jGDDUTVXU

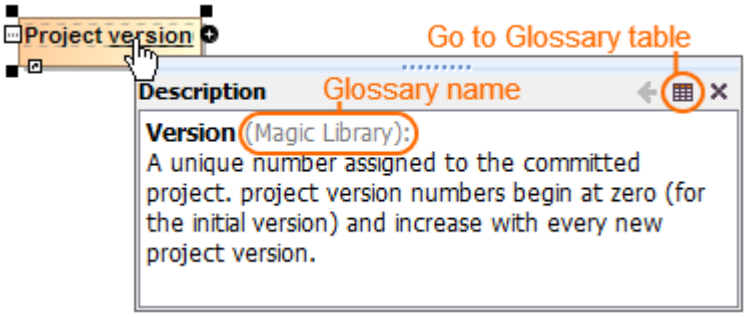
Example

The following figure illustrated the terms defined in The Glossary Table.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

#	Term	Description
1	Author	A user who has committed a new project version.
2	Comment	Optional description of changes in the committed version.
3	Project	A physical working unit that consists of model, model visualizations (diagrams, tables, matrices, etc.), configuration data, and references to other elements residing in modules.
4	Project category	Project category concept which enables visual grouping of projects in Teamwork Server repository.
5	Repository	A storage place for projects and their versions managed by the Teamwork Server.
6	Version	A unique number assigned to the committed project. Project version numbers begin at zero (for the initial version) and increase with every new project version.

The figure below shows the term visibility on the diagram pane.



Specific tasks in the Glossary Table

The following outlines the specific procedures when working with Glossary Tables:

- Adding terms in the Glossary Table(see page 454)
- Using terms in your model(see page 457)

Related pages

- Legends(see page 530)

Adding terms in the Glossary Table

You can add terms in the Glossary Table in the following ways:

- Directly from the diagram pane(see page 455)
- Using the Generic Table toolbar(see page 455)
- By dragging any element(see page 455)
- By pasting from another resource(see page 455)
- Adding synonyms of the terms(see page 456)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

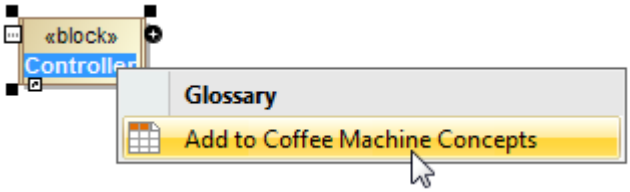
Directly from the diagram pane

To add a word or phrase to a glossary directly from the diagram pane

1. Select a word or phrase you want to be a term.

   > Even if you mark only a part of a word, the wholewordwill be added to the glossary as a term.

2. Right-click it and select **Add To <Glossary Table name>.**
   The word or phrase is converted to a term and added to the Glossary Table.



Using the Generic Table toolbar

To add a term in the Glossary Table using a toolbar

- Click the **Add New** button to create a new row for a term in the table.
- Click the **Add Existing** button and from the **Select Element** dialog. Choose the element you wish to be a term.

By dragging any element

To drag an element to the Glossary Table

1. In the Containment tree, select an element you wish to be a term.
2. Drag it to the Glossary Table.

By pasting from another resource

To create terms in the table by pasting data from another resource

1. Copy a table from another resource.

   > - The number of columns in that table should be the same as the number of columns in the Glossary Table.
   > - The data types of columns in the Glossary table must be compatible with copied information.

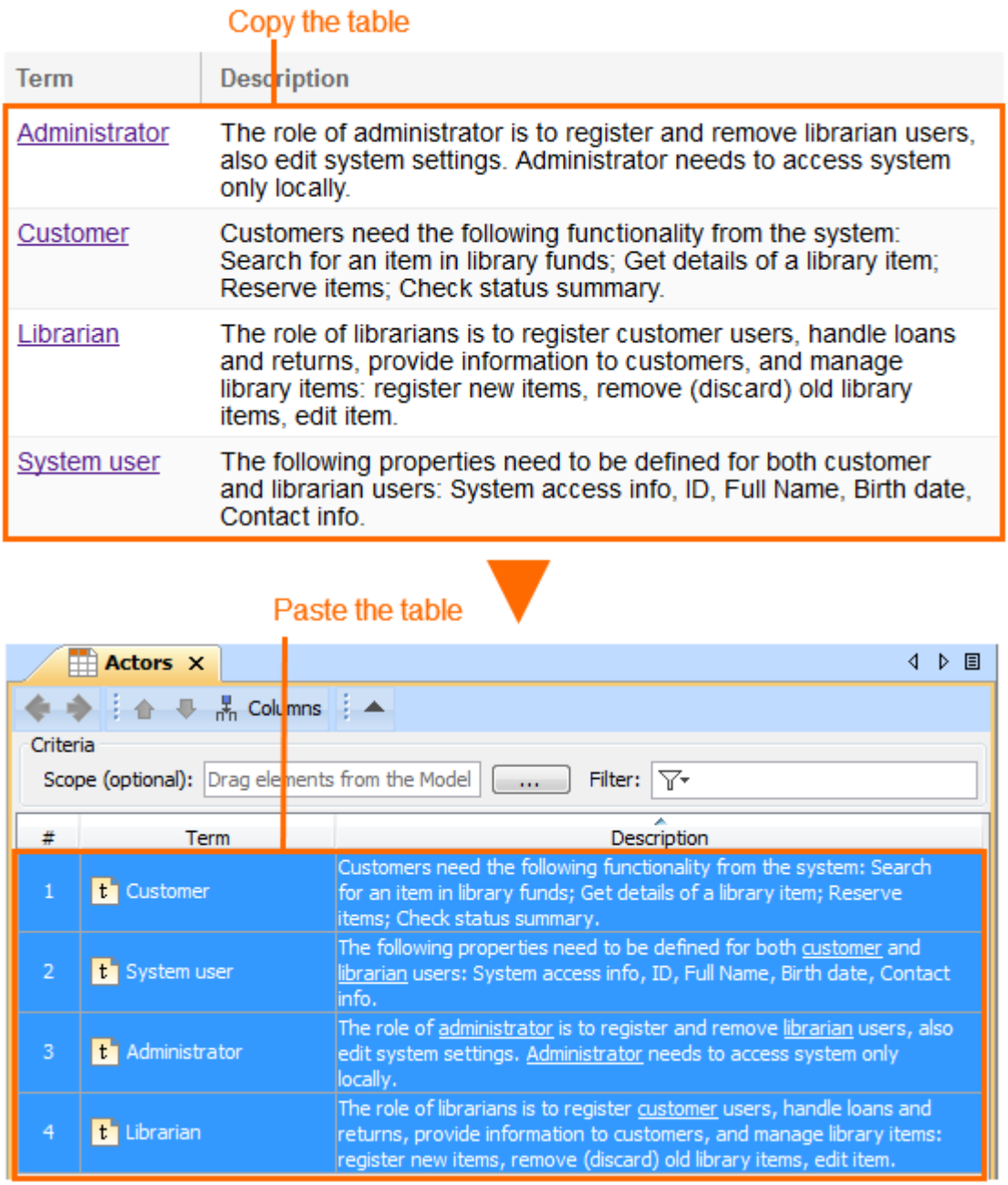
2. Open the Glossary Table where you want to paste the copied data.

   > - The cells must be editable in the Glossary Table.

3. Press Ctrl+V (Cmd+V on OS X).
   Each row displays one term with its description.

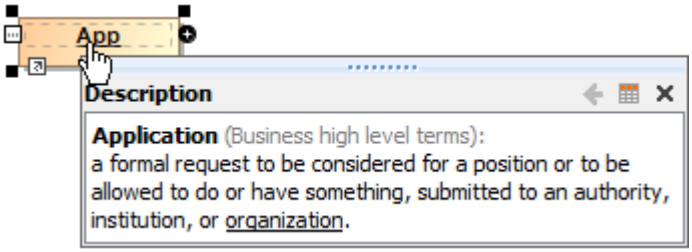
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- If the term already exists in the model, the information updates.
- If terms have ID prefixes in another resource, they are automatically set after the copy-and-paste command only for pasted terms in the Glossary Table. Manage ID prefixes in the **Element Numbering** dialog(see page 1046).



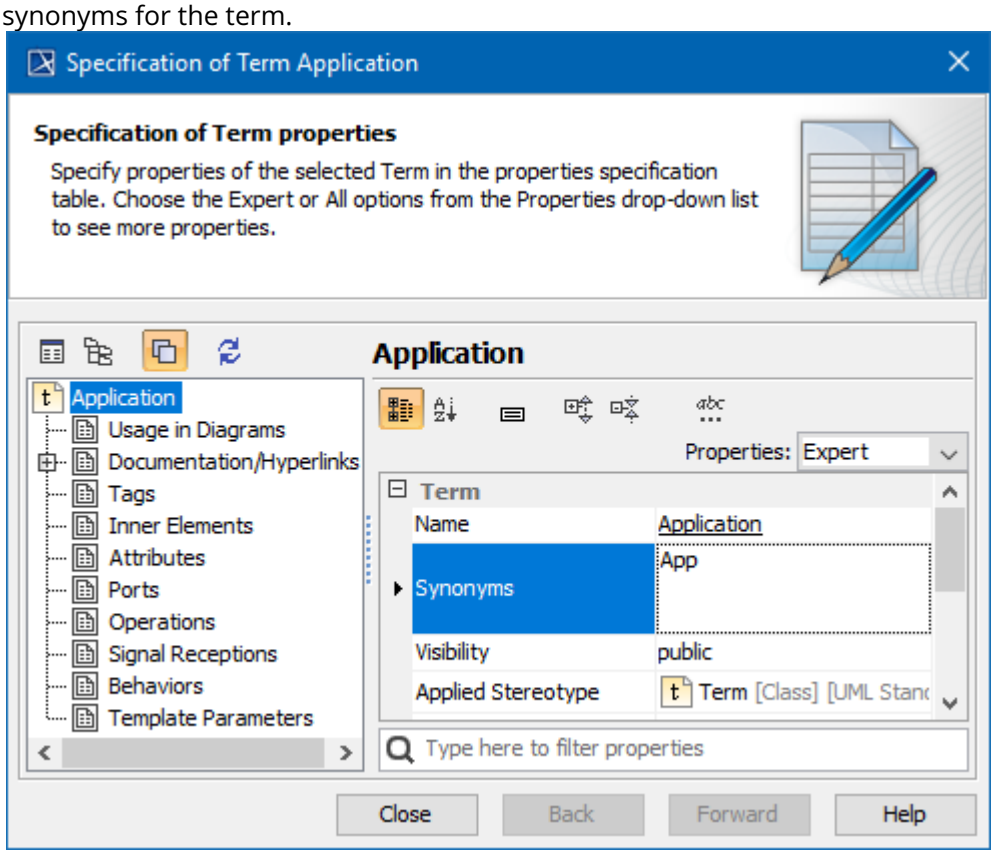
Adding synonyms of the terms

You can add one or more synonyms for every term. After that, the synonyms are available in the suggestion list. When you use a synonym, the original term is displayed in the description.



To add a synonym

- In the term's **Specification** window(see page 913), Synonym property specification cell, type the synonyms for the term.



Using terms in your model

After terms are defined in the glossary, it is easy to use them in your project.

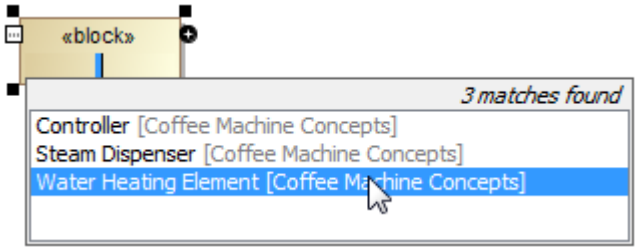
- Inserting terms(see page 458)
- Using acronyms(see page 458)
- Navigation through terms(see page 459)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Inserting terms

To use term in your model

1. Press Ctrl + Backspace while typing.
2. Select the term from the list.
   The term is inserted.



Using acronyms

You can define capitalized acronyms, such as ABS or MBSE, in the glossary (see the figure below).

> **Information**
> Only capitalized terms are recognized as acronyms. Lowercase words are not interpreted as acronyms.

Glossary table

#	Term	Description
1	Alternative Fuels	May be alcohol-based, such as ethanol or methanol; compressed natural gal; or combinations of gasoline and alcohol.
2	BE	Best Effort.
3	OR	O-ring.
4	SUV	Sport Utility Vehicle.
5	HSUV	The Hybrid Sport Utility Vehicle.

Acronyms

Requirement table

#	Name	Text
1	1 FuelEconomy	The Hybrid HSUV shall have dramatically better fuel economy than a typical SUV. Consumption can be expressed in terms of volume of fuel to travel a distance, or the distance travelled per unit volume of fuel consumed.

The option to recognize acronyms everywhere in the text is enabled by default.

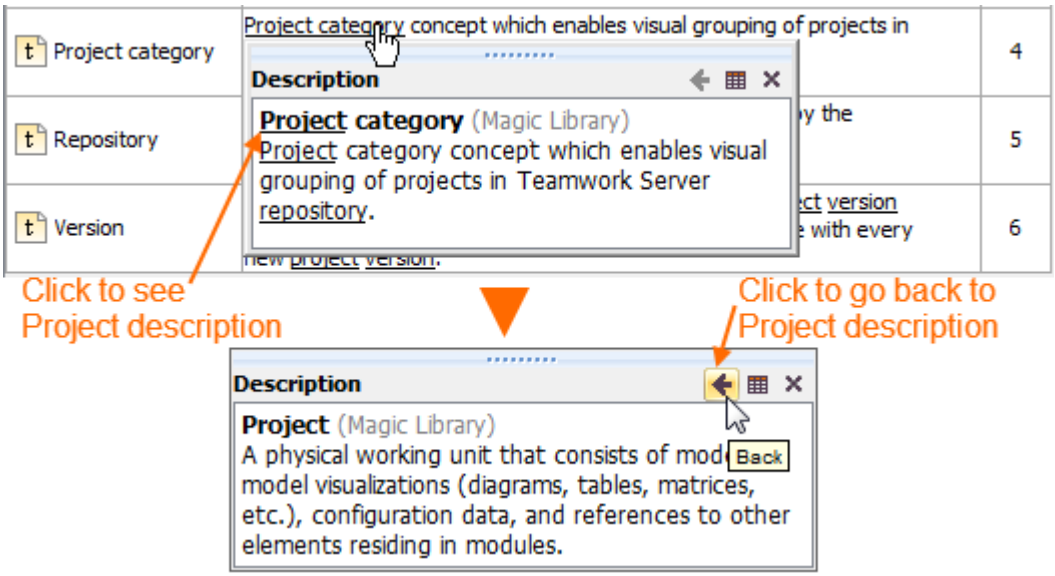
To disable acronyms usage

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

1. From the Glossary Table toolbar, select .
2. Click to clear the **Use Acronyms** check box.

Navigation through terms

When a term is a whole phrase and one or more words are also defined as terms, you can easily navigate through all those terms in the description window. We recommend creating a new package where the Glossary Table and all terms will be stored. In the new package, create a Glossary Table.



Instance table

On this page

- Description(see page 459)
- Purpose(see page 459)
- Example(see page 460)
- Specific tasks in the Instance Table(see page 460)

Description

This feature is available in Standard, Professional, Architect, and Enterprise editions.

The Instance Table is one of four MagicDraw tables which allow you to manage Instance Specifications of your model in a spreadsheet-like form. Each row in the table represents an instance.

Purpose

An Instance Table allows you to:

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Create Instance Specifications (instances) for the classifiers.
- Review the instances of one or more of the classifiers in a single place.
- Edit Slot values of the instances displayed in the table.
- Customize the representation of the table.
- Export the data into *.html*, *.csv,* or **.xlsx* files.

Example

#	Name	ma (kg)	me (kg)	mr (kg)	margin (kg)
1	spacecraft	130.0	95.0	15.0	35.0
2	telecom	35.0	27.0	5.0	8.0
3	amplifier	10.0	8.0	10.0	2.0
4	antenna	20.0	19.0	20.0	1.0
5	propulsion	80.0	68.0	7.0	12.0
6	tank	44.0	38.0	44.0	6.0
7	thruster	29.0	30.0	29.0	-1.0

Specific tasks in the Instance Table

The following outlines the specific procedures when working with the Instance Table:

- Creating an empty Instance Table(see page 460)
- Specifying the classifier(see page 461)
- Selecting columns to display(see page 462)
- Filling the table with instances(see page 465)
- Specifying instances as slot values(see page 467)

Related pages

- Legends(see page 530)

Creating an empty Instance Table

To create an empty Instance Table

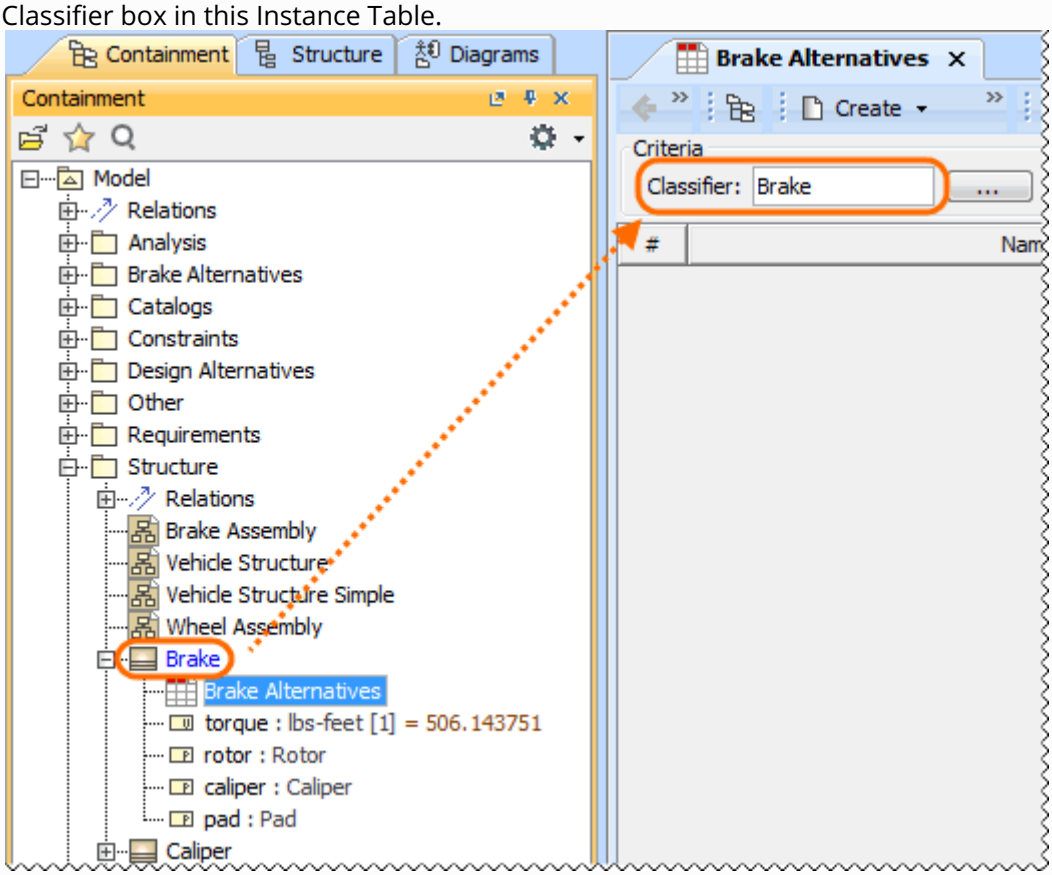
---

1. In the Containment tree, select an element to be the owner of the Instance Table.
2. Do one of the following:
   - From the main menu, select **Diagrams** > **Create Diagram**. Type “ins” and press Enter.
   - On the main toolbars, click the **Create Diagram** button. Type “ins” and press Enter.
   - Press Ctrl+N. Type “ins” and press Enter.
   - Right-click the element and from the shortcut menu select **Create Diagram** > **Other Diagrams** > **Instance Table**.

   The newly created Instance Table opens on the right side of the application window.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- If the default name of the Instance Table is not appropriate, type a new one directly in the Containment tree.
- If the owner of the Instance Table is also a classifier, it becomes the default value of the Classifier box in this Instance Table.



Specifying the classifier

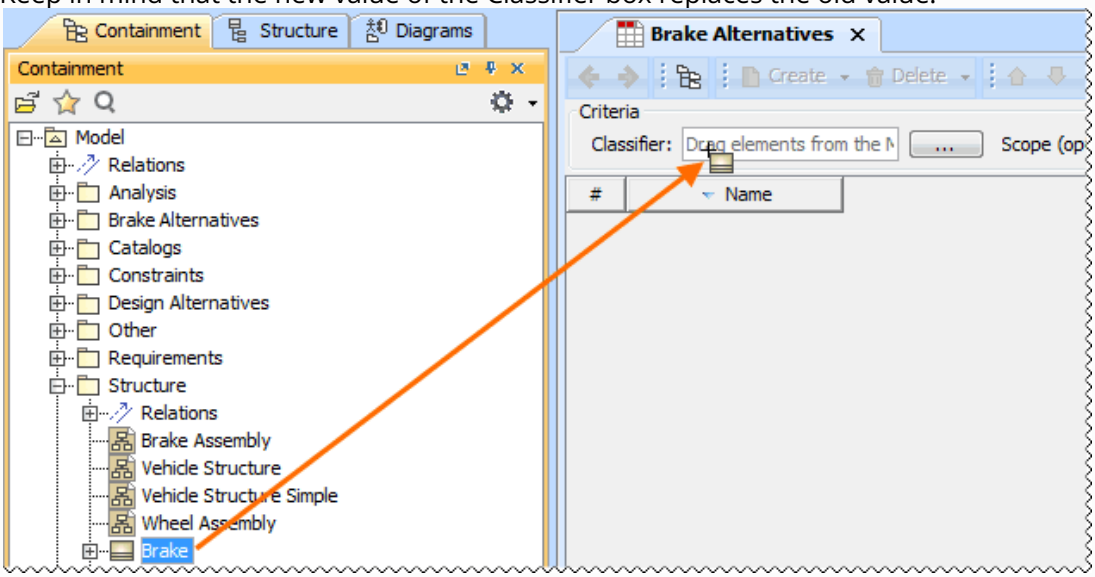
Specifying the classifier criteria is necessary. You are not able to add a single instance to the table until the classifier is known.

To specify the classifier for the Instance Table

1. In the Criteria area of the Instance Table, click the **...** button next to the Classifier box. The **Select Classifier** dialog opens.
2. Select one or more elements and add them to the Selected elements list. These are the elements that can be selected:
   - Classifier.
   - Classifier that owns other classifiers. In this case, the classifier and all the classifiers owned by it are specified as criteria.
   - Package that contains classifiers. In this case, the contents of the package, but not the package itself, are specified as criteria.
3. Click **OK** when you are done.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- To specify the classifier, you can also drag a selected element to the Classifier box directly from the Model Browser.
- To select adjacent elements, select a single element and then hold down the Shift key while you click other elements.
- To select nonadjacent elements, select a single element and then hold down the Ctrl key while you click other elements.
- Keep in mind that the new value of the Classifier box replaces the old value.



Once the classifier is specified, the following columns are displayed in the Instance Table (on the right of the row number # column):

- Name which is up to display the names of the instances of the specified classifier.
- Columns corresponding to the Data type attributes of the specified classifier and named after these attributes using sentence style capitalization. The columns are up to display the slot values of the instances of the specified classifier.

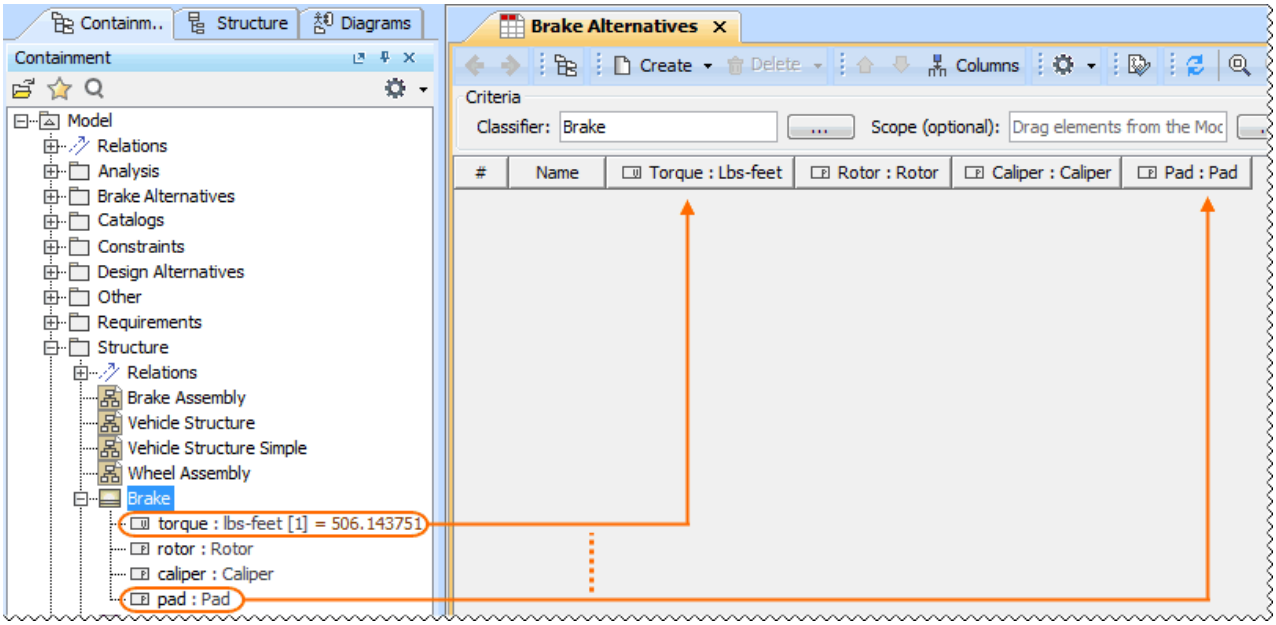
Selecting columns to display

By default, the Instance Table shows nine columns (including # (row number), Name (instance name), and the columns for instance slot values) that correspond to the first seven Data type attributes of the specified classifier. The rest of the columns are by default hidden, though you can show them if there is a need.

You can show any column that corresponds to one of the following:

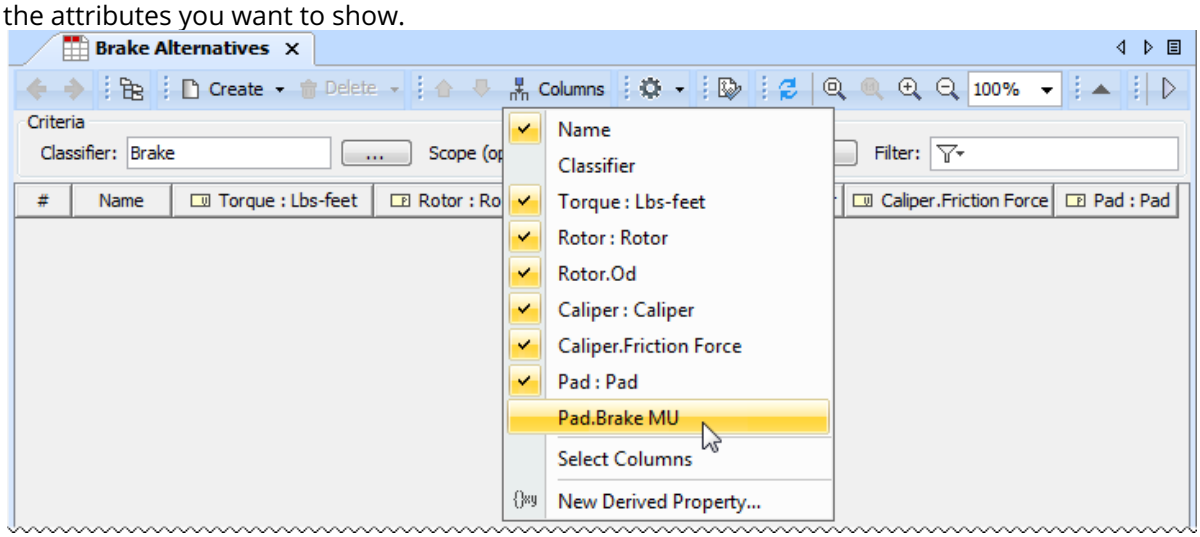
- Attribute of the classifier that is specified for the Instance Table(see page 463).
- Property of the instance of the specified classifier.(see page 463)
- Tag of a stereotype that can be applied on the instance of the specified classifier.(see page 463)
- Nested properties of the classifier that is specified for the Instance Table.(see page 464)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



To display attributes of the classifier

1. On the table toolbar, click the **Columns** button and then click to select the check boxes beside the attributes you want to show.



To display properties and tags

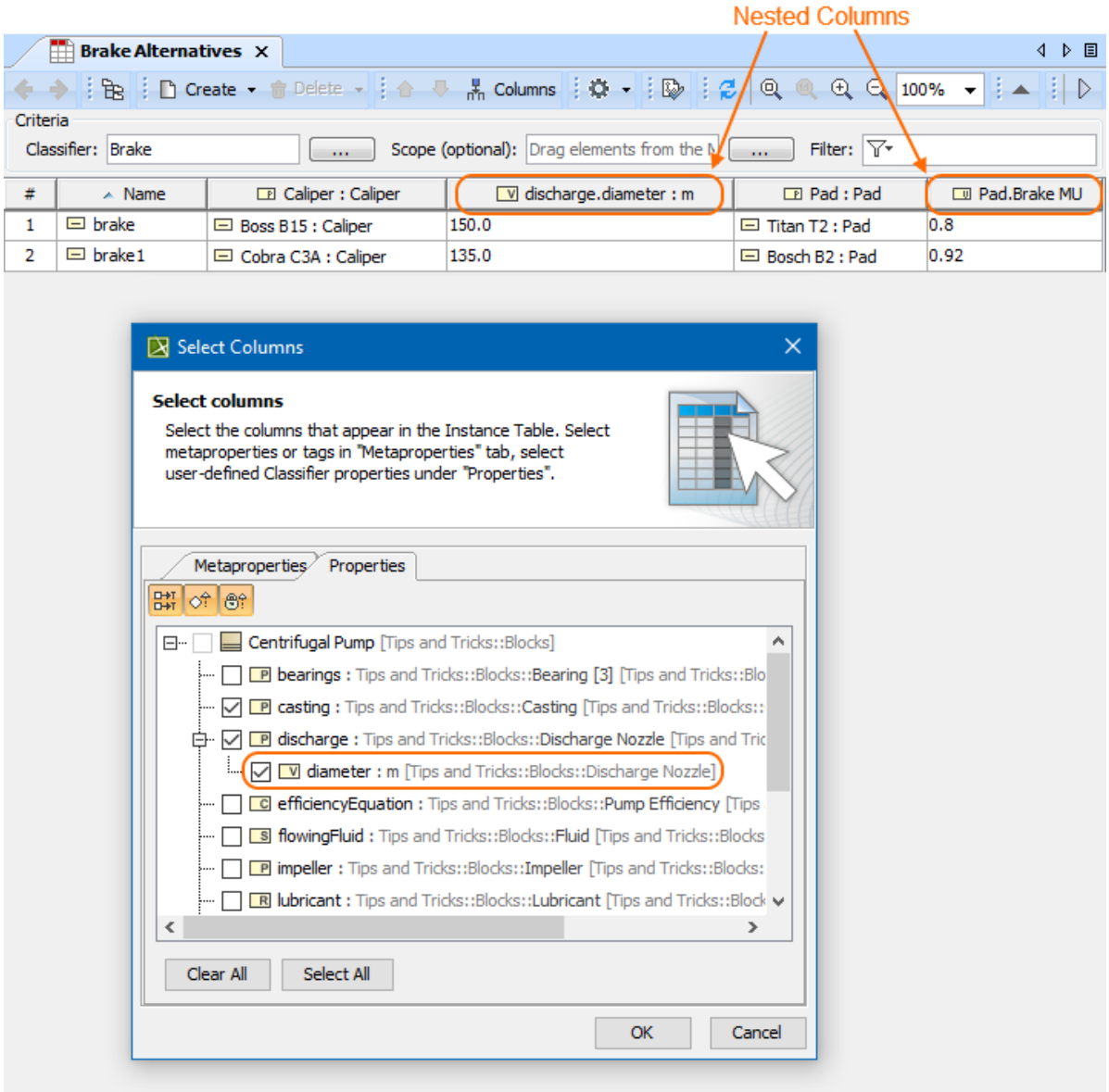
1. On the table toolbar, click **Columns** > **Select Columns** > **Metaproperties** tab. In the **Select Columns** dialog, choose the columns you want to display in the Instance Table by selecting the appropriate check boxes of the properties and tags.

> ⓘ All columns that you have chosen to display also appears on the menu of the **Columns** button.

To display nested properties of the classifier

1. On the table toolbar, click **Columns** > **Select Columns** > **Properties** tab. In the **Select Columns** dialog, choose the columns you want to display in the Instance Table by selecting the appropriate check boxes of the nested properties.

> To specify nested columns, click the **Create with Parts** button, when creating a new instance with parts(see page 0).



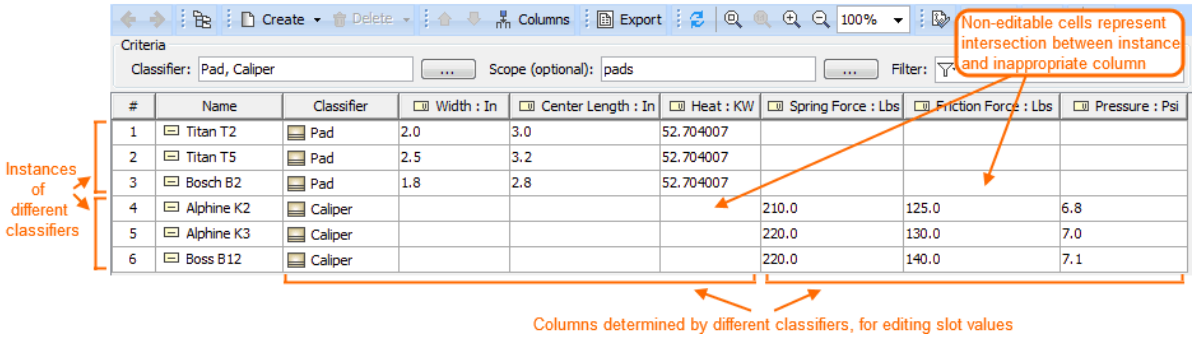
> If you add a new attribute to the classifier and need to display the corresponding column in the Instance Table, open the **Select Columns** dialog and click to select the appropriate check box. Keep in mind that columns corresponding to newly created attributes at first are available only in the **Select Columns** dialog (until you show them).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Filling the table with instances

The following outlines how to fill the Instance Table for the specified classifier:

- Create new instances.(see page 465)
- Create new instances with parts.(see page 465)
- Add already existing instances.(see page 465)
- Copy and paste instances from another resources (HTML, Excel, Word, etc.).(see page 466)



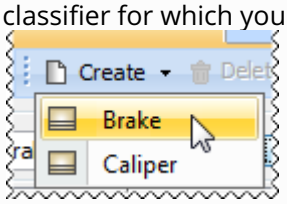
#	Name	Classifier	Width : In	Center Length : In	Heat : KW	Spring Force : Lbs	Friction Force : Lbs	Pressure : Psi
1	Titan T2	Pad	2.0	3.0	52.704007			
2	Titan T5	Pad	2.5	3.2	52.704007			
3	Bosch B2	Pad	1.8	2.8	52.704007			
4	Alphine K2	Caliper				210.0	125.0	6.8
5	Alphine K3	Caliper				220.0	130.0	7.0
6	Boss B12	Caliper				220.0	140.0	7.1

To create a new instance

1. On the table toolbar, click the **Create** button.

   The new instance can be created only if the classifier is already specified for the Instance Table. Otherwisethe **Create** button is not available.

   If there is more than the one classifier specified, from menuof the button, select a classifier for which you need to create an instance.

   The new instance is added to the Instance Table as the last row.
2. Type the instance name and specify slot values.

To create a new instance with parts

1. On the table toolbar, click **Create** > **Create with Parts**.

   The new instance with nested parts is created if:
   - the classifier is specified for the Instance Table. Otherwisethe **Create with Parts** button is not available.
   - the nested properties are displayed as columns(see page 0) in the Instance Table.

To add an existing instance

1. On the table toolbar, click **Create** > **Add Existing**. The **Select Instance Specification** dialog opens.
2. In the tree or list on the left side of the dialog, select an element and add it to the Selected elements list. You can select more than one element. Elements available for selection:
   - Instances of one or more classifiers specified for the table. An instance is available if it belongs to at least one of the classifiers.
   - Packages that contain these instances. In this case, the contents of a selected package, but not the package itself, will be added to the table.
3. Click **OK** when you are done.
   The selected instance is added to the table as the last row.

   - You can drag a selected element to the contents of the Instance Table directly from the Model Browser.
   - To select adjacent elements, select a single element and then hold down the Shift key while you click other elements.
   - To select nonadjacent elements, select a single element and then hold down the Ctrl key while you click other elements.

To copy and paste an instances from another resources

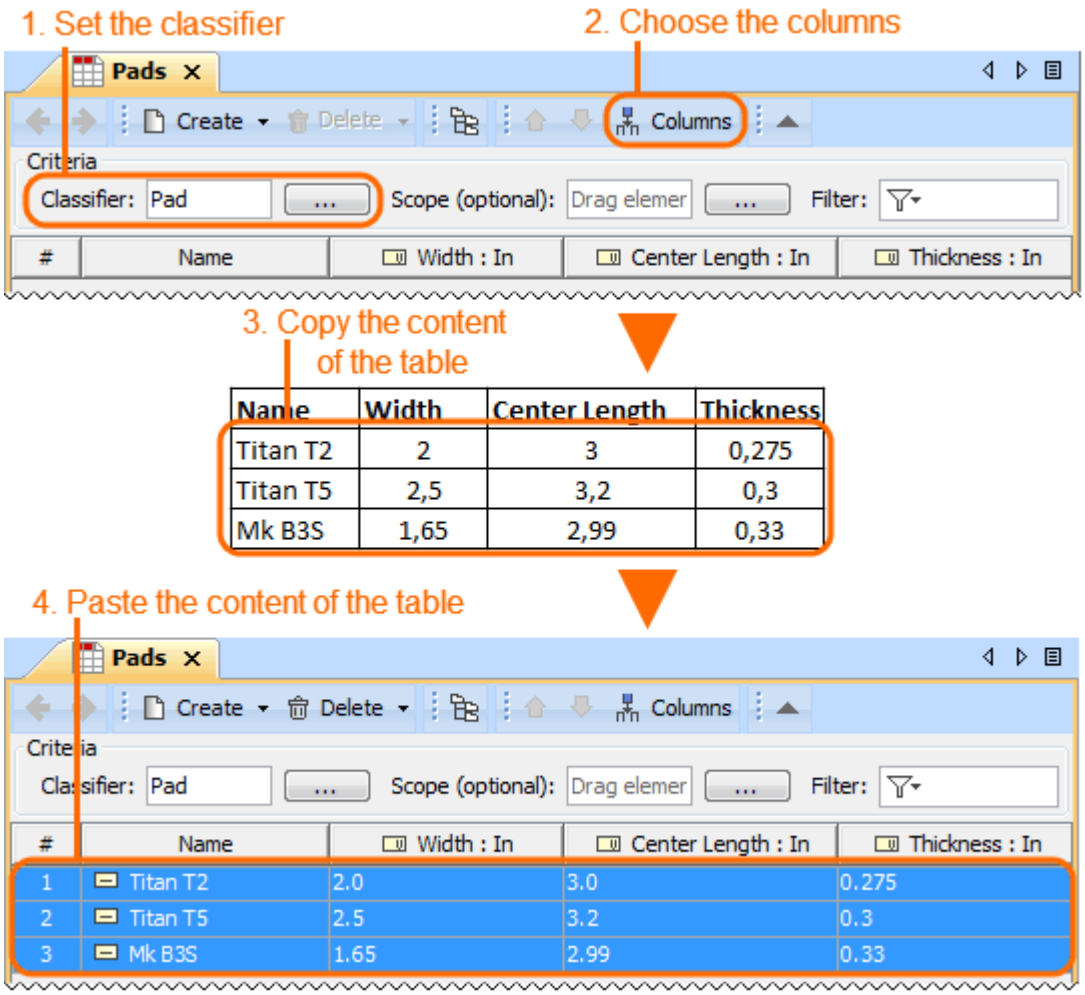
1. Copy the content of the table from another resource.

   - The number of columns in that table should be the same as the number of columns in the Instance Table.
   - The data types of columns in the Instance Table must be compatible with copied information.
2. Open the Instance Table where you want to paste the copied data.
3. Make sure the Classifier is specified in the Criteria area.
4. In the table toolbar, click the **Columns** button and choose columns of instance slot values that you want to fill with copied information.

   - The cells must be editable in the Instance Table.
5. Press Ctrl+V (Cmd+V on OS X).
   Each row displays one instance with its slot values.

   - If an instance already exists in the model, the information updates.
   - If instances have an ID prefixes in another resource, they are automatically set after the copy-and-paste command only for pasted instances in the Instance Table. Manage ID prefixes in the **Element Numbering** dialog(see page 1046).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Specifying instances as slot values

If the slot type is a classifier, you can specify an instance of this classifier as the slot value. Do this in one of the following ways:

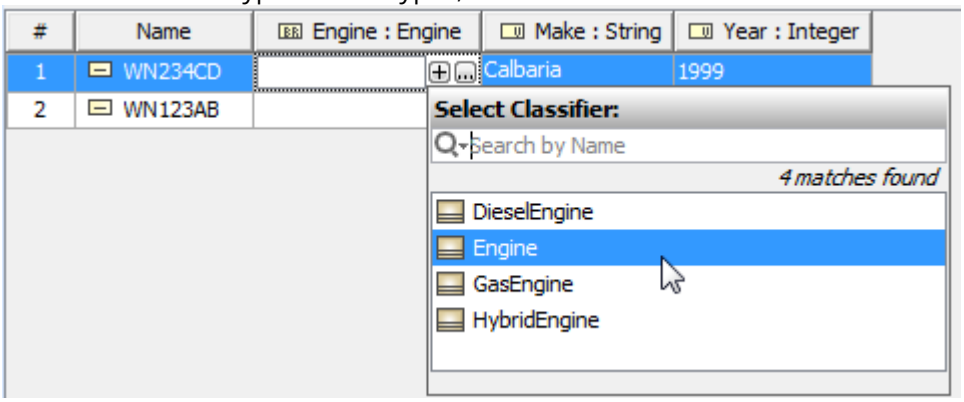
- By creating a new instance.
- By selecting an already existing instance.

To select an instance from the model, simply click ... in the slot value cell and choose the instance from the tree or list in the open dialog.

To create an instance as a slot value in the Instance Table

1. Select the slot value cell.

2. Click ⊞. If the slot type has subtypes, choose one of them.



The new instance is simultaneously created in the model and set as the slot value. You can see this instance in the Model Browser, where it is owned by the classifier, which is a type or a subtype of the selected slot.

To remove an instance from the cell

---

1. Select the cell.
2. Click ⊟.
   The instance is removed from the Instance Table, but not from the model.

Metric table

On this page

- Description(see page 468)
- Purpose(see page 469)
- Example(see page 470)
- Specific tasks in the Metric Table(see page 470)

Description

> ⚠ This feature is available in Architect and Enterprise editions only.

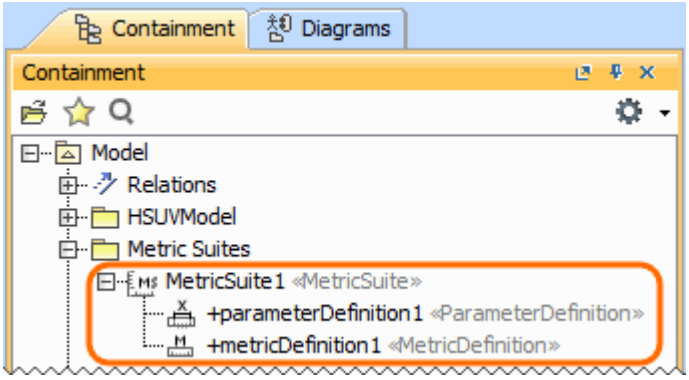
The Metric Table is one of four MagicDraw tables. The Metric Table allows you to collect various metrics used to evaluate the current state of your model. A Metric is an item of information about a specific aspect of the model. Each row in the table represents a set of metrics calculated according to the metric suites specified by the user.

To understand the material better, you need to be familiar with the following concepts.

Concept	Description
Metric suite	A collection of parameter and metric definitions used to calculate metrics that evaluate specific aspects of your model.

Concept	Description
Parameter definition	A variable used for calculating metrics. It can be an element type (including both relationship and DSL element types), a particular element, data type, or data type value.
Metric definition	A formula for calculating metrics according to the parameter values.
Validation-based metric definition	The attribute of a Metric Suite specifying the validation rule for calculating metrics according to the parameter values.

Actually, a metric suite is a class with the *«MetricSuite»* stereotype applied. Both parameter and metric definitions are attributes of the metric suite.



Purpose

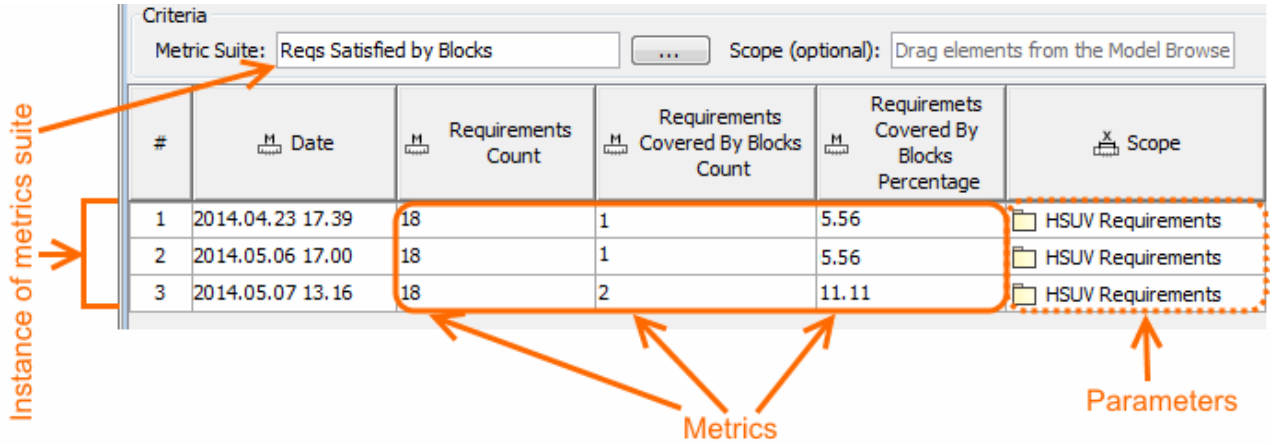
Calculated at regular intervals, metrics help track the evolution of the model development. One metric is the result of a single calculation of a metric definition. The metrics of all the metric definitions of the metric suite, together with relevant parameters, are stored in a single instance of the metric suite as slot values. Use the Metric Table to analyze and manage the instances of the metric suite.

With a help of a Metric Table you can:

- Calculate model metrics.
- Change the parameters of the metrics and recalculate them.
- Customize the representation of the metrics.
- Export the data to an *.html*, *.xlsx*, or *.csv* file.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example



Specific tasks in the Metric Table

The following links outline the specific procedures when working with the Metric Table:

- Creating a Metric Table(see page 470)
- Calculating metrics in the Metric Table(see page 471)

Related pages

- Legends(see page 530)
- Metric Suites(see page 1114)

Creating a Metric Table

Although you can create other types of tables in various ways, there is only one correct way to create a Metric Table.

To create a Metric Table

1. In the Containment tree(see page 32), right-click the element for which you need to calculate the metrics of a particular Metric Suite.

   Metric Suite target
   The element, or at least the type of the element, must be a target value of that Metric Suite(see page 1118).

2. From the shortcut menu of the element, select **Tools** > **Metrics** > **New Metric Table**. The **Create Metric Table** dialog opens.

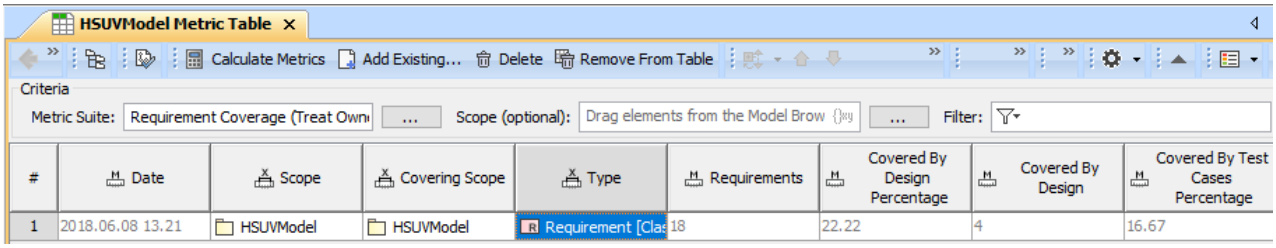
3. In the list of available metric suites, select the check boxes next to the Metric Suites you want to use. There are three groups of predefined Metric Suites:
   - Requirement Metric Suites (Obsolete)
   - Requirement Coverage (Treat Owner as Grouping Element)
   - Requirement Coverage (Treat Owner as Requirement)

   Metric Suite descriptions
   Select a Metric Suite and click to read its description in the description area of the **Create Metric Table** dialog. Carefully read descriptions of each metric suite. This helps you decide which one best suits your needs.

4. Click **OK**.

Important
If you cannot see a particular Metric Suite in the **Create Metric Table** dialog, check the target value of that Metric Suite(see page 1118). It is possible that the element selected in step 1 is not the target of this Metric Suite.

The table of a particular metric suite is created for the selected element. It is already filled with data (metrics calculated at the time you created the table). These metrics are actually stored in the Package owned by the element for which they have been calculated. The name of the Package is *<element_name > Metrics*, for example, *HSUVModel Metrics*.



If one or more metrics are not calculated, make sure the Metric Suite is specified correctly. Here are some suggestions:
- Check the metric definition – the formula could be specified incorrectly.
- Check the parameter definition – the metric definition probably cannot access parameter values.

This method of creating Metric Tables saves much time compared to the more common ways of table (and even diagram) creation, as these methods for a Metric Table creation start with an empty table. This requires further effort on your part to fill it (specify the metric suite, create an empty row, define parameters, and then calculate the metrics).

Calculating metrics in the Metric Table

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

On this page

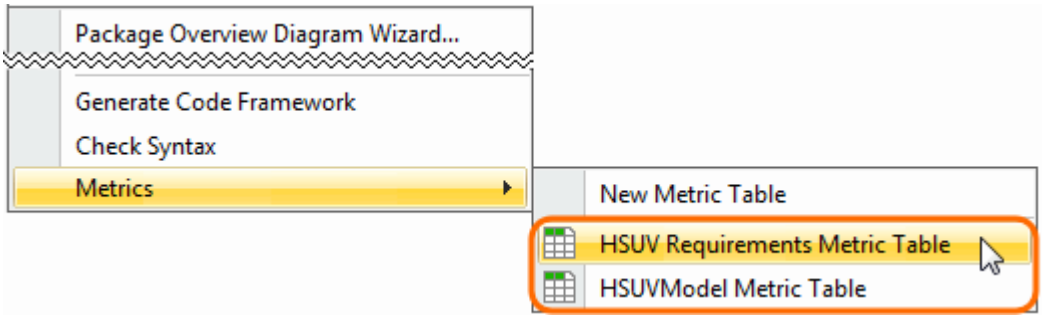
- Opening the Metric Table(see page 472)
- Calculating new metrics(see page 472)
- Recalculating metrics(see page 473)
- Calculating new metrics with different parameter values(see page 473)

Once you have created the Metric Table, which represents the metrics calculated at the time you created, you can make further calculations and track the evolution of your model development. If you need to append new metrics to your table, you can easily open it.

Opening the Metric Table

To open the Metric Table

1. In the Containment tree(see page 32), right-click the element whose Metric Table you need to append.
2. On the shortcut menu of the element, click **Tools** > **Metrics** and then select the Metric Table you need to append.



The selected Metric Table opens. Now you are ready to calculate new metrics.

Calculating new metrics

To calculate new metrics

1. In the Metric Table, select the row. Below that, you need to add a row with newly calculated metrics.
2. Do one of the following:
   - On the Edit toolbar, click Calculate Metrics and then select **Calculate New Metric** from the open menu.
   - Right-click the selected row and then click **Calculate New Metric** on the shortcut menu.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

The new metrics are calculated and represented in the table below the selected row.

> **Excluding auxiliary resources from validation-based metrics calculation**
> By default, validation-based metrics are calculated in used read-only projects (auxiliary resources) as well. To exclude auxiliary resources from metric calculation scope, click in the Metric Table toolbar and select the **Exclude Elements from Used Read-Only Projects** option.

Recalculating metrics

To recalculate metrics

1. In the Metric Table, select the row whose metrics you need to recalculate.
2. Do one of the following:
   - On the Edit toolbar, click Calculate Metrics and then select **Recalculate** from the open menu.
   - Right-click the selected row and then click **Recalculate** on the shortcut menu.

The new metrics are calculated and represented in the table, instead of the selected row.

Calculating new metrics with different parameter values

To calculate new metrics with different parameter values

1. In the Metric Table, select the row. Below that, you need to add a row for representing newly calculated metrics.
2. Do one of the following:
   - On the Edit toolbar, click Calculate Metrics and then select **Add New Metric with Different Parameters** from the open menu.
   - Right-click the selected row and then click **Add New Metric with Different Parameters** on the shortcut menu.
3. A new row with copied parameters is added to the Metric table.

   > If the new row is empty, it might be that you haven't selected any row. You may continue the procedure.

4. Change parameter values.
5. Select the newly created row and do one of the following:

- On the Edit toolbar, click Calculate Metrics and then select **Recalculate** from the open menu.
- Right-click the row and then select **Recalculate** from the shortcut menu.

The new metrics are calculated according to the specified parameter values.

Sync with Excel or CSV files

On this page

- Introduction(see page 474)
- How to sync your data(see page 474)
  - Turn on syncing(see page 475)
  - Set the sync options (optional)(see page 476)
  - Sync data(see page 476)
  - Open file (optional)(see page 478)
  - Turn off syncing (optional)(see page 479)

Introduction

You can transfer data between Excel/CSV files and modeling tool tables. After sync, the data on Excel file of selected sheet or CSV file matches the data in the modeling tool table. This mechanism is bidirectional and ensures continuous work between Excel/CSV files and modeling tool tables. If you don't have the ability to collaborate via the internet but you need to share information through Excel or CSV files, you can use this feature,. The Excel/CSV file syncing is available in all tables of the modeling tool by using the the **Excel/CSV Sync** button in the table toolbar.(see page 408)

How to sync your data

Follow these steps to manually sync data between an Excel/CSV file and a modeling tool table:

-
  - Turn on syncing(see page 475)
  - Set the sync options (optional)(see page 476)
  - Sync data(see page 476)
  - Open file (optional)(see page 478)
  - Turn off syncing (optional)(see page 479)

Turn on syncing

Before transferring data, you must turn on syncing by creating a reference between the Excel/CSV file and modeling tool table (i.e., link the Excel/CSV file with the modeling tool table).

> You can link only one Excel/CSV file with the same modeling tool table.

You can link:

- An existing Excel/CSV file.(see page 475)
- A new Excel/CSV file exported directly from the modeling tool table.(see page 475)

Linking an existing Excel/CSV file

To link an existing Excel/CSV file with a modeling tool table

---

1. Make sure the **Element Type** or **Classifier** is selected in the Criteria area(see page 415).
2. Do one of the following:

- Drag the Excel or CSV file directly onto the modeling tool table.
  Drag the Excel or CSV file directly onto the modeling tool table. Hold it down until the Tooltip appears (long drag). Drop it, and select the **Link Excel/CSV File** or **Attach and Link Excel/CSV File** command from the opened menu.
- On the table toolbar, click and select **Excel/CSV File** > **Select File**. Select **From file system** or **From model** in the **Excel/CSV Sync Options** dialog(see page 479), click , and select an appropriate Excel or CSV file.

  > **Selecting files from model**
  > If you want to link the file from your model, make sure the Excel or CSV file is attached to project. How to attach the file to a project >(see page 209)

  The Excel or CSV file is linked with the modeling tool table and ready to be synced.

> If you select or drag another file (e.g. Vehicle Requirements.xlsx) on a modeling tool table that is already linked with the file (e.g. Stakeholder Requirements.xlsx), the previous file named *Stakeholder Requirements.xlsx* is replaced with the latest one *Vehicle Requirements.xlsx*.

Linking a new Excel/CSV file exported directly from the modeling tool table

To create a new Excel/CSV file from the modeling tool table and link it at the same time

---

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- On the table toolbar, click , and select **Excel/CSV File** > **Write To File**. In the **Save** dialog, select the location from your file system, and save the new Excel or CSV file to your computer. The Excel or CSV file is exported, linked with the modeling tool table, and ready to be synced.

> The file cannot be saved if a file with the same name and in the same location already exists.

Back to top >(see page 474)

Set the sync options (optional)

You can specify different sync and mapping options than the default ones. For example, you can specify how the deleted rows in the Excel/CSV file will be transferred to the modeling tool table, define a specific Excel file sheet and cell, and create custom column mapping. All default settings are described next to each option in the **Excel/CSV Sync Options** dialog areas(see page 479).

To customize the sync and mapping options

---

1. On the table toolbar, click and do one of the following:
   - Select the **Sync Options** button.
   - Select the **Excel/CSV File** > **Select File**. Click the **Details** button on the left bottom corner of the dialog.
2. Specify the **If rows in the file are deleted option** value in the **Sync Options** area. How to >(see page 481)
3. Specify the mapping options in the **Mapping Options** area. How to >(see page 481)
4. Manage the column mapping in the mapping area. How to >(see page 482)
5. Click **OK**.
   The sync and mapping options are set. They will be applied next time you sync data.

Back to top >(see page 474)

Sync data

Data can be synced bidirectionally by using two commands from the :

- **Read From File** command.(see page 476)
- **Write To File** command.(see page 478)

Read From File command

You can import data from a selected sheet in an Excel file or CSV file to a modeling tool table by using the **Read From File** command. Read the following rules before using this command:

- The **Read From File** command is available only if the Excel/CSV file is linked with a modeling tool table. How to link it >>(see page 475)
- After making any changes in the Excel or CSV file you must save them before using the **Read From File** command. Otherwise, the changes will not be imported to the modeling tool table.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- The Excel or CSV file can be open while importing data from them.
- When syncing data from an Excel file, the **Read From File** command:
  - Syncs only one sheet of the Excel file. By default, the first sheet is synced. You can change the sheet number in the **Excel/CSV Sync Options** dialog(see page 479) > Mapping Options area(see page 481) > **Sheet** option.
  - Syncs the first cell of the sheet filled with data. If the selected sheet is empty, it will sync from the A1 cell. You can change the cell you want to start syncing from in the **Excel/CSV Sync Options** dialog(see page 479) > Mapping Options area(see page 481) > **First cell** option.
- When syncing data from a CSV file, the **Read From File** command syncs the comma delimiter (,) by default. You can change the delimiter in the **Excel/CSV Sync Options** dialog(see page 479) > Mapping Options area(see page 481) > **CSV delimiter** option.
- You can indicate the heading titles of the Excel/CSV file; change this option in the **Excel/CSV Sync Options** dialog(see page 479) > Mapping Options area(see page 481) > **First row contains headings** check box.
- After sync, the deleted rows in the file are marked as obsolete in the modeling tool table. You can change this behavior in the **Excel/CSV Sync Options** dialog(see page 479) > Sync Options area(see page 481) > **If rows in the file are deleted** option.
- The columns of the Excel/CSV file and modeling tool table by default are mapped on the ordinal position. You can create your own column mapping in the **Excel/CSV Sync Options** dialog(see page 479) > Mapping area(see page 482).

To import data from an Excel/CSV file to a modeling tool table

---

1. On the table toolbar, click .
2. Select the **Read From File** command.
3. In the **Select Type** dialog, select the element you want to create in the modeling tool table.
   The data is imported from the Excel/CSV file to the modeling tool table. If you made changes in the file and saved them, the table Legend(see page 530) displays automatically and highlights the **New**, **Obsolete**, **Unchanged**, and **Updated** rows. In the following figure, the first requirement is updated, the seventh is newly created, and the second is deleted.

Excel Import Status: New Obsolete Unchanged Updated

#	Name	Text
1	Original Statement	Describe a system for purifying dirty water. - Heat dirty water and condense steam are performed by a Counter Flow Heat Exchanger - Boil dirty water is performed by a Boiler. Drain residue is performed by a Drain. The water has properties: vol = 1 liter, density 1 gm/cm3, temp 19 deg C, specific heat 1cal/gm deg C, heat of vaporization 540 cal/gm.
2	Purify Water	The system shall purify dirty water.
3	Heat Exchanger	Heat dirty water and condense steam are performed by a Counter Flow Heat Exchanger
4	Boiler	Boil dirty water is performed by a Boiler.
5	Water Properties	Water has properties: density 1 gm/cm3, temp 20 deg C, specific heat 1cal/gm deg C, heat of vaporization 540 cal/gm.
6	Water Initial Temp	Water has an initial temp 20 deg C
7	Drain	Drain residue is performed by a Drain.

Read From File exceptions
Data cannot be imported or imported correctly if:

- The CSV file format is invalid.
- Cells are non-editable.
- Data types of columns are not compatible with source information.
- Columns do not support information updates.
- Elements with the same names but different types exist.
- The owner is not set or updated because this element does not exist in the model.

Also, some text formatting may not be transferred.

Write To File command

You can export data from a modeling tool table to an Excel/CSV file by using the **Write To File** command. Read the following rules before using this command:

- If the the file is not linked with the table, this command allows you to export a new Excel or CSV file to your computer and create a reference automatically between the table and the exported file. The file cannot be saved and linked with the table if a file with the same name and in the same location already exists. You can use the **Export** button from the table toolbar(see page 408) to export a table without creating a reference.

If the the file is linked with the table, this command transfers the data directly to your Excel/CSV file.

To export data from a modeling tool table to an Excel/CSV file

1. On the table toolbar, click .
2. Select the **Write To File** command.
   The data is exported from modeling tool table to Excel/CSV file.

Write To File exceptions
The data cannot be exported or exported correctly if:

- The Excel file is corrupted.
- The CSV file is open.
- The CSV file is locked or you do not have the necessary permissions to access it.

Open file (optional)

You can open the linked Excel or CSV file directly from a modeling tool table.

To open a linked Excel and/or CSV file directly from a modeling tool table

1. On the table toolbar, click .
2. Select **Excel/CSV File** > **Open File**.
   The linked file is opened.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Turn off syncing (optional)

You can turn off syncing if you do not want to import/export data between your modeling tool and an Excel/CSV file. For this, you must remove the reference between the Excel/CSV file and the modeling tool table.

To remove a reference between a Excel/CSV file and a modeling tool table

---

1. On the table toolbar, click .
2. Select **Excel/CSV File** > **Clear Selection**.
   The reference between tables is removed and syncing is turned off.

Back to top >(see page 474)

Related pages

- Legends(see page 530)

Excel and CSV Sync Options dialog

On this page

- Opening the Excel/CSV Sync Options dialog(see page 479)
- Description of the Excel/CSV Sync Options dialog areas(see page 479)
  - Select Excel/CSV file area(see page 480)
  - Sync Options area(see page 481)
  - Mapping Options area(see page 481)
  - Mapping area(see page 482)

The **Excel/CSV Sync Options** dialog allows you to specify settings before syncing data between an Excel/CSV file and a table. You can select files for syncing and define syncing and mapping options in this dialog.

Opening the Excel/CSV Sync Options dialog

To open the **Excel/CSV Sync Options** dialog

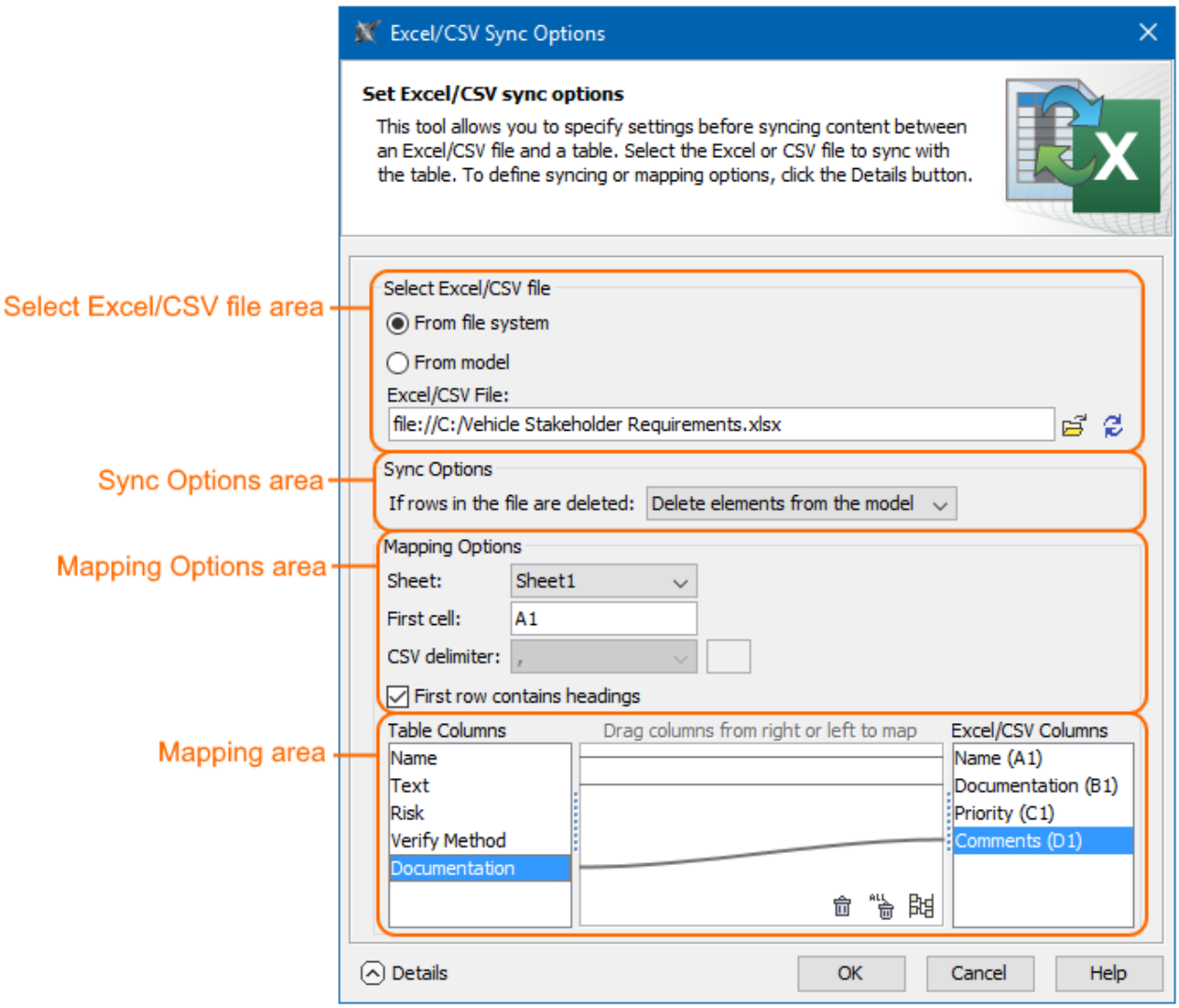
- In the table toolbar, click and do one of the following:
  - Select the **Sync Options** command.
  - Select the **Excel/CSV File** > **Select File**. Click the **Details** button on the left bottom corner of the dialog.
  The **Excel/CSV Sync Options** dialog opens.

Description of the Excel/CSV Sync Options dialog areas

The **Excel/CSV Sync Options** dialog consists of the following areas:

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

-
  - Select Excel/CSV file area(see page 480)
  - Sync Options area(see page 481)
  - Mapping Options area(see page 481)
  - Mapping area(see page 482)



The main areas, highlighted in orange, in the Excel/CSV Sync Options dialog: select file, sync, mapping options, and mapping.

Select Excel/CSV file area

The Select Excel/CSV file area allows you to select the Excel or CSV file from your file system or from your model if the Excel or CSV file is attached to a project. How to attach the file to a project >(see page 209)

The Select Excel/CSV file area contains the buttons described in the following table.

Button	Description
	If you select **From file system**, opens the **Open** dialog. If you select **From model**, opens the **Select Excel/CSV file** dialog.
	Refreshes the mapping area after changing the file.

Sync Options area

The Sync Options area contains one option that allows you to specify the sync mechanism behavior when importing data to a table if rows in your file are deleted. All available values are described below.

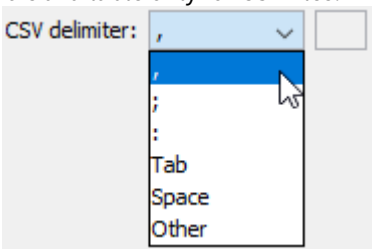
Sync option	Description
If rows in the file are deleted	• **Delete elements from the model** - deletes the same elements that were deleted in the file from your model. If some elements are read-only in Excel file, they cannot be deleted from the model. • **Remove elements form the table** - removes the same elements that were deleted in the file **only** from the modeling tool table, **not** from the model. • **Mark as obsolete** (default) - The table Legend(see page 530) automatically displays and highlights the removed elements (obsolete) in red.

Mapping Options area

The Mapping Options area allows you to specify mapping options. The following table lists all available options.

Mapping option	Description
Sheet	Select the Excel file sheet you want to sync. The first sheet is selected by default. It is available only for Excel files.
First cell	Select the Excel file cell you want to start syncing from. The first cell filled with data is read by default. If the selected sheet is empty, it reads from the A1 cell. It is available only for Excel files.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Mapping option	Description
CSV delimiter	Select the delimiter of the CSV file from the drop-down list. If you want to specify the custom delimiter in the next box, select **Other**. By default, the comma (,) is selected. It is available only for CSV files. 
First row contains headings check box	Clear the check box if your Excel/CSV file does not contain headings. The check box is selected by default.

Mapping area

The Mapping area allows you to manage mapping between modeling tool table columns and Excel/CSV table columns by creating new mappings, re-mapping, or deleting them:

- To create a new map or re-map one by one, drag columns from the left tree/right tree onto columns in the right tree/ left tree.
- To create a new map or re-map all columns in the ordinal position, click at the bottom of the mapping area.
- To delete mapping one by one, select an appropriate map and click at the bottom of the mapping area.
- To delete all mappings, click at the bottom of the mapping area.

If you do not define your own mapping - that is, the mapping area is empty - the columns by default are mapped on the ordinal position.

The hashtag (#) column from the modeling tool table is not included in the mapping.

Related pages

- Sync with Excel or CSV files(see page 474)
- Table toolbars(see page 408)
- Legends(see page 530)

Diagramming

When working with diagrams it is helpful to keep the following concepts in mind:

- A shape refers to the notation of a model element, such as a package, class, state, use case, object, etc.
- A path refers to the notation for the various kinds of relationships, such as associations, aggregations, dependency, message, and links.
- Both paths and shapes are defined as symbols.

The following pages walk you through the procedures for working with the various types of diagrams.

Creating Diagrams(see page 483)

Creating diagrams(see page 483)

Dragging objects(see page 487)

Displaying elements(see page 490)

Copying and pasting objects(see page 517)

Using diagram tabs(see page 519)

Specifying diagram properties(see page 520)

Specifying diagram style properties(see page 522)

Diagram name and its context name synchronization(see page 523)

Diagram frame(see page 523)

Drawing diagram shapes(see page 525)

Creating diagrams

You can create any diagram in one of the following ways:

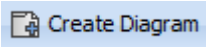
- By using the diagram creation dialog(see page 483)
- From the Model Browser(see page 484)
- From the element's Specification window(see page 485)
- By using the diagram creation wizard(see page 485)

To create a diagram using the diagram creation dialog

---

1. In the Model Browser, select the owner for the new diagram.
2. Do one of the following:
    - Press Ctrl+N.

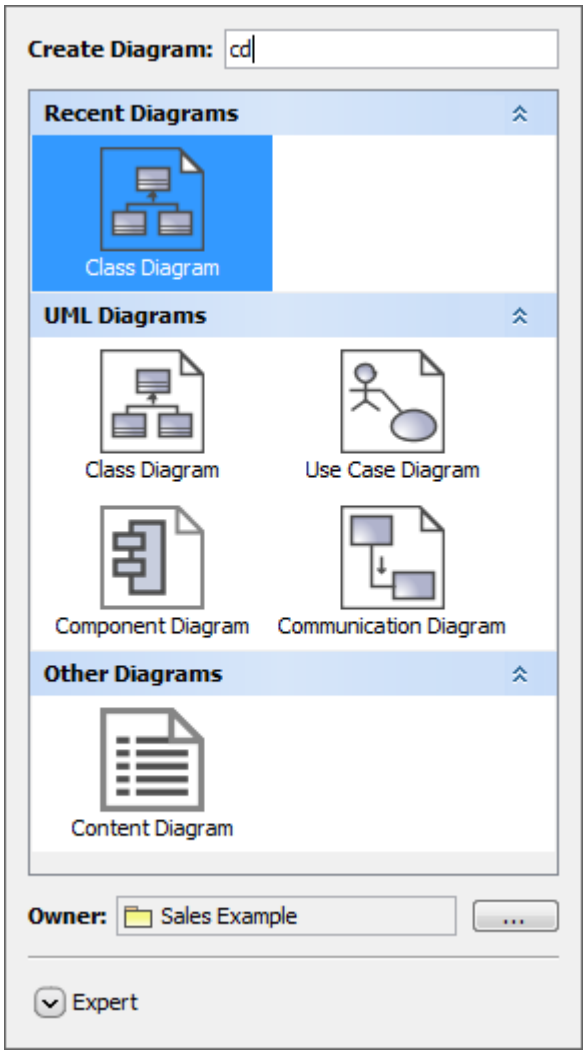
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- On the main toolbar, click 

.
- From the main menu, select **Diagrams** > **Create Diagram**.
- Open the Content diagram and on its palette, click the **Create Diagram** button.

3. When the diagram creation dialog opens, in the **Create Diagram** box, enter the diagram type you are searching for.

> Use the Camel Case, when searching for diagrams:
> - Type the first letters (for example, "CD" to find "Class Diagram")
> - Type lower-case letters (for example, "cd" instead of "CD")
> - Skip the spacers (for example, "cv3" works as well as "CV-3" or "cv-3")
> - Perform a partial search (for example, "ibd" finds SysML Internal Block diagram)



4. Select the diagram.

> Press Esc or click outside the dialog to close the diagram creation dialog without creating the diagram.

To create a diagram from the Model Browser

1. Right-click the Package, Profile, Model, or other element where you want to create the diagram.
2. From the shortcut menu, point to **Create Diagram**, then click the diagram type you want to create.

> You can also create diagrams for multiple elements at the same time. Press and hold the **Shift** key while selecting several elements in your model.

To create a diagram from the element's Specification window

---

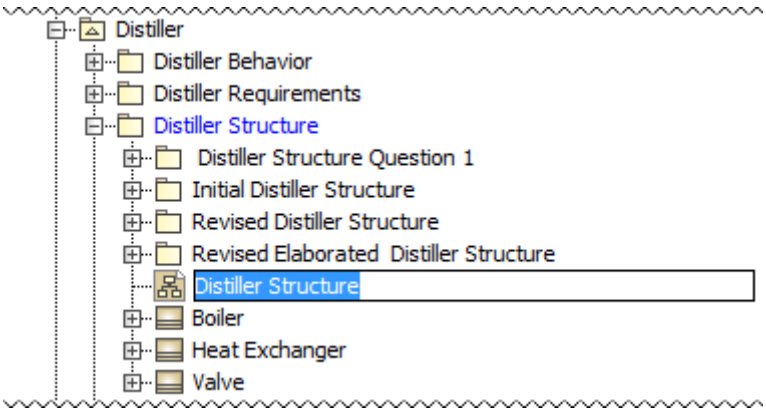
1. Open the **Inner Elements** property group in the Package's, Profile's, Model's, or other element's Specification window.
2. Click the **Create** button. From the button's shortcut menu select **Create Diagram**, then choose the new diagram type. The Specification window of the new diagram opens.
3. In the open window, type the diagram name and click **Close**.

To create a diagram using the diagram creation wizard

---

1. Do one of the following:
   - On the main menu, click **Diagrams** > **Diagram Wizards**, then select the appropriate wizard depending on the type of diagram you want to create.
   - On the main menu, click **Analyze** > **Model Visualizer**. The **Model Visualizer** dialog with the list of available diagram creation wizards is opened. Select one and click the **Start** button.
2. Follow the steps of the selected diagram type creation wizard.

The newly created diagram is automatically named after the owner. If you create several diagrams of the same type under the same owner, their names are numbered.



Related pages

- Diagramming(see page 482)

Opening diagrams

You can open diagrams either in the same tab or in a new tab – just like in the most popular Internet browsers.

The benefits of using tabbed browsing include:

- Ability to analyze different aspects of the system (such as requirements, architecture, test cases, or domain models) in separate tabs. Both backward and forward navigation are supported for each aspect (tab) separately. Navigation does not open new tabs by default, which greatly helps in limiting the number of open tabs.
- Ability to compare separate views of the system side-by-side. For example, business and IT architectures can be displayed in two tabs side-by-side and analyzed by drilling down in their hierarchies. Navigation deeper in the structures will not open new tabs, which allows smooth analysis of the modeled system.
- Smooth work with multiple screens. Browsing in a tab of the second screen never opens diagrams in the first screen, which means you stay in the same screen. New tabs open in the same screen, to the right of the active one.

To open a diagram do one of the following:

- Double-click the diagram.
- Select the diagram and press Enter.
- Right-click the diagram and then click **Open in New Tab**.
- Select the diagram and press Ctrl+Enter. The diagram opens on the top of the active tab.
- Right-click the diagram and then click **Open**. The diagram opens on the top of the active tab.

To open a diagram from the diagram pane:

- Double-click the model element to which the diagram is assigned. The new diagram opens in the same diagram tab.

To open a diagram from the content diagram (available in the Standart, Professional, Architect, and Enterprise editions):

If the diagram is added to the table of contents or a shape of the diagram is drawn on the diagram pane, do one of the following:

- Double-click the diagram shape. The diagram opens in the same diagram tab.
- On the diagram pane, right-click the diagram shape, and then click **Open**. The diagram opens in the same diagram tab.
- On the diagram pane, right-click the diagram shape and then click **Open in New Tab**

To open a diagram from Diagrams menu, do one of the following:

- On the main menu point to **Diagrams** and then click **Recently Opened**. The **Recently Opened Diagrams** dialog opens. Double-click the diagram you want to open or press Enter.

- Press F12. The **Recently Opened Diagrams** dialog opens. Double-click the diagram you want to open or press Enter.

> - You can change the default diagram opening behavior in the **Environment Options** dialog, the **General** options group, under the **Navigation** category.
> - Press the Ctrl key when opening diagram, to turn on the opposite diagram opening mode. For example, if by default opening in the same tab is active, by using the Ctrl key, the diagram will open in the new tab and vice versa.
> - To load all diagrams that have been created in the project, from the **Diagrams** menu, select **Load All Diagrams**.
> - In the **General** pane of the **Environment Options** dialog box, you can select a method for loading diagrams while opening a project. Three options are available:
>
> a) **Load all Diagrams** – loads all diagrams existing in the project.
> b) **Load Only Open Diagrams** – loads only diagrams that were not closed in earlier usages of the project.
> c) **Do not Load Diagrams** – all diagrams are not loaded and closed after opening a project.

To navigate backward and forward between diagrams in the same diagram tab, do one of the following

---

- On the diagram toolbar, click the or button.
- Press Alt+Left Arrow or Alt+Right Arrow.

Related pages

- Diagramming(see page 482)

Dragging objects

On this page

- Dragging files and elements(see page 487)
- Dragging in diagrams(see page 488)

Dragging files and elements

The drag-and-drop capability allows you to:

- Drag files from your file system onto an element in the Model Browser or a symbol on a diagram.

> - A hyperlink to the dragged file is automatically created.
> - Dragged images are set as the **Image** property value of the element.
> - If an image is dragged to a Stereotype, it is set as a Stereotype icon.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Drag a project to the diagram pane to open the project.
- Drag source code files from your file system to the MagicDraw Code Engineering Set.
- Drag any elements to any properties in the Specification window(see page 913).

  For example, you can drag a Class element from the Model Browser to the **Type** propert y in the **Operation** Specification window. That Class element is assigned as the Operation type in the Specification window.

- Drag any elements from the Specification window(see page 913) onto:
  - An empty place on a diagram. In this case, the class symbol is created on the diagram.
  - A shape on a diagram. In this case, a new attribute is created.
  - An element in the Model Browser. In this case, a new model element is created.
- Drag a property to create an association(see page 651):
  - Drag a property from the classifier (for example, class, component, subsystem) compartment to a diagram pane.
  - Drag a property from the Model Browser(see page 29) to a diagram pane.

    In both cases, an association (see page 651)is drawn from the classifier (see page 666)to a property's type. The property itself is set as an association end. If the classifier symbol is not present on the diagram, it is added as a result of the created association.

If selected model elements are not compatible with an open diagram, you will not be allowed to drag those model elements.

Dragging in diagrams

You can drag-and-drop relationships to a diagram. The path for the relationship with shapes at the end of the path and the path between the shapes (if the shapes at the end of the path are already represented on the diagram) is created.

You can also drag-and-drop model elements to a note or a text box symbol in a diagram to add a hyperlink to that element.

Dragging in Sequence diagrams

In a Sequence diagram(see page 304) you can:

- Drag an operation(see page 677) from the Model Browser(see page 29) to a Message(see page 756) in a Sequence diagram. The Message becomes a Call Message once the operation is assigned.

  The Lifeline(see page 746) type must have or inherit this Operation.

- Drag an Interaction to an Interaction Use(see page 740). The Interaction will be set as the **Refers To** element.
- Drag a Signal to a Message. The Message is converted into a Send Signal Message and the Signal is assigned to the Message.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Dragging in State Machine diagrams

- In a State Machine diagram(see page 308) you can drag an Event(see page 708) element to a Transition(see page 872) element. A Trigger with this Event is created for the Transition element.
- You can drag behaviors from one state shape to another.

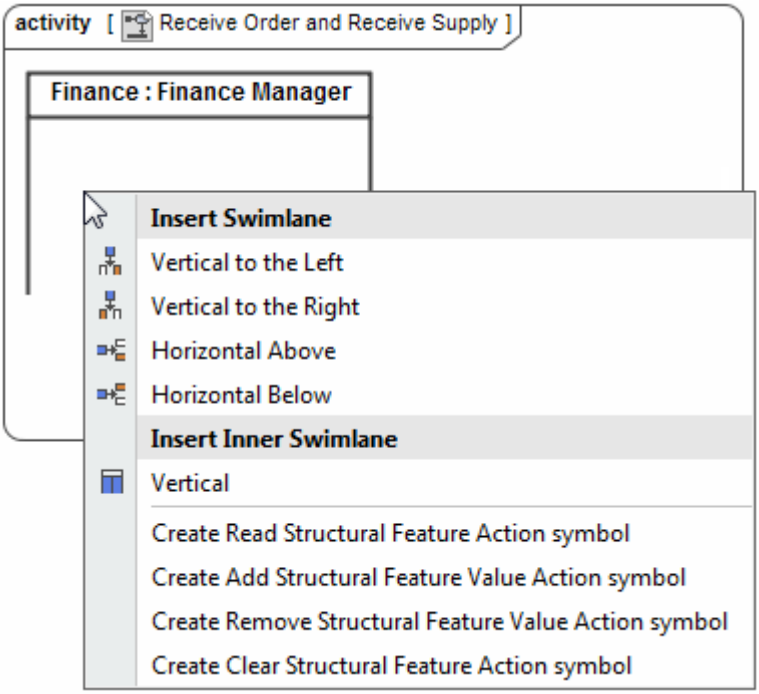
Dragging in Activity diagrams

In an Activity diagram(see page 311) you can:

- Drag an Activity, Interaction(see page 739), or a State Machine onto an Activity diagram to create a Call Behavior Action(see page 633).
- Drag a Signal onto an Activity diagram to create a Send Signal Action(see page 638).
- Drag a Signal onto a Send Signal Action to set or change the Signal.
- Drag a Signal onto an Accept Event Action(see page 630) to set or change the Signal.

> ⚠ Once the Signal is assigned to the Accept Event Action, the Signal Event for the diagram and the Trigger for the action will be created automatically.

- Drag an Event onto an Activity diagram to create an Accept Event Action.
- Drag an Event onto an Accept Event Action to set the Event.
- Drag a Signal Event to an Activity diagram to create an Accept Event Action.
- Drag a Property, Actor(see page 646), Class(see page 666) or Instance(see page 735) to an Activity diagram to create a Swimlanes(see page 861). Elements will be set as representative elements.
- Drag a Property, Actor, Class or Instance Specification to an Activity diagram by using the right mouse button. From the shortcut menu, select to create a Swimlane, Central Buffer Node, or available Actions.



Related pages

- Sequence diagram(see page 304)

- State Machine diagram(see page 308)
- Activity diagram(see page 311)
- Diagramming(see page 482)

Displaying elements

MagicDraw includes a number of display commands that allow you to represent elements in different diagrams. However, displaying elements is only possible if they are already created somewhere in the model. The display commands are interactive, and depend upon the selection on the diagram pane. The table below lists all of the commands which are available, and a short description of each. The table also provides a link to a more in depth description of its functionality, and lists the diagrams in which they can be used.

Command names	Description	Usage in diagrams
Display Related Elements	Opens the **Display Related Elements** dialog in which you can specify the relation type(s), their direction, scope, depth, and other options. All specified criteria define which elements and relations will be displayed on the diagram pane of the selected element. Read more >(see page 508)	All diagrams
Display Paths	Displays paths among shapes that already exist in the model. Read more >(see page 513)	All diagrams
Display Internal Structure	Displays internal structure inside a selected classifier shape that contains its own internal structure. Read more >(see page 507)	All diagrams
Display Parts/Ports	Opens the **Display Parts/Ports** dialog from where you can specify the exact properties and/or ports you want to display on the diagram pane. Read more >(see page 491)	Class, Composite Structure, Component, Package, Deployment, Object diagrams
Hide All Ports	Hides all ports of the selected shape(s) or diagram frame. Read more >(see page 491)	Class, Composite Structure, Component, Package, Deployment, Object diagrams

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Command names	Description	Usage in diagrams
Display Inner Elements	Displays internal elements and paths inside selected shape or diagram frame. There are different procedures when trying to display inner elements in: • The Package diagram. Read more >(see page 801) • The Activity diagram. Read more >(see page 311) • The State Machine diagram. Read more >(see page 308)	All diagrams
Display Provided/Required Interfaces	Displays provided or required interfaces of the selected port shape. Read more >(see page 742)	Class, Composite Structure, Component, Package, Deployment, Object diagrams
Display Pins	Opens the **Select Pins** dialog from where you can select the exact pins you want to display on the selected shape. Read more >(see page 805)	Activity diagram
Display Parameter Nodes	Opens the **Select Parameters** dialog from where you can select exactly which Activity Parameter Nodes you want to display on the frame of the Activity Diagram. Read more >(see page 783)	Activity diagram
Display Expansion Nodes	Displays the input or output Expansion Nodes on the Expansion Region border. Read more >(see page 788)	Activity diagram
Display Points	Opens the **Select Points** dialog in which you can select which Connection Point References you want to display on a submachine State shape. Read more >(see page 853)	State Machine diagram
Display Lifelines	Opens the **Display Lifelines** dialog from where you can select which lifelines you want to display. Read more >(see page 746)	Sequence diagram Communication diagram
Display Messages	Opens the **Select Messages** dialog from where you can select which messages of the selected connector you want to display. Read more >(see page 774)	Communication diagram

Displaying parts and ports

You can display ports, properties, and deep nested properties that have already been created in all structure diagrams[75]. This functionality is driven by stereotypes, which allow you to display properties on which stereotypes(see page 856) have been applied. If you have applied your own stereotypes(see page 856) for the elements (e.g., electrical, mechanical, optical), you can use those stereotypes when selecting the properties you want to display on the diagram pane.

> **Note**
> For more information about how to create custom stereotypes and apply them, see UML Profiling and DSL Guide(see page 2034).

This page contains the following information:

- Ways to display parts and ports(see page 492)
  - Using the Display Parts/Ports dialog(see page 492)
  - Using the diagram toolbar and the shortcut menu(see page 497)
  - Using the smart manipulator toolbar(see page 499)
  - Using the drag-and-drop operation in the Composite Structure diagram(see page 500)
- Displaying properties of a Port(see page 504)

Ways to display parts and ports

You can display ports and properties in the following ways:

-
  - Using the Display Parts/Ports dialog(see page 492)
  - Using the diagram toolbar and the shortcut menu(see page 497)
  - Using the smart manipulator toolbar(see page 499)
  - Using the drag-and-drop operation in the Composite Structure diagram(see page 500)

Using the Display Parts/Ports dialog

You can use the **Display Parts/Ports** dialog to control how much of a structure should be displayed. The **Display Parts/Ports** dialog opens automatically when creating a new Composite Structure diagram(see page 325) for a Class(see page 666) element that owns properties. It can also be accessed and opened in the structure diagrams that have already been created. The **Display Parts/Ports** dialog allows you to:

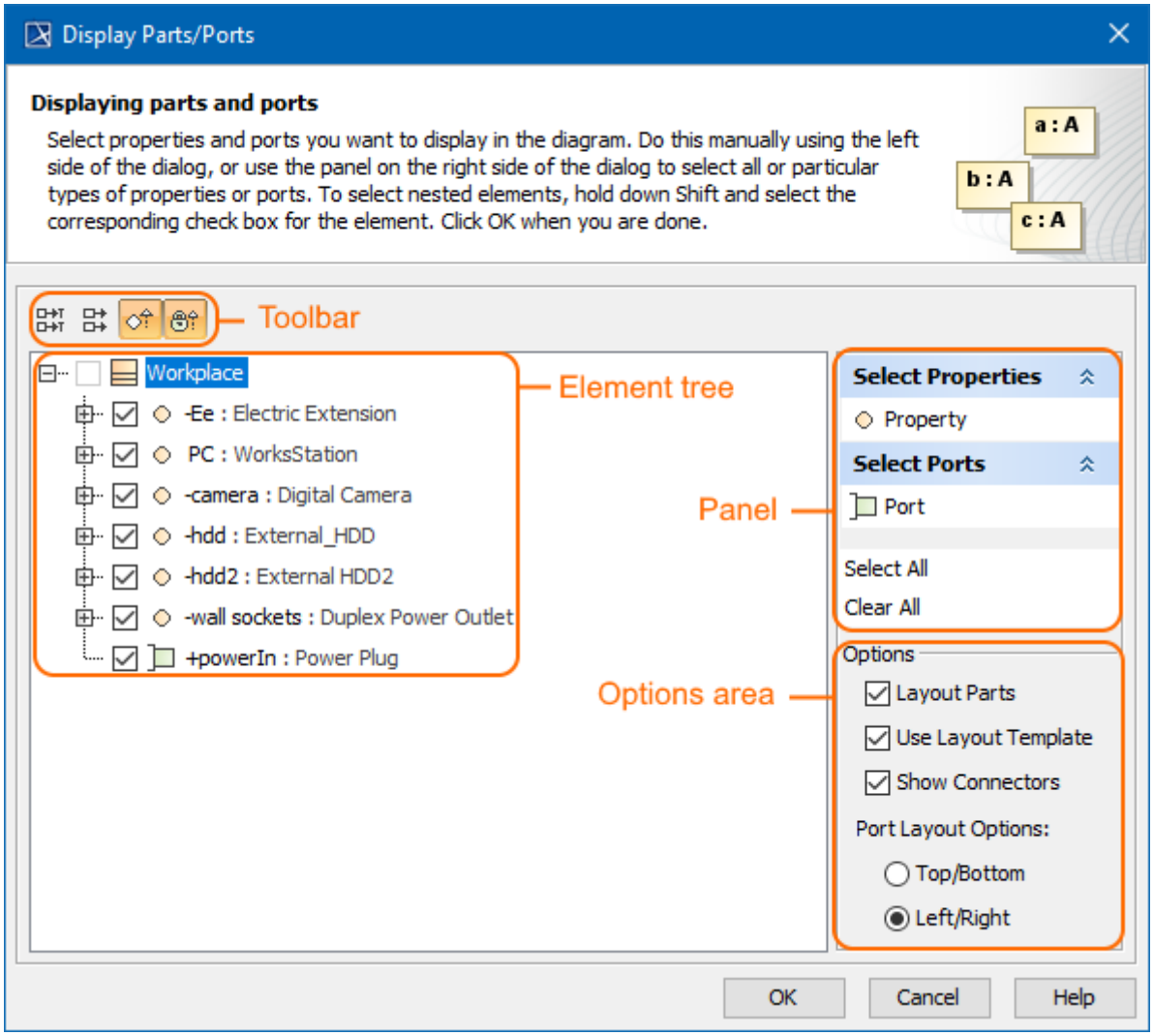
- Preview all Parts(see page 804), Ports(see page 809), and other properties in one place.
- Manually or automatically select required Parts(see page 804), Ports(see page 809), and other properties you want to display.
- Manually or automatically choose deeply nested(see page 327) Parts(see page 804), Ports(see page 809), and other properties you want to display.
- Define layout options for Parts(see page 804).
- Specify layout options for Ports(see page 809).

The **Display Parts/Ports** dialog consists of the following areas:

---

75 https://docs.nomagic.com/download/attachments/55846387/uml_diagrams.png?api=v2&modificationDate=1476892380940&version=1

- *Dialog toolbar* allows you to show or hide additional information about listed elements.
- *Element tree* displays all owned internal structure of context Class(see page 666).
- *Panel* is a context-sensitive area that depends on the selected property from the Element tree. It allows you to batch select Parts(see page 804)/Ports (see page 809) in the Element tree. You can select all, clear all, or choose only properties by type, stereotype, or aspect.
- *Options* area allows you to choose the layout of the selected parts and Ports (see page 809)when they appear in the diagram.



The following procedure will guide in working with the **Display Parts/Ports** dialog when you want to display required properties on the diagram pane.

To display Ports and Properties using the **Display Parts/Ports** dialog

1. Open the **Display Parts/Ports** dialog:
    - When creating a new Composite Structure diagram(see page 325), the dialog opens automatically.

- When editing an already created structure diagram[76] , the dialog can be opened in the following ways:
  - Select either a diagram frame or single/multiple element shapes, and, on the diagram toolbar, click and choose **Display Parts/Ports**.
  - Right-click either a diagram frame or single/multiple element shapes, and select **Display** > **Display Parts/Ports**.

2. (Optional) Enable buttons from the **Display Parts/Ports** dialog toolbar if you want to show or hide additional information about listed elements:
   - Click to show the full path of elements.
   - Click to show the full types of elements.
   - Click to show the inherited elements.
   - Click to show the inherited properties.

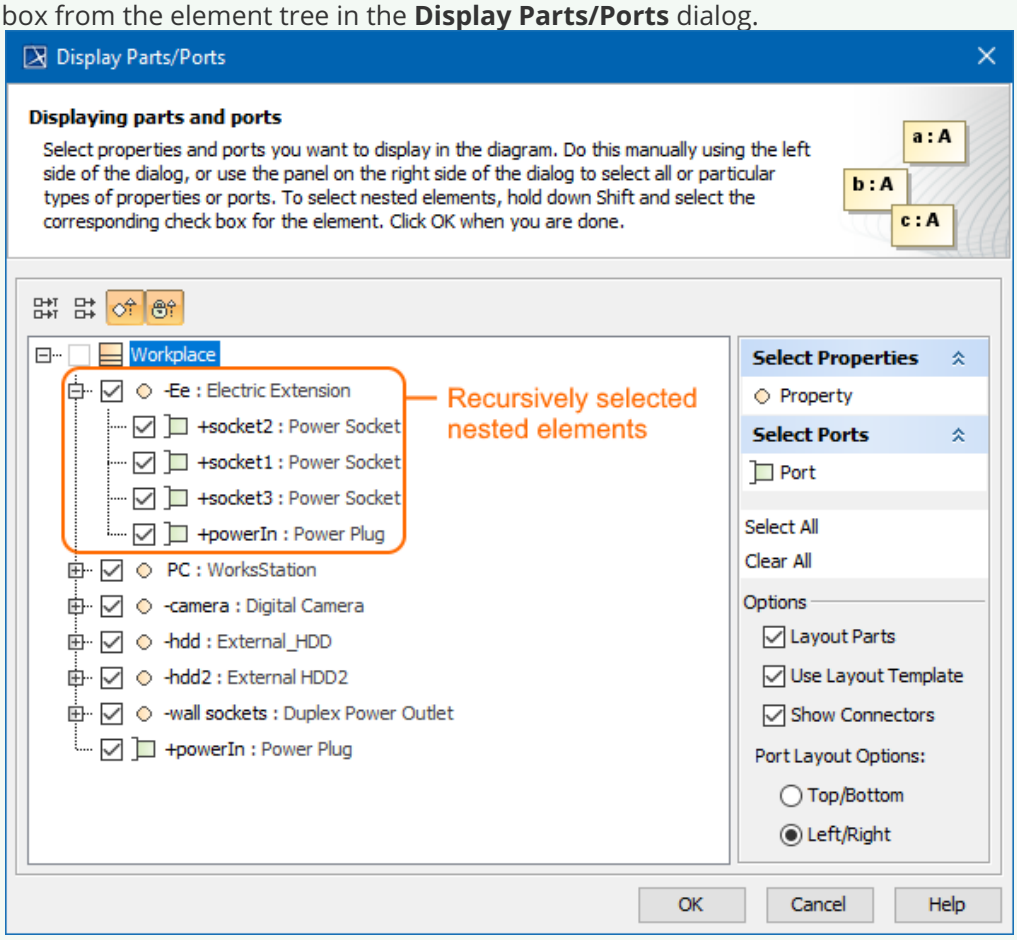
3. Choose the properties you want to display by using one or more of the following methods:

   Warning
   If some check boxes are inactive in the element list, those elements are already displayed in the diagram.

   - Use the element tree located on the left side of the dialog:
     - Manually select individual check boxes one by one.
     - Select recursively nested elements by holding down the Shift key while selecting the corresponding check box for the element.

76 https://docs.nomagic.com/download/attachments/55846387/uml_diagrams.png?api=v2&modificationDate=1476892380940&version=1

Example 1
All properties (*socket1, socket2, socket3, powerIn*) of the *Ee* Property are selected recursively by holding down the Shift key while selecting the corresponding check box from the element tree in the **Display Parts/Ports** dialog.

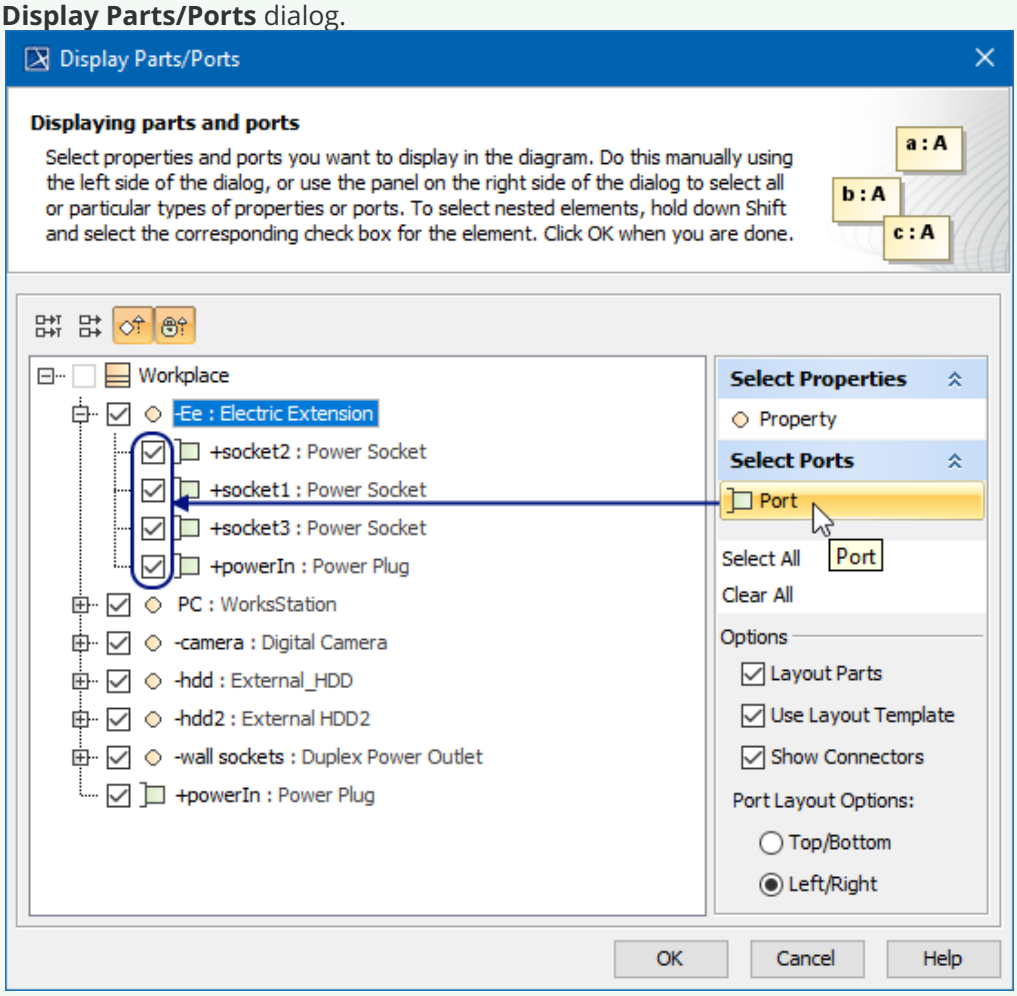


- Use the context-sensitive panel located on the right side of the dialog:
  - **Select Properties** group - displays all property types (including your own stereotypes) that are owned by the selected element from the element tree located on the left side of the dialog.
  - **Select Ports** group - displays all Port types (including your own stereotypes) that are owned by a selected element from the element tree located on the left side of the dialog.
  - **Select All** - selects all check boxes that are expanded from the element tree located on the left side of the dialog.
  - **Clear All** - clears all check boxes, even those that are collapsed (deep nested), from the element tree located on the left side of the dialog.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example 2

All ports (*socket1, socket2, socket3, powerIn*) of the *Ee* Property are selected in the element tree by using the **Select Ports** group in the context-sensitive panel of the **Display Parts/Ports** dialog.



4. Specify the layout options in the **Options** area:
   - Parts layout: select the **Layout Parts** check box if you want the Part shapes to be arranged in the automatic layout.
   - Usage of layout template: select the **Use Layout Template** check box if you want the layout template to be applied. When this check box is selected, the Part, which is included in the layout template, and all its Ports from the layout template are selected automatically.
   - Showing connectors: select the **Show Connectors** check box if you want the connectors to be displayed.
   - Ports layout on Parts shape:
     - **Top/Bottom** - displays Ports on the Part shape from top to bottom.
     - **Left/Right** - displays Ports on the Part shape from left to right.
5. Click **OK**.

After those steps, all parts, ports, other properties that were selected in the **Display Parts/Ports** dialog are displayed on the diagram pane.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

In the structure diagrams that have already been created, you can display Ports(see page 809), Properties(see page 2308), and hide all Ports(see page 809) by using the commands from:

- The **Display** button on the diagram toolbar.
- The **Display** command group under the shortcut menu.

You can also use the **Display** button and **Display** command group from the shortcut menu as their functions are identical. All those commands are interactive and depend upon the selection on the diagram pane. For example, if the Part shape is selected on the diagram pane, the both menu contains only the commands that can be applied to the selected Part shape. If you select nothing in the diagram pane, both the menu shows commands that can be applied to the whole diagram.

These commands under the **Display** button and shortcut menu are comprised of elements that are still not displayed on the diagram according to their:

- Metatypes.
- Stereotypes.
- Types stereotypes.

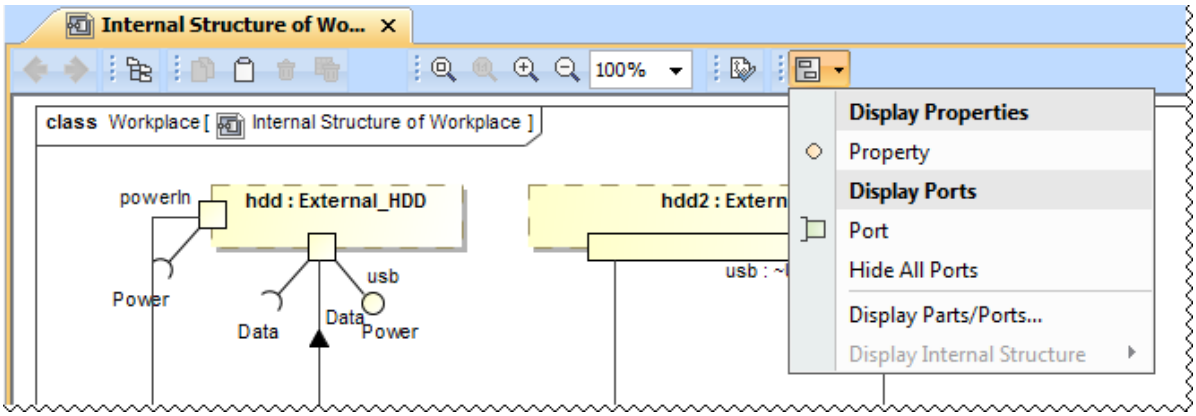
The following procedures will show you how to:

- How to display Ports, Properties using the **Display** button.(see page 497)
- How to display Ports, Properties using the shortcut menu.(see page 498)
- How to hide Ports.(see page 498)

To display Ports, Properties using the **Display** button

---

1. Select either a diagram frame or single/multiple element shapes.
2. On the diagram toolbar, click .



3. From the opened command menu, select the required properties to display:
   - All Ports that are available to display are listed under the **Display Ports** command group.

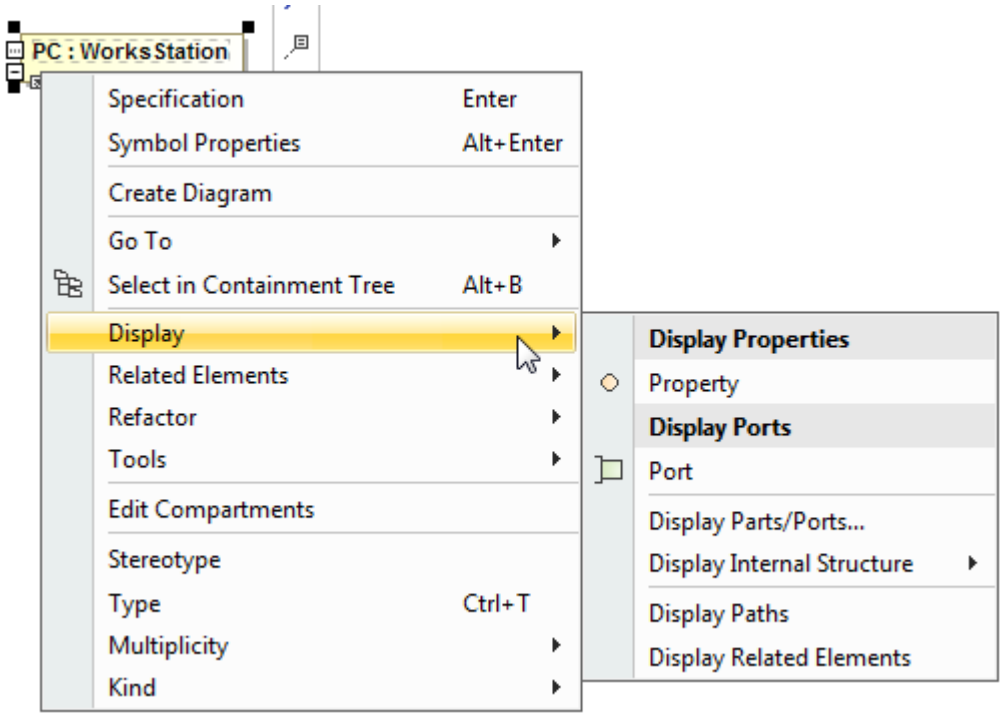
> **Important**
> If nothing is selected on the diagram pane, the **Display Ports** command group will contain commands that allow you to display two levels of structure: ports are displayed on the diagram frame and on the first level parts.

- All Properties that are available to display are listed under the **Display Properties** command group.
After selecting the required command from the **Display** button, the appropriate properties are displayed on the diagram pane.

To display Ports, Properties using the shortcut menu

---

1. Right-click either the diagram pane or single/multiple element shapes.
2. From the shortcut menu, select the **Display** command group.



3. From the opened command menu, select the required properties to display:
   - All Ports that are available to display are listed under the **Display Ports** command group.

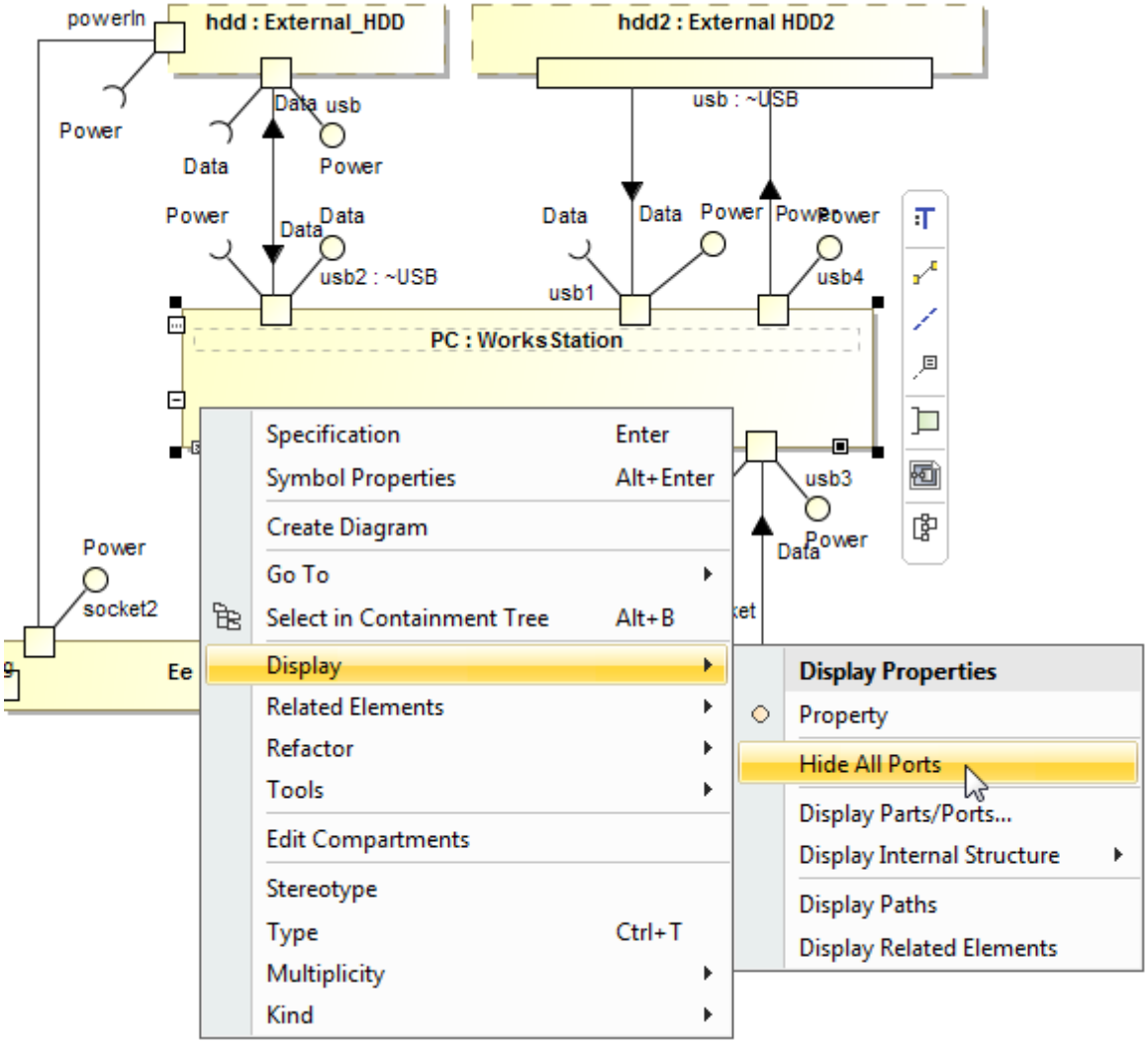
> **Important**
> If nothing is selected on the diagram pane, the **Display Ports** command group allows you to display two levels of structure: ports are displayed on the diagram frame and on the first level parts.

- All Properties that are available to display are listed under the **Display Properties** command group.
After selecting the required command from the Display command menu, the appropriate properties, Ports are displayed onto the diagram pane.

To hide Ports

- Select either a diagram pane or any number of element shapes, and, on the diagram toolbar, click and choose **Hide All Ports**.
- Right-click either a diagram pane or any number of element shapes, and select **Display** > **Hide All Ports**.
  After selecting the command, the Ports (see page 809)of the selected shape will be hidden from the diagram pane, but not removed from the model.

> **Warning**
> If the **Hide All Ports** command doesn't appear, the selection or diagram doesn't contain any Ports (see page 809)that can be hidden.



Using the smart manipulator toolbar

When the properties are already displayed on the structure diagrams[77], you can display their direct ports using the **Display All Ports** button from the smart manipulator toolbar. It does not allow you to display deeply nested ports, only first level ports. To display deeply nested ports, you should use the

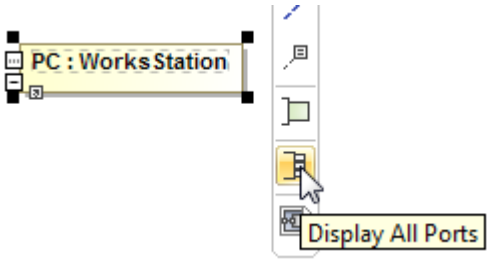
77 https://docs.nomagic.com/download/attachments/55846387/uml_diagrams.png?api=v2&modificationDate=1476892380940&version=1

Display Parts/Ports dialog. Follow the steps below to learn how to display ports of a selected property shape.

To display Ports on shapes

1. Select the required element shape or a diagram frame.
2. From the smart manipulator toolbar, click .
   All first level Ports of the selected shape will be displayed on the diagram pane.

   Warning
   If the **Display All Ports** button is not available, the selected element or diagram doesn't own any Port.



Using the drag-and-drop operation in the Composite Structure diagram

The most efficient way to create a new property and display it with all Ports(see page 809) in the Composite Structure diagram(see page 325) is by dragging the Class(see page 666) or Property elements. This method does not allow you to display deeply nested properties, only first level properties. To display deeply nested elements, use the **Display Parts/Ports** dialog. The following procedures will show you:

- How to create a Property shape and how to display all its Ports when dragging the Class element. (see page 500)
- How to display a Property shape and all its Ports by dragging the Property element.(see page 502)

You can create a new Property by dragging the Class (see page 666)element from the Model Browser (see page 29)to the Composite Structure diagram(see page 325) pane. The newly created Property will be typed by the dragged Class(see page 666) and displayed on the diagram pane. If this Class(see page 666) has Ports(see page 809) defined, you can display them by using the long drag, and selecting an appropriate option from the opened menu (see Example 3).

To create and display a Property by dragging a Class element

- Drag a single or a multiple Class element from the Model Browser onto the diagram pane.
- Drag a single or a multiple Class element from the Model Browser onto the diagram pane, and hold it down until the Tooltip appears (long drag). Drop it, and, from the opened menu, select the **Create Property Symbol** command.
  The new Property is created in the model and displayed on the diagram pane. The newly created Property is typed by the dragged Class automatically.

To create and display a Property and display all its Ports by dragging a Class element

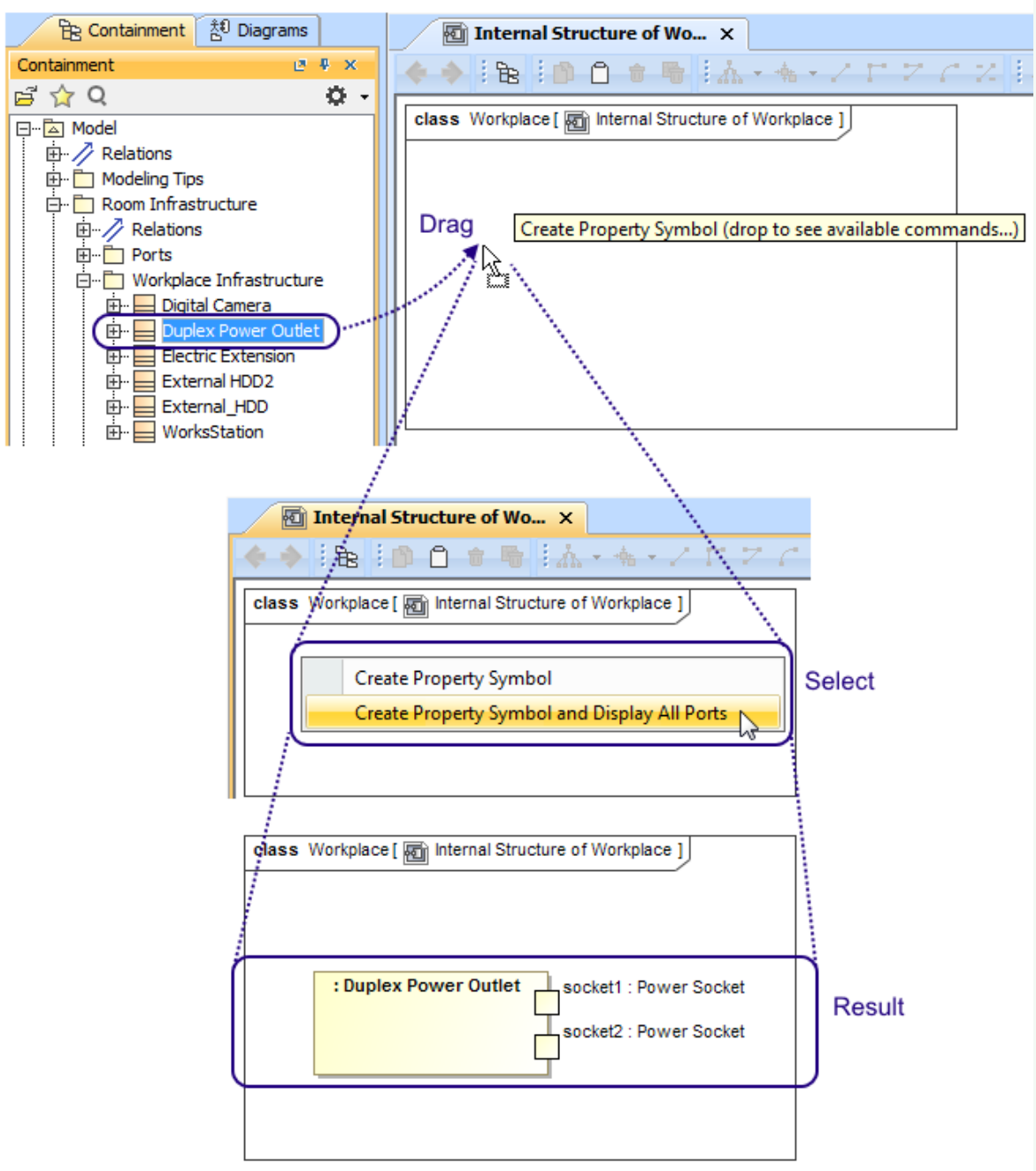
- Drag a single or a multiple Class element from the Model Browser onto the diagram pane, and hold it down until the Tooltip appears (long drag).
- Drop it, and, from the opened menu, select the **Create Property Symbol and Display All Ports** command.
  The new Property is created in the model and displayed with all its Ports on the diagram pane. The newly created Property is typed by the dragged Class automatically.

> **Tip**
> After creating an element, it is beneficial to find its location in the model tree by performing one of the following:
> - Right-click the shape, and, from the shortcut menu, choose the **Select in Containment Tree** command.
> - Select the shape and press Alt+B.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example 3

After selecting the **Create Property Symbol and Display All Ports** command, and dragging (long drag) the Class element to *Internal Structure of Workplace* diagram pane, it creates a new Property typed by a *Duplex Power Outlet* Class and displays all Ports on it.



You can display a Property shape by dragging a Property element from the Model Browser to the Composite Structure diagram(see page 325) pane. If this property has Ports(see page 809) defined, you can display them by using the long drag, and selecting an appropriate option from the opened menu (see Example 4).

The following procedures describe how to display Property shape and all its Ports(see page 809) by dragging the Property element.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To display a Property by dragging a Property element

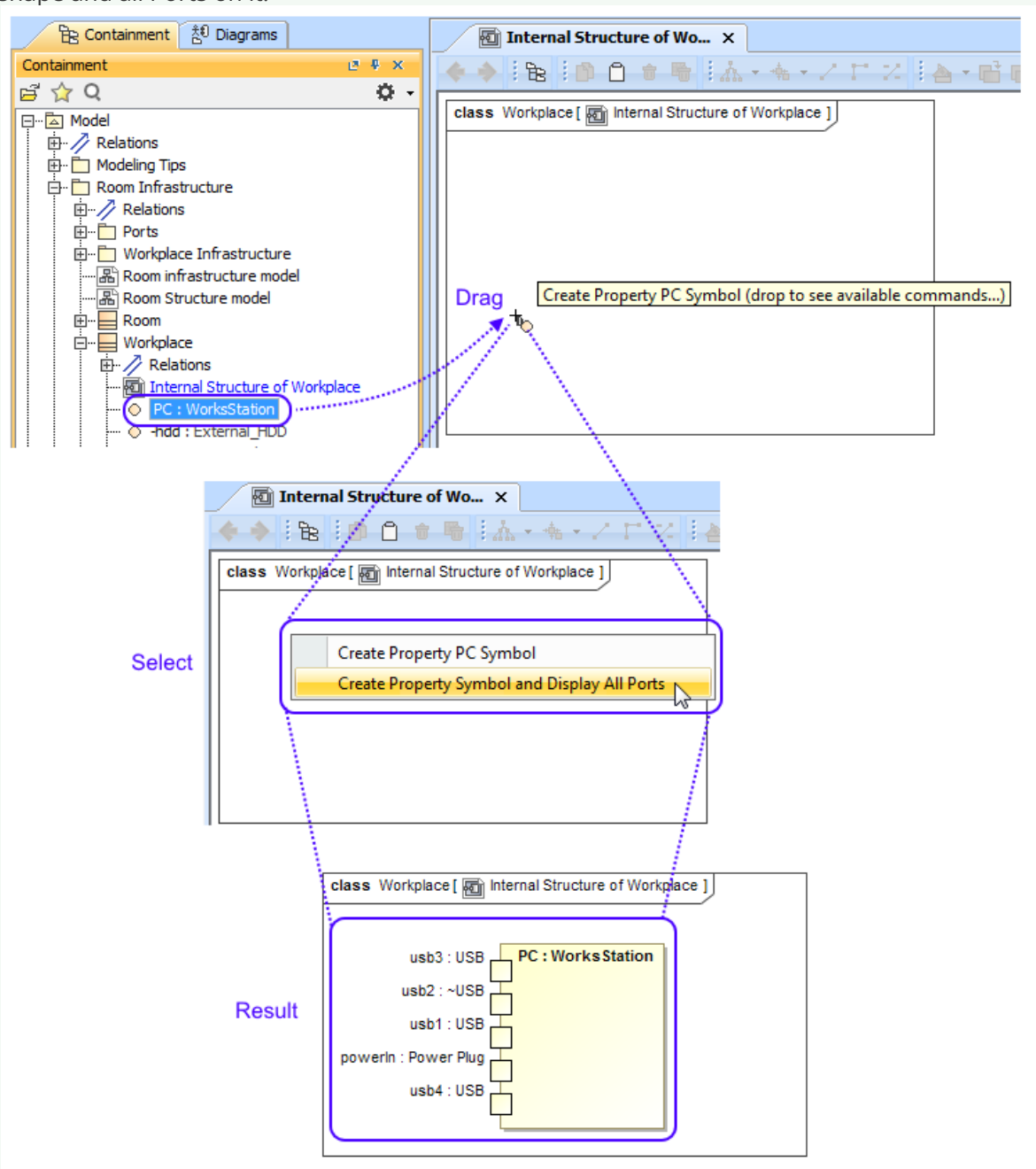
- Drag a single or a multiple Property element from the Model Browser onto the diagram pane.
- Drag a single or a multiple Property element from the Model Browser onto the diagram pane, and hold it down until the Tooltip appears (long drag). Drop it, and, from the opened menu, select the **Create Property Symbol** command.
  The Property shape is displayed on the diagram pane.

To display a Property with all of its Ports by dragging a Property element

- Drag a single or a multiple Property element from the Model Browser onto the diagram pane, and hold it down until the Tooltip appears (long drag).
- Drop it, and, from the opened menu, select the **Create Property Symbol and Display All Ports** command.
  The Property shape is displayed with all its Ports on that shape.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

After selecting the **Create Property Symbol and Display All Ports** command, and dragging (long drag) the Property element to *Internal Structure of Workplace* diagram, it displays Property shape and all Ports on it.



After completing the steps outlined above, the Property shape and/or all its Ports are displayed on the Composite Structure diagram.

Displaying properties of a Port

If you display properties of a Port, all properties of its type will be displayed inside the Port shape and part compartments will be enabled for that Port.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

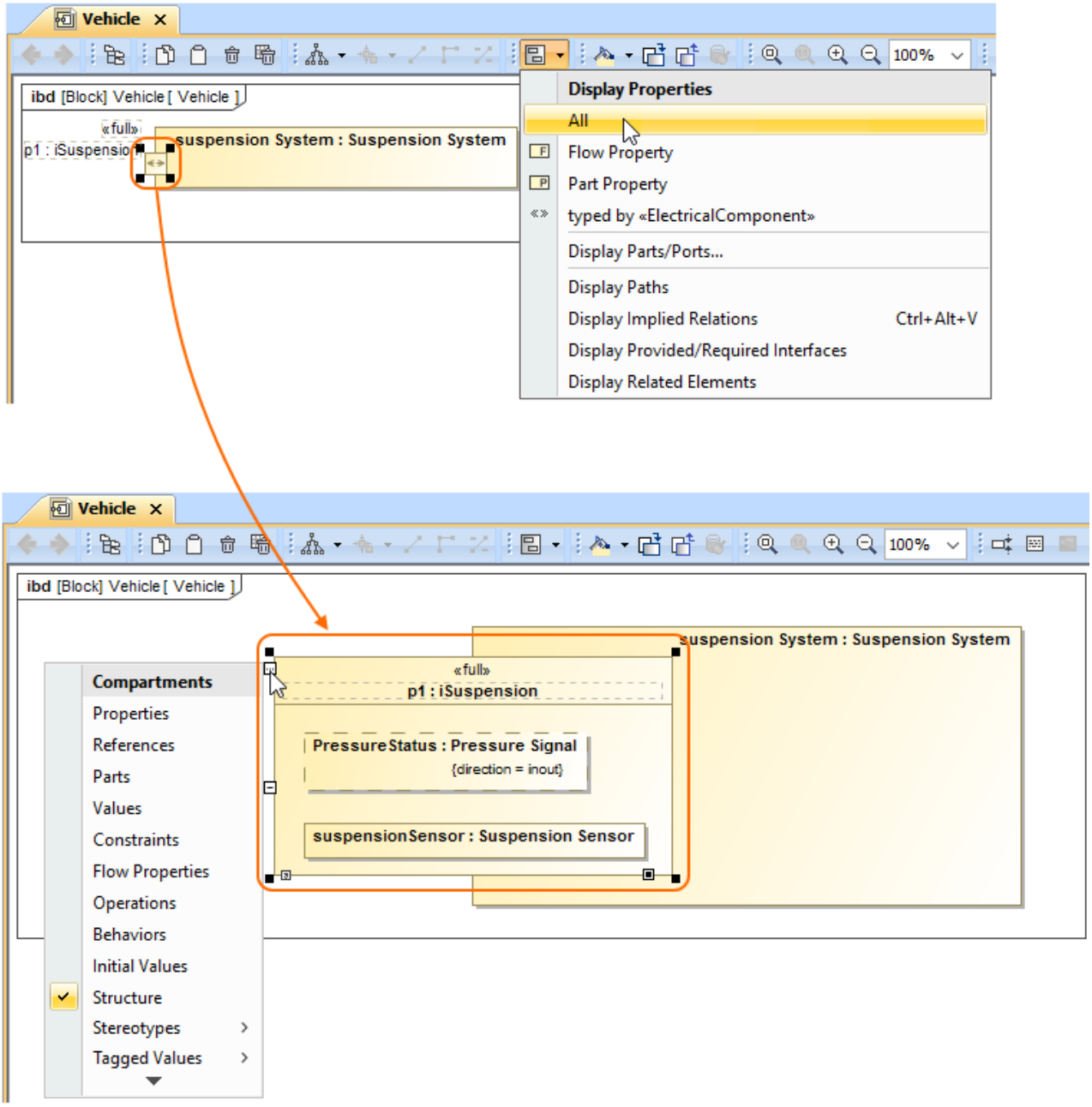
To enable Part compartments on the Port shape

1. Select a Port shape whose type has properties.
2. Do one of the following:
   - On the diagram toolbar, click .
   - From the shortcut menu, point to the **Display** command group.
3. Select the required properties to display.

The properties are displayed inside the Port shape. Part compartments are enabled for the Port.

The example below shows the *p1* Port both before and after displaying its type properties. The properties displayed on the Port shape are the *Pressure Status* Flow Property and the *suspensionSensor* Part Property, which are the properties of the selected Port type *iSuspension* Block. After display of properties has been selected, compartments are automatically enabled for the *p1* Port.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Related pages

- Displaying elements(see page 490)
- Applying Stereotypes(see page 856)
- Nesting parts that are association ends(see page 327)
- Mapping ports on a Part type change(see page 328)
- Class diagram(see page 297)
- Composite Structure diagram(see page 325)
- Component diagram(see page 318)
- Package diagram(see page 321)
- Deployment diagram(see page 319)
- Object diagram (see page 299)

Sample model

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

The models used in the figures of this page are:

- The **Composite structure diagram** sample model that comes with MagicDraw. To open this sample do one of the following: download composite structure diagram.mdzip[78] or find in modeling tool *<modeling tool installation directory>\samples\diagrams\composite structure diagram.mdzip.*
- The **Vehicle parts under ports** sample model which is available only with SysML Plugin(see page 491). To open this sample d*ownload* Vehicle parts under ports.mdzip[79].

Displaying internal structure on structured classifiers

You can represent the internal structure of the structured classifier on the diagram pane in all diagrams. The internal structure of the selected shape can be represented only if the selected element contains its own internal structure. For example, in a Composite Structure diagram on a Part(see page 804) shape or in a Class diagram(see page 297) on a Class(see page 666) shape, you can display its ports(see page 809), parts, and connectors.

The following procedure will show you how to display the internal structure of a selected classifier.

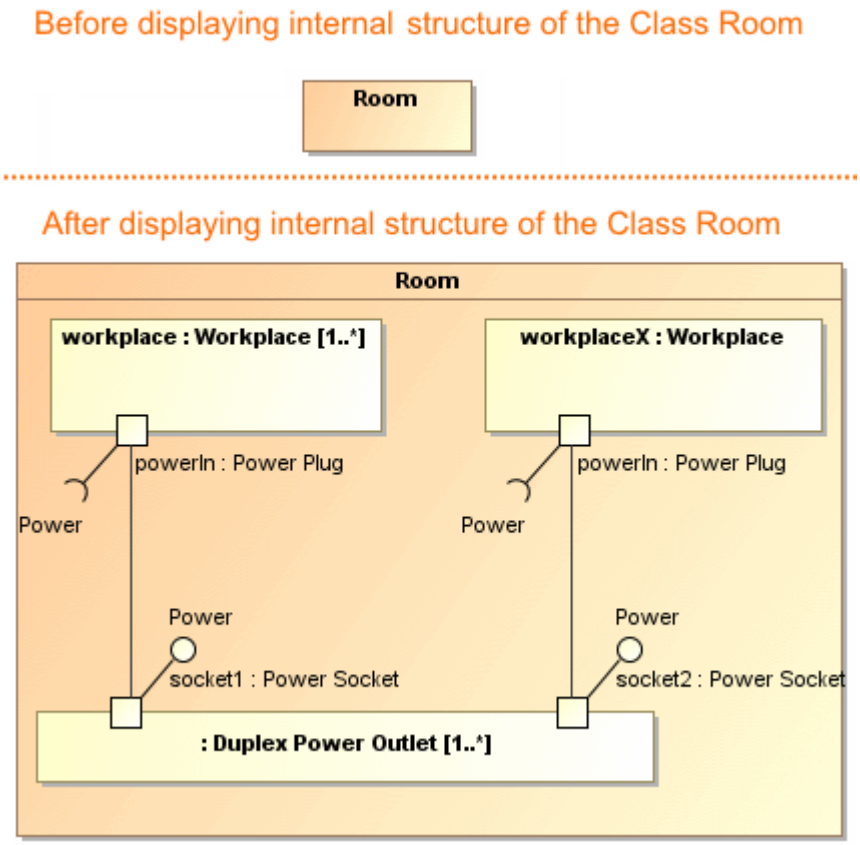
To display the internal structure of a selected classifier

---

1. On the diagram pane, select the classifier shape in which you want to display the internal structure.
2. Do one of the following:
   - Open the selected shape's shortcut menu, click **Display** > **Display Internal Structure**, select the name of the classifier, and then select a diagram containing the internal structure.

   - On the diagram toolbar click , choose the **Display Internal Structure** command, select the name of the classifier, and then select a diagram containing the internal structure.
   After those steps are completed, the internal structure is displayed inside the selected classifier (see figure below).

78https://docs.nomagic.com/download/attachments/55847477/composite%20structure%20diagram.mdzip?api=v2&modificationDate=1478089178031&version=1
79https://docs.nomagic.com/download/attachments/55847477/Vehicle%20parts%20under%20ports.mdzip?api=v2&modificationDate=1490011099593&version=1



Related pages

- Displaying elements(see page 490)

Sample model

The model used in the figures of this page is the **Composite structure diagram** sample model that comes with MagicDraw. To open this sample do one of the following:

- Download composite structure diagram.mdzip[80].
- Find in modeling tool *<modeling tool installation directory>\samples\diagrams\composite structure diagram.mdzip.*

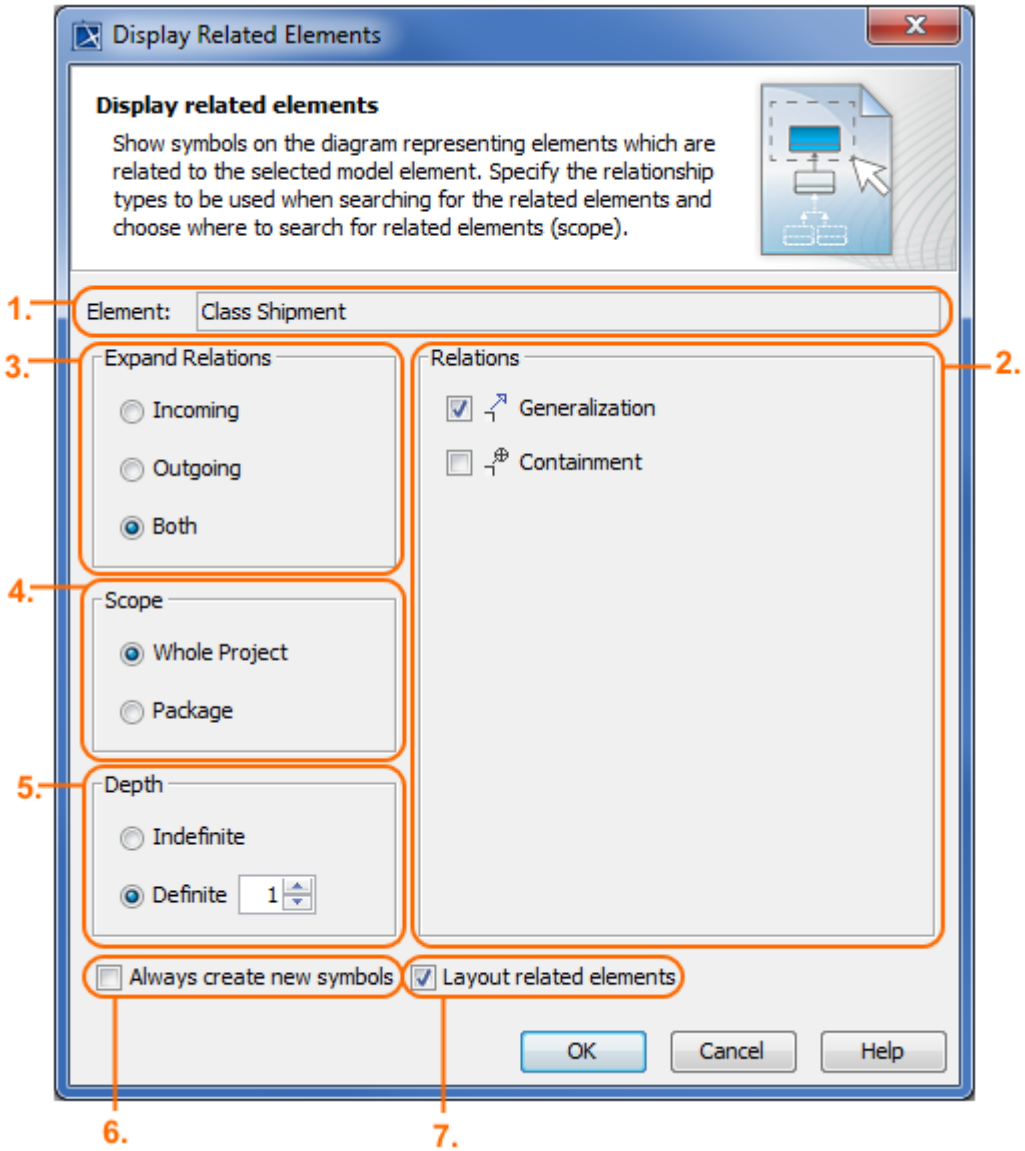
Displaying related elements

When trying to extend your model or visualize additional elements on the diagram pane, you can represent symbols that are related to a selected symbol via relationships.This functionality allows for displaying shapes and paths among those shapes which have already been created in the model.

After selecting the **Display Related Elements** command, the **Display Related Elements** dialog opens. This dialog consists of the following sections (see the figure below):

80https://docs.nomagic.com/download/attachments/55847477/composite%20structure%20diagram.mdzip?api=v2&modificationDate=1478089178031&version=1

- *Element* box (1) shows the name of the selected element shape of which you are going to display its related elements.
- *Relations* area (2) lists all relationship types which connect the selected element shape with other elements.
- *Expand Relations* area (3) allows you to select the direction of the relationships which you want to display on the diagram pane.
- *Scope* area (4) allows you to select the search scope of the elements related to the selected element.
- *Depth* area (5) allows you to specify the search range for the selected relationships.
- Always create new symbols option (6) allows you to create a symbol even if a related element already has a symbol in the diagram.
- *Layout related elements* option (7) lays out new symbols by using the default diagram layout.



The following procedure will guide you through using the **Display Related Elements** command and dialog when you want to represent all related elements via selected relationship types of the selected shape on the diagram pane. The functionality works for the entire diagram, single shapes, or numerous selected shapes.

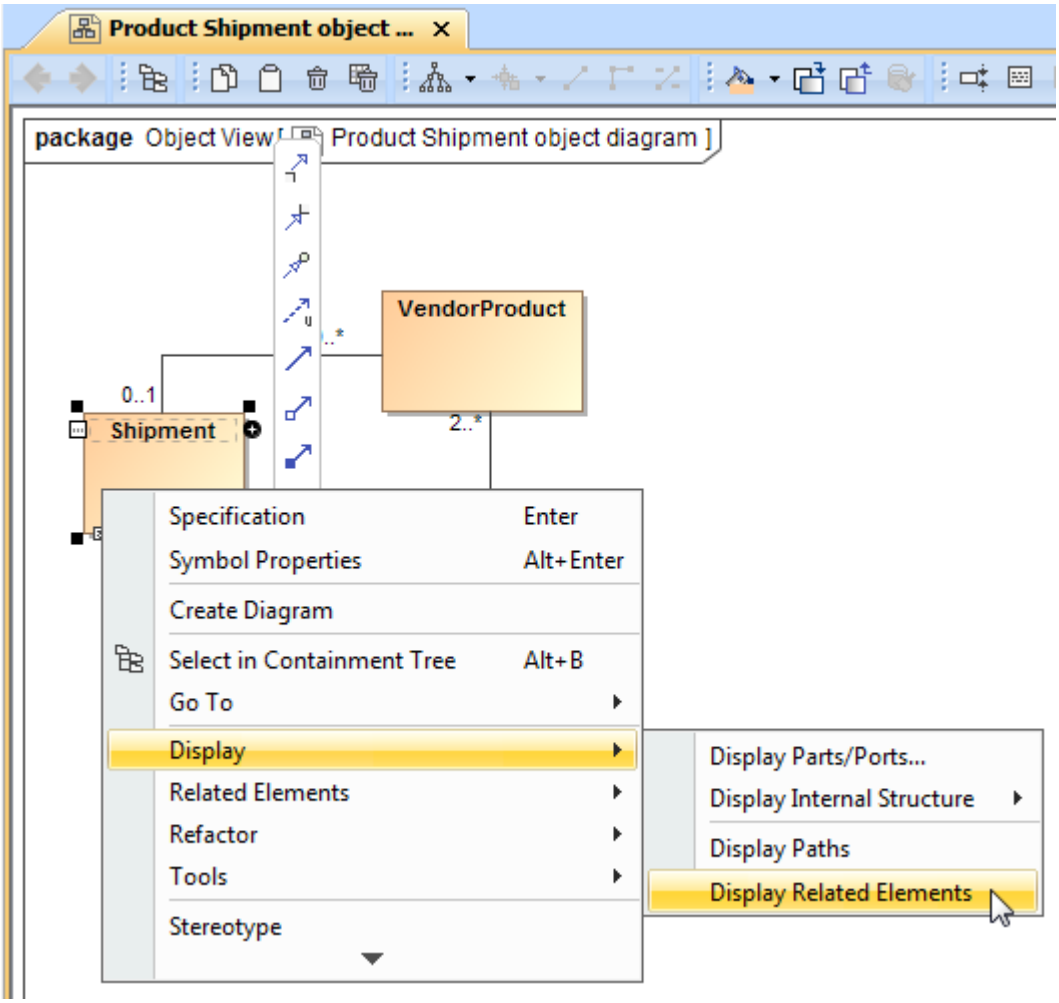
To display related elements

1. Select a diagram frame or any number of shapes on the diagram pane.
2. Open the **Display Related Elements** dialog in one of the following ways:
   - Right-click to open the shortcut menu, and select **Display** > **Display Related Elements**.
   - On the diagram toolbar click , and select **Display Related Elements**.
   - On the main menu, click **Analyze** > **Display Related Elements**.

   > **Important**
   > The **Display Related Elements** dialog cannot be opened if the selected element shape or diagram doesn't contain any related elements.

   Example 1

   A quicker way to display related elements of the selected shape is to use its shortcut menu, and select **Display** > **Display Related Elements**. The example below demonstrates the shortcut menu of the Class named *Shipment*. It is showing the need to display related elements of the *Shipment* Class in the *Product Shipment object diagram*.



3. In the opened dialog, specify:

- In the Relations area (2), select check boxes of the listed relationship types.
- In the Expand Relations area (3), select the direction of the relationships:
  - **Incoming** - in which the element participates as client.
  - **Outgoing** - in which the element participates as supplier.
  - **Both-**in which the element participates as both client and supplier.
- In the Scope area (4), choose the location in which the related elements should be found:
  - **Whole Project** - search in the whole model.
  - **Package** - search in the package under which the selected element is created.
- In the Depth area (5), select the search range for the relationships:
  - **Indefinite** - search for all possible relationships.
  - **Definite** - search for a specified level of relationships.

4. Select the check box of the **Always create new symbols** option to create a symbol even when the related element already has a symbol displayed on the diagram pane.
5. Select the check box of the **Layout related elements** option to layout(see page 579) new symbols by using the default diagram layout(see page 579).

> **Note**
> The position of the selected element shape can be changed after selecting the **Layout related elements** check box.

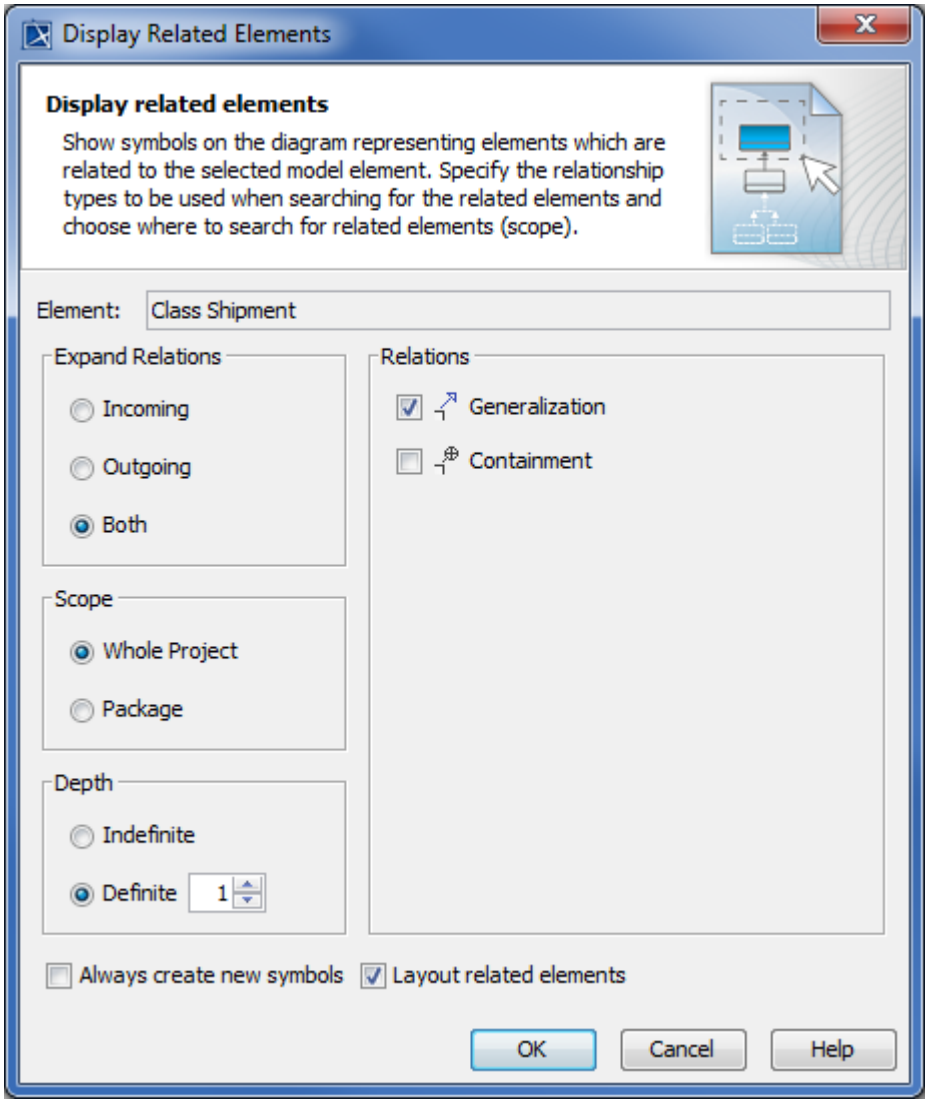
Example 2

For example, if you would like to display all related elements of the *Shipment* Class that are connected through the Generalization relationship, you can find them according to the following criteria:

- The direction of the generalization relation is both incoming and outgoing.
- The related elements to be searched for are in the whole project.
- The search depth of the relationship is one level.

If you want to layout related elements after the display, but don't want to create new symbols which are already displayed in the *Product Shipment object diagram*.

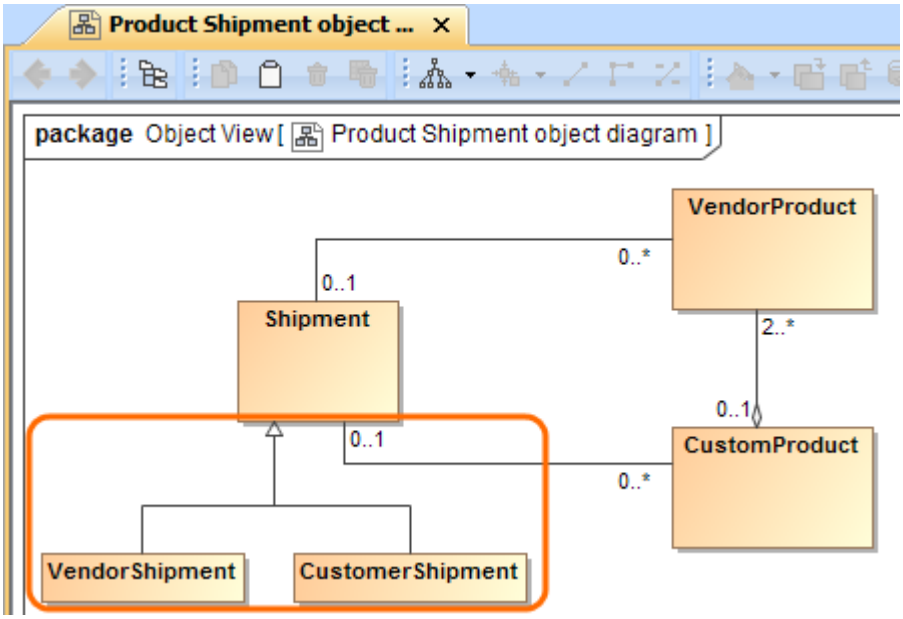
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



6. Click **OK**.
   According to the specified relationship type, scope, depth, direction, layout, and creation options, all related elements of the selected shape are represented as shapes and paths on the diagram pane.

   Example 3

   After opening the **Display Related Elements** dialog (Example 1), and defining the search criteria in it (Example 2), two Classes (*VendorShipment* and *CustomerShipment*) related with the Shipment Class via the Generalization relationship are displayed on the *Product Shipment object diagram* pane. The figure below is the result of following the steps described in Example 1 and Example 2.



Related pages

- Displaying elements(see page 490)
- Layout(see page 579)
- Displaying related elements in the diagram(see page 2289) (Open API)

Sample model

The model used in the figures of this page is the **Inventory Control System** sample model that comes with MagicDraw. To open this sample do one of the following:

- Download Inventory Control System.mdzip.[81]
- Find in modeling tool *<modeling tool installation directory>\samples\case studies\Inventory Control System.mdzip.*

Displaying paths

This functionality allows you to visualize relationships that are created between elements in the model, but are not displayed on the diagram pane. There are two options to display paths among the element shapes:

- Displaying paths only between selected element shapes.(see page 513)
- Displaying all existing paths among the shapes.(see page 514)

To display paths between selected element shapes

81 https://docs.nomagic.com/download/attachments/55847477/Inventory%20Control%20System.mdzip?api=v2&modificationDate=1478093857390&version=1

1. On a diagram pane, select two or more elements.
2. Do one of the following:
   - Right-click to open the shortcut menu, and select **Display** > **Display In-Between Paths**.
   - On the diagram toolbar, click and select **Display In-Between Paths**.
   - On the main menu, click the **Analyze** > **Display In-Between Paths**.

     The **Display In-Between Paths** command is available only when two or more shapes are selected.

   The existing paths between selected element shapes are displayed on the diagram pane.

Warning

The paths won't be displayed if:

- The selected shape doesn't have any relationships with the shapes displayed on the diagram.
- All paths are already displayed on the diagram pane.

Paths to self also will not be displayed.

To display all existing paths of the selected shape

---

1. On a diagram pane, select one or more elements.
2. Do one of the following:
   - Right-click to open the shortcut menu, and select **Display** > **Display All Paths**.
   - On the diagram toolbar, click and select **Display All Paths**.
   - On the main menu, click the **Analyze** > **Display All Paths**.

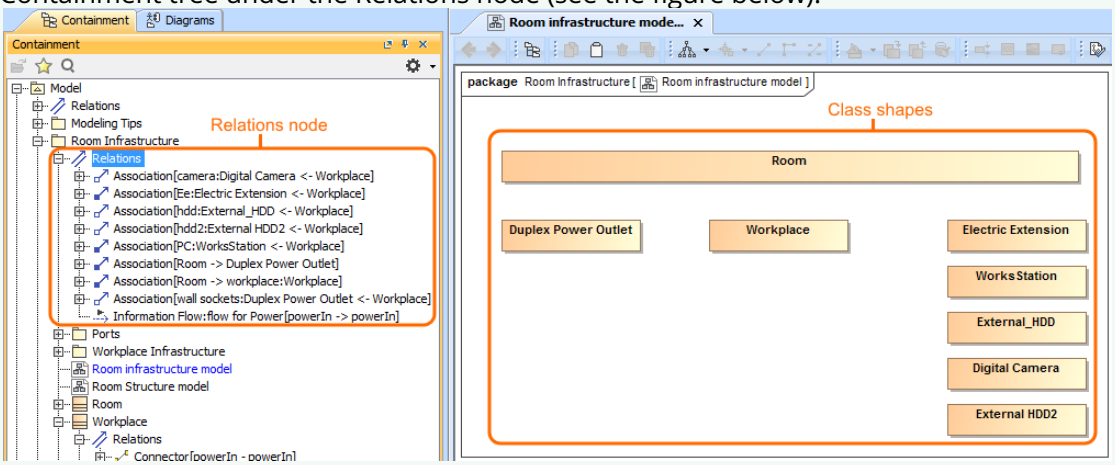
   All the existing paths among the selected shape and other shapes on the diagram pane are displayed. If the element has a path to self, this path will be displayed as well.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

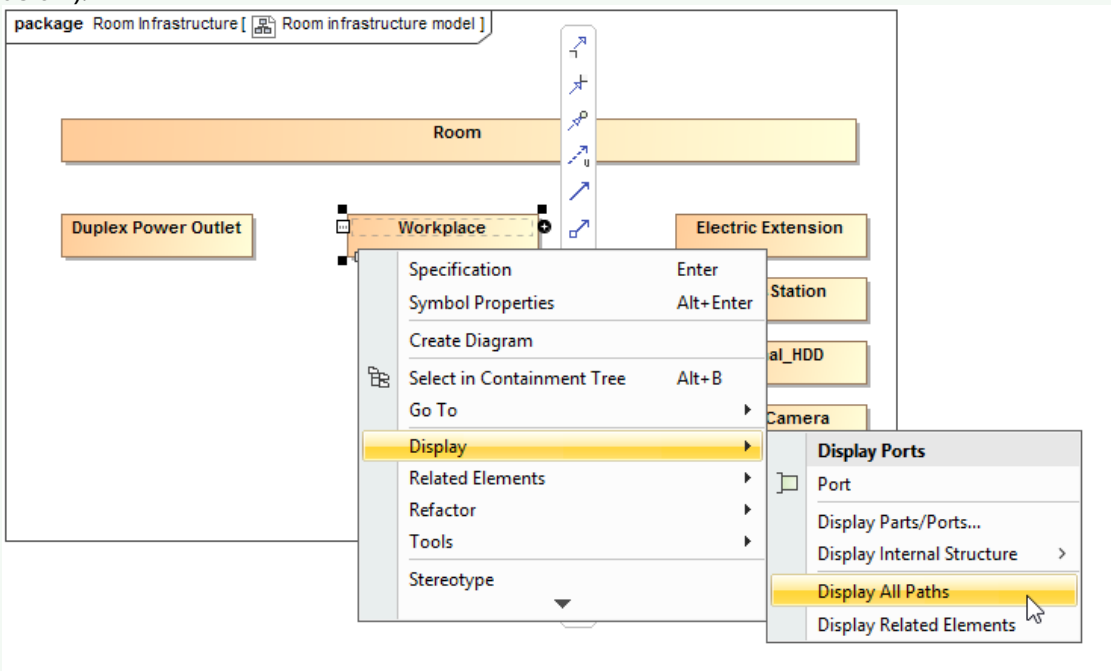
Example of the Display All Paths command

This example presents the workflow on representing the paths between shapes on the diagram pane when they are already created in the model.

The Class shapes without any paths between them are represented in the *Room infrastructure model* Class diagram. In this case, paths between those classes exist in the Containment tree under the Relations node (see the figure below).

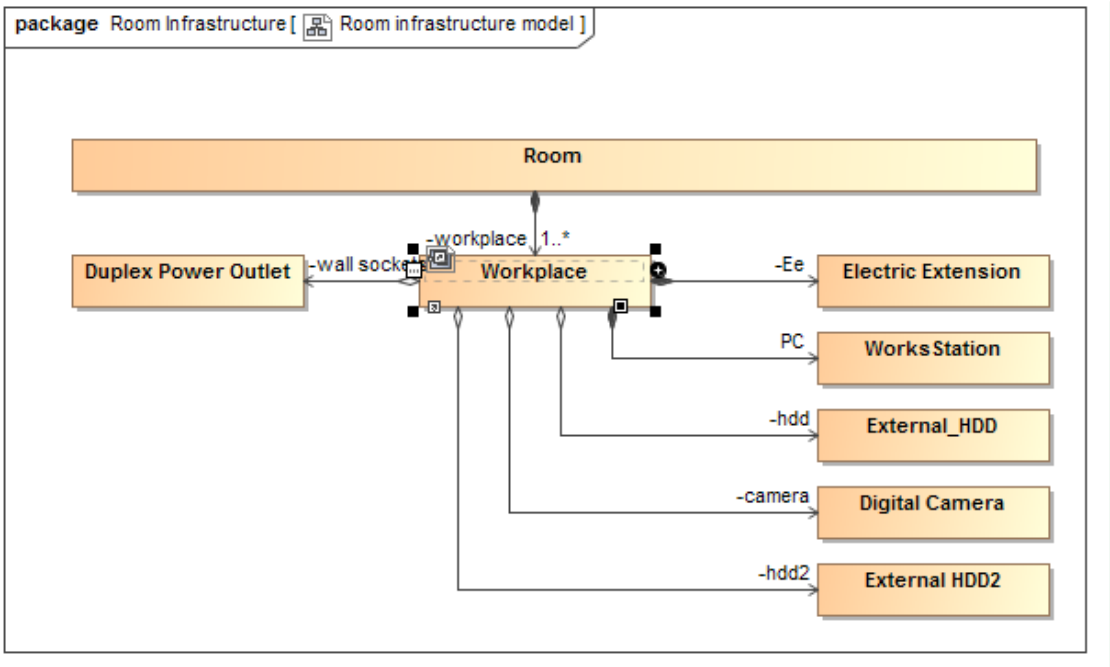


If you want to visualize how the *Workplace* Class is related with other shapes displayed on the diagram pane, you can do so easily by using the **Display All Paths** command. One way to reach this command is from the shortcut menu of the selected shape (see the figure below).



After selecting the **Display All Paths** command, all paths between the *Workplace* Class and other shapes on the diagram pane are displayed (see the figure below). The diagram shows that the *Room* consists of between one and an arbitrary number of *Workplaces*. The *Workplace* consists of *Electric Extension* and *Works Station*. Also, the *Workplace* can contain *Duplex Power Outlet*, *External HDD2*, *Digital Camera*, and *External_HDD*.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



To display paths among element shapes automatically when displaying elements

1. On the main menu, select **Options** > **Environment**.
2. In the **Environment Options** dialog(see page 2310), find the **Display Paths on Element Drop** option and set it's value to *true*.
3. Click **OK**.
4. Select the elements in the Model Browser(see page 29) and drag them on the diagram pane(see page 24).
   The shapes and paths between them are displayed.

> **Warning**
> The paths won't be displayed if:
> - The selected shape doesn't have any relationships with the shapes displayed on the diagram.
> - All paths are already displayed on the diagram pane.

Related pages

- Displaying elements(see page 490)
- Working with paths and relationships(see page 557)

Related page in Developer Guide

- Paths(see page 2279)

Sample model

The model used in the figures of this page is the **Composite structure diagram** sample model that comes with MagicDraw. To open this sample do one of the following:

- Download composite structure diagram.mdzip[82].
- Find in modeling tool *<modeling tool installation directory>\samples\diagrams\composite structure diagram.mdzip.*

Copying and pasting objects

Modeling tools support all common copying/cutting and pasting functions, as well as advanced copying and pasting functions. You can copy and paste objects from one project to another. You can find these options in the **Edit** menu.

Advanced commands for copying:

Command	Shortcut		Description
	Windows	Mac OS	
Copy URL			Copies the URL of the selected project element or element symbol to the clipboard (the project element and the element symbol have different URLs). To copy the URL of an element, select **Copy URL** from the element shortcut menu in the Containment tree. To copy the URL of an element symbol, select the the symbol and click **Edit** > **Copy URL**. You can use URLs to open any elements by clicking the **Open Element from URL** command. The element is then highlighted in the Containment tree or in a diagram. Activating the URL in another application allows you to start MagicDraw, open a project (if possible), and select any elements. You can paste URLs from the clipboard to any MagicDraw diagram. Hyperlinks can also hold URLs of any model elements.
Copy as BMP Image	Ctrl+Shift+B	⌘⇧B	Copies the selected shapes to the clipboard. If no shapes are selected, the active diagram is copied. The .bmp image format provides the best image quality. The image files of this format are typically not compressed, resulting in a large file.

82 http://docs.nomagic.com/download/attachments/9906189/composite%20structure%20diagram.mdzip?version=1&modificationDate=1478089178031&api=v2

Command	Shortcut		Description
Copy as EMF Image	Ctrl+Shift+E	⌘⇧E	Copies the selected shapes to the clipboard. If no shapes are selected, the active diagram is copied. The .emf format image files are smaller, so using this format reduces network traffic. Additionally, the .emf format supports language specific symbols. EMF does not support gradient fill. Because icons of the modeling tool are stored in the SVG format, it may be exported incorrectly. To render SVG icons as raster images in EMF, on the main menu, click **Options** > **Environment** > **General** > **Image Export** and set the **Render SVG Icons as Raster Images in EMF** option to *true.*
Copy as JPG Image	Ctrl+Shift+J	⌘⇧J	Copies the selected shapes to the clipboard. If no shapes are selected, the active diagram is copied. The .jpg format produces relatively small file sizes, but when compressing, the .jpg image file loses data, which results in lower quality. Thus, this format is suitable for final distribution of images, but is not recommended if images are to be edited.
Copy as PNG Image	Ctrl+Shift+P	⌘⇧P	Copies the selected shapes to the clipboard. If no shapes are selected, the active diagram is copied. The .png format is best suited for storing images during the editing process because of its lossless compression. Compared to the .jpg format, the .png excels when the image has large areas of uniform color. It also works well in online viewing applications, such as web browsers.

Advanced commands for pasting:

Command	Shortcut		Description
	Windows	Mac OS	
Paste with New Data	Ctrl+E	⌘E	Pastes one or more copied model elements by creating new model elements. Use this command when copying or cutting owned symbols from one shape to another in a diagram.

Command	Shortcut		Description
Paste Style	Ctrl+Shift+V	⌘⇧V	Pastes symbol properties to the selected symbol. You can copy and paste symbol properties of more than one symbol at the same time. Symbol properties are pasted according to the element type. For example, on a class symbol, the class symbol properties are pasted, and on a package symbol, the package symbol properties are pasted, etc.
Paste as Diagram Overview			Creates a new diagram overview representing selected elements. A diagram overview shape provides the ability to overview other diagrams on a diagram pane. Moreover, using diagram overview shape you can overview dependency matrices, tables, and relation maps.

In addition to the described commands, there you have the ability to copy the whole diagram or the selected shapes and paste them to Microsoft Office or other applications.

Not all symbols can be copied/cut and pasted.

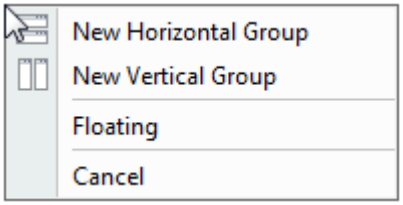
Related pages

- Saving as image(see page 584)
- Diagramming(see page 482)

Using diagram tabs

To split diagrams in new horizontal or vertical group

1. Do one of the following:
   - Right-click the diagram tab. The shortcut menu opens.
   - Select the open diagram tab, then drag it to the diagram pane. The shortcut menu opens.



2. Select New Horizontal Group or New Vertical Group to split diagram pane and have more than one diagram opened at the same time.

To split diagrams, more than one diagram should be open.

To turn the diagram window to floating

- From the diagram tab shortcut menu, select **Floating**.

To show the diagram owner on the diagram tab

1. Select a package and create a diagram in it.
2. From the open diagram tab shortcut menu, select **Show Owner**.

Showing diagrams in Full Screen

If you want to see your diagrams in full screen and work exclusively from the diagram, use the **Show Tabs in Full Screen** functionality. In a full screen mode, all necessary modeling commands are visible with an option to hide, and the Model Browser is in the auto hide mode. You can also set the interface components to be displayed or hidden.

To turn on diagram full screen mode, do one of the following

- Double-click the diagram tab.
- From the diagram shortcut menu, select **Show Tabs in Full Screen**.
- From the **View** menu, select **Show Tabs in Full Screen**.
- Press F11.

> You can change the **Show Tabs in Full Screen** shortcut key in the **Environment Options** dialog, **Keyboard** option group.

You can turn off the diagram full screen mode in the same way as turning it on.

Managing Interface Components in Diagram Full Screen Mode

The Model Browser(see page 29) is in the auto hide mode when the diagram full screen mode is turned on. The Model Browser (see page 29)tabs are located in the left side of the window. To display the browser, move the mouse pointer over the desired browser window tab (for example, on the Containment tab) and the tab opens.

The main toolbar is hidden when the diagram full screen mode is turned on. To display the main toolbar, set the **Hide Toolbars in the Full Screen Mode** property to false in the **Environment Options** dialog, General pane, **Display** option group.

The diagram toolbar is displayed when the diagram full screen mode is turned on. Right-click the diagram toolbar to manage it.

Related pages

- Diagramming(see page 482)

Specifying diagram properties

You can specify your diagram's properties by accessing the diagram's Specification Window(see page 913).

To open the diagram's specification window

- Right - Click on the diagram's window and click on Specification in the drop down menu.

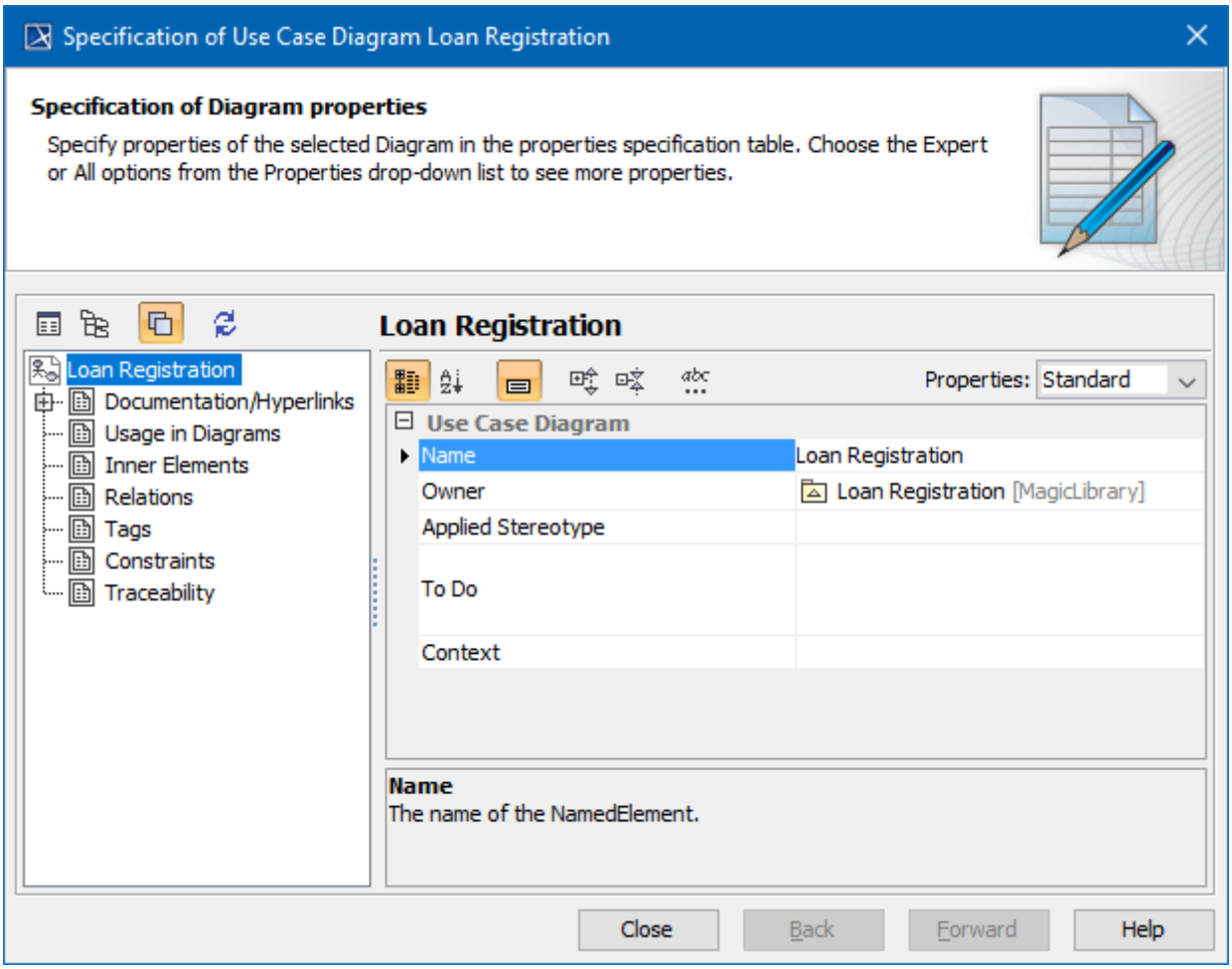


Diagram Specification window.

Related pages

- Specification window(see page 913)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Specifying diagram style properties

In the **Diagram Properties** dialog, you can customize the diagram's style: color, grid, diagram info, font, etc. This allows for your diagram to stand out from other diagrams in your project, or simply just for aesthetic purposes.

To open the **Diagram Properties** dialog

There are multiple ways to bring up the dialog, so do one of the following:

- Select **Diagram Properties** from the diagram shortcut menu.
- Press Shift+Enter.

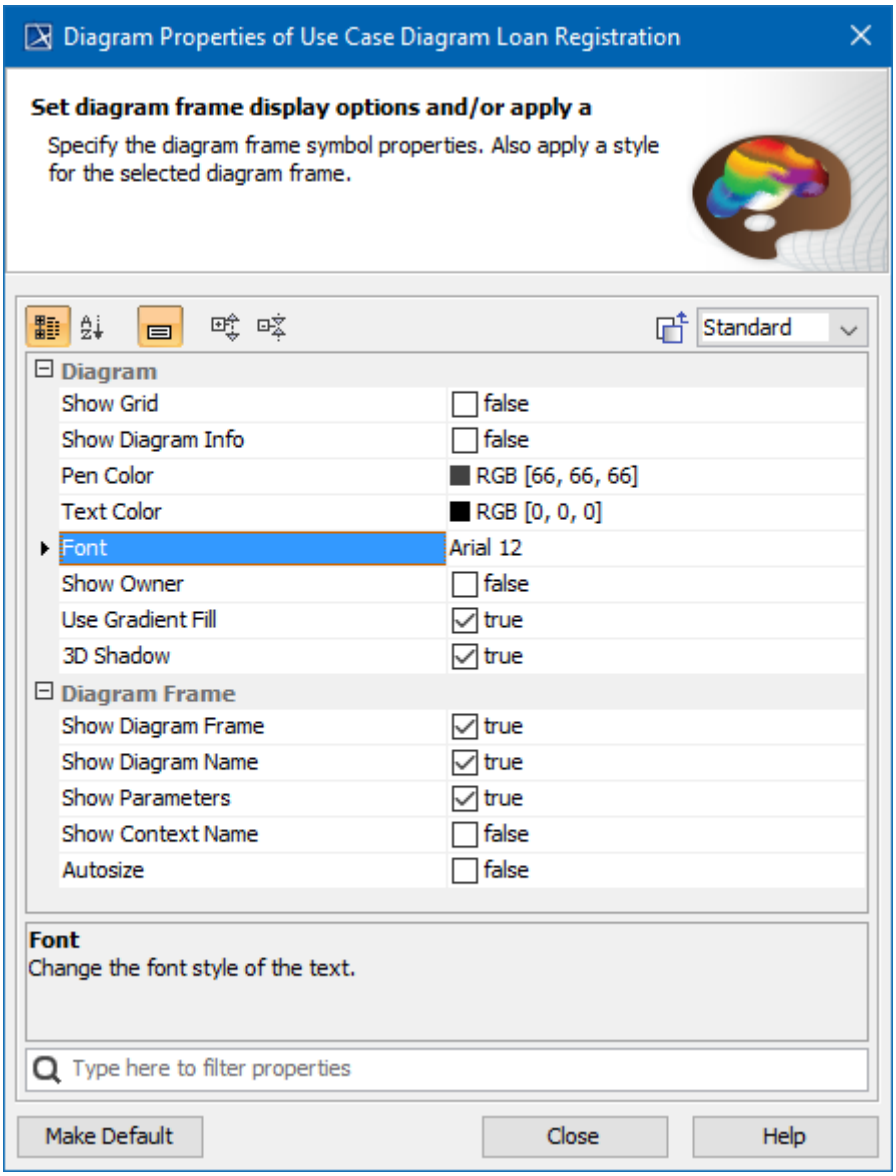


Diagram name and its context name synchronization

The diagram name and its context name are synchronized automatically.

> Create an Activity diagram. Type a name for the Activity diagram, for example, Receive. The name of the Activity automatically changes to Receive. And conversely - change the name of the Activity and the Activity diagram name will be changed automatically. This is synchronization of a diagram name and its context name:
> Synchronization works in the following cases:
> - Activity and Activity diagram inside.
> - Interaction and Communication or Sequence diagram inside.
> - (Protocol) State Machine and (Protocol) State Machine diagram inside.
> - Class and all available inner diagrams inside.

To turn off the synchronization

- In the **Environment** options dialog, **General** > **Editing** options group, cancel the selection of the **Synchronize the diagram name with its contextname** option.

Related pages

- Diagramming(see page 482)
- Customizing environment options(see page 65)

Diagram frame

All UML diagrams have the content area that is restricted by the diagram frame as it is stated in UML specification. The diagram frame is primarily used in cases where the diagrammed element has graphical border elements (like ports for classes and components, entry/exit points on state machines).

By default the diagram frame is displayed on the new diagram pane. The frame is a rectangle in all diagrams, except state machine and activity. State machine and activity diagram frames have rounded corners.

The frame can be re-sized manually by dragging its corners or borders.

To hide the diagram frame, do one of the following

- Right-click the diagram and on the shortcut menu, cancel the selection of the **Show Diagram Frame**.
- Open the **Diagram Properties** dialog and set the **Show Diagram Frame** property to false.

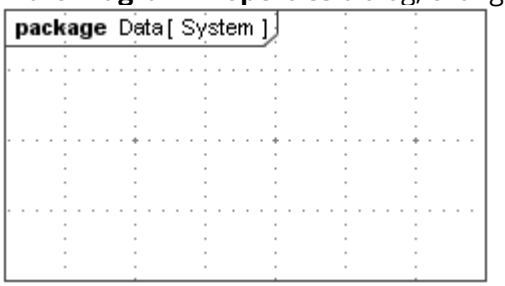
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To change the diagram frame properties, do one of the following

- In the Project Options dialog > Symbol styles > Diagram > Diagram Frame options group, change the desired options.
- In the **Diagram Properties** dialog > **Diagram Frame** property group, change the properties.

To hide the icon on the diagram frame

- In the **Diagram Properties** dialog, change the **Show Stereotypes** property value to **Text**.



When a new diagram is created, the diagram stereotype icon is displayed on the diagram frame header by default, though you can change it to the context stereotype icon.

To display the context stereotype icon on the diagram frame

- In the **Diagram Properties** dialog > **Diagram Frame** property group, change the **Use Stereotype** property value to **Context**.

There is a possibility to show the abbreviation of a diagram type instead of the full diagram type in the diagram frame header.

To display the abbreviated diagram type

- In the **Diagram Properties** dialog > **Diagram Frame** property group, select the **Show Abbreviated Type** check box.

The abbreviated diagram types are listed in the following table.

Diagram name	Abbreviation
Use Case	uc
Communication	comm
Sequence	sd
State Machine	stm
Activity	sct
Composite Structure	cs

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related pages

- Diagramming(see page 482)
- Note(see page 791)

Drawing diagram shapes

You can draw diagram shapes in any type of diagram.

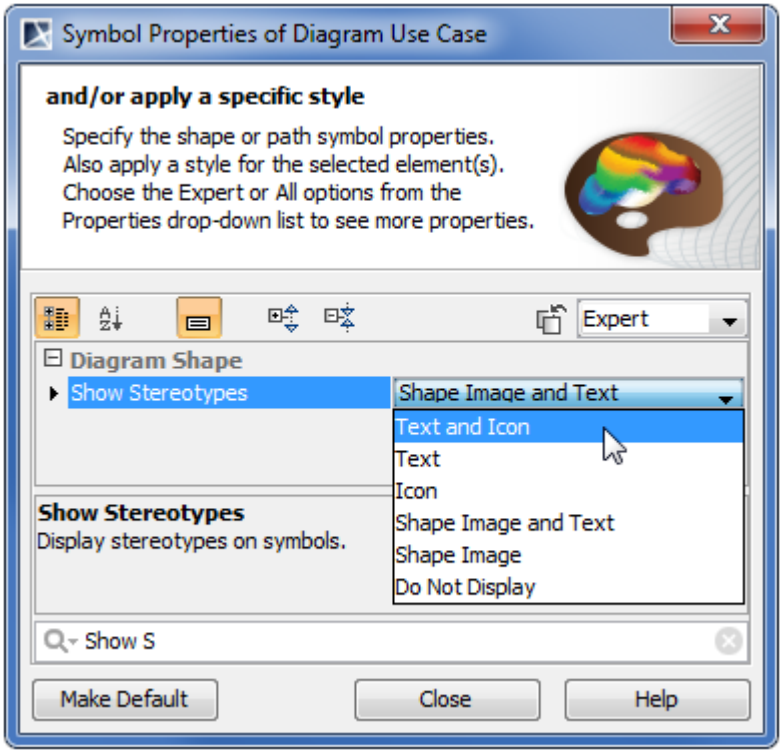
To display a diagram shape on a diagram pane

1. In the Model Browser(see page 29), select the diagram.
2. Drag the diagram to the diagram pane.

You can change the representation of the diagram shape by changing the diagram stereotype display mode.

To change the representation of the diagram shape

1. From the diagram shape's shortcut menu, select **Symbol Properties**.
2. In the **Symbol Properties** dialog(see page 975), select a new value for the **Show Stereotypes** property.



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

You can also select wether to show or hide the diagram constraints or tagged values on the diagram shape.

To show diagram stereotype constraints and tagged values on the diagram shape

1. From the diagram shape's shortcut menu, select **Symbol Properties**.
2. In the **Symbol Properties** dialog(see page 975), select the **Show Constraints** and **Show Tagged Values** check boxes.

To display the context stereotype icon instead of the diagram stereotype icon on the diagram shape

1. From the diagram shape's shortcut menu, select **Symbol Properties**.
2. In the **Symbol Properties** dialog(see page 975), change the **Use Stereotype** property value to **Context**.

The **Use Stereotype** property takes effect only in case the **Show Stereotypes** property value is **Text and Icon** or **Icon**.

There is a possibility to show the abbreviation (see page 523)of a diagram type instead of the full diagram type on the diagram shape.

To display the abbreviated diagram type

- From the diagram shape shortcut menu, open the **Symbol Properties** dialog(see page 975) and select the **Show Abbreviated Type** check box.

Related pages

- Diagramming(see page 482)

Overviewing other diagrams

This feature is available in Standard, Professional, Architect, and Enterprise editions.

You can also learn about overviewing diagrams while analyzing the Diagram overview sample. To open the sample, do one of the following:
- On the Welcome screen, select **Samples** and then in the **Product Features** section click **Diagram overview**.
- Go to *<MagicDraw installation directory>\samples\product features* and open the diagram *overview.mdzip* file.

As of version 17.0 you can overview other diagrams, including dependency matrices, tables, and relation maps, on a diagram pane. Use the diagram overview shape in order to do that.

Read and learn how to use the diagram overview shape in the following sections:

How to...

This section gives some remarks that can be useful when struggling to arrange sizes of your diagram overview shapes.

How to make a couple of diagram overview shapes to be of the same size?

Let's say you have two diagram overview shapes. You want their zoom ratio to be 1:1 and shape areas to be of the same height (or width) at the same time.

Solution

- For both diagram overview shapes set the **Fit Content to the Shape Area** property value to false.
- For both diagram overview shapes set the **Autosize** property value to false.
- Make both diagram overview shapes of preferred size.
- Make both diagram overview shapes of the same height (or width) by dragging corners of the shape areas.

How to set desired shape sizes in the a diagram overview shape?

Solution

- Set the **Fit Content to the Shape Area** property value to true for this diagram overview shape.
- Set the desired shape content size by dragging corners of the shape area.
- Set the **Fit Content to the Shape Area** property value to false for this diagram overview shape.

In case the diagram overview shape shows the full content of the corresponding diagram, all new shapes added to the diagram pane in the diagram overview shape will be of the previously set size.

If you want to reset the shape sizes to ratio zoom 1:1, do one of the following:

- Set for the diagram overview shape the **Autosize** property value to *true*.
- Make the diagram overview shape of the preferred size.

Related pages

- Creating a diagram overview shape(see page 527)
- Modifying the content of a diagram overview shape(see page 530)
- Diagramming(see page 482)

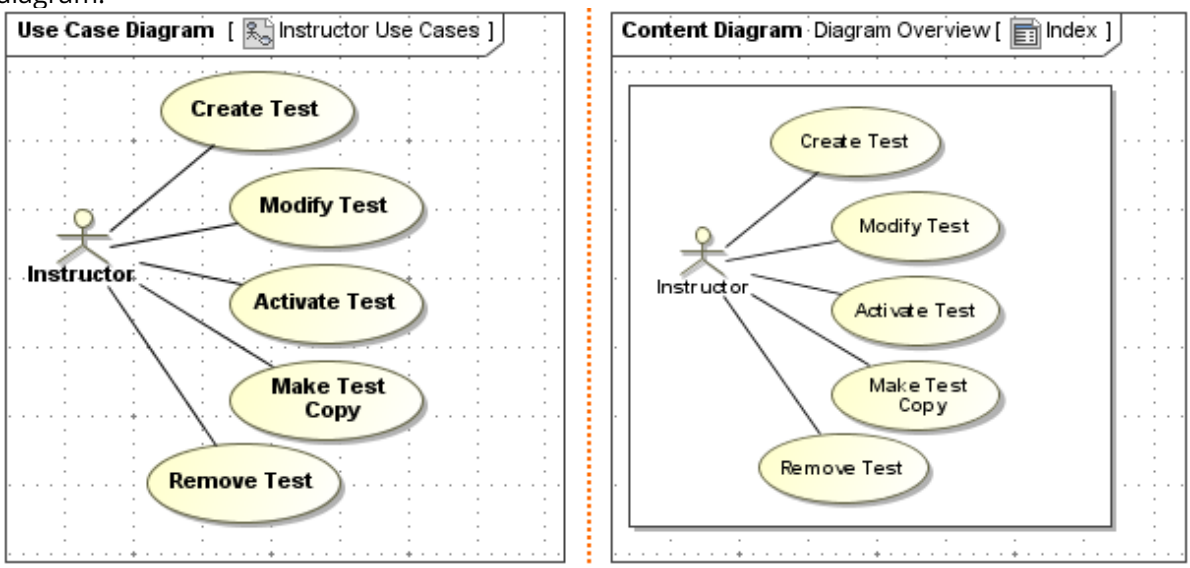
Creating a diagram overview shape

You can use one of the following ways to create a diagram overview shape:

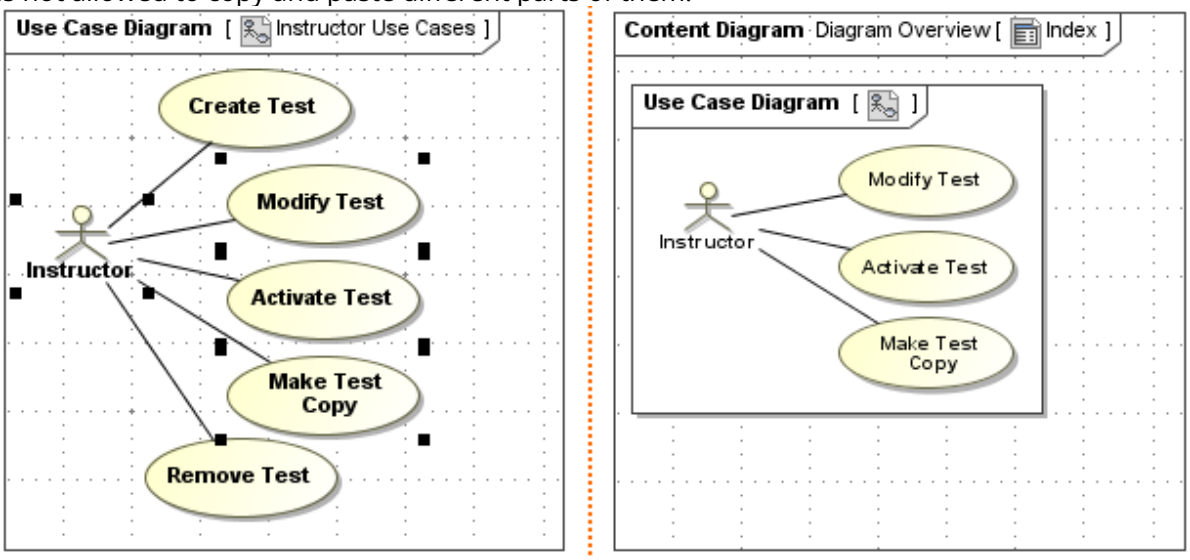
- **Drag the diagram shape from the Model Browser and change it into the diagram overview shape afterwards**. This way created diagram overview shape shows the full content of the corresponding diagram and is updated automatically according to all changes made in this

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

diagram.



- **Use the Diagram Overview button that is located in the diagram palette**. This way created diagram overview shape shows the full content of the corresponding diagram and is updated automatically according to all changes made in this diagram.
- **Paste a copied part of the diagram's content using a special command from the main menu**. This way created diagram overview shape shows any copied part of the corresponding diagram's content and is updated automatically according only to *the changes made in this copied part*. Note that this way cannot be used for dependency matrices, tables, and relation maps, as it is not allowed to copy and paste different parts of them.

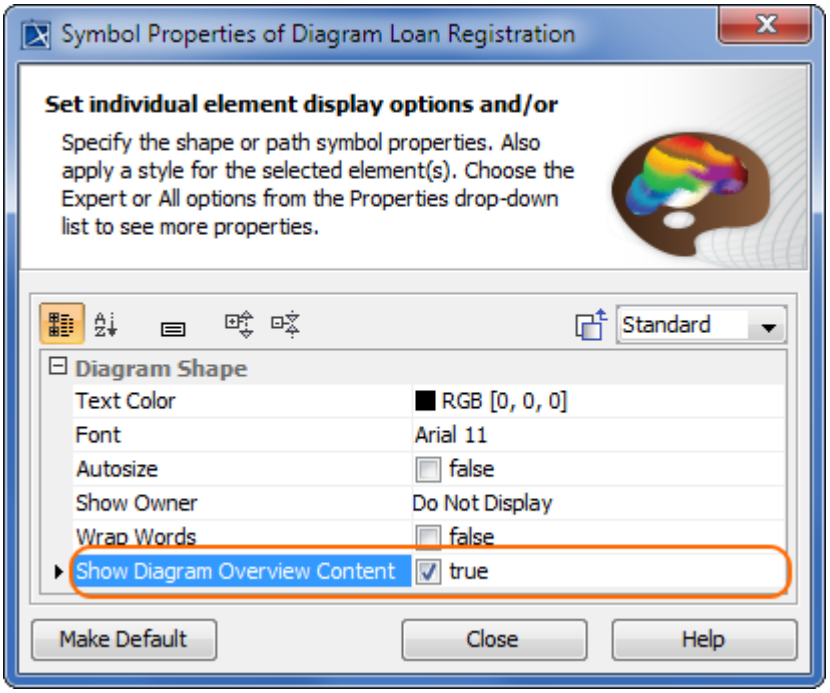


To create a diagram overview shape by dragging it to the diagram pane

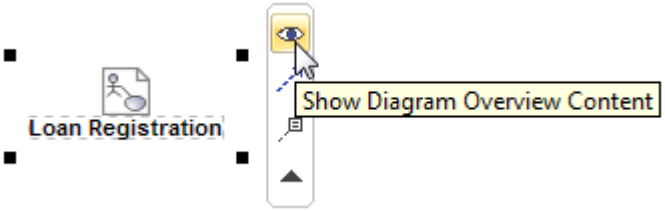
1. Open a diagram wherein you want to create the diagram overview shape.
2. In the Model Browser(see page 29) select a diagram you want to overview.
3. Drag the diagram shape to the open diagram pane.

> ⚠ If you cannot drag the diagram shape to the open diagram pane when working with a teamwork project, make sure you have the right to edit model of this project and then try to lock for edit this diagram.

4. Make the diagram shape show the full content of the corresponding diagram by doing one of the following:
   - Right-click the diagram shape and from the shortcut menu select Symbol Properties(see page 975). In the open dialog, set the **Show Diagram Overview Content** property value to *true*.



   - Click the diagram shape and on the smart manipulator toolbar select the button.



   The diagram shape will be changed to the diagram overview shape that shows the full content of the corresponding diagram.

   > The content of the diagram overview shape will be updated automatically according to *all changes* made in the corresponding diagram.

   > Double-click the diagram overview shape to open the corresponding diagram.

To create a diagram overview shape by using the diagram palette

---

1. Open a diagram wherein you want to create the diagram overview shape.
2. On the diagram palette, click the **Diagram Overview** button that is located within the **Common** buttons group. The **Select Diagram** dialog is open.

   > If the diagram palette is inactive within a teamwork project, make sure you have the right to edit model of this project and then try to lock for edit this diagram.

3. Select a diagram for which you want create a diagram overview shape and click **OK**.
4. Click a free space of the diagram pane.

To create a diagram overview shape by pasting any part of diagram's content

1. Select the element shapes that you want to appear in the diagram overview shape and then copy them.
2. Open the diagram wherein you want to create the diagram overview shape.
3. From the **Edit** menu, select **Paste as Diagram Overview**.

> ⚠ The content will be updated automatically according to the changes made in the copied part of the corresponding diagram only.

Related pages

- Modifying the content of a diagram overview shape(see page 527)

Modifying the content of a diagram overview shape

Once the diagram overview shape is created, you can modify its content: add more element shapes or remove no more needed ones by using the **Compartment Edit** dialog.

To open the **Compartment Edit** dialog, do one of the following

- From the diagram overview shape shortcut menu, select **Edit Compartments**.
- Select the diagram overview shape, click the "..." Compartments button and from the menu, select **Edit Compartments**.

To modify the content of the diagram over view shape

1. In the **Compartment Edit** dialog, use the **>**, **>>**, **<**, and **<<** buttons to move the items among the **Hidden** and **Selected** lists.
2. Click **OK** when you are finished.

Related pages

- Creating a diagram overview shape(see page 527)

Legends

A Legend is a model element allowing to define the styles of diagram symbols and table rows. It helps you to visually group model elements according to specified criteria. Use Legends to increase the readability of diagram and table contents by highlighting important information, e.g., specific element property values or statuses. In Addition, the styles applied to model elements are dynamically updated when you modify your model, so all changes are instantly visible in diagrams and tables where Legends are used.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

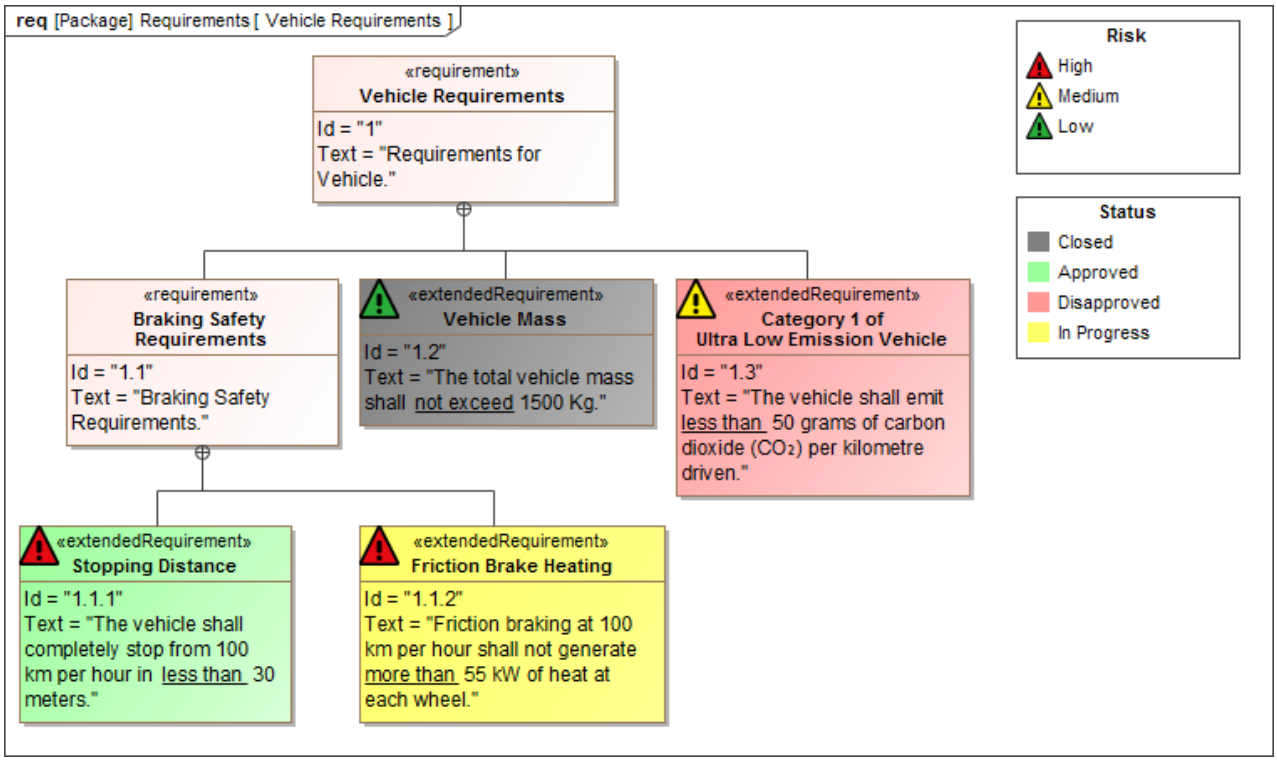
Sorry, the widget is not supported in this export.
But you can reach it using the following URL:

https://www.youtube.com/watch?v=ShM8jQL8mes

Tips for working with Legends

- You can apply multiple Legend Items to a single model element.
- You can reuse Legends in different diagrams and tables of a model, as well as in different models.

For more information about working with Legends, see the following pages:



The Risk and Status legends used in a Requirement Diagram.

Related pages

- Diagramming(see page 482)
  - Opening diagrams(see page 485)
  - Creating diagrams(see page 483)
  - Drawing diagram shapes(see page 525)
  - Working with symbols of model elements(see page 555)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Working with paths and relationships(see page 557)
- Smart manipulation(see page 567)
- Compartments(see page 569)

Creating Legends

On this page

- Creating Legends in a symbol diagram(see page 532)
- Creating Legends in a table(see page 533)

You can create a Legend right in a diagram or table where you want to use it. After creating a Legend, you also have to create its Legend Items(see page 534). If you want Legend Items to be created and specified for you, create Legends automatically(see page 547).

Creating Legends in a symbol diagram

When creating a Legend in a symbol diagram, you can use either the diagram palette, or the diagram tool bar, as described below.

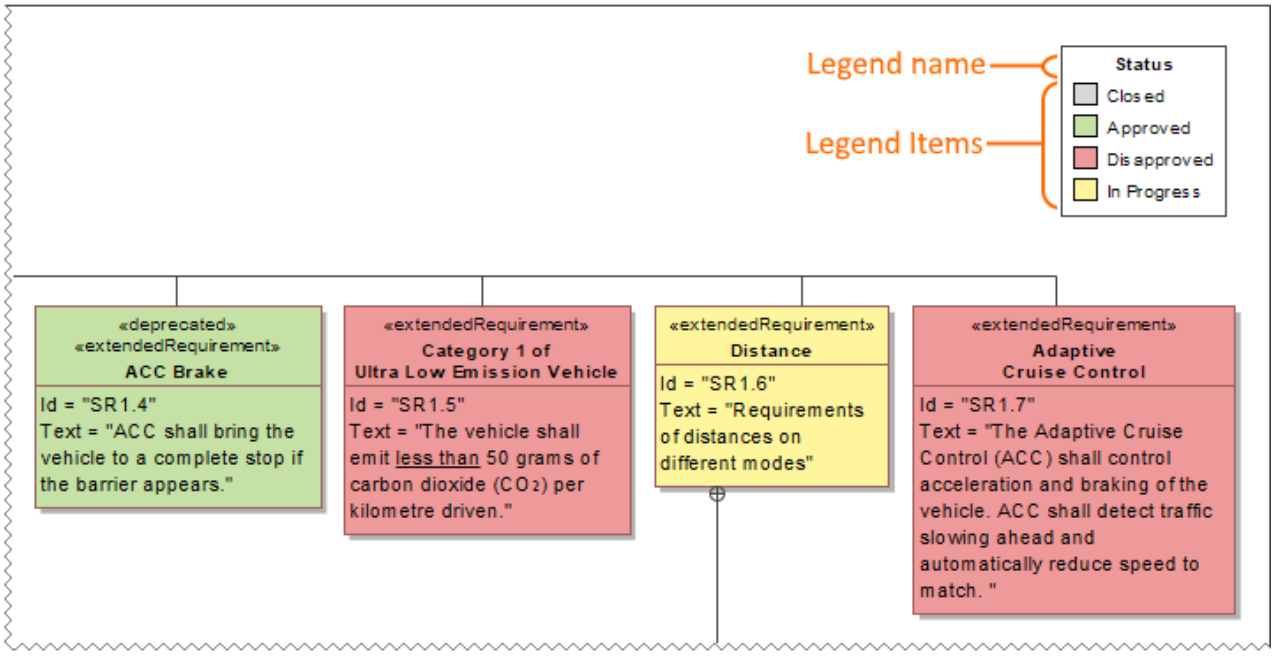
To create a Legend in a symbol diagram

---

1. Open the diagram in which you want to create a Legend.
2. Do one of the following:
   - Under the **Common** group of the diagram palette, click **Legend**, and click an empty space on the diagram pane.
   - Click in the diagram toolbar and select **Create Legend**.

After creating a Legend, use the smart manipulator on the Legend shape to create Legend Items(see page 534).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



A Legend shape with specified Legend Items in a Requirement Diagram.

Creating Legends in a table

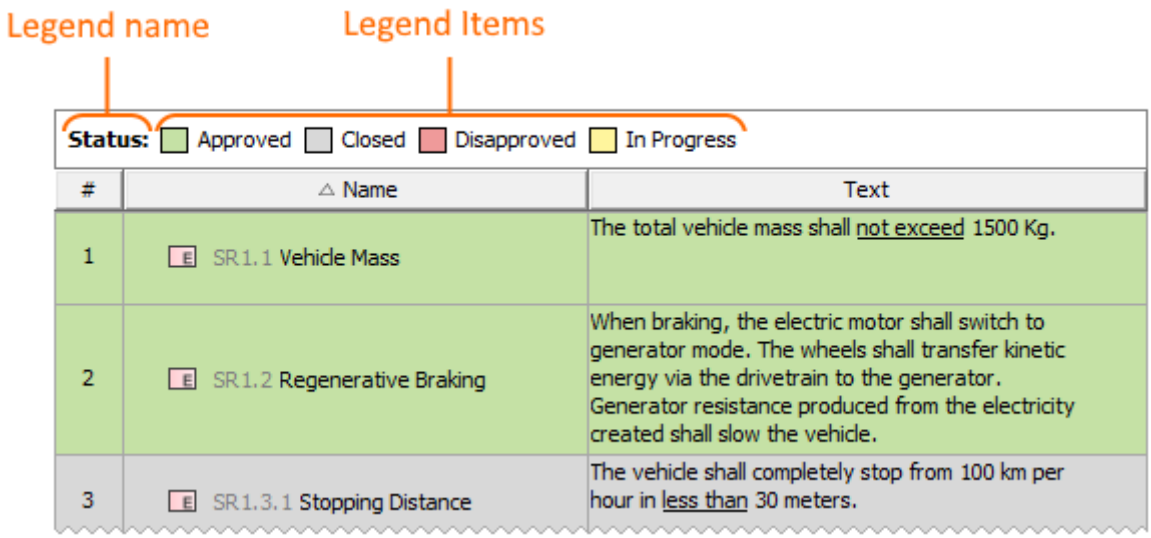
When working with a table, you can create a Legend by using the table toolbar, as described below.

To create a Legend in a table

1. Open the table in which you want to create a Legend.
2. Click ▾ in the table toolbar and select **Create Legend**.

In a table, a Legend is displayed as a bar right above the table. After creating a Legend in a table, the Specification window of the newly created Legend opens allowing you to create Legend Items(see page 534).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



A Legend in a table with specified Legend Items.

Sample model

The model used in the figures of this page is the *Legends* sample model that comes with SysML plugin. To open this model do one of the following:

- Download legends.mdzip[83].
- Open the model from the *<modeling tool installation directory>\samples\SysML directory*.

Related pages

- Legends(see page 530)
  - Creating Legends automatically(see page 547)
  - Creating and managing Legend Items(see page 534)
  - Displaying and hiding Legends(see page 549)

Creating and managing Legend Items

On this page

- Creating Legend Items(see page 535)
- Managing Legend Items(see page 536)

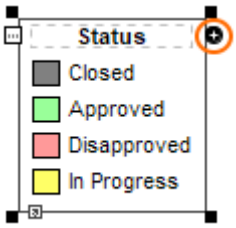
After creating a Legend(see page 532), you must create and specify its Legend Items. Legend Items define different styles used in a diagram or table in a Legend.

83 https://docs.nomagic.com/download/attachments/55847477/legends.mdzip?api=v2&modificationDate=1488812669828&version=1

You can create Legend Items directly from a Legend shape or in the Specification window of a Legend. If you are working with a Legend in a symbol diagram, you can use both methods. However, if you are working with a Legend in a table, you can create Legend Items only in the Specification window of the Legend.

To create a Legend Item from a Legend shape

1. In a diagram, select the shape of the Legend for which you want to create a Legend Item.
2. Click the smart manipulator on the top right corner of the Legend shape, highlighted in the figure below.



3. In the Specification window(see page 913) of the newly created Legend Item, specify its properties(see page 537).
4. Close the Specification window.

The newly created Legend Item is displayed in the Legend shape and applied to diagram symbols according to the properties you specified in the **Applied To** property group(see page 542).

To create a Legend Item in the Specification window of a Legend

1. Open the Specification window of the Legend for which you want to create a Legend Item by doing one of the following:
   - In a symbol diagram, double-click the shape of the desired Legend.
   - In a table, hover the mouse pointer over the desired Legend, click ... and select **Specification**.
2. Click the **Legend Items** group on the left side the Specification window.
3. Click the **Create** button.
4. In the open Specification window of the newly created Legend Item, specify its properties(see page 537).
5. Close the Specification window.

The newly created Legend Item is displayed in the Legend and applied to diagram symbols or table rows according to the properties you specified in **Applied To** property group(see page 542).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Managing Legend Items

The **Legend Items** group in the Specification window of a Legend allows you to create new Legend Items and modify or reorder the existing ones.

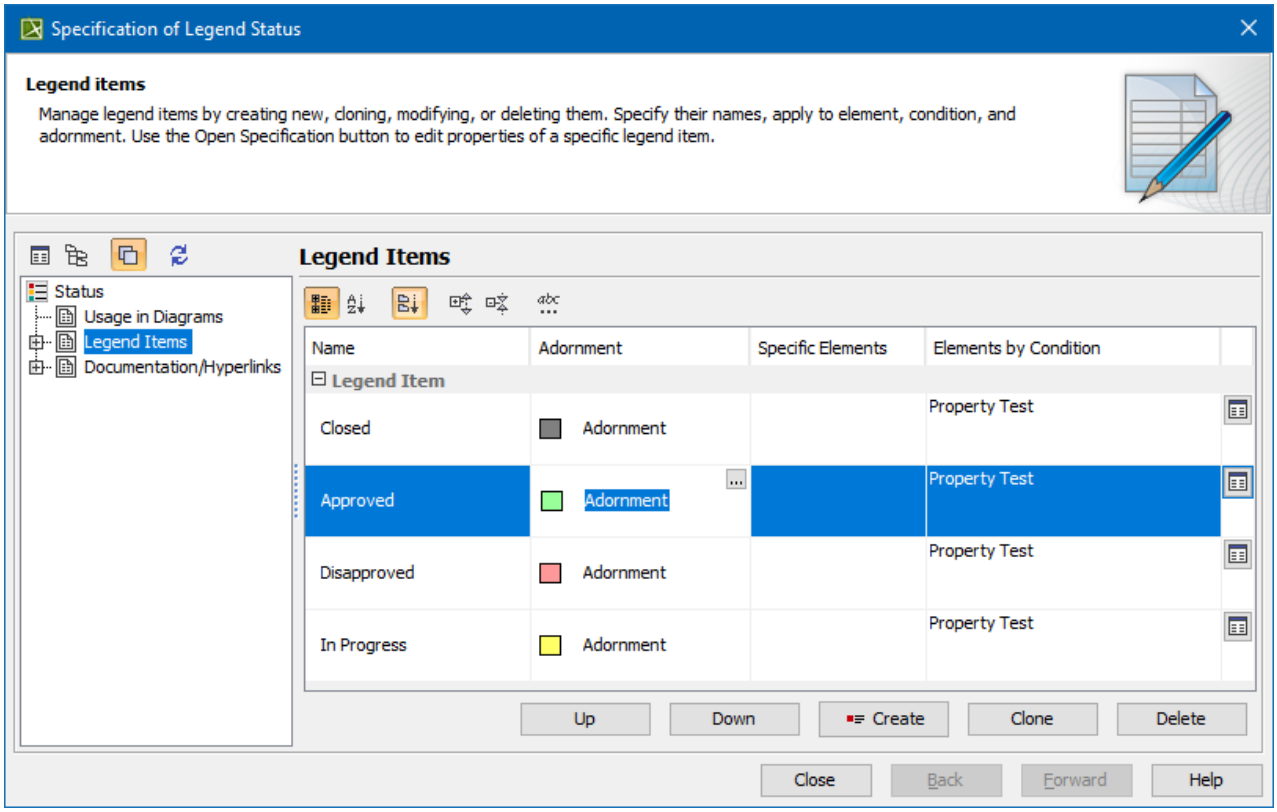
To access the Legend Item management area

1. Open the Specification window of a Legend.
2. Click the **Legend Items** group on the left side the Specification window.

The Legend Item management area is displayed on the right side of the Specification window. The following table explains how to use the buttons found there to create and modify Legend Items.

Button name	Description
Up	Select a Legend Item and click the button to move the Legend Item one position up in the list.
Down	Select a Legend Item and click the button to move the Legend Item one position down in the list.
Create	Click the button to create a Legend Item. The Specification window of the newly created Legend Item opens automatically, allowing you to specify its properties.
Clone	Select a Legend Item and click the button to create its copy. The Specification window of the newly created Legend Item clone opens automatically, allowing you to modify its properties. By default, the clone inherits all property values of the Legend Item from which it was cloned.
Delete	Select a Legend Item and click the button to remove it.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Legend Item management area in the Specification window of a Legend.

Alternative method to manage Legend Items

You can also manage Legend Items in the **Legend Items** property specification cell:

1. Open the Specification window of a Legend.
2. Select the specification cell of the **Legend Items** property.
3. Complete one of the following actions:
   - To create a Legend Item, click ⊞.
   - To remove a Legend Item, select the Legend Item you want to remove and click ⊟.
   - To reorder Legend Items, click ⇅ and in the open dialog use the **Up** and **Downd** buttons to change the position of the selected Legend Item.

Related pages

- Legends(see page 530)
  - Creating Legends(see page 532)
  - Creating Legends automatically(see page 547)
  - Displaying and hiding Legends(see page 549)
  - Specifying Legend Items(see page 537)

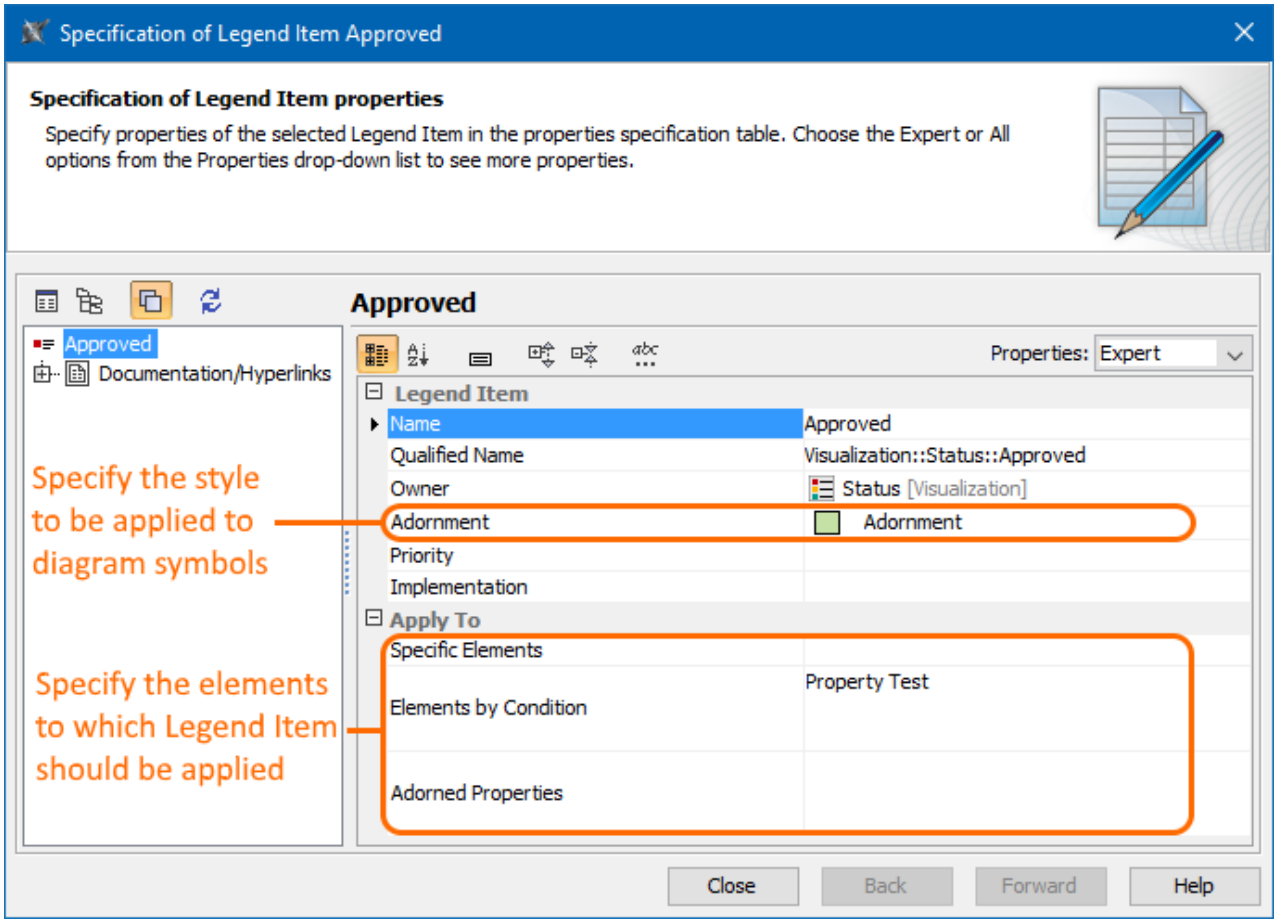
Specifying Legend Items

When you create a new Legend Item(see page 534), you have to specify its name, adornment and application properties. The following pages explain how to do it:

- Specifying Legend Item adornment(see page 538)
-

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Applying Legend Items(see page 542)



The Specification window of a Legend Item.

Related pages

- Legends(see page 530)
  - Creating Legends(see page 532)
  - Creating Legends automatically(see page 547)
  - Displaying and hiding Legends(see page 549)
  - Creating and managing Legend Items(see page 534)

Specifying Legend Item adornment

On this page

- How to avoid conflicting Legend Items(see page 540)

Each Legend Item of a Legend should have a unique style. This style is defined by specifying Legend Item adornment properties. After creating(see page 532) or displaying(see page 549) a Legend in a diagram or table, Legend Item adornment (style) is applied(see page 542) to the symbols or table rows of

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

corresponding elements and represented as Legend Item icon inside the Legend. There are a wide range of Legend Item adornment properties including:

- Fill color
- Text color
- Shape or path line color, style and width
- Icon and icon position on shape or path

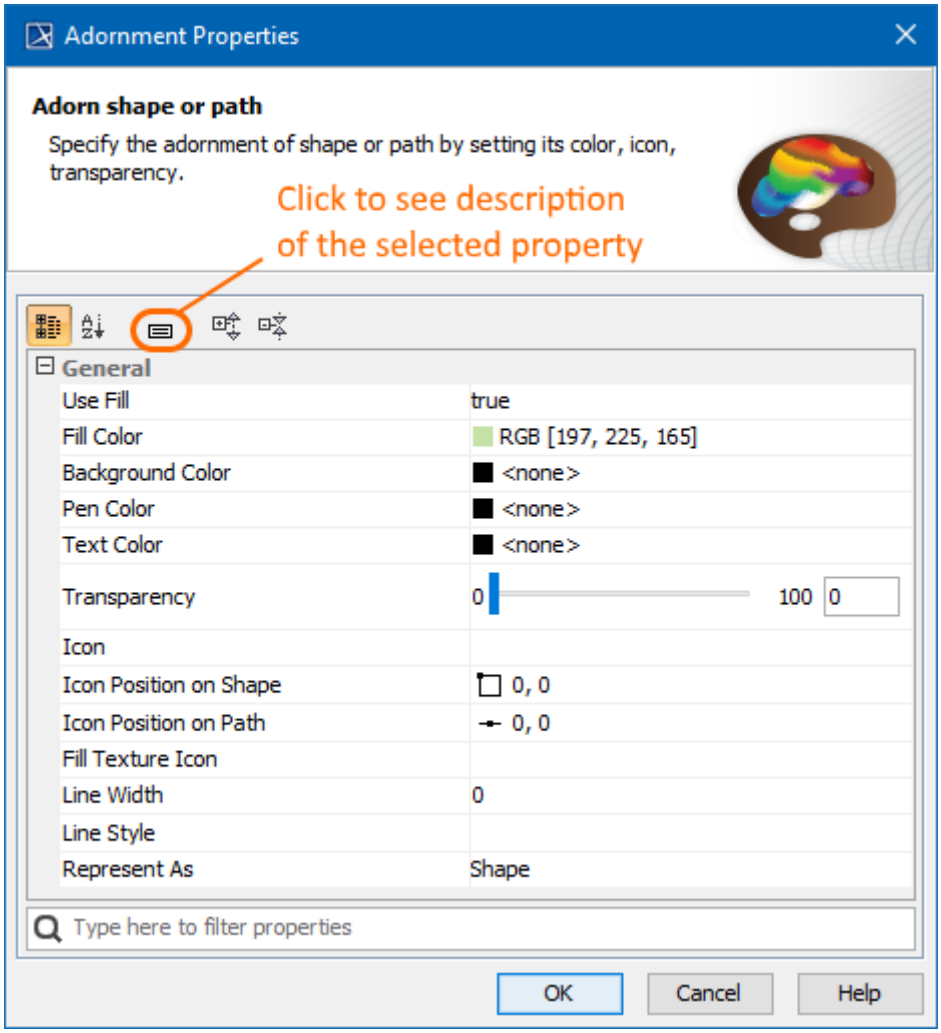
Table adornment
If you specify Legend Item adornment for the Legend that is applied to a table, you should only specify the **Fill Color** property. Other adornment properties, e.g., **Icon**, **Pen Color** or **Line Style**, are not applied to table rows.

You can specify the adornment properties of a Legend Item in the **Adornment Properties** dialog.

To access the adornment properties of a Legend Item

1. Open the Specification window of a Legend Item.
2. Click the specification cell of the **Adornment** property, and then click ... .
3. In the **Adornment Properties** dialog, specify the desired adornment properties.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Adornment Properties dialog.

How to avoid conflicting Legend Items

If you use multiple Legends in one diagram, you should specify a different set of adornment properties for Legend Items of each Legend. This way Legend Items of different Legends do not conflict when applied to the same element. For example, try using **Fill Color** in one Legend, and **Icon** in another. If you want to use the **Icon** property in multiple Legends, then specify different **Icon Position on Shape**/ **Path** for Legend Items of each Legend. I addition, you can define Legend Item adornment by specifying the **Pen Color**, **Line Width** and **Line Style** properties describing the appearance of a shape border or path.

Legend Item priority

If you can not avoid conflicting Legend Items in a diagram or table, specify the **Priorit**y property for each Legend Item. It defines which Legend Item should be applied in case several Legend Items conflict.

Example

Let's say you want to use two legends in a Requirements Diagram: the *Status* Legend indicating the status of each Requirement and the *Risk* Legend showing the risk level of Requirements. It is likely that Legend Items from both Legends will be applied to the same Requirement. For example, a single Requirement can have both status and risk values specified. Let's analyse this specific use case by creating the Legends mentioned above and defining their Legend Item adornment properties so that Legend Items would not conflict when applied to the same Requirement:

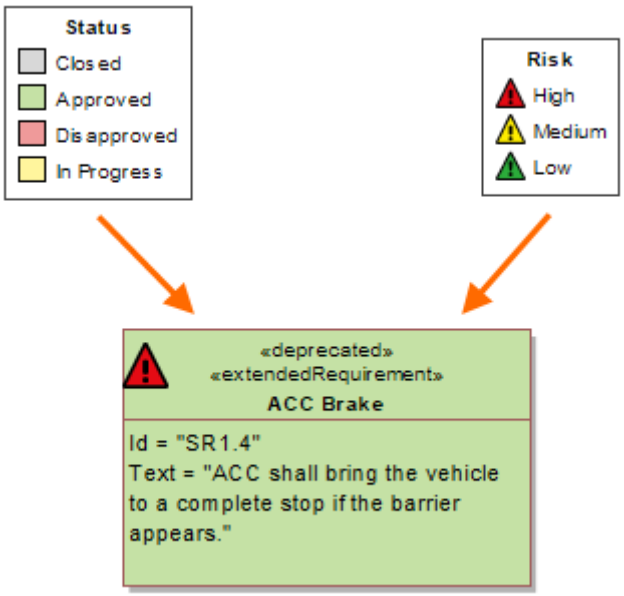
1. Open the desired Requirements Diagram.
2. In the diagram, create a legend(see page 532) intended to show the status of Requirements and name the Legend *Status*.
3. Create and define the Legend Items for the *Status* Legend by completing the following steps:
   a. Create four Legend Items(see page 534) indicating four possible Requirement status values and name them *Closed*, *Approved*, *Disapproved* and *In Progress*.
   b. Open the Specification window of the first Legend Item.
   c. Specify the Legend Item adornment by selecting the desired **Fill Color**.
   d. Specify the **Elements by Condition** property(see page 542) by creating a Property Test operation and selecting the **Status** property value the Legend Item represents.
   e. Repeat the steps from b. to d. with the rest of the Legend Items.
4. In the same agram, create a Legend to indicate the risk level of Requirements and name the Legend *Risk*.
5. Create and define the Legend Items for the *Risk* Legend by completing the following steps:
   a. Create three Legend Items indicating three possible Requirement risk levels and name them *High*, *Medium*, and *Low*.
   b. Open the Specification window of the first Legend Item.
   c. Specify the Legend Item adornment by selecting the desired **Icon** and **Icon Position on Shape**.

      Specifying Icon
      When specifying the **Icon** property:
      - Click to select an image from the Image Library of your modeling tool.
      - Click to select a custom image from the local file system.
      - Click to specify the URL of an image.

   d. Specify the **Elements by Condition** property by creating a Property Test operation and selecting the **Risk** property value the Legend Item represents.
   e. Repeat the steps from b. to d. with the rest of the Legend Items.

If you followed the above steps, you have a Requirements Diagram with the *Status* and *Risk* Legends applied to it. The Legend Items of both Legends do not conflict when applied to the same shape, as shown below.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The figure displays a Requirement shape with two Legend Items applied to it.
The Legend Items do not conflict due to different adornment properties used to define them.

Related pages

- Specifying Legend Items(see page 537)
  - Applying Legend Items(see page 542)

Applying Legend Items

On this page

- Applying Legend Items to specific elements(see page 542)
- Applying Legend Items to elements by condition(see page 543)
- Applying Legend Items to specific cells of a table(see page 545)

When you create(see page 532) or display(see page 549) a Legend in a diagram or table, the adornment(see page 538) of its Legend Items is applied to the symbols or table rows of corresponding elements. You can define the elements to which a Legend Item should be applied by specifying one of its properties in the **Apply To** property group. Legend Items can be applied either to **Specific Elements** or to **Elements By Condition**. If you apply a Legend to a table, you can also specify the **Adorned Properties** property which defines what cells of the relevant table rows should be adorned.

> **Important**
> You can specify either the **Specific Elements** or the **Elements by Condition** property of a Legend Item, but not both.

Applying Legend Items to specific elements

You can apply a Legend Item to one or multiple specific elements as described below.

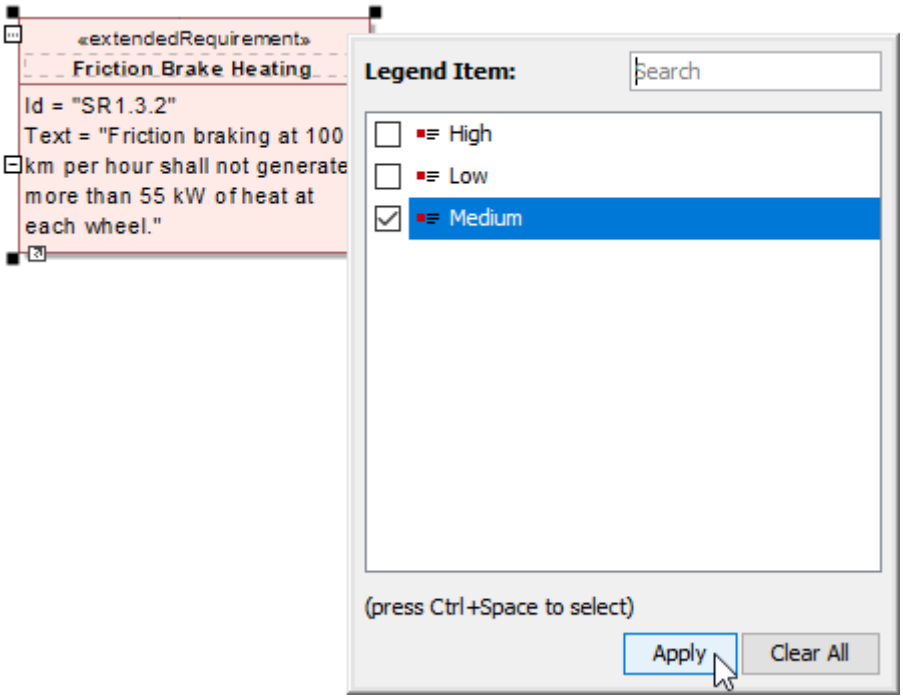
To apply a Legend Item to specific elements

1. Open the Specification window of a Legend Item you want to apply.
2. Select the property specification cell of the **Specific Elements** property, and click ...
3. On the left side of the **Select Elements** dialog, select the elements to which you want to apply the Legend Item.
4. Click + to add them to the **Selected elements** area, and click **OK**.
5. Close he Specification window.

You can also apply one or multiple Legend Items to an element from its symbol.

To apply Legend Items to an element from its symbol

1. In a diagram, right-click the shape of the element to which you want to apply Legend Items, and in the menu select **Legend Item**.
2. In the open dialog, select the Legend Items you want to apply, and click the **Apply** button.



The example how to apply Legend Items from an element shape.

Applying Legend Items to elements by condition

Applying a Legend Item to **Elements by Condition** ensures that it is consistently applied to all the elements that meet the defined condition, even after you change your model.

Example

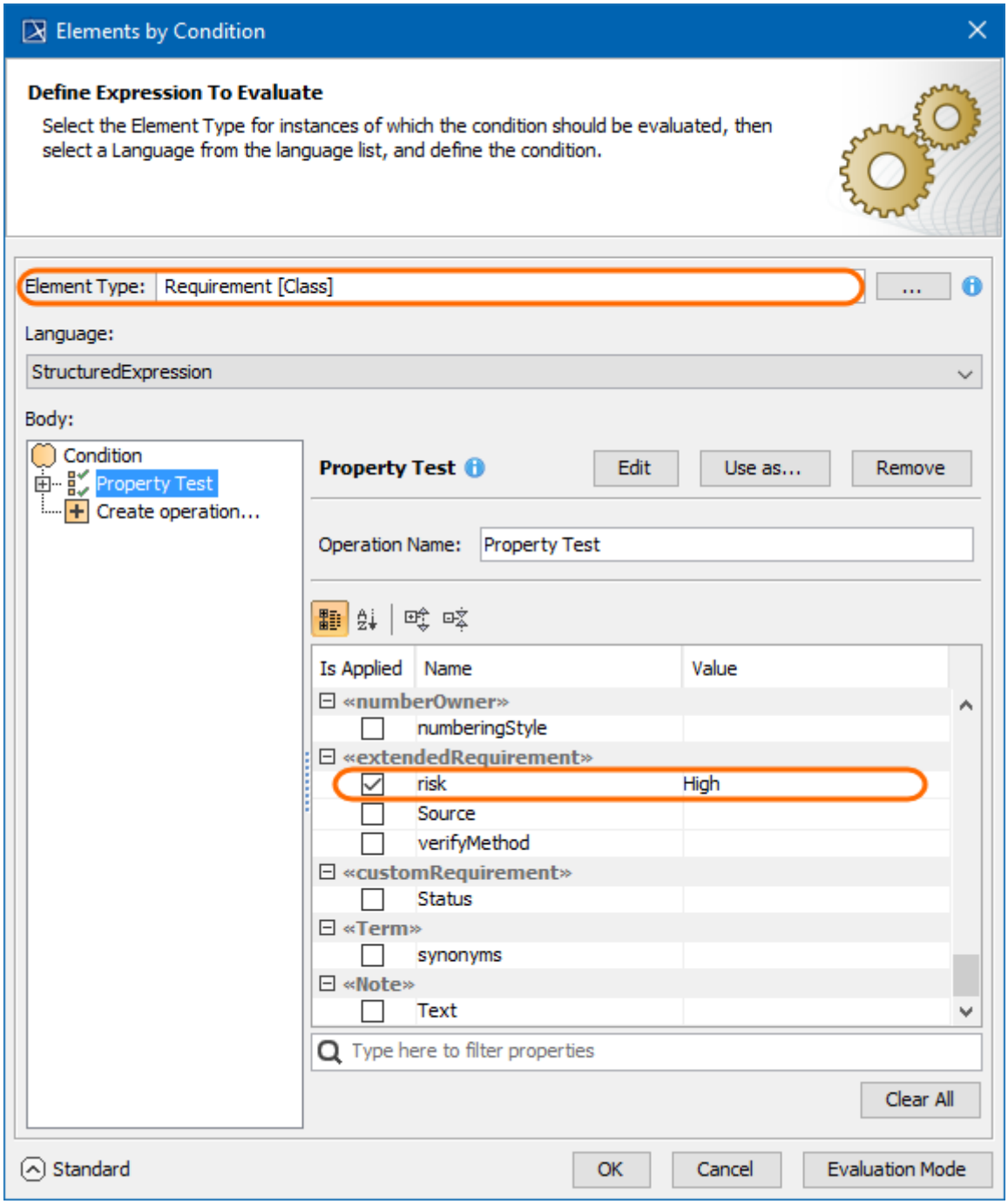
Let's say that you want to apply a Legend Item to all requirements that have a high risk level. The quickest way to do it is to create a Property Test operation. Follow the steps bellow to analyze this specific use case and define the **Elements by Condition** property for the Legend Item:

1. Open the Specification window of a Legend Item.
2. Select the specification cell of the **Elements by Condition** property, and click ...
3. In the **Element by Condition** dialog, click ... next to the **Element Type** box.
4. In the **Select Element Type** dialog, select the **Requirement** type, and click **OK**.
5. In the Property Test operation specification area of the **Elements by Condition** dialog, select the **risk** property and set its value to **High**.

   Default options
   By default, the **Language** is set to **StructuredExpression,** and the Property Test operation is selected in the **Elements by Condition** dialog.

6. Click **OK**, and close the Specification window.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The **Elements by Condition** dialog with the Property Test operation defined to test if Requirements have a high risk level.

After following the above steps, the Legend Item is applied to all Requirements that have the **Risk** property set to **high**. If this use case does not meet your needs, you may enable the Expert mode, and define advanced Property Test operation properties. You may also create an operation(see page 1249) of a different type, or create several operations for a single condition, as long as it returns Boolean data type value (*true* or *false*). In addition, you can define a condition using a different **Language** setting.

Applying Legend Items to specific cells of a table

When a Legend is used in a table, you can specify not only the elements to which specific Legend Items should applied (the **Specific Elements** or **Elements By Condition** properties), but also the desired properties of these elements. In other words, you can adorn specific cells of a table instead of adorning entire table rows. To do this, use the **Adorned Properties** property of a Legend Item.

> **Adorned Properties**
> The **Adorned Properties** property of a Legend Item is only effective if you use the Legend in a table.

Example

Let's say you have a Requirements Table with three columns: *Name, Text* and *Risk*., and you want to apply two legends in that table: *Risk* and *Status*. When you create or use a Legend in a table, Legend Item adornments are normally applied to enitre table rows. So in this case, you also have to specify certain element properties (table columns) to which Legend Items of each Legend should be applied. To reproduce this spesific use case, follow these steps:

1. Create a Requirement Table with the *Name*, *Text* and *Risk* columns.
2. In the table, create a Legend(see page 532) to indicate the risk level of Requirements and name the Legend *Risk*.
3. Create and define the Legend Items for the *Risk* Legend by completing the following steps:
   a. Create three Legend Items(see page 534) indicating three possible Requirement risk levels and name them *High*, *Medium*, and *Low*.
   b. Open the Specification window of the first Legend Item.
   c. Specify the Legend Item adornment by selecting the desired **Fill Color**.
   d. Specify the **Elements by Condition** property(see page 542) by creating a Property Test operation and selecting the **Risk** property value the Legend Item represents.
   e. Specify **Adorned Properties** property by selecting the **Risk** property as its value.
   f. Repeat the steps from b. to e. with the rest of the Legend Items.
4. In the same table, create a legend intended to show the status of Requirements and name the Legend *Status*.
5. Create and define the Legend Items for the *Status* Legend by completing the following steps:
   a. Create four Legend Items indicating four possible Requirement status values and name them *Closed*, *Approved*, *Disapproved* and *In Progress*.
   b. Open the Specification window of the first Legend Item.
   c. Specify the Legend Item adornment by selecting the desired **Fill Color**.
   d. Specify the **Elements by Condition** property by creating a Property Test operation and selecting the **Status** property value the Legend Item represents.
   e. Specify **Adorned Properties** property by selecting the **Name** and **Text** properties as its values.
   f. Repeat the steps from b. to e. with the rest of the Legend Items.

After following the above steps, you have a Requirements Table with the *Status* and *Risk* Legends applied to it. The Legend Items of both Legends do not conflict because they are applied to different columns of the table, as shown below.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Risk: High Medium Low **Status:** Closed Approved Disapproved In Progress

#	Name	Text	Risk
1	SR1.1 Vehicle Mass	The total vehicle mass shall not exceed 1500 Kg.	Low
2	SR1.2 Regenerative Braking	When braking, the electric motor shall switch to generator mode. The wheels shall transfer kinetic energy via the drivetrain to the generator. Generator resistance produced from the electricity created shall slow the vehicle.	
3	SR1.3.1 Stopping Distance	The vehicle shall completely stop from 100 km per hour in less than 30 meters.	High
4	SR1.3.2 Friction Brake Heating	Friction braking at 100 km per hour shall not generate more than 55 kW of heat at each wheel.	High
5	SR1.4 ACC Brake	ACC shall bring the vehicle to a complete stop if the barrier appears.	High
6	SR1.5 Category 1 of Ultra Low Emission Vehicle	The vehicle shall emit less than 50 grams of carbon dioxide (CO_2) per kilometre driven.	Low
7	SR1.6.1 Full charge mode	The vehicle shall drive at least 200 km on full charge electric-only propulsion mode and shall be charged in less than 150 minutes.	Low
8	SR1.6.2 Quick charge mode	The vehicle should be charged in less than 30 minutes and it should be possible to drive at least 70 km.	Medium

The figure shows a Requirement Table with the *Risk* and *Status* Legends applied to it. The Legend Items of each Legend are applied to different columns of a table.

Related pages

- Specifying Legend Items(see page 537)
  - Specifying Legend Item adornment(see page 538)

Creating Legends automatically

You can create Legends automatically by extracting them from model elements. When you create a Legend automatically, main Legend properties, such as **Name** and **Legend Items**, are specified for you. The property values of an automatically created Legend are extracted from the stereotype tags of the selected element. If you want to create and specify Legend Items by yourself, create Legends manually(see page 532).

Extracting Legends
You can only extract Legends from the elements that have the stereotype tags with Enumeration type.

To create a Legend automatically

1. Open the diagram or table in which you want to create a Legend.
2. Right-click the shape or table row of the element from which you want to generate a Legend, and select **Tools** > **Extract Legends**.

   Alternative methods
   You can also access the **Extract Legends** command by right-clicking the actual element in the model browser. But in this case, you have to apply the Legend to desired diagrams by yourself.

3. In the **Select Tags** dialog select the stereotype tag from which you want to generate a Legend. You can also select multiple tags to generate multiple legends at once.
4. Click **OK**.

The Legend is automatically displayed and applied to the diagram. The name of the selected tag is used as the Legend name, and every possible tag value creates an individual Legend Item.

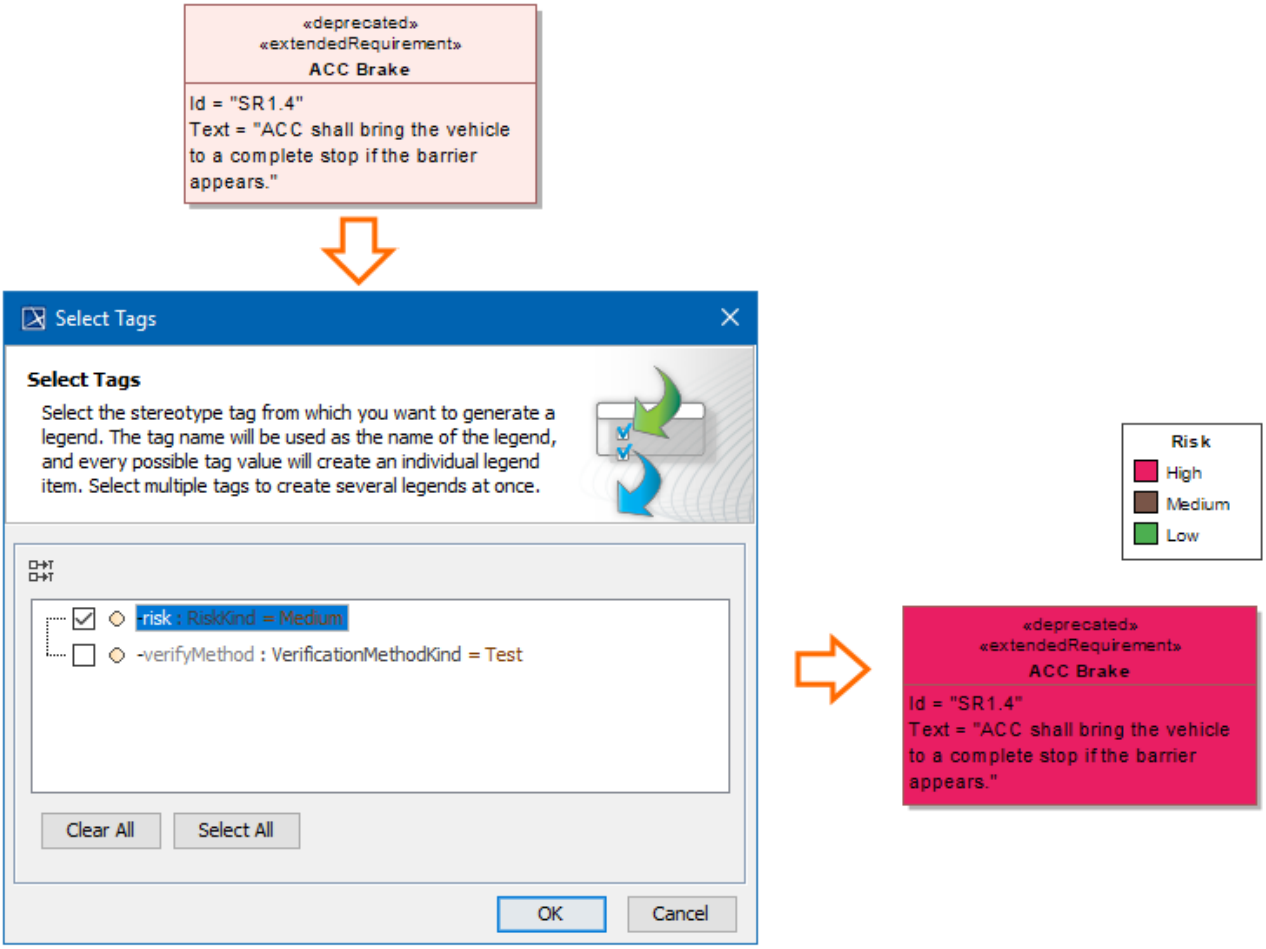
Example

Let's say that you want to extract the Legend identifying different **Risk** property values of Requirements in your model. To do this, complete the following steps:

1. Open the Requirement Diagram in which you want to create the Legend identifying different Requirement risk values.
2. Right-click the shape of any Requiremenn and select **Tools** > **Extract Legends**.
3. In the **Select Tags** dialog select the risk tag of the «extendedRequirement», as highlighted in the figure below.
4. Click **OK**.

The *Risk* Legend is automatically displayed and applied in the Requirements Diagram. The name of the tag is used as the Legend name, and every possible **Risk** property value creates an individual Legend Item.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The process of creating the Legend identifying different Requirement risk values in a Requirement Diagram.

Sample model

The model used in the figures of this page is the *Legends* sample model that comes with SysML plugin. To open this model do one of the following:

- Download legends.mdzip[84].
- Open the model from the *<modeling tool installation directory>\samples\SysML directory*.

Related pages

- Legends(see page 530)
  - Creating Legends(see page 532)
  - Creating and managing Legend Items(see page 534)
  - Displaying and hiding Legends(see page 549)

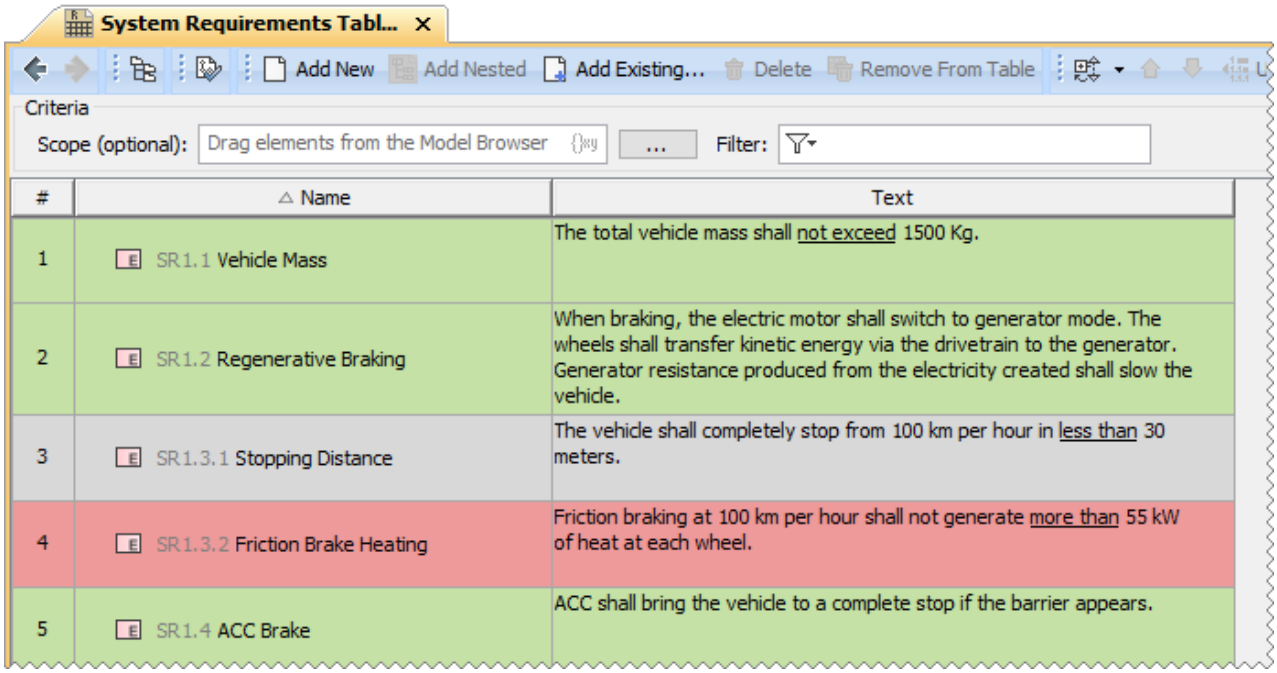
Displaying and hiding Legends

84 https://docs.nomagic.com/download/attachments/55847477/legends.mdzip?api=v2&modificationDate=1488812669828&version=1

On this page

- Applying adornments without displaying Legend(see page 551)

You can display or remove a Legend in an active diagram or table. It is also possible to apply Legend adornments(see page 551) in a diagram or table without displaying the Legend itself, as shown below.



A Requirement Table displaying only Legend adornments

To display a Legend in an open diagram or table

- Create a Legend in a diagram or table.
- Drag an existing Legend from the model browser to a diagram.
- Complete the following steps:
  a. Click in the diagram or table toolbar.
  b. In the drop-down menu, select **Display Legends**.
  c. Select the legend(s) you want to display.
  d. Click the **Display** button.

Completing one of the actions described above displays the Legend in an open diagram or table, and applies its Legend Item adornments to the relevant diagram shapes or table rows.

To remove a Legend from an open diagram or table

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Delete the Legend shape from a diagram.
- Hover the mouse pointer over a legend in a table, click ... and select **Hide**.
- Complete the following steps:
    a. Click ▾ in the diagram or table toolbar.
    b. In the drop-down menu, select **Remove Legends**.
    c. Select the legend(s) you want to remove.
    d. Click the **Remove** button.

The actions described above remove a Legend from an open diagram or table and Legend Item adornments are no longer applied to the relevant diagram shapes or table rows.

> **Keeping Legend adornments**
> When you delete a Legend shape in a diagram or select the **Hide** command for a Legend in a table, the question message appears allowing you to keep Legend Item adornments after the Legend is removed.

Applying adornments without displaying Legend

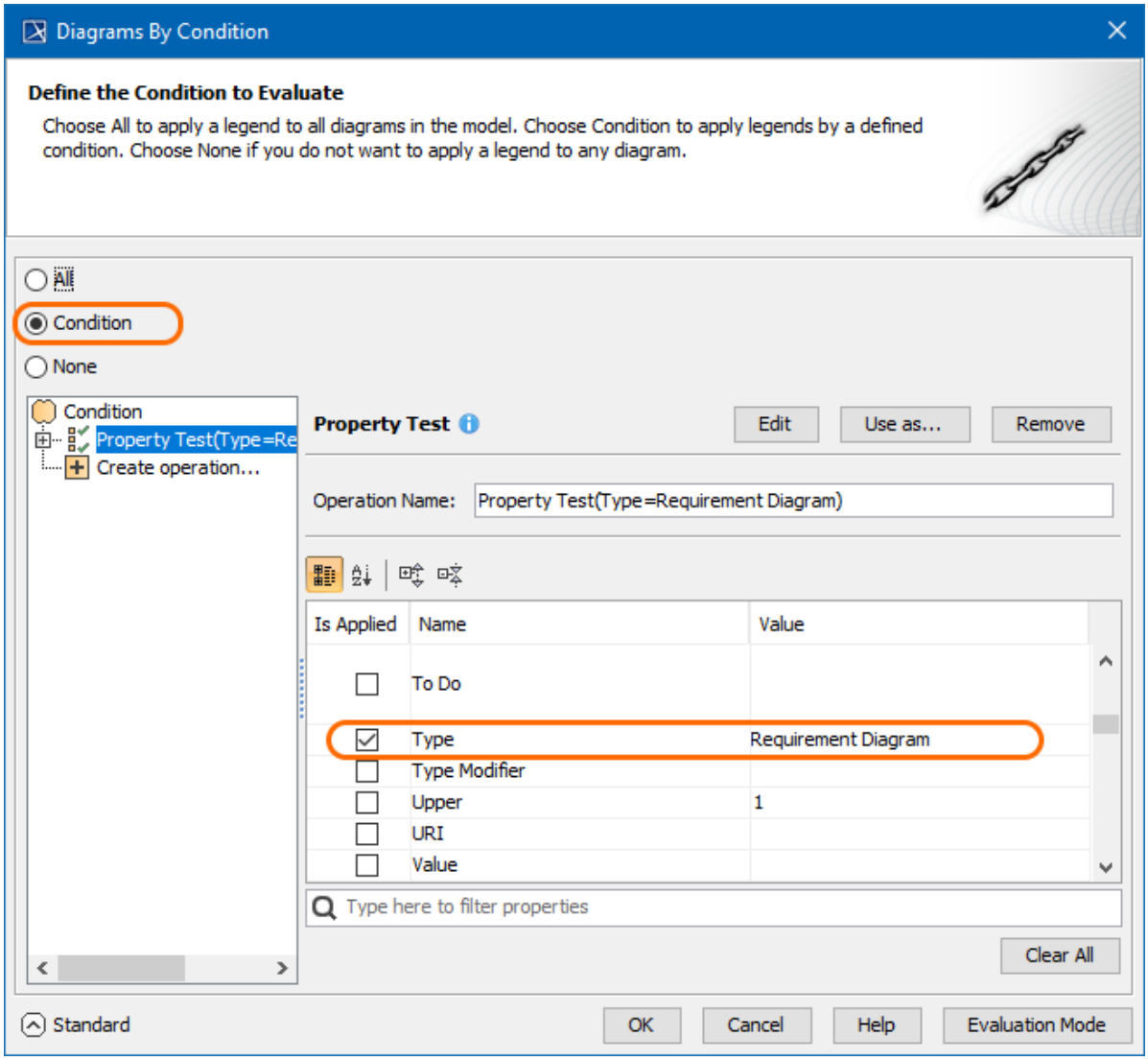
You can apply Legend adornments in a diagram or table without displaying the Legend. To achieve this, define the Legend scope by selecting specific diagrams of your model (the **Specific Diagrams** property) and/or defining the condition that diagrams should meet for the Legend to be applied (the **Diagrams by Condition** property).

Example

Let's say that you want to apply the adornments of a Legend to all Requirement Diagrams and one specific Requirement Table of your model. To do this, follow these steps:

1. Open the Specification window of a Legend.
2. Select the **Diagrams by Condition** property specification cell and click ...
3. In the **Diagrams by Condition** dialog, select the **Condition** radio button.
4. In the Property Test operation specification area, select the **Type** property and enter the **Requirement Diagram** as its value.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



5. Click **OK**.
6. Select the **Specific Diagrams** property specification cell and click ...
7. In the **Select Diagram** dialog, select the desired Requirement Table.
8. Click **OK**.

After completing the above steps, the Legend adornments are applied to all Requirement Diagrams and the selected Requirement Table. By specifying the **Diagrams by Condition** property, you also ensure that the Legend adornments will be automatically applied to newly created Requirement Diagrams as well, because they meet the defined condition.

Related pages

- Legends(see page 530)
  - Creating Legends(see page 532)
  - Creating Legends automatically(see page 547)
  - Creating and managing Legend Items(see page 534)
  - Specifying Legend Items(see page 537)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Table with diagram information

On the diagram, you can display a table containing various diagram details: its name, author, status, the dates it was created and modified, etc. By default, the Diagram info table is displayed at the right top corner of the diagram frame, but you can drag and drop it to any other position on the diagram.

The table includes the following fields:

- Diagram name
- Author
- Creation date
- Modification date
- Last Modified by
- Other available tag definitions

To show the table containing the diagram information

- From the diagram shortcut menu, select **Show Diagram Info**.

Diagram name	Requirements
Author	andre
Creation date	5/18/07 9:02 AM
Modification date	2/26/10 11:34 AM
Last modified by	martin

To define information for the table

---

1. Do one of the following:
   - From the diagram information table shortcut menu, select **Customize.**
   - From the **Options** menu, select **Project**.
2. In the Project Options dialog > **Diagram Info** > **Source**, select the type of information you want to include in the table:
   - **Standard Mode**. This mode contains the following fields that will be shown in the table: Author, Creation date, Modification date, and all other tag definitions that can be assigned to the diagram.
   - **Custom Mode**. In this mode, you can create your own table or any other object in HTML.
3. Preview the selected table or other created object in the **Preview**.

Related pages

- Diagramming(see page 482)

Changing diagram type

The diagram type can be changed to another type of diagram if both diagram types are compatible.

Diagram Elements
Diagram elements are not converted.

Uses for changing the diagram type:

- To migrate with existing project to a diagram type which was not available until then. For example, migrating a Class diagram(see page 297) to the SysML Block Definition diagram[85].
- To migrate with existing project from the diagram type, which will be erased from your project. For example, if the user has decided to erase a plug-in, they would need to convert that plug-in's specific diagram to standard diagrams.

Diagram conversion scenarios:

- Any static diagram can be converted to another static diagram.
- Any dynamic diagram can be converted to another diagram, if and only if both diagrams are based on the same diagram type and both diagrams are compatible.

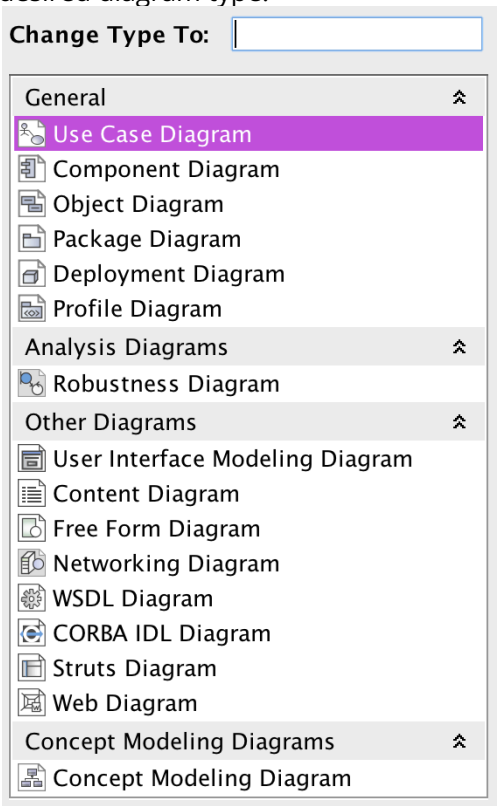
To change the diagram type

1. Select one or more, of the same type, diagrams in the Model Browser(see page 29) (Containment or Diagram tree).

85 https://docs.nomagic.com/display/SYSMLP190SP4/SysML+Block+Definition+Diagram

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

2. From the shortcut menu, choose **Refactor** > **Change Type To**. In the open dialog, select the desired diagram type.



Working with symbols of model elements

Sorry, the widget is not supported in this export.
But you can reach it using the following URL:

https://www.youtube.com/watch?v=9tPfgYQLNd0

To draw an element on the diagram pane

1. Select an element button on the diagram palette.
2. Click anywhere on the diagram pane.

To draw several elements of the same type on the diagram pane, do one of the following

Use the Shift key

1. Press and hold Shift.
2. Click an element button on the diagram palette.
3. Click the diagram pane. With each click, the new element of the same type is drawn.

Use the Sticky button

1. On the diagram palette, under the **Tools** category, click the Sticky button or press Z.
2. From the diagram palette, select an element button.
3. Click the diagram pane. With each click, the new element of the same type is drawn.

To draw a symbol of the existing element

---

1. Activate a diagram on which you wish to draw a symbol.
2. From the Model Browser(see page 29), select an item you wish to draw.
3. Drag the selected model element onto the diagram pane.

To specify a name of the element, do one of the following

- In the Specification window(see page 913), Name property specification cell, type the element name.
- Type the symbol name directly on the selected symbol on the diagram pane.
- In the Model Browser(see page 29), select an element and name it.

To draw several symbols for one element, do one of the following

---

- In the Model Browser(see page 29), select an element and drag it to the diagram pane. Repeat it if needed.
- On the diagram pane, select a symbol, copy and then paste by pressing Ctrl+C and Ctrl+V.

To resize selected shapes to a preferred size automatically

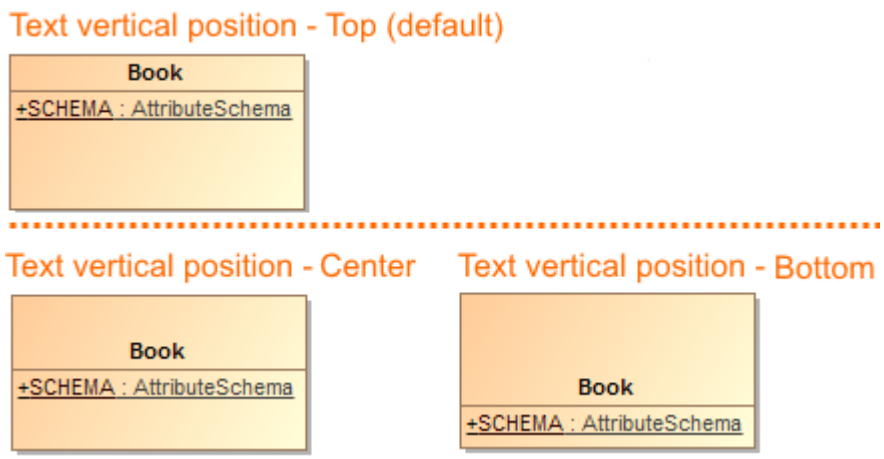
• Click the shape and then click the ▣Make Preferred Size smart manipulator.

> Note that the smart manipulator is always displayed on any of the visible borders. Just select the shape, and the manipulator appears at the right of the shape's horizontal border or at the bottom of the vertical shape's border.

To change text and compartments vertical position on shapes

- In the **Symbol Properties** dialog > **Text Vertical Position**, select one of the following:
  - Top
  - Center
  - Bottom

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



To delete selected model elements, do one of the following

- From the **Edit** menu, select **Delete**.
- On the diagram toolbar, click the *Delete* button.
- Press Ctrl+D.
- In the Activity and Sequence diagrams, press Delete.

> The model elements will be removed from the model with all their data.

To delete selected symbols, do one of the following

- From the **Edit** menu, select **Remove from Diagram**.
- On the diagram toolbar, click the *Remove from Diagram* button.
- Press Delete.

Related pages

- Diagramming(see page 482)

Working with paths and relationships

The difference between relationship and paths(see page 2279) is that a relationship is a link between model elements while the path is a representation of the relationship on a diagram.

You can learn how to work with paths and relationships in the following pages:

Related pages

- Displaying paths(see page 513)

Related pages in Developer Guide

- Paths(see page 2279)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Creating a relationship

You can create the relationships between elements in one of the following ways:

- Draw on the diagram pane(see page 558)
- From the selected element shortcut menu(see page 558)
- In the Specification window(see page 559)

To draw a path on the diagram pane(see page 24)

---

1. In the Diagram palette(see page 24), click the needed relationship.
2. Select the source shape of the path in the diagram pane.
3. Drop the path to the target shape.

> ⓘ
> - When drawing a new path the optimal route is chosen automatically (path avoids other shapes).
> - When drawing a path between two elements, those shapes are bordered with a red or blue rectangle. The red highlight indicates that the path can not be drawn between these shapes. The blue highlight allows a path to be drawn. You can disable this highlighting in the **Environment Options** dialog under the Diagram node by setting the **Show Manipulation Highlighting** property value to *false*. For more information, see Customizing environment options(see page 65).
> - To draw a path faster use smart manipulator toolbar(see page 568).

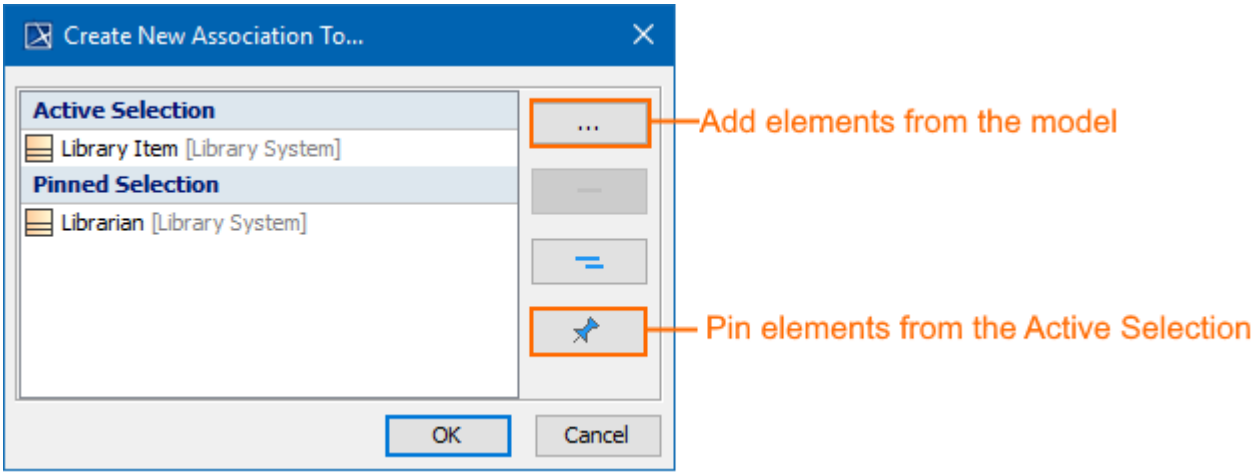
To create relationships between selected elements

---

1. In the Model Browser(see page 29), diagram pane, or table right-click an element.
2. Choose **Create Relation** > **Outgoing** / **Incoming** and select the desired relationship type. The **Create New <relation name> To** / **From** dialog opens.
3. Do one or more of the following:
   - In the Model Browser, click the element to / from which you want to create a relationship.
   - On a diagram, click and element to / from which you want to create a relationship.
   - In the **Create New <relation name> To** / **From** dialog, click ... . In the opened **Create New <relation name> To** / **From** dialog, choose the element to / from which you want to create a relationship.

     > ⓘ If you do not know the exact location of the needed element, you can search for it in this step.
4. Click **OK**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The **Active Selection** area shows the elements currently selected in the active diagram, table, matrix, map, or Model Browser. It changes with each click of the different elements of when you switch to another diagram.

To choose one or several elements at a time, do one of the following

- Press Shift and click as many elements as you need.
- Click and drag the mouse pointer to select as many elements as you need.

The multiple selection works in one area at a time. For example, diagram pane, table, matrix, relation map, or Model Browser. If you need to select elements from several locations, you need to pin your current selection.

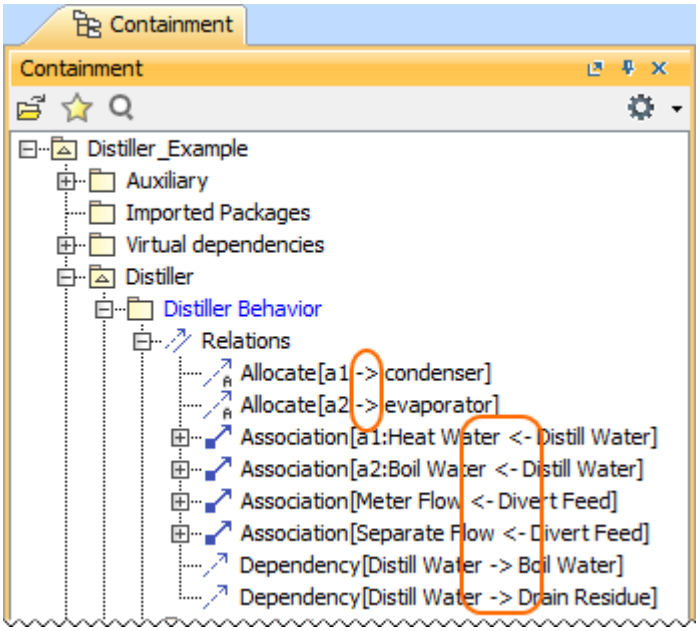
The **Pinned Selection** area show the elements you choose to pin from the **Active Selection**. This is needed when you want to select elements from different locations (for example, you need to select elements from different diagrams, or from the diagram and Model Browser). In this case, you should pin each selection.

To create a relationship in the Specification window(see page 913)

1. Open element's Specification window and select the **Relations** property group.
2. Click the **Create Outgoing** button or **Create Incoming** button and then select the desired relation type from the list.
3. In the **Create New <relation name> To/ From** dialog select an element to/ from which you want to create a relation and click **OK**.
   The relation appears in the **Relations** property group.

When the relationship is created, you can visually identify its direction (source and target elements) under the Containment tree(see page 32) > **Relations**.

> The arrow between source and target elements appears if the path is created using only directed relationships.



Removing a relationship

To remove a relationship from the diagram pane

1. Select a relationship and click in the Diagram toolbar.

To remove a relationship from the model

1. Select a relationship and click in the Diagram toolbar.
2. In the **Question** dialog, click **Yes** to confirm the deletion.

Changing a path style

To change a path style, do either

- Right-click a path(s) on the diagram pane and select **Reset Labels Positions** or **Remove Break Points**.
- Select a path(s) on the diagram pane and on the diagram toolbar, click , , or .
- Select a path(s) on the diagram pane and on the **Edit** menu click **Paths**, and then choose one of the following:
  - **Path Style** to select one of the path styles - rectilinear, oblique, or bezier.
  - **Change Path Style** (shortcut keys Ctrl+L) to change the path style to rectilinear, oblique, or bezier.
  - **Reset Labels Positions** to reset the changed path labels position to default.
  - **Remove Break Points** to remove all angles of the path.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Right-click a path(s) on the diagram pane and select **Symbol Properties**. In the **Symbol Properties** dialog, change the **Path Style** property value by selecting rectilinear, oblique, or bezier.

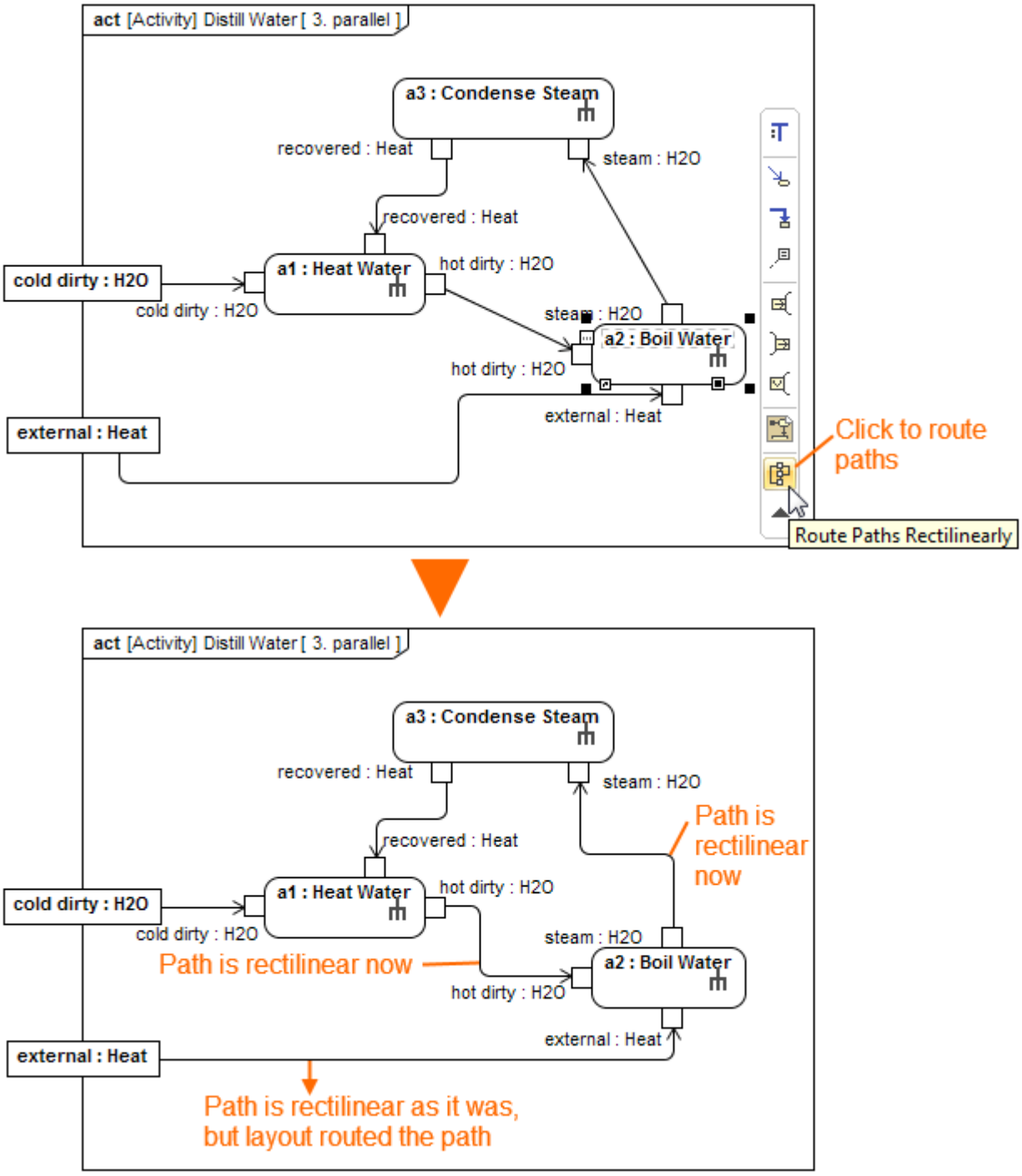
Routing paths

To route paths using the **Route Paths Rectilinearly** button, do either

---

- Select a path on the diagram pane and on the smart manipulator toolbar, click . The selected path is routed rectilinearly.
- Select a shape on the diagram pane and on the smart manipulator toolbar, click . All the paths that are connected to the shape are routed rectilinearly.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Making the paths corners rounded

To make the path corners rounded

1. Right-click the path and click **Symbol Properties**.
2. In the **Symbol Properties** dialog, set the **Rounded Corners** property value to *true*.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

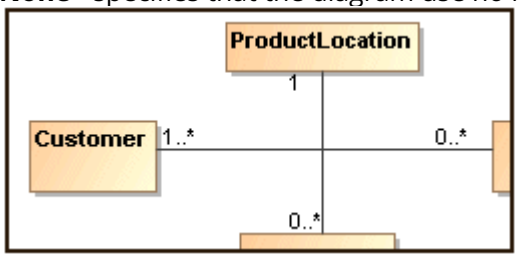
Creating line jump

Line jumps represent an intersection of lines. If you have a large diagram with lots of intersecting paths, line jumps make the diagram easier to understand. The line jumps are not displayed by default.

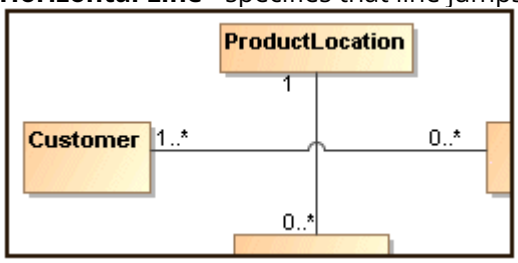
To add line jumps

---

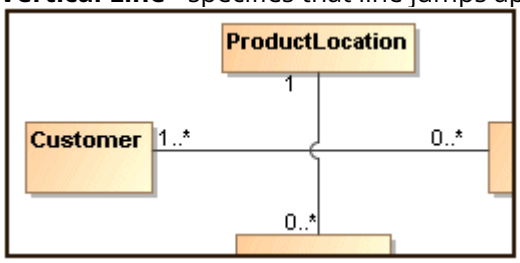
1. Right-click the diagram pane and select **Diagram Properties**.
2. In the **Diagram Properties** dialog, select the **Add Line Jumps To** property value from the drop down list:
   - **None** - specifies that the diagram use no line jumps (default value).



   - **Horizontal Line** - specifies that line jumps appear only on horizontal line.



   - **Vertical Line** - specifies that line jumps appear only on vertical line.



> ⓘ The line jump changes are applied for the current diagram only. How to change line jumps for the whole project, see Formatting symbols(see page 975).

Inserting a shape on the path

> ⓘ This functionality is available in the State Machine(see page 308) and Activity(see page 311) diagrams.

In the State and Activity diagrams you can split a path into two paths, by dropping a shape on it. This is valid for Transition/ Control Flow/ Object Flow relationships and allowed to connect with these path elements.

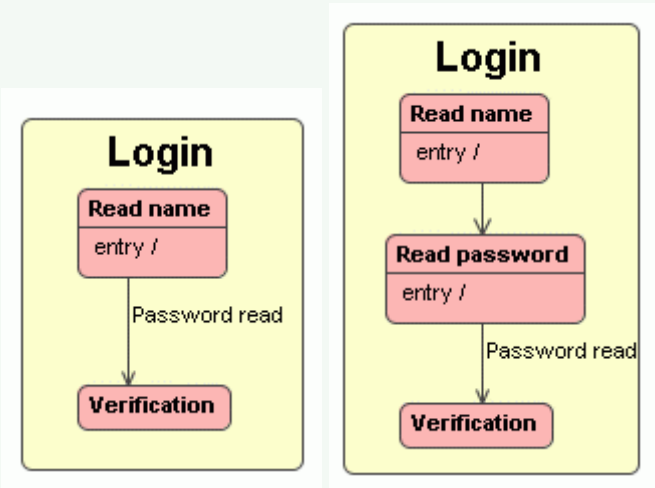
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To insert a new shape by splitting path on the diagram pane

1. Select the shape you want to insert or create a new one.
2. Drag the shape on the path (the path is highlighted in blue) and drop it.
3. In the **Split Transition** dialog click the appropriate button according to were do you want to insert the shape on the path:

- **Before Transition** button. The shape is inserted before the path. It means that a new path is created, then the dropped shape is drawn and then the existing path is drawn.

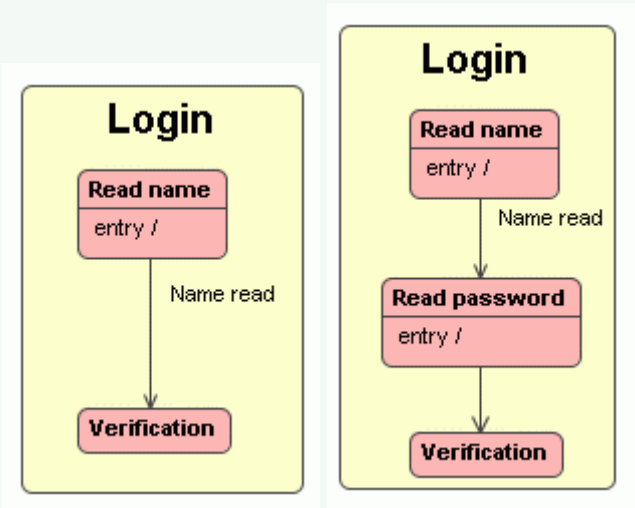
  For example: *Password read* transition is drawn from *Read Name* State to *Verification* State. If you want to insert the *Read password* State before the *Password read* Transition, drop the *Read password* State on the Transition and in the **Split Transition** dialog, choose the **Before Transition** button.



- **After Transition** button. The shape is inserted before path. It means, the existing path is created, then the dropped element symbol is drawn, and then a new path is drawn.

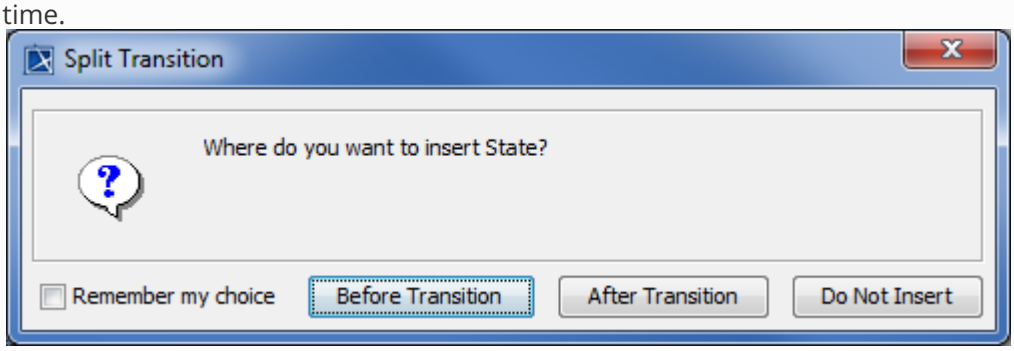
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

For example: *Name read* Transition is drawn from *Read Name* State to *Verification* State. If you want to insert the *Read password* State after the *Name read* Transition, drop the *Read password* State on the Transition and in the **Split Transition** dialog, choose the **After Transition** button.



- **Do Not Insert** button. Action is cancelled and the dialog is closed.

In the **Split Transition** dialog, select the **Remember my choice** check box and the next time the shape will be inserted before or after the path, depending on your selection made this time.

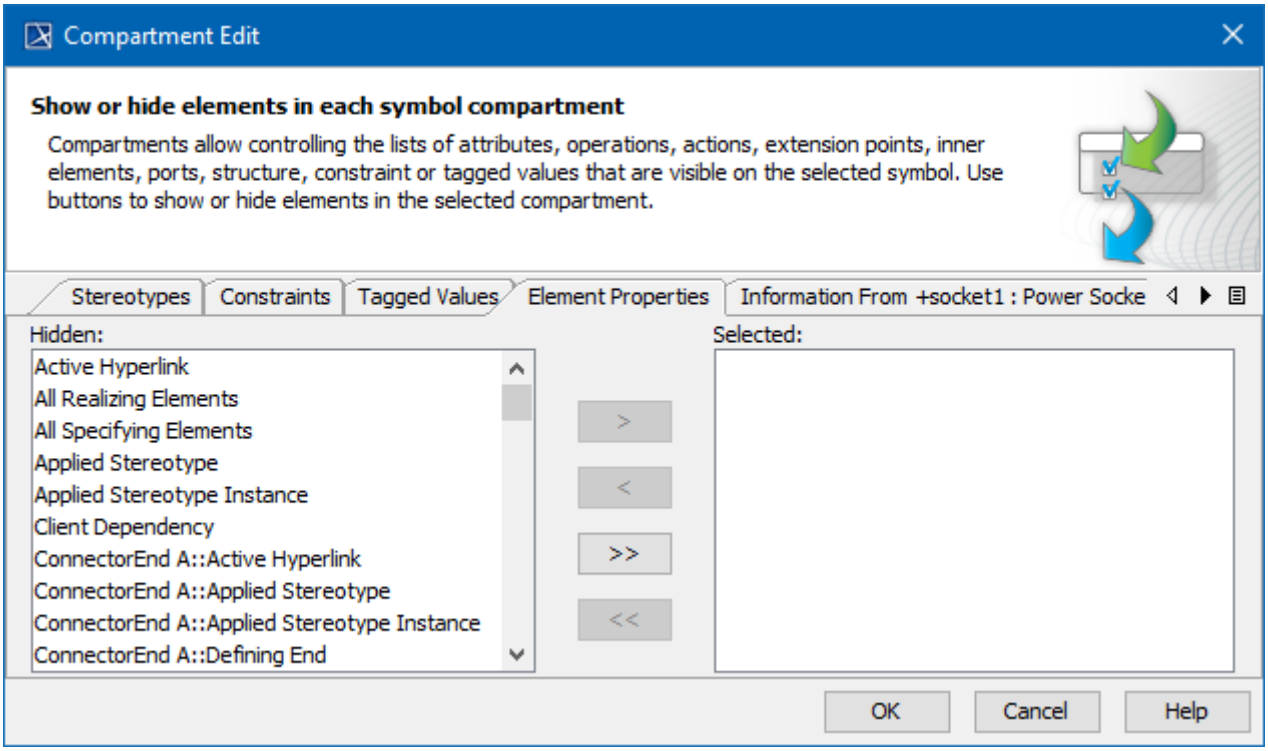


Path compartments

Use the **Compartment Edit** dialog to control the list of elements and properties visible in compartments on paths(see page 2279).

To open the **Compartment Edit** dialog from the diagram pane

1. Right-click a single or any number of paths.
2. From its shortcut menu, select **Edit Compartments**.
   The **Compartment Edit** dialog opens.



The Compartment Edit dialog of the path.

You can display or hide stereotypes, constraints, tagged values, element properties, and others for any number of paths at once. You can display or hide the element specification properties, such as *Name*, *Is Active*, *Owner*, *To Do* in the **Compartment Edit** dialog under theElement Propertiestab.

To display/hide elements in path compartments

---

1. Open the **Compartment Edit** dialog. How to >(see page 565)
2. In the **Compartment Edit** dialog, select the tab for the items you want to show or hide.
3. Do the following:
   - Move items from the Hidden area to the Selected area to display them in path compartments.
   - Move items from the Selected area to the Hidden area to hide them in path compartments.

When you edit compartments of more than one path at once, the **Compartment Edit** dialog works with the following rules:

- If you select multiple paths of the same element, the **Compartment Edit** dialog is the same as for a single path, but changes are applied for all selected paths after clicking **OK**.
- If you select multiple paths of different elements, the **Compartment Edit** dialog shows a union of compartments and their items.
- If you select multiple paths but at least one of them hides the common compartment item while another shows it, the **Compartment Edit** dialog displays that item in italics in the **Element Properties** tab > Selected area. If you move the conflicting item to the Hidden area, it becomes non-italicized and changes are applied for both paths after clicking **OK**.

Related pages

- Creating a relationship(see page 558)
- Removing a relationship(see page 560)
- Changing a path style(see page 560)
- Routing paths(see page 561)
- Making the paths corners rounded(see page 562)
- Creating line jump(see page 563)
- Inserting a shape on the path(see page 563)
- Displaying paths(see page 513)
- Paths(see page 2279)
- Compartments on shapes(see page 569)

Smart manipulation

Sorry, the widget is not supported in this export.
But you can reach it using the following URL:

https://www.youtube.com/watch?v=dptM-r2LIwg

Smart manipulation is a feature designed to make modeling with MagicDraw diagrams easier. Use smart manipulation to display or suppress compartments, create elements, set an auto-size option, reset a label position on a path, and draw relationships with most commonly used elements. MagicDraw offers varying smart
mechanisms depending on the shapes involved.

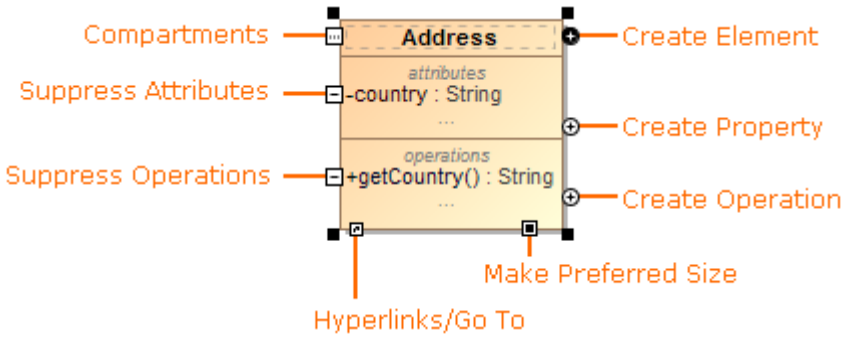
There are the following types of smart manipulation:

Related pages

- Diagramming(see page 482)

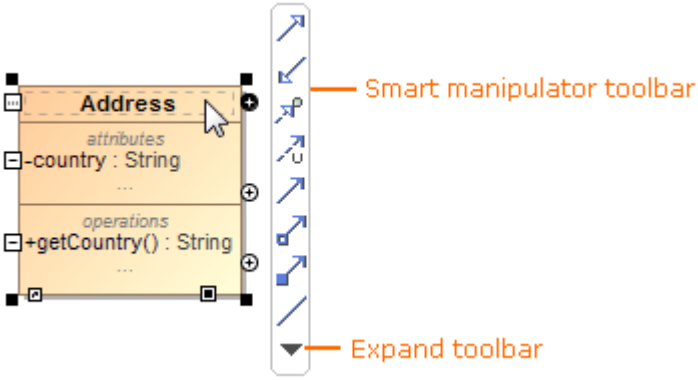
Smart manipulators

Small buttons are displayed on the borders of a symbol on the diagram pane.



Smart manipulator toolbar

Smart manipulator toolbar appears when symbols are selected on the diagram pane.



In the smart manipulator toolbar, buttons are divided into standard and extra modes. You can toggle between these two modes by clicking the Expand button - the arrow symbol - located at the bottom of every smart manipulator toolbar. Your choice will be remembered for all elements.

Use the smart manipulator toolbar to perform simple actions and create new elements quickly.

To create a new element connected to a particular element

1. From the selected element's smart manipulator toolbar, select the relationship that you want to draw.

   Create a class symbol. In the class smart manipulator toolbar, select the Directed Association relationship for drawing. The drawing of this Directed Association relationship is initiated and the mouse cursor displays a class icon. Click the left mouse button. The element displayed on the mouse cursor is created together with the relationship.

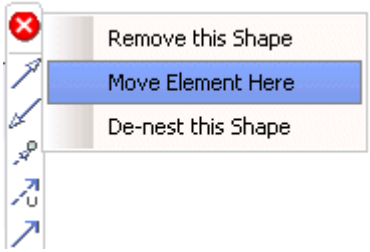
2. Right-click the mouse button and select the element from the list to be created at the end of the relationship you are drawing.

- To create a path breakpoint use the following keyboard combination: Ctrl key + Mouse click.
- To cancel the drawing of an element, press Esc.

To solve the detected symbol ownership problem

1. Select the element with an incorrect ownership on the diagram pane (it is highlighted in red). The smart manipulator toolbar appears.
2. Click the red button at the top of the smart manipulator toolbar. The menu with the possible problem solving solutions appears.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



To hide smart manipulators and the smart manipulator toolbar

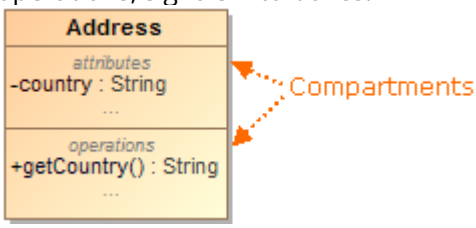
1. From the **Options** menu, select **Environment**. The **Environment Options** dialog opens.
2. In the **Diagram** options group, **Smart Manipulators** category, click to cancel the selection of the **Show Smart Manipulators** check box.
3. Click **OK**.

Compartments

The compartment is area on the shape or next to the shape that is dedicated for the specific type of information.

There are the following types of compartments:

- Compartment boxes. For example, the class shape has compartments which puts attributes, operations, signals into boxes.



> - You can hide compartments by suppressing them or expand the hidden compartments to see their data.
> - Compartment names are displayed by default. However, you can hide them by changing their visibility settings either in the **Symbol Properties** dialog or the **Project Options** dialog. Each compartment has its own property for visibility settings (for example, **Show Attributes Compartment Name**).

- Compartments that are displayed next to the element name on the shape. For example, a shape has a compartment stereotypes, constraints, and others.

> - Stereotypes (or stereotypes compartment) are displayed above the element name.
> - Constraints (or constraints compartment) are displayed under the element name.
> - By default, tagged values are displayed under the element name or you can select to display tagged values in the compartment box.
> - Element Properties are displayed in the compartment box.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Compartments that are displayed next to the element name, next to the shape. For example, an Actor has a compartment for element properties, a compartment for stereotypes, behaviors, and others.
- Compartments on paths that are displayed next to the path label. For example, a compartment for element properties, a compartment for stereotypes, and others.

Related pages

- Creating elements in compartments on shapes(see page 570)
- Displaying and suppressing compartments on shapes(see page 570)
- Displaying or hiding elements in compartments on shapes(see page 571)
- Diagramming(see page 482)
- Path compartments(see page 565)

Creating elements in compartments on shapes

To quickly create elements in compartments on shapes, do one of the following

---

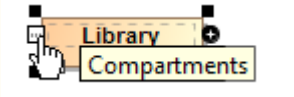
- On the diagram, select a shape and click the *Create Element* smart manipulator. Then in the menu, select the element that you want to create.
- On the diagram, select a shape. Then next to the particular compartment, click the *Create <element name>* smart manipulator.

Displaying and suppressing compartments on shapes

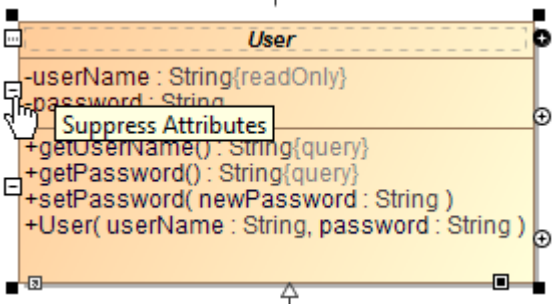
To display or suppress compartments, do one of the following

- On the diagram, click the shape and click the *Compartments* smart manipulator. Then select or or cancel the selection of the particular compartment, for example, Attributes, Operations.

  The Compartments smart manipulator exists on shapes that have the compartment boxes. For example, pins, initial nodes do not have the Compartments smart manipulator.

- On the diagram, select a shape and click the *Suppress <compartment name>* smart manipulator.

- On the diagram, right-click the shape and select **Symbol Properties**. Then select or cancel the selection of the **Suppress** <compartment name> check box.

Displaying or hiding elements in compartments on shapes

Use the **Compartment Edit** dialog to control the list of elements and properties visible in compartments on shapes. You can display or hide stereotypes, constraints, tagged values, attributes, operations, and others. If there are hidden properties on the boxable compartment, the **...** is displayed.

You can also display or hide the element specification properties, such as *Name*, *Is Active*, *Owner*, *To Do* in the **Compartment Edit** dialog under the **Element Properties** tab.

To open the **Compartment Edit** dialog from the diagram pane

---

- Select shape and click the *Compartments* smart manipulator. On the menu, click the arrow to expand and then select **Edit Compartments**.
- Right-click a single or any number of shapes or classifier compartments (e.g. Properties, Operations, Receptions, Ports, and etc.) and from the shortcut menu, select **Edit Compartments**.

  Compartment Edit dialog of multiple shapes
  When you edit compartments of more than one shape at once, the **Compartment Edit** dialog works with the following rules:
  - If you select multiple shapes of the same element, the **Compartment Edit** dialog is the same as for a single shape, but changes are applied for all selected shapes after clicking **OK**.
  - If you select multiple shapes of different elements, the **Compartment Edit** dialog shows a union of compartments and their items.
  - If you select multiple shapes but at least one of them hides the common compartment item while another shows it, the **Compartment Edit** dialog displays that item in italics in the **Element Properties** tab > Selected area. If you move the conflicting item to the Hidden area, it becomes non-italicized and changes are applied for both shapes after clicking **OK**.

- On the Note shape, click the *Edit Element Properties* smart manipulator.

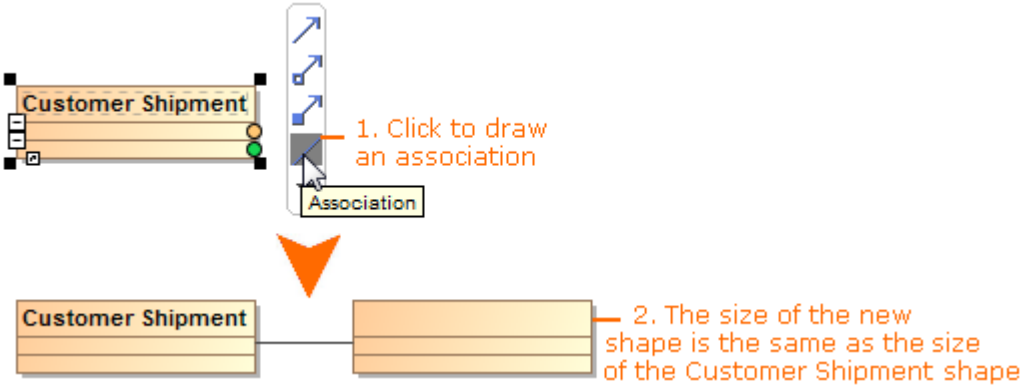
Smart shape sizing

MagicDraw introduces the smart shape sizing when drawing new elements from other elements via paths. The size of the newly created shape is kept the same as the size of the shape from which the path is drawn:

- Drawing a shape from another shape using a path from smart manipulator. The size of the new shape is the same as the size of the shape from which you draw a path. Note that this is valid if the source shape is not large, for example, if it is not more than 3 times larger than the new

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

shape.



- Inserting a new shape into the flow in the activity or state diagram. The size of the newly inserted shape is the same as the size of the existing shape.

Related pages

- Diagramming(see page 482)

Selection and multiple selections

To select a shape

- Click the desired shape on the Diagram pane.

To select all shapes of the same type

- Press and hold the Alt key, and click one shape

To select all shapes on the diagram, do one of the following

- From the **Edit** menu, choose **Select All**.
- Press Ctrl+A.

To make multiple selections, do one of the following

- Select a shape on the diagram pane and holding down Shift, select as many shapes as you need.
- Drag the cursor diagonally across the area you wish to select.

Select a group of shapes

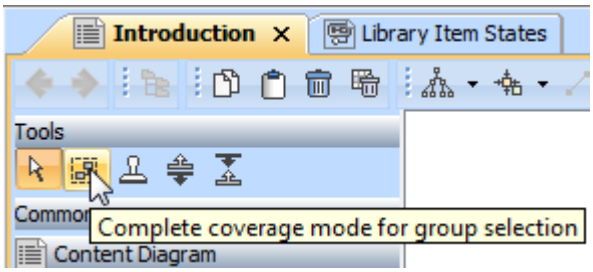
To select a group of shapes

- Drag the cursor diagonally across the area you wish to select.

This is a simple and fast way to select a group of shapes on the diagram.

The following rectangular selection modes are available:

1. Partial coverage. Symbols which are only partly covered with the rectangular selector are selected.
2. Complete coverage. Only those symbols that are fully covered by the rectangular selector are selected.



Default selection mode is Partial coverage.

To change the group selection mode from Partial coverage to Complete coverage mode and vice versa

- Press Ctrl and drag the cursor diagonally across the area you want to select.

To change the group selection mode for the whole project, do one of the following

- In the main diagram toolbar, click the *Complete coverage mode for group selection* button.
- In the Environment Options dialog, **Diagram** > **Symbols Manipulation** group, change the property of the Group selection mode option.

Related pages

- Diagramming(see page 482)

Copying text or images to diagrams

You can copy the text or images to a diagram. The copied text or image is pasted into a text box or an image shape. MagicDraw supports the HTML and plain text, and images of gif, jpg, svg, and png image

file types. This copy and paste functionality allows you to drag and drop data from other applications, for example, web browsers.

To copy the text or image to a diagram

1. Select the text or image.
2. Press Ctrl+C.
3. Open the diagram.
4. Press Ctrl+V. The **Paste Special** dialog opens.
5. Select **Image**, **Plain Text**, or **HTML Text** according to what you want to paste on the diagram.

Related pages

- Diagramming(see page 482)

Nesting image shapes

You can now drag an image to any elements in a diagram as nested a element.

To drag an image to an element

1. Select an image in the diagram pane.
2. Drag it to the image shape.

Dragged images will be nested by the following elements: Package, Model, Subsystem, Instance, Node, Part, Combined Fragment, Composite State (State diagram), Interruptible Activity Region, Structured Activity Node, Expansion Region, and Conditional Node (Activity diagram).

Related pages

- Diagramming(see page 482)

Zooming

Zooming allows to select a particular part of a diagram, zoom in, and make changes while working with a finer level of detail. You can also gain an overview of a diagram by zooming out.

The **Diagram Zoom Slider**, located in the right bottom of the modeling tool helps you with the zooming options. You can hide the slider if you do not need it.

To hide the Diagram Zoom Slider

1. In the main menu, click **View** > **Status Line**.
2. Click **Show Diagram Zoom Slider** to cancel the selection.

To fit the current diagram in the window, do one of the following

- From the diagram shortcut menu, select **Fit in Window** (shortcut keys Ctrl+W). The same is valid from the **View** menu.
- At the right bottom of the modeling tool, click the Fit to Window button.
- In the Model Browser > **Zoom** tab, click **Fit to Window**.

To zoom in the current diagram, do one of the following

- In the main menu, click **View** and select **Zoom In** (shortcut keys Ctrl+NumPad + or scroll).
- At the right bottom of the modeling tool, click the Zoom In button.

To zoom out from the current diagram, do one of the following

- In the main menu, click **View** and select **Zoom Out** (shortcut keys Ctrl+NumPad - or scroll).
- At the right bottom of the modeling tool, click the Zoom Out button.

To restore the diagram to the original size, do one of the following

- In the main menu, click **View** and select **Zoom 1:1** (shortcut keys Ctrl+NumPad /).
- At the right bottom of the modeling tool, click the Zoom 1:1 button.

To view the selected shapes at maximum size

- Select the shapes and from the **View** menu, select **Zoom to Selection** (shortcut keys Ctrl+NumPad *).

To select the zoom settings

- In the **Environment Options** dialog, **Diagram** options group, change the **Zoom Step Size** property. The maximum number is 1.0 (you can zoom a diagram twice.)

> ⚠ You can also zoom the diagram in or out using the zoom tab(see page 53) in the Model Browser.

Related pages

- Diagramming(see page 482)

Using the grid

The grid helps to arrange diagram symbols on the diagram pane. By default, the grid is not visible.

To show the grid on the diagram, do one of the following

- From the main menu, select **View** > **Grid** > **Show Grid**.
- From the diagram shortcut menu, select **Show Grid**.
- In the **Diagram Properties** dialog, set the **Show Grid** property to *true*.

To snap the paths to gridlines

- From the diagram shortcut menu, select **Grid** and select **Snap Paths to Grid**. The same is valid from the **View** menu.

To snap the shapes to gridlines

- From the diagram shortcut menu, select **Grid** and select **Snap Shapes to Grid**. The same is valid from **View** menu.

To change the grid size

1. From the diagram shortcut menu, select **Grid** > **Grid Size**. The **Grid Size** dialog opens.

   > ⚠ The same is valid from **View** menu.

2. Enter a grid size between 2 and 30 (default is 10).
3. Click **OK**.

To change the grid style

1. From the **Options** menu, select **Environment**. The **Environment Options** dialog opens.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

2. From the **Grid Style** drop-down list, select one of the following styles:
    - Dense
    - Sparse (default)

Related pages

- Diagramming(see page 482)

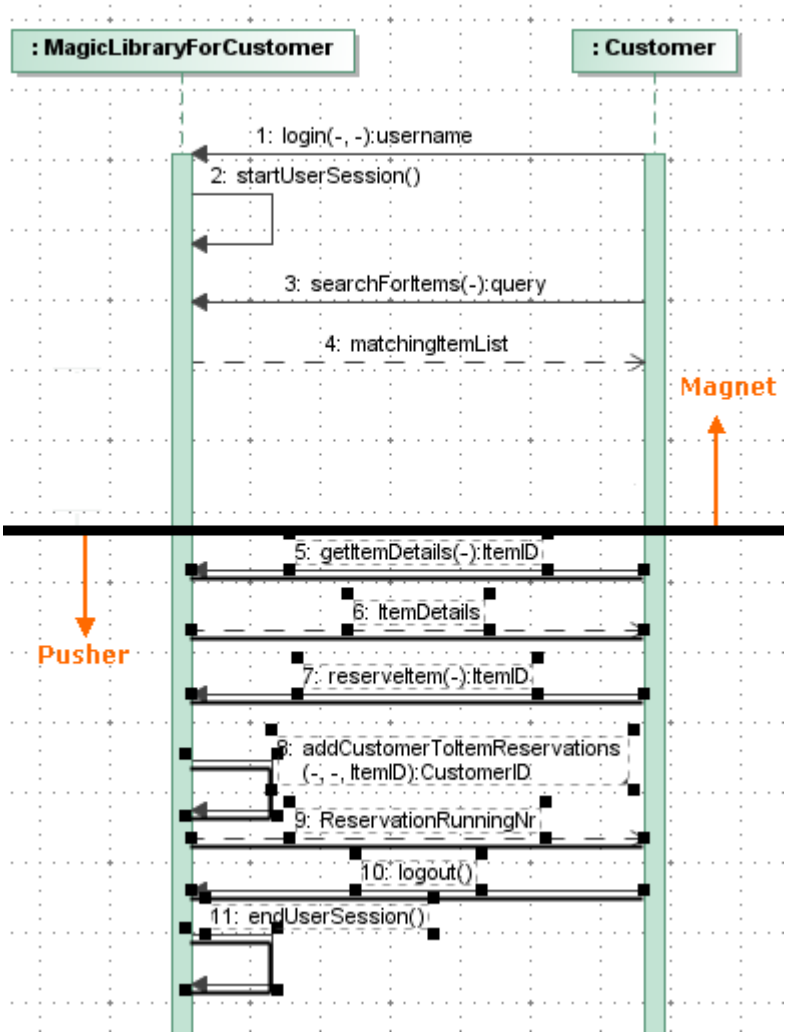
Pusher and magnet

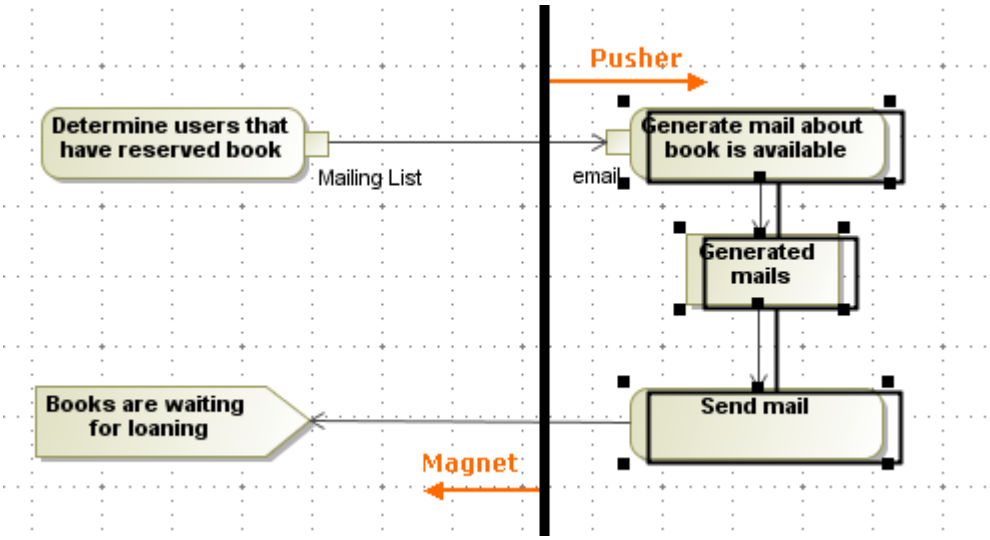
This feature allows for rearranging element symbols on a diagram easily. Pusher moves a group of symbols towards the pushing direction thus creating an empty space for adding new symbols to a diagram.

Magnet works in the opposite way. It moves the selected group of symbols to an empty space on a diagram.

Now you do not need to select separate symbols on a diagram to move them to another location. Just click the **Pusher** or **Magnet** button on the diagram palette and move the whole group of symbols toward the direction you need.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.





To push the diagram fragment

---

1. On the diagram palette, click the Pusher button.

2. On the diagram pane, click the diagram fragment you need to move and holding the right mouse button, drag it to the left or down.

To pull the diagram fragment

---

1. On the diagram palette, click the Magnet button.
2. On the diagram pane, click the diagram fragment you need to move and holding the right mouse button, drag it to the right or up.

Related pages

- Diagramming(see page 482)

Layout

> The diagram layout engine is available in Standard, Professional, Architect, and Enterprise editions.

It is easy to manage simple or complex diagrams using the automated layout features that optimize diagram layout.

MagicDraw automatically layouts symbols on diagrams when displaying related elements, creating diagrams by using wizards, when creating related diagrams.

You can layout diagrams quickly by using the Quick Diagram layout. The quick layout is a recommended type of layout, and it is specific for each diagram.

The layout mechanism is built on various layout tools. There are 2 groups of layout tools: general layout tools and specific diagram layout tools. These are the general layout tools:

- Orthogonal Layout
- Hierarchic Layout
- Organic Layout
- Tree Layout
- Circular Layout
- Oblique Path Router
- Rectilinear Path Router

These layout tools are provided by yFiles layout tool component. You can arrange each diagram (except Sequence diagram) by using any of the following general layout tools:

- Class Diagram Layout
- Composite Structure Diagram Layout
- Activity Diagram Layout
- State Machine Diagram Layout
- Business Process Diagram Layout

To layout diagrams by using the quick diagram layout, do one of the following

---

- Press Ctrl+Q.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- On the diagram toolbar, click the Quick Diagram Layout button.
- On the **Layout** menu, click **Quick Diagram Layout**.

To layout diagrams by using layout tools, do one of the following

---

- On the **Layout** menu, click one of the layout tools.
- On the diagram toolbar, click the small arrow next to the Quick Diagram Layout button, and select one of the layout tools.
- In the diagram shortcut menu, select **Layout**, and then select one of the layout tools.

To change the diagram layout options

---

1. On the **Layout** menu, click **Layout Options**.
2. In the **Diagram Layout Options** dialog, specify the layout options.
3. Click **OK**.

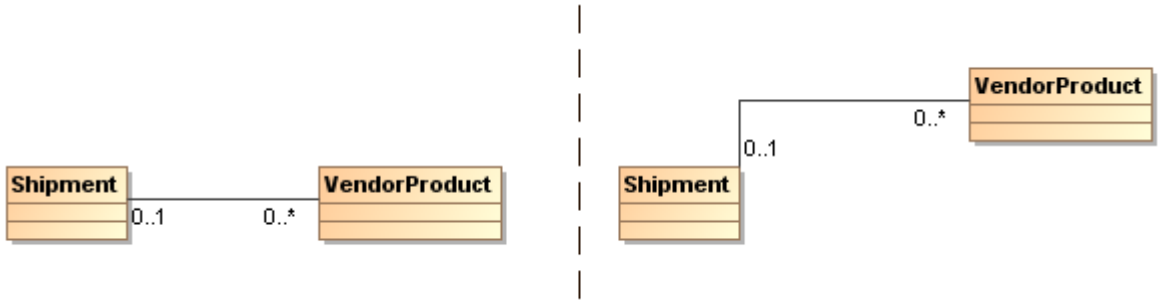
Related pages

- Label layout in the diagram(see page 580)
- Label orientation(see page 582)
- Diagramming(see page 482)

Label layout in the diagram

Default label positions

Default label positions are retained after moving a path, shape, or related element if it is a semantically logical decision. See the example below, where association multiplicities remain at their default positions after class is moved.



> ⚠ If labels are at their default position, reset labels position functionality is not available.

Label positions after moving a path, shape, or related element

After moving a path, shape, or related element, default label positions are retained if it is a semantically logical decision.

For a nicer representation of labels in a diagram, the label positions are reset to default automatically if:

1. Symbol properties are edited. When symbol properties are edited by adding or removing a label text box in the diagram pane, label positions are reset.
2. Path, path end, or port properties are edited. When path, path end, or port data are edited by adding or removing label text box in the diagram pane, label positions are reset.
3. Path, shape, or related element are moved.

To check the label deviation from its default position

- Select a label and begin to move it. The dashed line shows the deviation from the default position. This helps to show the current label's owner.

> When the label is not in its default position, you may indicate it by the icon on the bottom right corner, which appears after selecting the label's owner.

Label consideration mode

When arranging shapes for the hierarchic layout mechanisms, you may choose the **Label Consideration** mode:

- Hierarchic Layout.
- Class Diagram Layout.
- Composite Diagram Layout.
- Activity Diagram Layout.
- State Machine Diagram Layout.
- Business Process Diagram Layout.
- Ordered Hierarchic Layout.

In order to do this, you must specify the **Label consideration** mode (Full, Partial, or None) in the **Diagram Layout Options** dialog.

To specify the label consideration mode

---

1. From the diagram toolbar, expand and select **Layout Options** command.
2. On the left side of the dialog, select one of the following layout style:
   - Under the **General Layout** group, select **Hierarchic Layout**.
   - Under the **Specialized Layout** group, select any layout style:
     - Class Diagram Layout.
     - Composite Diagram Layout.
     - Activity Diagram Layout.
     - State Machine Diagram Layout.
     - Business Process Diagram Layout.
     - Ordered Hierarchic Layout.
3. On the right side of the dialog, find the **Label consideration** option.
4. Click on the value of this option and select one of the following modes:
   - **None** - does not consider any labels, does not increase the shape size.
   - **Partial** - considers only the center path labels, does not increase shape sizes by labels.
   - **Full** - considers all labels, creates additional space for them and increases the shape sizes.
5. Click the **Layout** button.
6. Click **OK**.

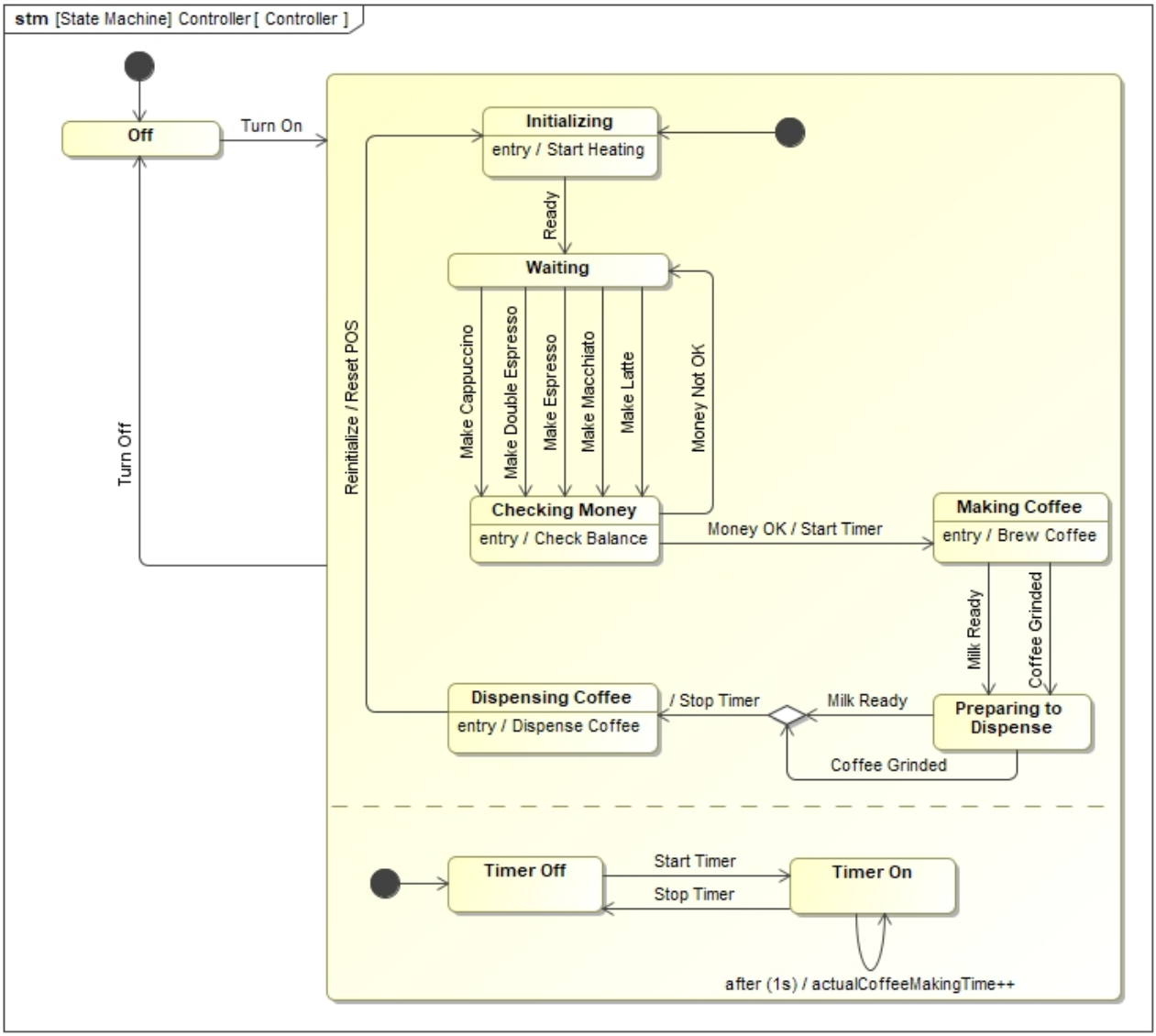
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Label orientation

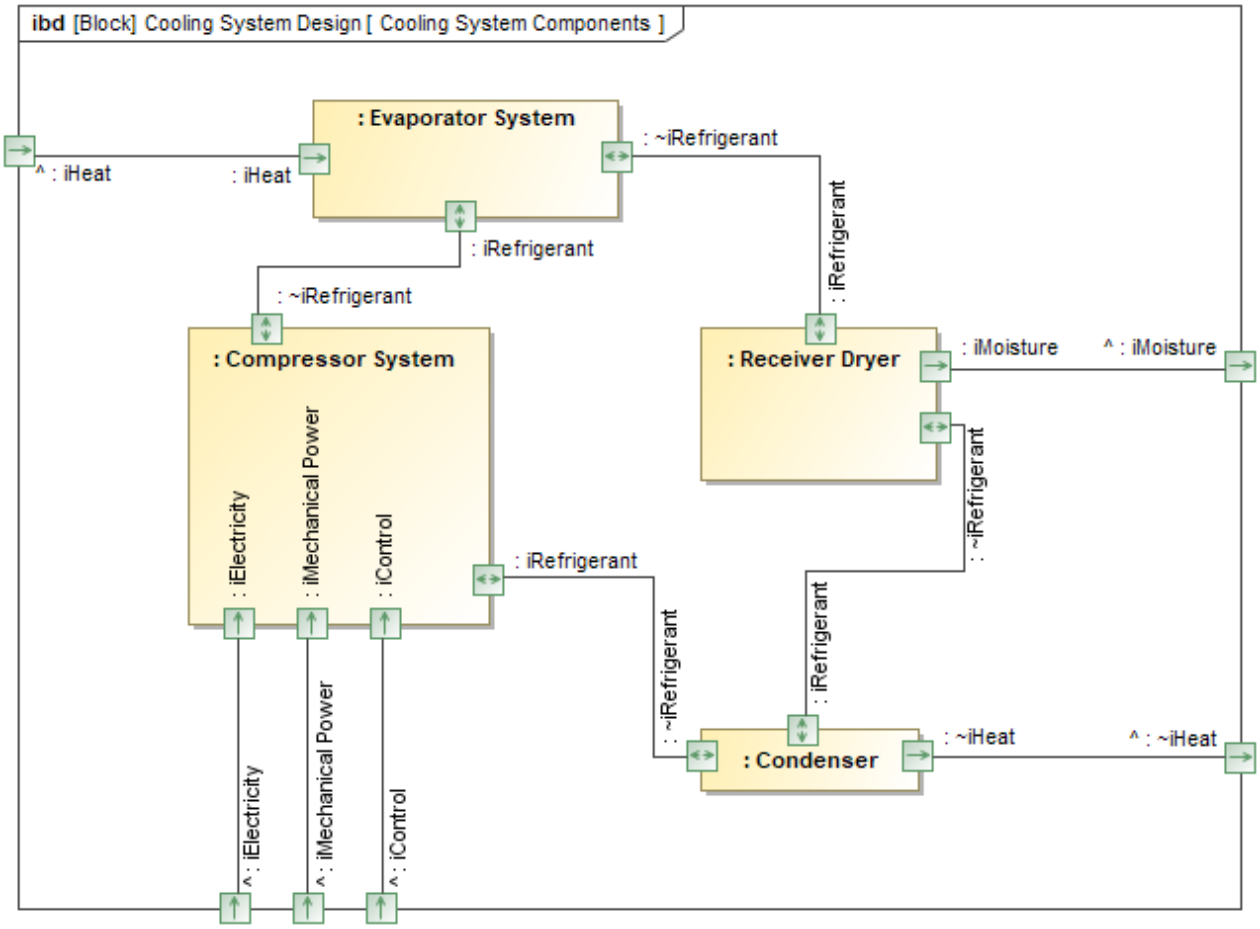
On this page

- Default label orientation(see page 584)

You can optimize diagram layout by changing the orientation of symbol labels. Rotate labels horizontally or vertically, or select the automatic label orientation option, which adjusts labels according to the position of shapes or paths.



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



This sample diagrams illustrate how changing label orientation can optimize diagram layout.

To change the orientation of independent labels

1. In an open diagram, select one or several labels that you want to rotate.
2. Do one of the following:
   - In the diagram toolbar, click to change the label orientation from horizontal to vertical or vice versa.
   - In the diagram toolbar, click the arrow next to and from the menu select the desired label orientation option: **Horizontal**, **Vertical**, or **Automatic**.

The labels are rotated to the selected position. If you select the **Automatic** label orientation option, labels are rotated according to the position of shapes or paths they are on.

To change the orientation of labels on a shape or path

1. In an open diagram, select the path or shape on which you want to rotate labels.
2. Do one of the following:

- In the diagram toolbar, click to change the label orientation from horizontal to vertical or vice versa.
- In the diagram toolbar, click the arrow next to and from the menu select the desired label orientation option: **Horizontal**, **Vertical**, or **Automatic**.

The labels on the shape or path are rotated to the selected position. If you select the **Automatic** label orientation option, labels are rotated according to the position of the shape or path.

> **Label Orientation symbol property**
> When you change the orientation of individual labels or labels displayed on a shape or path, the value of the **Label Orientation** symbol property of the related shapes or paths changes as well. To open the **Symbol Properties** dialog, right-click a symbol in a diagram and select **Symbol Properties**.

Default label orientation

By default, all labels on shapes and paths are displayed horizontally. You can change the default label orientation for a project as explained below.

To change the default label orientation

---

1. In the main menu of a modeling tool, select **Options** > **Project**.
2. On the left side of the **Project Options** dialog, select the **Shapes** project option group under the **Default** node.
3. On the right side of the dialog, click the property value box of the **Label Orientation** property and select one of the following values:
   - **Horizontal** - to display the labels of newly created shapes horizontally.
   - **Vertical** - to display the labels of newly created shapes vertically.
   - **Automatic** - to display the labels of newly created shapes according to their position in a diagram.
4. On the left side of the **Project Options** dialog, select the **Paths** project option group under the **Default** node.
5. On the right side of the dialog, click the property value box of the **Label Orientation** property and select one of the following values:
   - **Horizontal** - to display the labels of newly created paths horizontally.
   - **Vertical** - to display the labels of newly created paths vertically.
   - **Automatic** - to display the labels of newly created paths according to their position in a diagram.
6. Click **OK** on the bottom of the **Project Options** dialog.

The labels of newly created symbols in the diagrams of a project will be displayed according to your selected property values.

Saving as image

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

You can save an image as an active diagram, selected symbols, or selected diagrams.

Diagrams and symbols that were created in the model can be saved as images in the following formats:

- Enhanced Metafile Format (*.emf) (supports language specific symbols)
- Encapsulated PostScript (*.eps)
- Joint Photographic Experts Group (*.jpg, *.jpeg)
- Portable Network Graphics (*.png)
- Scalable Vector Graphics (*.svg)
- Tagged Image File Format (*.tif, *.tiff)
- Windows Metafile Format (*.wmf)

To save an image

1. Open the diagram you want to save as an image. If you want to save only particular symbols, select them on the diagram.
2. On the main menu, select **File** > **Save as Image**. The **Save As Image** dialog(see page 585) opens.
3. In the dialog:
   - Select whether to save an **Active diagram**, **Selected symbols**, or **Selected diagrams**.

     Selected diagrams
     Selected diagrams allow you to search for a particular diagram(see page 1033) in your model.

   - Specify the directory you want to save the image to.
   - Select the image format. Click **Options** to specify image options(see page 587) according to the selected format.
4. Click **Save**. The image is saved in the directory you have specified.

Related pages

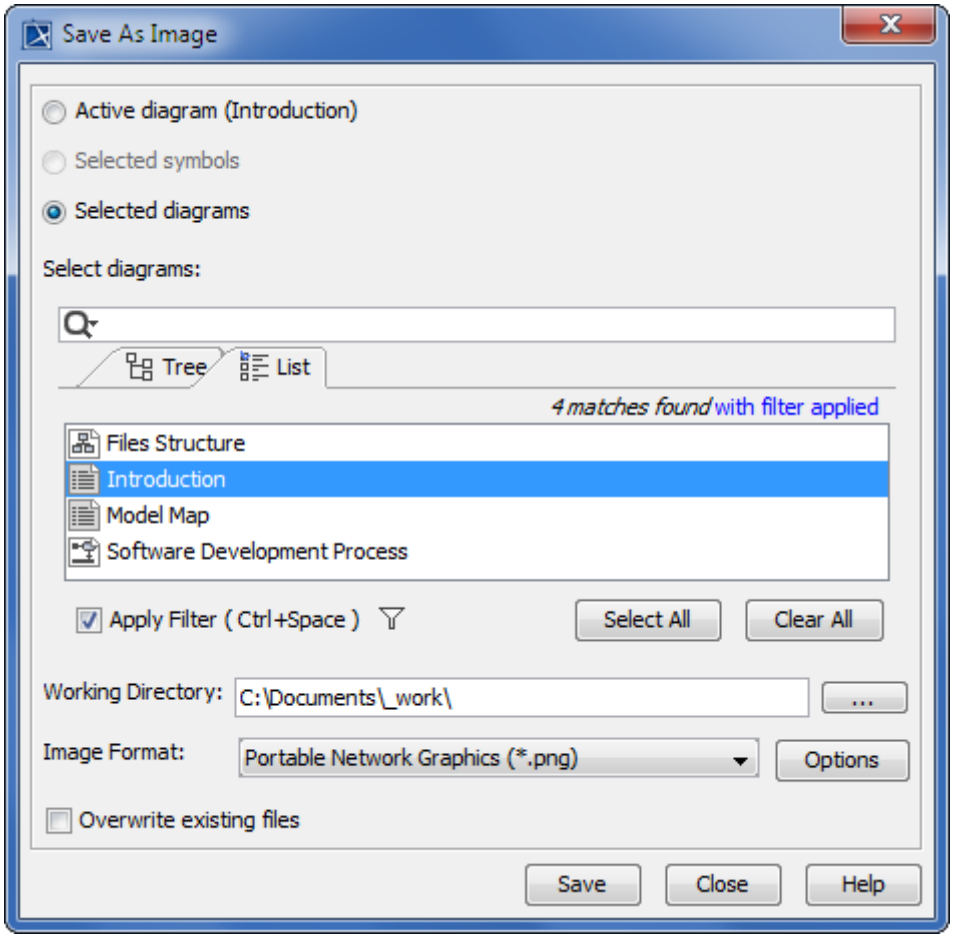
- Save As Image dialog(see page 585)
- Properties dialog(see page 587)
- Diagramming(see page 482)

Save As Image dialog

The **Save As Image** dialog allows you to save one or multiple diagrams of your model as images. Also, you can save only the selected symbols instead of an entire diagram, and choose the desired image format.

To open the **Save As Image** dialog

- From the **File** menu, select **Save as Image**.



The view of the Save As Image dialog when the Selected diagrams option button is selected.

Dialog element	Element type	Description
Active diagram	Option button	Select to save as image the currently active diagram. The active diagram name is displayed in brackets next to the option button name.
Selected symbols	Option button	Select to save as image specific symbols that are selected in the diagram. The option is active only if one or multiple symbols are selected in the diagram.
Selected diagrams	Option button	Select the option in order to save as image the selected diagrams in the model tree or diagram list.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Dialog element	Element type	Description
Select diagrams	Text box	The text box is available only if the **Selected diagrams** option button is selected. In the **Select diagrams** box, type the name of the diagram you want to find. Search results are displayed as a model tree or a plain list right under the text box. Click the **Tree** or **List** tab in the search result area to switch to the desired view, and select the diagram(s) you want to save as image(s). To optimize your search, use search modes and scope filters(see page 1033). The **Apply Filter** check box and the filter options button are located below the search result area.
Image File / Working Directory	Text box	Select the working directory in which the image will be saved.
Image Format	Drop-down list	Select the image format.
Options	Button	Click to open the Properties dialog(see page 587) where you can specify image export properties, such as image resolution and exported image size. The properties provided in the **Properties** dialog depend on the selected image format.
Overwrite existing files	Check box	Select to overwrite already saved image files.

Rrelated pages

- Saving as image(see page 584)
- Properties dialog(see page 587)

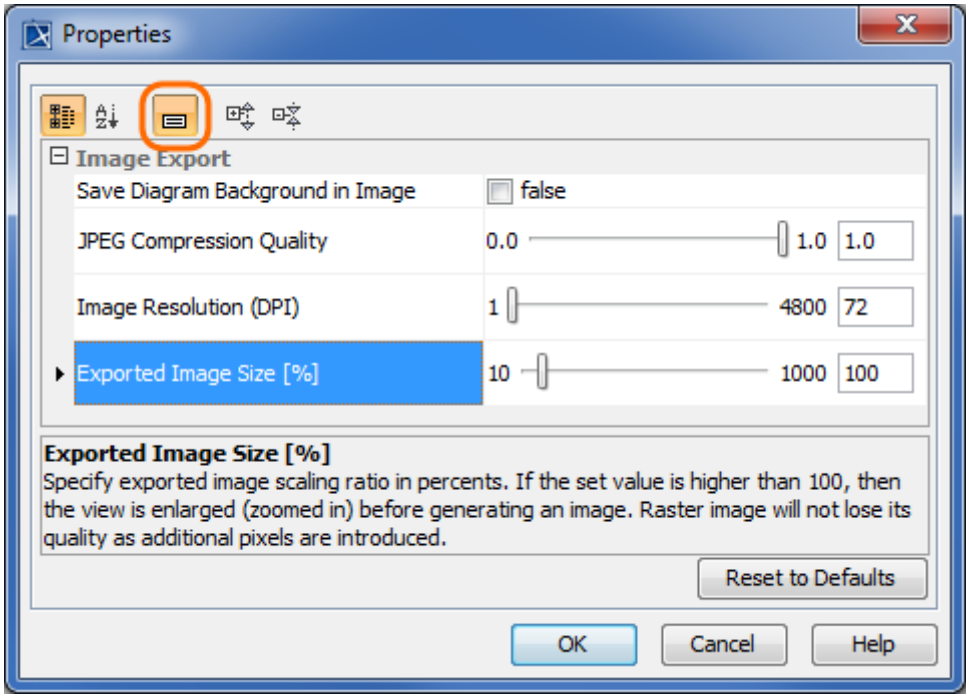
Properties dialog

In the **Properties** dialog, you can specify image export properties specific to the image format selected in the S(see page 585)ave(see page 585) A(see page 585)s(see page 585) Image(see page 585) dialog. For example, if the selected image format is .jpg, or .jpeg, the **Properties** dialog allows you to specify JPEG compression quality, image resolution (DPI), and exported image size.

To specify image export options

1. From the **File** menu, select **Save as Image**.
2. In the **Save As Image** dialog, specify the desired image format, and click the **Options** button.
3. In the **Properties** dialog, specify given properties.
4. Click **OK**.

> **Property related information**
> If you need more information about the properties provided in the **Properties** dialog, make sure that the **Show Description** button is pressed (as shown in the figure below), then select the desired property, and read its description in the description area on the bottom of the dialog.



This Properties dialog displays the image export properties specific to the .jpg, .jpeg image format. If the highlighted button is pressed, the description of the selected property is provided in the description area on the bottom of the dialog.

> **Image Resolution and Alternative specification method**
> The default image resolution for Retina Display is 144 DPI, and the default image resolution for other displays is 72 DPI. The resolution and other export properties can be changed manually in the **Environment Options** dialog(see page 65). Image export properties are located under the **General** group.

Related pages:

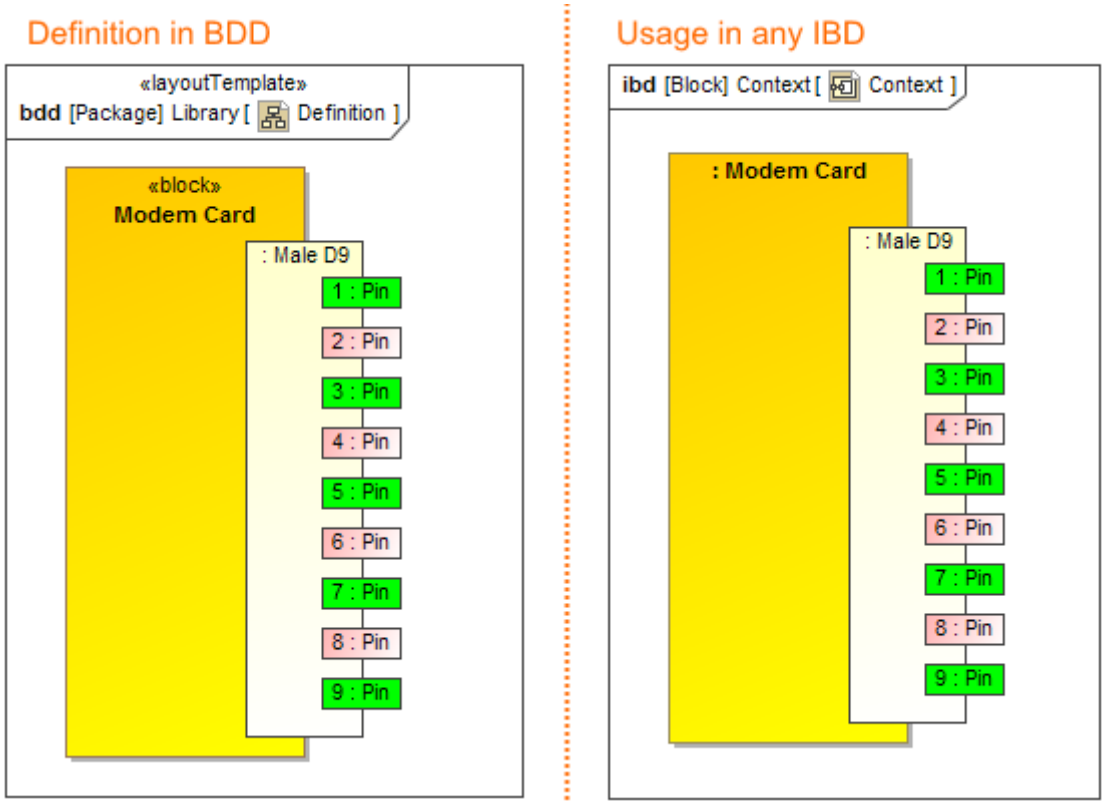
- Saving as image(see page 584)
- Save As Image dialog(see page 585)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Layout templates

The layout templates functionality, which is driven by <<layoutTemplate>> stereotype, assists you in creating well-formed diagrams. It allows you to define Class appearance once, and reuse it later when creating Parts typed by that Class and Ports in different diagrams. Parts(see page 804) and their Ports are represented identically as part's type Class(see page 666) whose layout and style is defined in the layout template diagram. Often, Port position reflects its physical location in the system, so it is beneficial to maintain the same appearance of Parts, layout, and style of Ports throughout the whole model. In the figure below, the appearance of the *Modem Card* Block is the same in two different diagrams, SysML Block Defiition diagram[86] and SysML Internal Block diagram[87].

> **Additional information**
> You can reuse layout templates in other projects by exporting packages to new projects.(see page 237)



Usage of a layout template. The illustration displays concepts from SysML Plugin.

You can create a layout template in the Class Diagram or derive it from an existing Composite Structure Diagram as described below.

86 https://docs.nomagic.com/display/SYSMLP190SP4/SysML+Block+Definition+Diagram
87 https://docs.nomagic.com/display/SYSMLP190SP4/SysML+Internal+Block+Diagram

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To define a layout template in the Class Diagram and apply it to other diagrams:

1. Define the appearance of Class, layout, and style of its Ports in the Class diagram.(see page 590)
2. Set that Class diagram as the layout template.(see page 592)
3. Apply the layout template to other diagrams.(see page 597)

To derive a layout template for the Composite Structure Diagram and apply it to other diagrams:

1. Specify whether layout templates are created as Class Diagrams or Composite Structure Diagrams.(see page 595)
2. Create a layout template from a Part Property in the Composite Structure Diagram.(see page 594)
3. Apply the layout template to other diagrams.(see page 597)

Important
Only one Class shape with its Ports can be defined in one layout template diagram.

Read the following pages to learn how to manage layout templates:

- Stop using Class diagram as a layout template[88]
- Removing applied layout template from shape(see page 600)
- Multiple layout template(see page 600)
- Setting layout template as default(see page 603)
- Navigation to layout templates(see page 606)
- Using layout templates with diagram aspects(see page 607)

Related pages

- Class diagram(see page 297)
- Composite Structure diagram(see page 325)
- Diagram aspects(see page 608)
- Exporting packages to new projects(see page 237)

Sample model

The sample models used in the figure of this page are **Modem Cable** that comes with SysML Plugin[89]. To open those samples properly you need to install the SysML plugin[90] in the MagicDraw and download Modem Cable Layout Templates.mdzip[91].

Defining layout template

88 https://docs.nomagic.com/display/MD190SP4/Stop+using+Class+diagram+as+a+layout+template
89 https://docs.nomagic.com/display/SYSMLP190SP4/SysML+Plugin+Documentation
90 https://docs.nomagic.com/display/NMDOC/Installing+plugins
91 https://docs.nomagic.com/download/attachments/55847477/Modem%20Cable%20Layout%20Templates.mdzip?api=v2&modificationDate=1481817717837&version=1

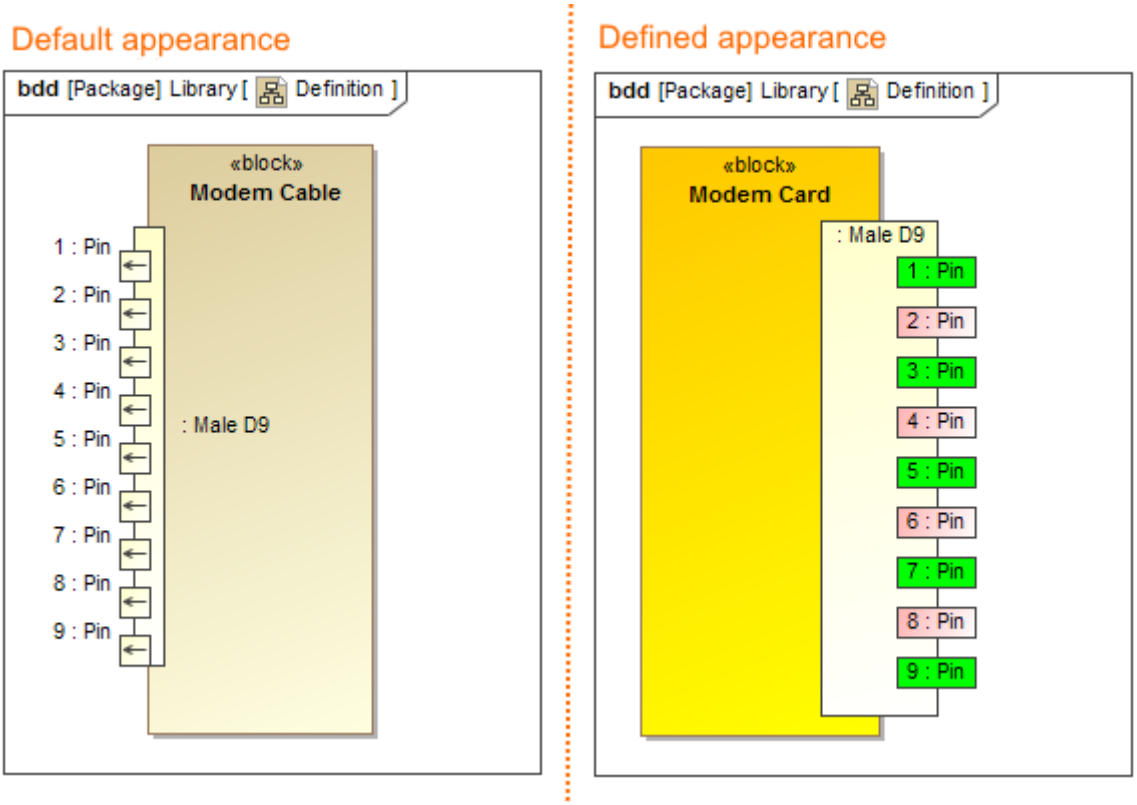
If you want to reuse the style and layout of Class with its Ports in another diagram, you need to define the appearance of the Class, the style, and position of its Ports in the Class diagram. It means that you define a new layout template. When defining the appearance of the Class, you can change its color, font, size, layout, and other style properties. To do this, follow the procedure below.

> **Important**
> Only one Class shape with its Ports can be defined in one layout template diagram.

To define the layout template

1. Create the Class diagram. How to create diagrams >(see page 483)
2. Create a new or represent an existing Class with its Ports on the diagram pane.
3. Specify the appearance of all shapes by using the Symbol Properties dialog(see page 975) and diagram toolbar(see page 24).
4. Set the positions of Ports on Class shape manually.
   The appearance of Class and its Ports shapes are defined (see the figure below).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The comparison of default and defined appearance of the Modem Card Block, style, and the position of its ports. The illustration displays concepts from SysML Plugin.

Composite Structure Diagram as a layout template

- From the version 19.0 SP1, you can define the appearance and layout of a Part Property with its Ports in the Composite Structure Diagram and set this diagram as layout template.
- If a Composite Structure Diagram displays several Part Properties, create a layout template(see page 594) directly from a particular Part Property. It's appearance and layout will be copied to the template automatically.

Sample model

The sample models used in the figure of this page are **Modem Cable** that comes with SysML Plugin[92]. To open those samples properly you need to install the SysML plugin[93] in the MagicDraw and download Modem Cable Layout Templates.mdzip[94].

Setting diagram as layout template

92 https://docs.nomagic.com/display/SYSMLP190SP4/SysML+Plugin+Documentation
93 https://docs.nomagic.com/display/NMDOC/Installing+plugins
94 https://docs.nomagic.com/download/attachments/9915051/Modem%20Cable%20Layout%20Templates.mdzip?api=v2&modificationDate=1481817717837&version=1

After the layout template (appearance of the required Class and its Ports in the Class diagram) is defined, you can now use it as the layout template, which you can reuse in other diagrams. The procedure below describes how to set the diagram as the layout template. Note that the layout template name automatically becomes the same as diagram name. This will be important when applying the layout template, and choosing which template you want to apply.

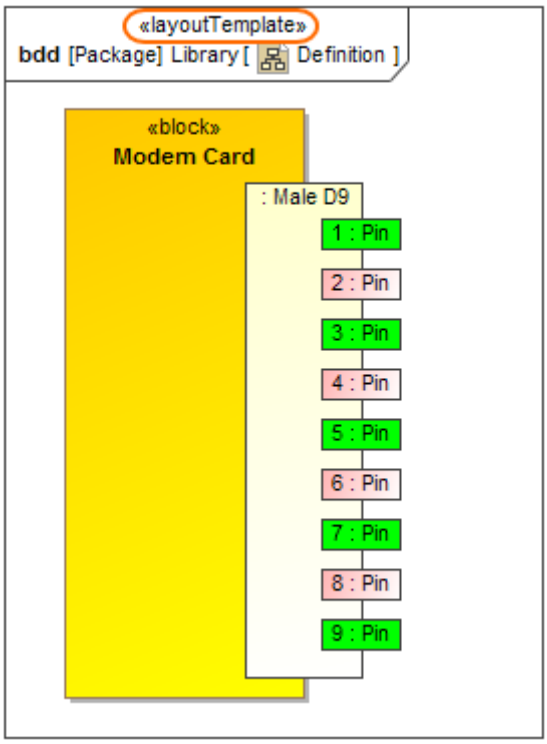
To set the Class Diagram as a layout template

---

1. Open the Class Diagram that you want to set as layout template.
2. Do one of the following:
   - Right click the diagram and select **Display** > **Layout Template** > **Use as Layout Template**.
   - Ensure that nothing is selected on the diagram pane(see page 24), then in the diagram toolbar, click and select **Layout Template** > **Use as Layout Template**.

The «layoutTemplate» stereotype appears on the diagram header (see the figure below).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The Definition BDD is set as the layout template named Definition. The illustration displays concepts from SysML Plugin.

Composite Structure Diagram as a layout template

- From the version 19.0 SP1, you can define the appearance and layout of a Part Property with its Ports in the Composite Structure Diagram and set this diagram as layout template.
- If a Composite Structure Diagram displays several Part Properties, create a layout template(see page 594) directly from a particular Part Property. It's appearance and layout will be copied to the template automatically.

Sample model

The sample models used in the figure of this page are **Modem Cable** that comes with SysML Plugin[95]. To open those samples properly you need to install the SysML plugin[96] in the MagicDraw and download Modem Cable Layout Templates.mdzip[97].

Creating layout template

When you create a Composite Structure Diagram and finish perfecting its layout and looks, you can reuse the same layout in other diagrams of your model by creating layout templates. Create layout templates directly from Composite Structure Diagrams as described below.

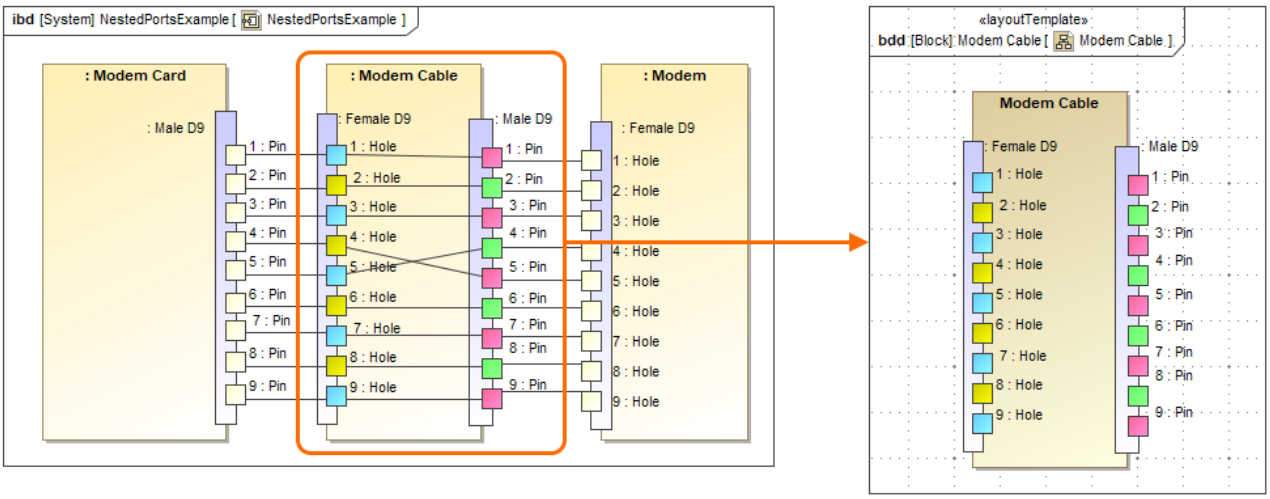
95 https://docs.nomagic.com/display/SYSMLP190SP4/SysML+Plugin+Documentation
96 https://docs.nomagic.com/display/NMDOC/Installing+plugins
97 https://docs.nomagic.com/download/attachments/9915051/Modem%20Cable%20Layout%20Templates.mdzip?api=v2&modificationDate=1481817717837&version=1

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To create a layout template

1. Create a Composite Structure Diagram with Part Properties and their Ports or open an existing one.
2. Right click the Part Property the the layout of which you want to reuse and select **Display** > **Layout Template** > **Create Template**.

A layout template is created as the Class or Composite Structure Diagram depending on the layout template creation mode(see page 595) (see the figure below).



Creating a layout template from the SysML Internal Block Diagram (Composite Structure Diagram in a UML project).

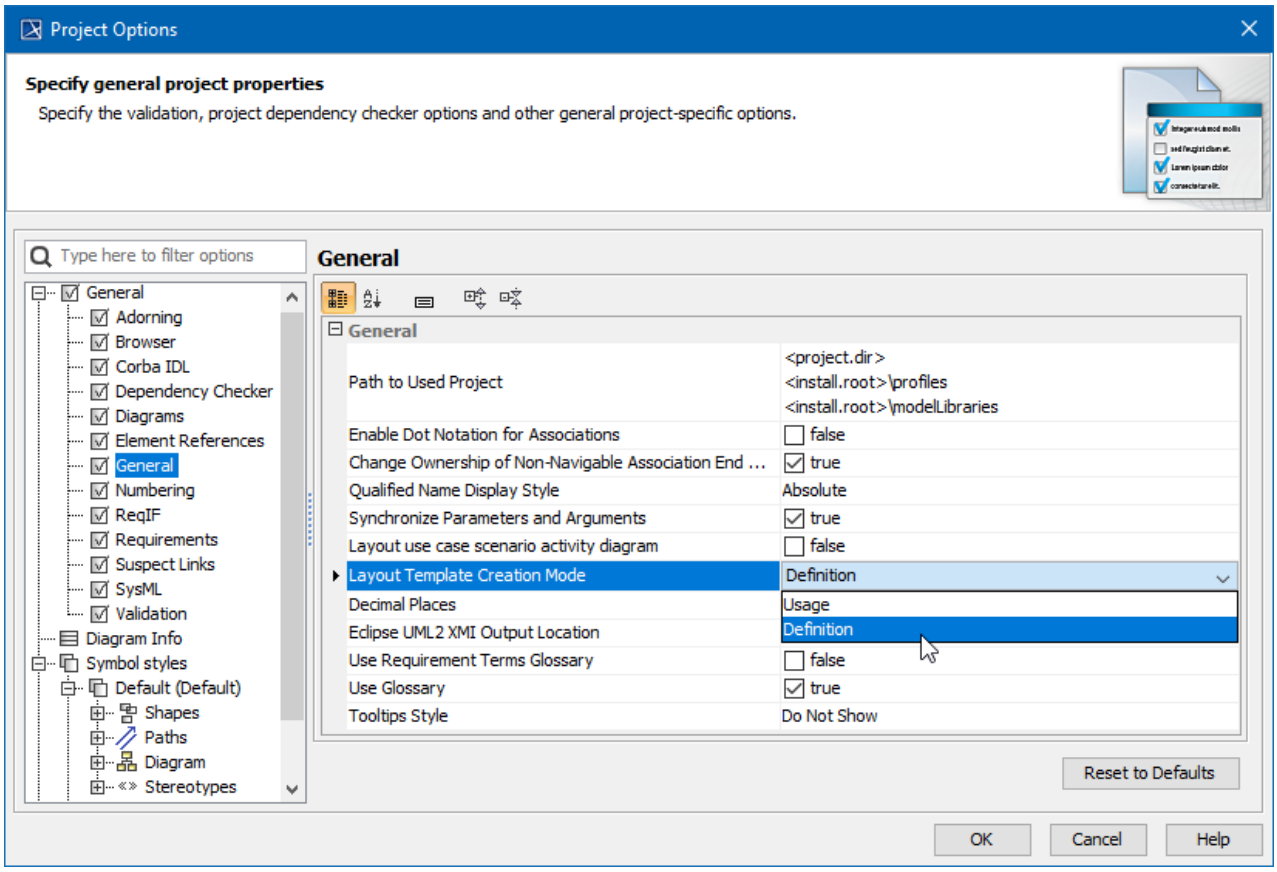
Layout template creation mode

Before creating layout templates from Composite Structure Diagrams, you can select whether layout templates are created as Class Diagrams (default option) or Composite Structure Diagrams. To do this, specify layout template creation mode as described below.

To specify layout template creation mode

1. In the main menu of your modeling tool, select **Options** > **Project**.
2. On the left side of the **Project Options** dialog, expand the **General** node and select the **General** option group.
3. Click the specification cell of the **Layout Template Creation Mode** option on the left side of the dialog and select one of the following values as shown below:
    - **Usage** - if you want to create layout templates as Composite Structure Diagrams.
    - **Definition** - if you want to create layout templates as Class Diagrams.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Specifying layout template creation mode.

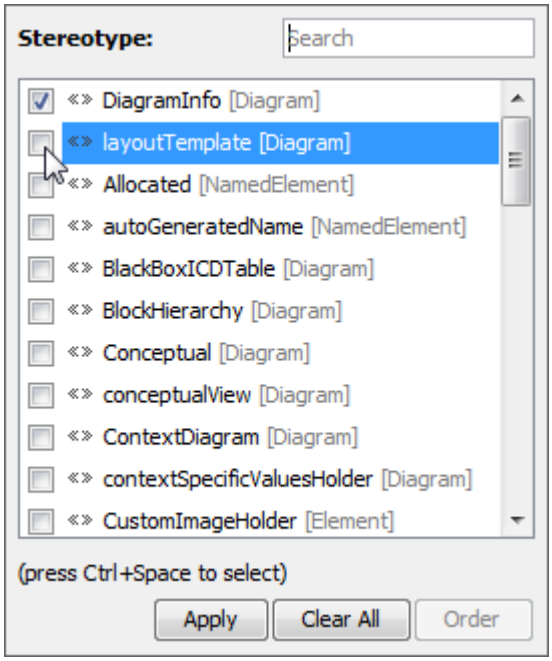
Stopping to use diagram as layout template

If you no longer want to use a diagram as a layout template, you need to remove the «layoutTemplate» stereotype from it. To do this, follow the steps below.

To stop using a diagram as a layout template

1. Ensure that nothing is selected on the diagram pane.(see page 24)
2. Right click the diagram and select **Stereotype**.
3. In the **Stereotype** dialog, clear the **layoutTemplate** check box.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



4. Click **Apply**.

The «layoutTemplate» stereotype disappears from the header of the diagram, and it can no longer be applied as a layout template.

Applying layout template

If you want to use the layout template in another diagram (usually the Composite Structure diagram), the following conditions must be met beforehand:

- The Part must be typed by the Class or subtype of Class in the Composite Structure diagram.
- The appearance of the Part type Class or subtype of Class must be defined in the Class diagram.
- That Class diagram must be set as the layout template.

You can apply layout templates directly from the diagram toolbar or shape's shortcut menu by using the **Apply Template** command. This command provides different suggestions in the following situations:

- **Apply Template** without drop-down arrow appears when only one layout template is available for the selected element.
- **Apply Template** with drop-down arrow appears when multiple layout templates are available for the selected element. The multiple layout templates are listed according to the following rules:

If you set the default layout template, it is always the first on the list, and other templates are listed alphabetically. Click to learn how to set the default layout template(see page 603). If you have not set the default layout template, all layout templates are listed alphabetically.

The procedures below explain how to apply the layout template to:

- Part(see page 598)
- Class and all classifiers(see page 598)
- Diagram frame(see page 599)

Applying a layout template to the Part

There are several different ways you can apply a layout template to a Part. Use one of the workflows below depending on your situation.

To apply a layout template to the Part Property displayed in the Composite Structure Diagram

---

- Do one of the following:
  - Right click the Part Property shape to which you want to apply the layout template and select **Display** > **Layout Template** > **Apply Template**. Then select the name of the desired layout template if you can apply more than one template.
  - In the diagram toolbar, click and select **Layout Template** > **Apply Template**. Then select the name of the desired layout template if you can apply more than one template.

To apply a layout template when creating a new Part Property in the Composite Structure diagram

---

- Drag the Class that is the type of the Part Property you want to create on the diagram pane and do one of the following:
  - Drop the Class.
  - Hold down the mouse button until the tooltip appears (long drag), then release the mouse button and select the **Create Property Symbol and Apply Default Template** command.

Applying a layout template to the Class and all classifiers

You can apply a layout template for the Class and all classifiers that are set as the layout template, but are represented in other Class diagrams. To do this, follow the procedure below.

To apply a layout template to the Class and all classifiers

---

1. Select the Class shape or another classifier shape that has the layout template defined.
2. Do one of the following:

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Right click the shape and select **Display** > **Layout Template** > **Apply Template**. Then select the name of the desired layout template if you can apply more than one template.
- In the diagram toolbar, click and select **Layout Template** > **Apply Template**. Then select the name of the desired layout template if you can apply more than one template.

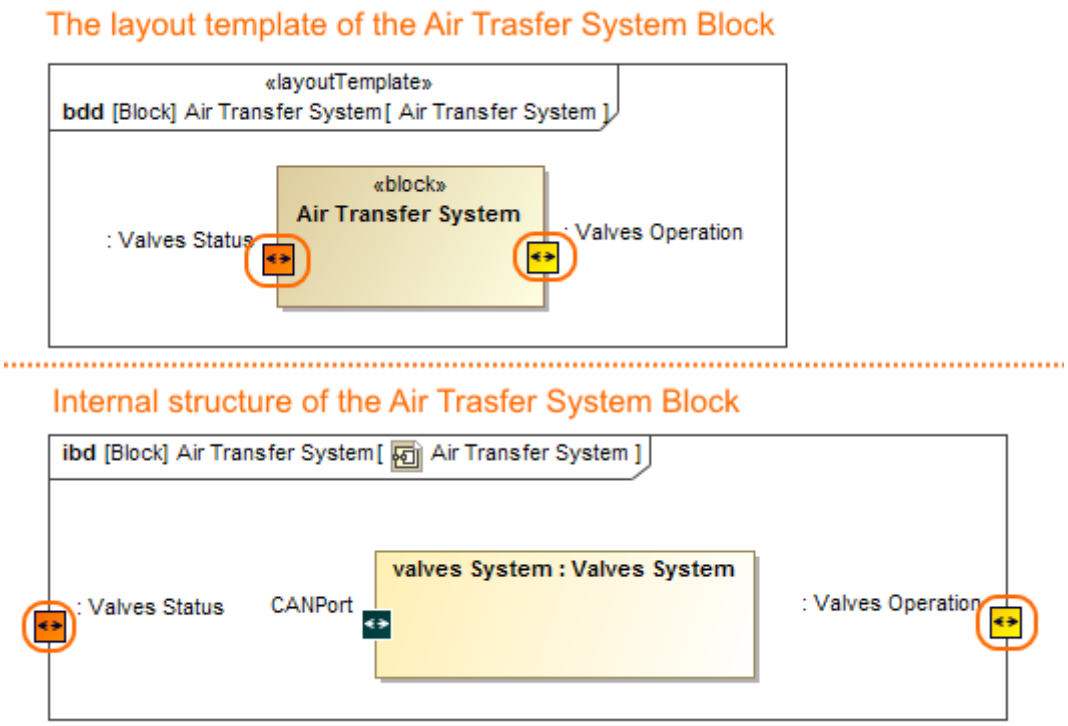
Applying a layout template to the diagram frame

When creating the internal structure for the Class that has a layout template defined, you can keep the same layout and style of its Ports on the Composite Structure Diagram frame.

To apply a layout template to the diagram frame

1. Select the diagram frame.
2. Do one of the following:
   - Right click the diagram and select **Display** > **Layout Template** > **Apply Template**. Then select the name of the desired layout template if you can apply more than one template.
   - In the diagram toolbar, click and select **Layout Template** > **Apply Template**. Then select the name of the desired layout template if you can apply more than one template.

The example below illustrates how the appearance of Ports defined in *the Air Transfer System* layout template is applied to the Internal Block Diagram of the *Air Transfer System* Block.



The layout template applied on the *Air Transfer System* Internal Block Diagram frame.

Sample model

The sample models used in the figure of this page are **Modem Cable** that comes with SysML Plugin[98]. To open those samples properly you need to install the SysML plugin[99] in the MagicDraw and download Modem Cable Layout Templates.mdzip[100].

Removing applied layout template from shape

You can remove the applied layout template from the Part, Class, or classifier shape, if you want to see their predefined style notations.

To remove an applied layout template from a shape

- Revert the Apply Template command by using the **Undo** button from the main toolbar(see page 24), or press Ctrl+Z.
- Delete the shape and display it again by using the **Display Parts/Ports** dialog.(see page 491) Ensure that the **Use Layout Template** check box is clear.

Multiple layout template

You can represent the same Class with a different appearance according to the context of Class usage in the Composite Structure diagram. To accomplish this, define multiple diagrams for the same Class element with different names, and set them as layout templates. The diagram names identify the differences between layout templates and simplify layout template selection when applying it.
The procedures below explain how to create and apply multiple layout templates.

To create multiple layout templates

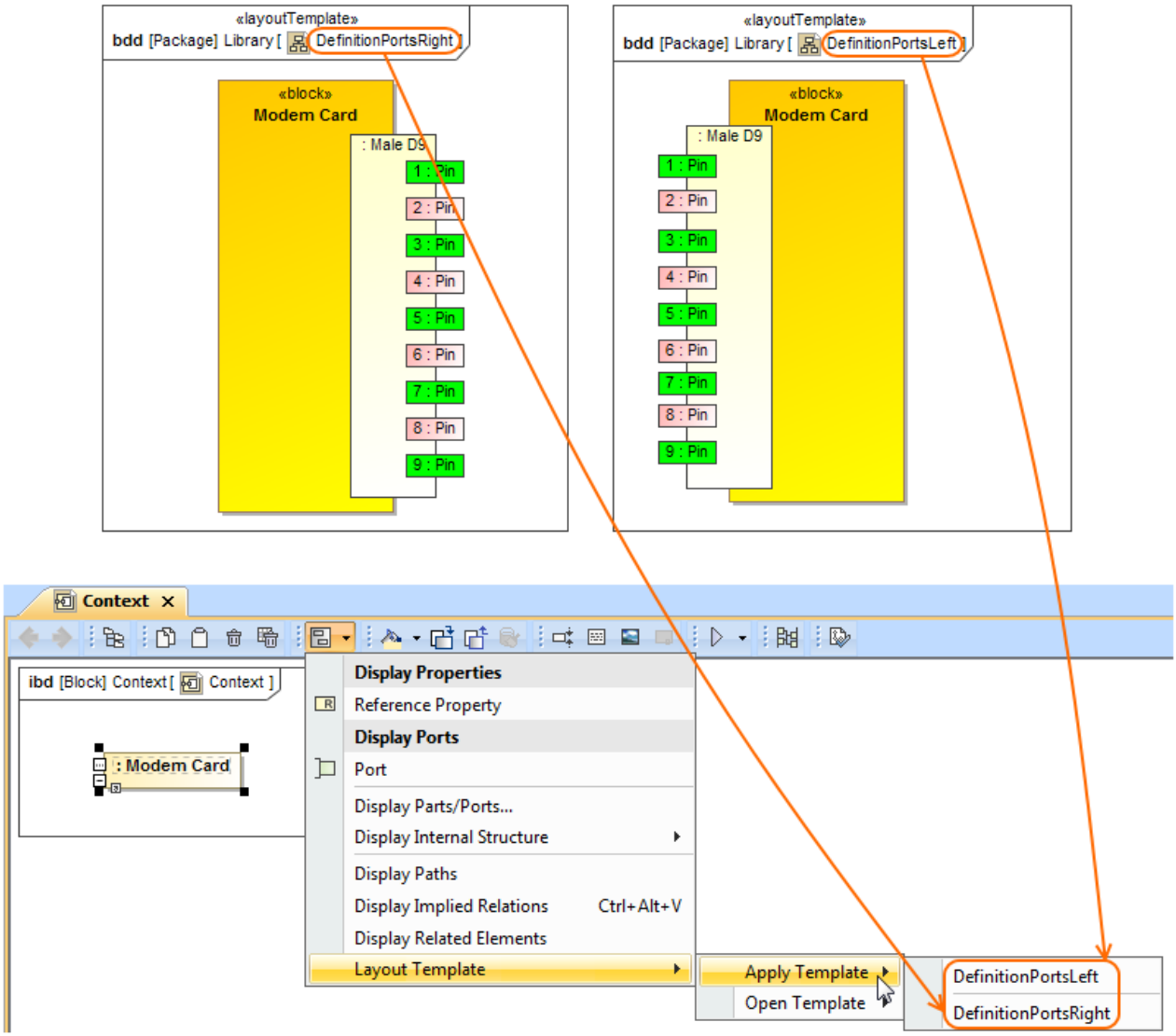
1. Create the Class diagram with a name that identifies the layout template name.
2. Define the appearance of Class and its Ports shapes in the Class diagram. How to define the layout template >>(see page 590)
3. Set that Class diagram as the layout template. How to set the Class diagram as layout template >>(see page 592)
4. Repeat steps 1, 2, and 3 until you have the required number of layout templates of the same Class element.
5. Apply multiple layout templates to other diagrams. How to apply layout template >>(see page 597)

98 https://docs.nomagic.com/display/SYSMLP190SP4/SysML+Plugin+Documentation
99 https://docs.nomagic.com/display/NMDOC/Installing+plugins
100 https://docs.nomagic.com/download/attachments/9915051/Modem%20Cable%20Layout%20Templates.mdzip?api=v2&modificationDate=1481817717837&version=1

The example below shows how the port's position on the *Modem Card* Block is represented differently: on the left side and on the right side of the Block shape. According to the illustration, the names of the Block Definition Diagrams are appropriate: *DefinitionPortsRight* and *DefinitionPortsLeft*. You can find the same layout templates when trying to apply one of those layout templates for the Part shape in the Internal Block Diagram.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Two different layout templates of the Modem Card Block



Two different layout templates of the Modem Card Block can be found as selection when applying it in any Internal Block Diagram. The illustration displays concepts from SysML Plugin.

Composite Structure Diagram as a layout template

- From the version 19.0 SP1, you can define the appearance and layout of a Part Property with its Ports in the Composite Structure Diagram and set this diagram as layout template.
- If a Composite Structure Diagram displays several Part Properties, create a layout template(see page 594) directly from a particular Part Property. It's appearance and layout will be copied to the template automatically.

Sample model

The sample models used in the figure of this page are **Modem Cable** that comes with SysML Plugin[101]. To open those samples properly you need to install the SysML plugin[102] in the MagicDraw and download Modem Cable Layout Templates.mdzip[103].

Setting layout template as default

If you have a commonly used layout template, you can set it as the default. The default layout template will always be the first and separated by a line when trying to apply or open the layout templates. Only one layout template can be set as the default. The procedures below explain how to:

- Set the layout template as default template(see page 603).
- Change the default layout template(see page 605).

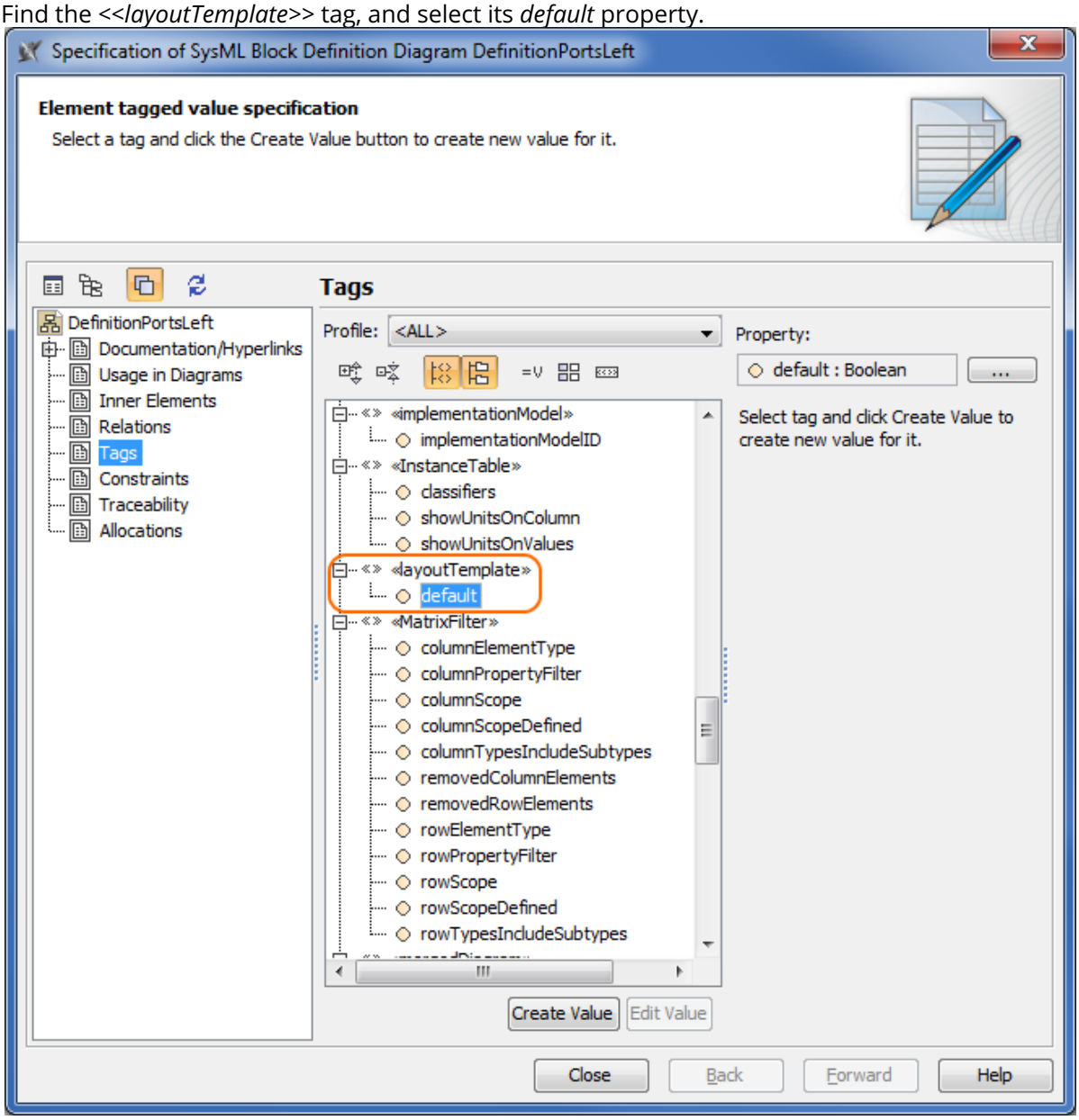
Setting the layout template as default template

To set the layout template as default template

1. From the diagram shortcut menu, select **Specification**.
2. In the Specification window(see page 913), select the **Tags** property group.

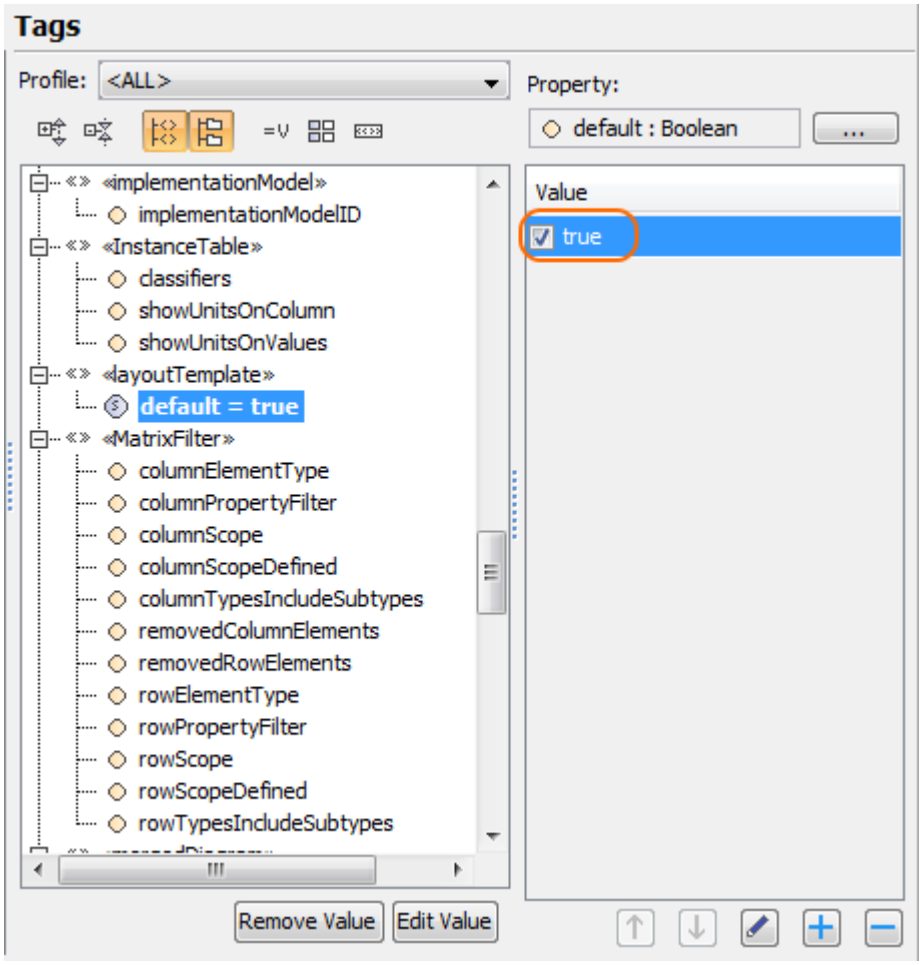
101 https://docs.nomagic.com/display/SYSMLP190SP4/SysML+Plugin+Documentation
102 https://docs.nomagic.com/display/NMDOC/Installing+plugins
103 https://docs.nomagic.com/download/attachments/9915051/Modem%20Cable%20Layout%20Templates.mdzip?api=v2&modificationDate=1481817717837&version=1

3. Find the <<*layoutTemplate*>> tag, and select its *default* property.



4. Click the **Create Value** button to create a new value for the <<*layoutTemplate*>> tag, if it is not already created.

5. Ensure that the **Value** check box is selected and set *to true*.



6. Click **Close**.
   The layout template diagram is now set as the default layout.

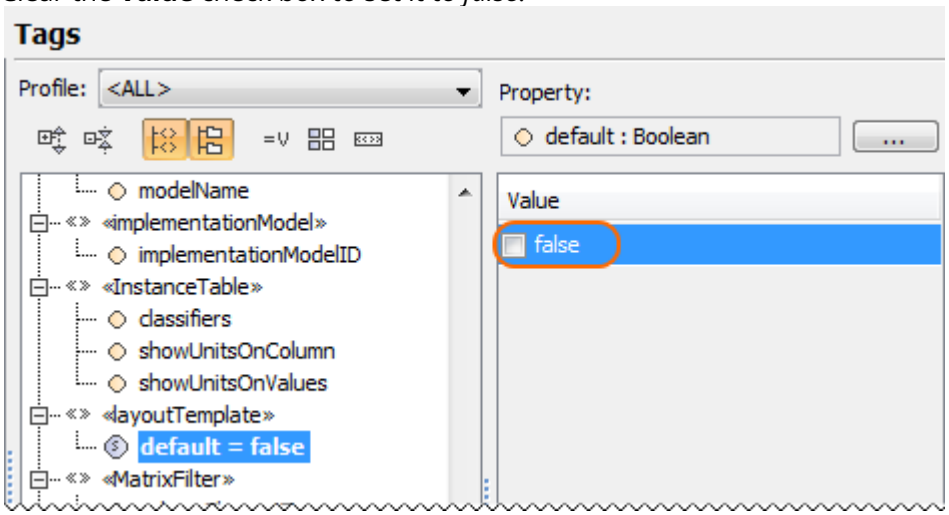
Changing the default layout template

To change the default layout template

---

1. Open the Specification window(see page 913) of the default layout template diagram.
2. Select the **Tags** property group.

3. Clear the **Value** check box to set it to *false*.



4. Click **Close**.
5. Open the layout diagram template that you want to be the default template.
6. Repeat all steps of the previous procedure(see page 603).

Sample model

The sample models used in the figure of this page are **Modem Cable** that comes with SysML Plugin[104]. To open those samples properly you need to install the SysML plugin[105] in the MagicDraw and download Modem Cable Layout Templates.mdzip[106].

Navigation to layout templates

You can navigate from the layout template usage to definition by using the Open Template command. This command provides different options in the following situations:

- **Open Template** without drop-down arrow appears when only one layout template is available for the selected element.
- **Open Template** with drop-down arrow appears when multiple layout templates are available for the selected element. The multiple layout templates are listed according to the following rules:
  - If you set the default layout template, it is always the first in the list, and other templates are listed alphabetically. Click to learn how to set the default layout template.(see page 603)
  - If you haven't set the default layout template, all layout templates are listed alphabetically.

To open the layout template diagram

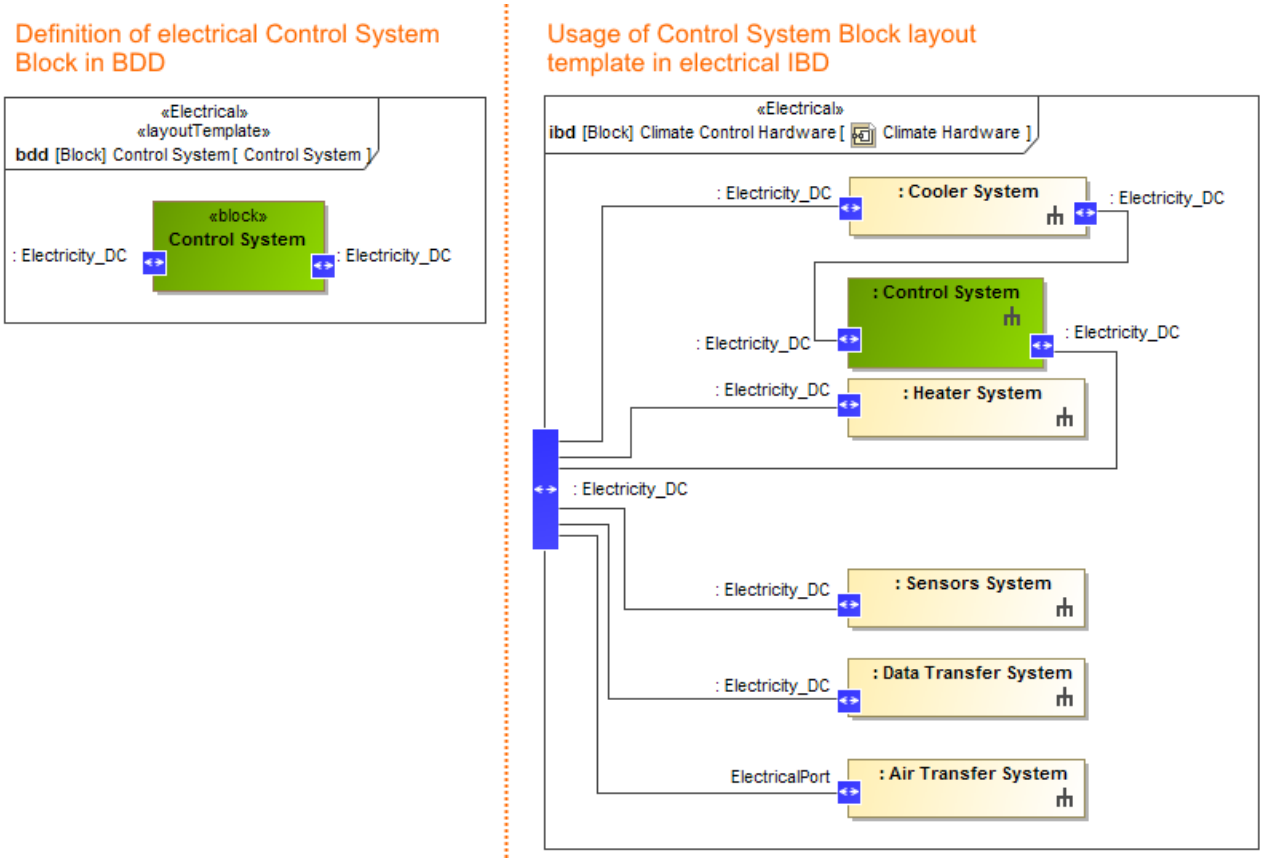
- Right click the Part Property or Class shape and select **Display** > **Layout Template** > **Open Template**. Then select the name of the desired layout template if more than one template is available.

104 https://docs.nomagic.com/display/SYSMLP190SP4/SysML+Plugin+Documentation
105 https://docs.nomagic.com/display/NMDOC/Installing+plugins
106 https://docs.nomagic.com/download/attachments/9915051/Modem%20Cable%20Layout%20Templates.mdzip?api=v2&modificationDate=1481817717837&version=1

- Select the Part Property or Class shape, click , and select **Layout Template** > **Open Template**. Then select the name of the desired layout template if more than one template is available.

Using layout templates with diagram aspects

The Class Diagram that is used as as a layout template can have a specific diagram aspect(see page 608) defined. In other words, the layout template can only be used in another diagram with the same aspect. In the example below, the *Control System* Block Definition Diagram is a layout template with the *electrical* aspect defined, so the *Control System* layout template can only be used in the *electrical* view of the system. In this case, the *Control System* layout template is used in the *Climate Hardware* Internal Block Diagram that is defined as electrical. This layout template can't be applied in the diagram with a different aspect defined, e.g., Communication.



Usage of a layout template with aspect. The illustration displays concepts from SysML Plugin.

Sample model

The sample models used in the figure of this page is **Diagram Aspects** that comes with SysML Plugin[107]. To open those samples properly you need to install the SysML plugin[108] in the MagicDraw and download diagram aspects.mdzip[109], or find in modeling tool *<modeling tool installation directory>\samples\SysML\diagram aspects.mdzip.*

Diagram aspects

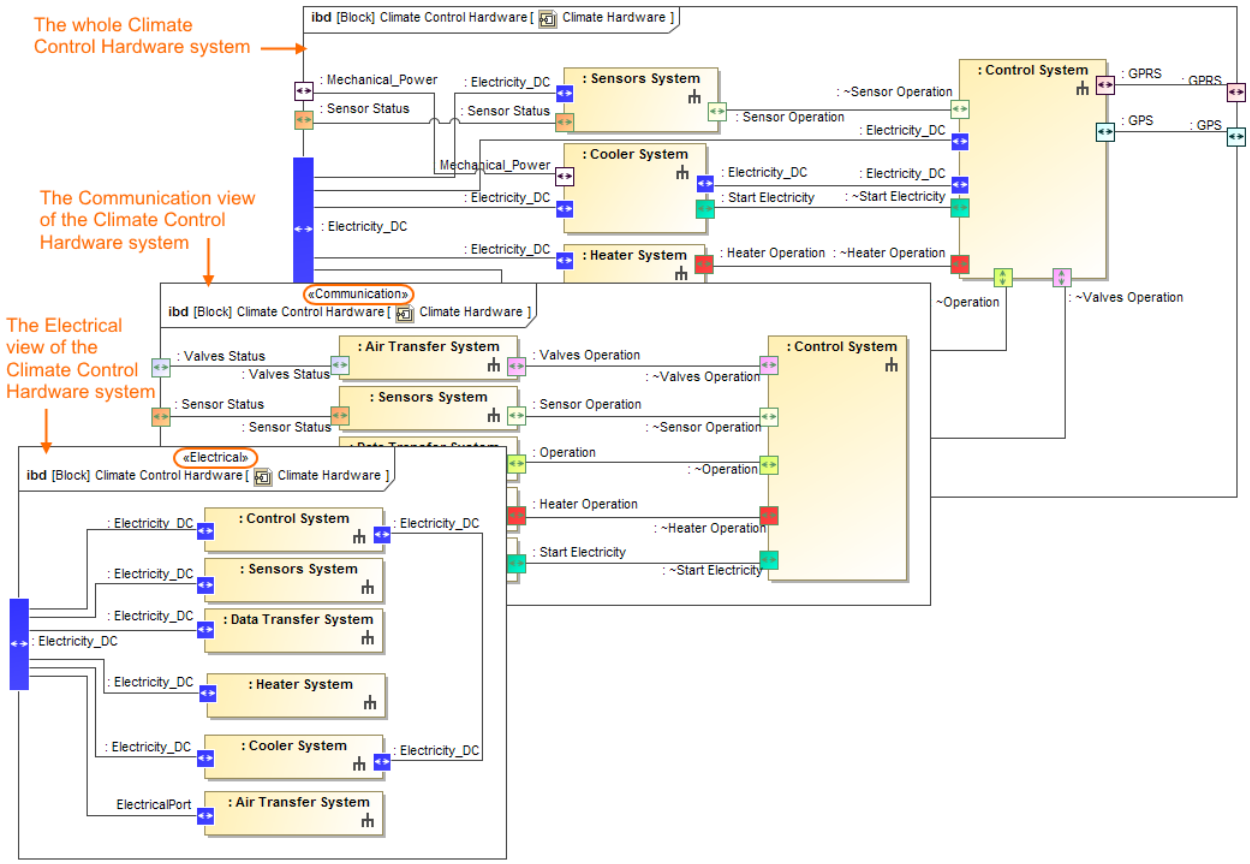
Diagram aspects allows you to create the different views, e.g. electrical, mechanical, optical, for the same system structure.The aspects functionality is driven by stereotypes, which allows you to use predefined aspects (only included in SysML Plugin[110] or other customizations) or custom aspects for diagrams. This enables you to:

- Set any number of aspects for one diagram.
- Drill down navigation from the Part shape to its internal structure.
- Create an Aspect diagram directly from the Part shape.
- Display parts and ports according to aspects.
- Use aspects with multiple layout templates(see page 589).

The figure below demonstrates the usage of predefined diagram aspects(see page 611). Three SysML Internal Block Diagrams of the same *Climate Control Hardware* system are of different aspects: electrical and communication. The Internal Block Diagram with the electrical aspect shows only the electrical structure of *Climate Control Hardware* system. Correspondingly, the IBD with the communication aspect shows only the communication structure.

107 https://docs.nomagic.com/display/SYSMLP190SP4/SysML+Plugin+Documentation
108 https://docs.nomagic.com/display/NMDOC/Installing+plugins
109 https://docs.nomagic.com/download/attachments/9915051/diagram%20aspects.mdzip?api=v2&modificationDate=1483357535810&version=1
110 https://docs.nomagic.com/display/SYSMLP190SP4/SysML+Plugin+Documentation

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The Climate Control Hardware views through different aspects. The illustration displays concepts from SysML Plugin.

You can represent different views of the same diagram by using different aspects. The procedures below describe how to:

- Enabling or disabling diagram aspects option(see page 610)
- Specifying or removing ignored aspects(see page 610)
- Finding predefined diagram aspects(see page 611)
- Creating custom aspects(see page 612)
- Creating an aspect diagram(see page 614)
- Applying or removing aspect for preexisting diagrams(see page 616)
- Navigation among aspect diagrams(see page 618)
- Displaying parts and ports according to aspects(see page 619)

Related pages

- Diagramming(see page 482)
- Layout templates(see page 589)
- Using layout templates with diagram aspects(see page 589)
- Displaying rake icon(see page 984)
- Displaying parts and ports(see page 491)
- Stereotype(see page 856)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Profile Diagram(see page 324)
- Creating diagrams(see page 483)
- Generalization(see page 722)

Sample model

The model used in the figures of this page is the **Diagram aspects** sample models that comes with SysML Plugin[111]. To open this sample properly you need to install the SysML plugin[112] in the MagicDraw and:

- Download diagram aspects.mdzip[113].
- Find in the modeling tool: *<modeling tool installation directory>\samples\SysML\diagram aspects.mdzip.*

Enabling or disabling diagram aspects option

The **Use Aspects** option allows you to use definable diagram aspects in your project. By default, its value is *false* in the **Project Options** dialog. The procedures below describe how to enable/disable the diagram aspects option in your project.

To enable/disable the diagram aspects option in your project

---

1. From the main menu, select **Options** > **Project**.
2. In the **Project Options** dialog, expand the **General** property group, and select the **Diagram Aspects**.
3. On the right side of the **Project Options** dialog, set the **Use Aspects** option value to *true* to enable the functionality, *false* - to disable it.
4. Click **OK**.

After enabling the diagram aspect functionality, you are able to create new aspect diagrams, and apply aspects for preexisting diagrams. The following procedures describe how to perform these functions:

- Specifying or removing ignored aspects(see page 610)
- Finding predefined diagram aspects(see page 611)
- Creating custom aspects(see page 612)
- Creating an aspect diagram(see page 614)
- Applying or removing aspect for preexisting diagrams(see page 616)
- Navigation among aspect diagrams(see page 618)
- Displaying parts and ports according to aspects(see page 619)

Specifying or removing ignored aspects

All of the available aspects are visible when trying:

- To create a new aspect diagram(see page 614): in the **Create Diagram** dialog(see page 483) under the **Aspect Diagrams** group.
- To apply aspect for a diagram(see page 616): in the diagram shortcut menu under the **Aspect** command group.

---

111 https://docs.nomagic.com/display/SYSMLP190SP4/SysML+Plugin+Documentation
112 https://docs.nomagic.com/display/NMDOC/Installing+plugins
113 https://docs.nomagic.com/download/attachments/55847477/diagram%20aspects.mdzip?api=v2&modificationDate=1483357535810&version=1

If you have many diagram aspects defined, but you need to use only a few of them in your project, you can restrict the visibility of any unnecessary aspects in your project. The procedure below describes how to specify/remove ignored aspects.

To specify/remove ignored aspects in your project

1. From the main menu, select **Options** > **Project**.
2. In the **Project Options** dialog, expand the **General** property group, and select the **Diagram Aspects**.
3. On the right side of the **Project Options** dialog, select the **Ignored Aspect** option value.
4. Click the **Edit** button.
5. In the **Select Stereotype** dialog, select aspects that you do not want to use in your project. If you want to use them in your project again, remove them from this list by double-clicking on the aspect in the Selected elements area. Click **OK**.

   > **Information**
   > If you have SysML Plugin[114] installed, you can see predefined aspects in the **List** tab by clearing the **Apply Filter** check box in the **Select Stereotype** dialog. How to install SysML Plugin >>[115]

6. Click **OK**.

All aspects added into **Ignored Aspect** option value are no longer visible when trying to create a new aspect diagram or to apply it to a pre-existing diagram. To see them again in the model, you must remove them from the **Ignored Aspect** option value.

Finding predefined diagram aspects

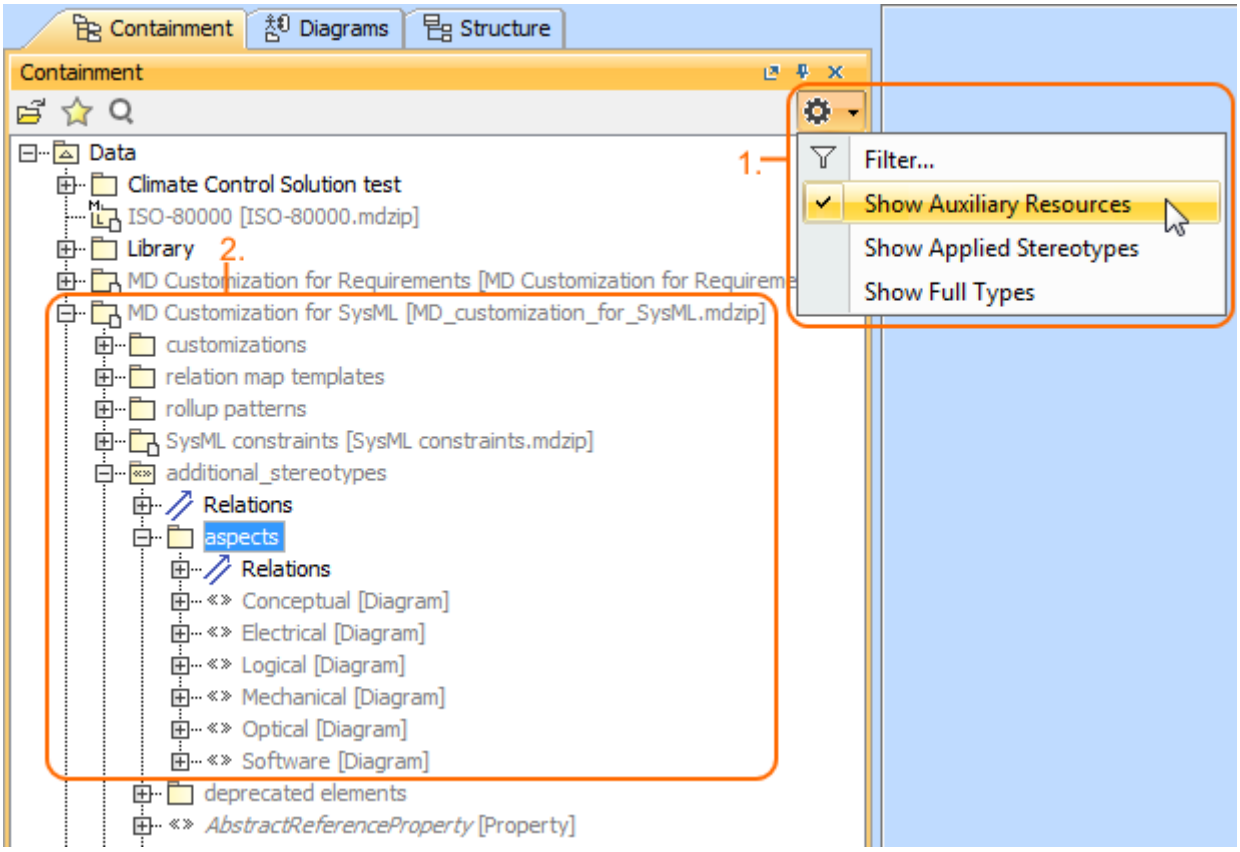
There are predefined aspects that are only included in SysML Plugin[116] or other customizations. The predefined diagram aspects are the following: <<optical>>, <<mechanical>>, <<electrical>>, <<software>>, <<conceptual>>, and <<logical>>. The procedure below explains how to find them in your project. You can use predefined aspects in your project, or you can create custom aspects. How to create custom aspects >>(see page 612)

To find predefined aspects in the project when SysML Plugin is installed

1. In the Containment tree, click
   ,
   and select the **Show Auxiliary Resources** option.
2. Expand packages *MD Customization for SysML* > *additional_stereotypes* > *aspects*.

114 https://docs.nomagic.com/display/SYSMLP190SP4/SysML+Plugin+Documentation
115 https://docs.nomagic.com/display/NMDOC/Installing+plugins
116 https://docs.nomagic.com/display/SYSMLP190SP4/SysML+Plugin+Documentation

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The location of the predefined diagram aspects in the project when SysML plugin is installed. The illustration displays concepts from SysML Plugin.

Sample model

The model used in the figures of this page is the **Diagram aspects** sample models that comes with SysML Plugin[117]. To open this sample properly you need to install the SysML plugin[118] in the MagicDraw and:

- Download diagram aspects.mdzip[119].
- Find in the modeling tool: *<modeling tool installation directory>\samples\SysML\diagram aspects.mdzip.*

Creating custom aspects

You can define your own diagram aspects that you want to use in the project. You can create custom aspects in the same way you create a new stereotype(see page 856). The procedures below describe how to create a custom diagram aspect, e.g., Air, in two different ways:

- On the Profile Diagram pane(see page 613).

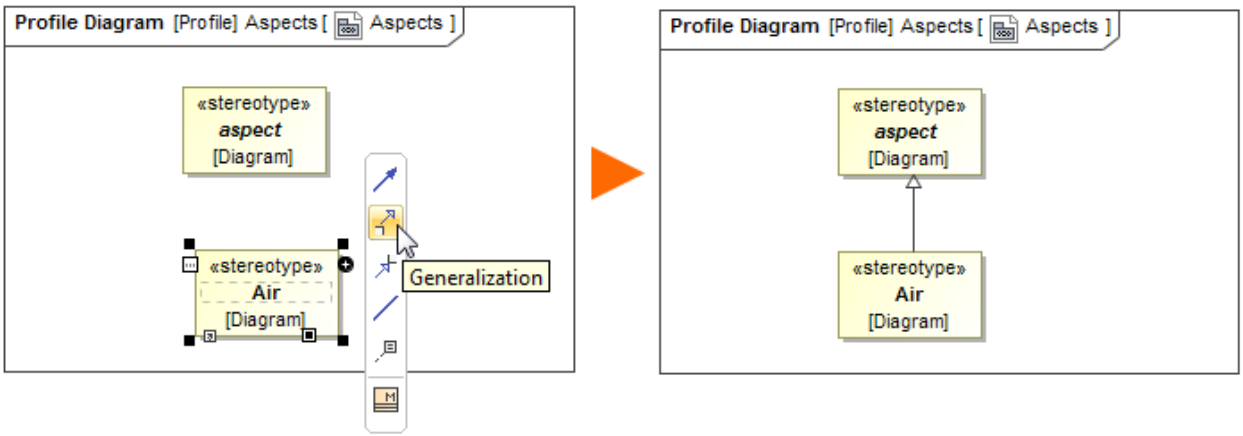
117 https://docs.nomagic.com/display/SYSMLP190SP4/SysML+Plugin+Documentation
118 https://docs.nomagic.com/display/NMDOC/Installing+plugins
91 https://docs.nomagic.com/download/attachments/55847477/diagram%20aspects.mdzip?api=v2&modificationDate=1483357535810&version=1

- In the Containment tree.(see page 613)

To create a new aspect on the Profile Diagram pane

1. Open or create the Profile Diagram(see page 324).
2. On the diagram palette(see page 24), click the **Stereotype** button, place it on the diagram pane, and type its name, e.g., Air.
3. In the Containment tree, click

   ,
   and select the **Show Auxiliary Resources** option.
4. Expand packages *UML Standard Profile* > *MagicDraw Profile*, and drag the *aspect* Stereotype on the same diagram pane.
5. Create the Generalization(see page 722) relationship from the newly created Stereotype to the *aspect* Stereotype.
   The new aspect is created, and can be used in the project.

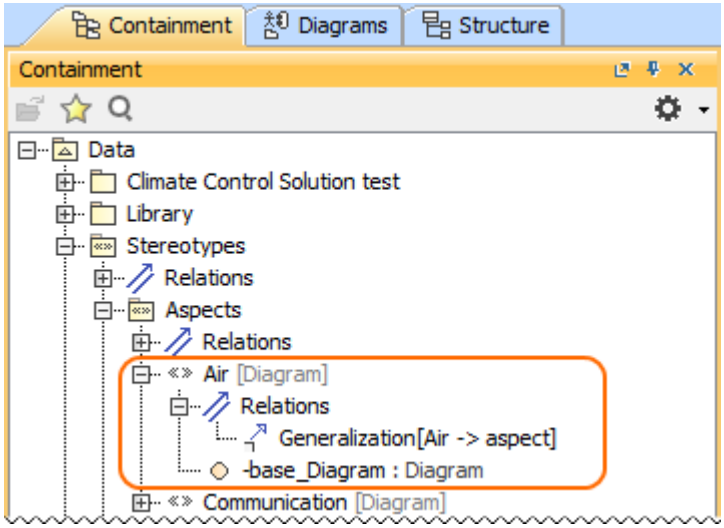


The creation of the new diagram aspect on the Profile Diagram pane.

To create a new aspect in the Containment tree

1. In the Containment tree, select either a Profile or Package.(see page 800)
2. From its shortcut menu**,** click the **Create Element** button.
3. In the **Create Element** dialog, select **Stereotype,** and type its name, e.g., Air.
4. Open its Specification window, select the **All** properties mode, and find the **Base Classifier** property.
5. Select the value cell of the **Base Classifier** property, click the **Edit** button.
6. In the Select Stereotype dialog, clear the **Apply Filter** check box.
7. In the Tree tab, expand packages *UML Standard Profile* > *MagicDraw Profile*, and double click to select the *aspect* Stereotype.
8. Click **OK** and **Close**.
   The new aspect is created, and can be used in the project.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The creation of the new diagram aspect in the Containment tree.

After the custom stereotype is created in the model, the Composite Structure diagram with the custom aspect applied can be found in the **Create Diagram** dialog(see page 483) under the **Aspect Diagrams** group when creating a new aspect diagram. How to create a new aspect diagram >>(see page 614)

Sample model

The model used in the figures of this page is the **Diagram aspects** sample models that comes with SysML Plugin[120]. To open this sample properly you need to install the SysML plugin[121] in the MagicDraw and:

- Download diagram aspects.mdzip[122].
- Find in the modeling tool: *<modeling tool installation directory>\samples\SysML\diagram aspects.mdzip.*

Creating an aspect diagram

Before creating an aspect diagram, the following conditions must be met:

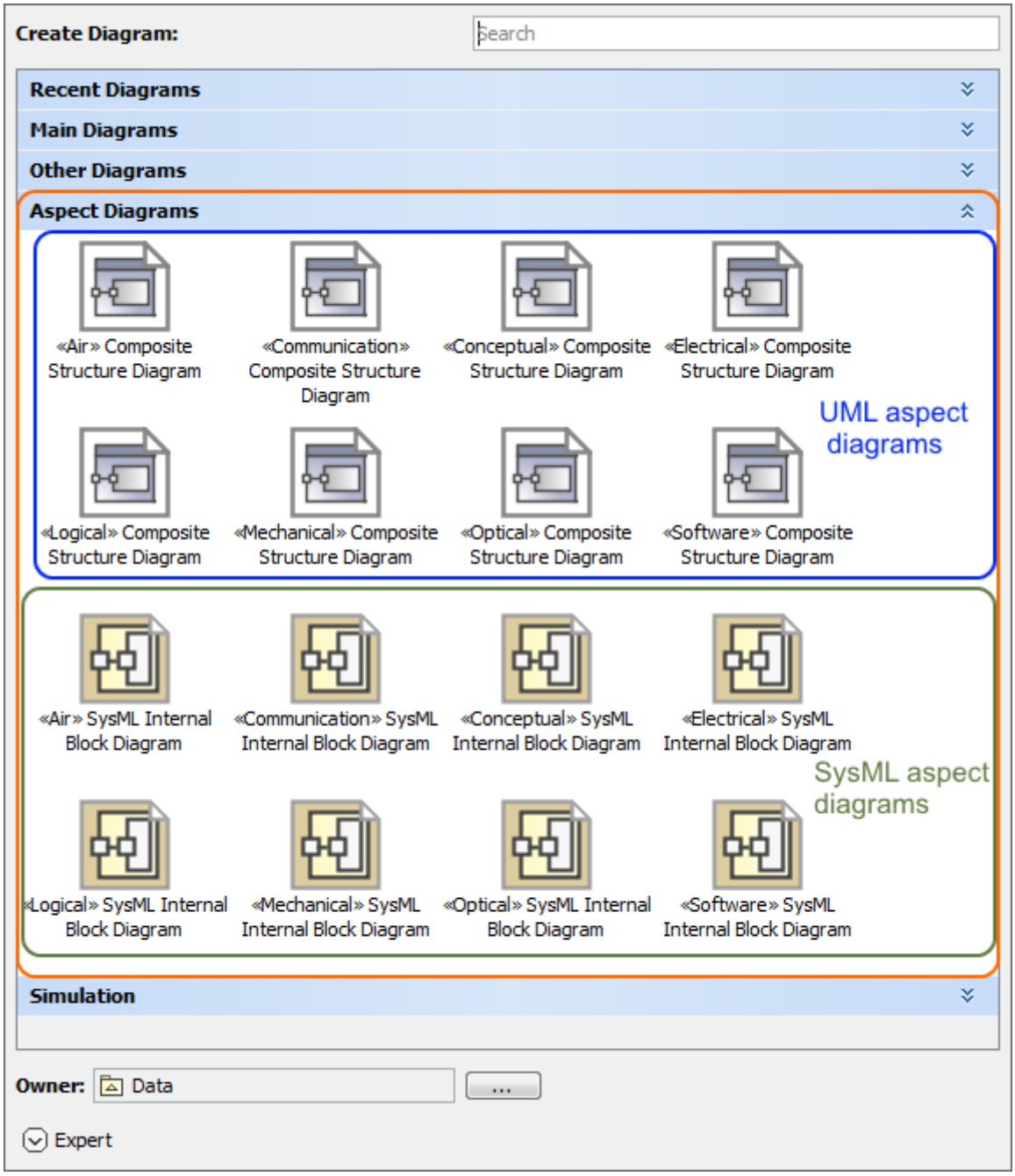
- The custom aspects are created in the project. How to create a custom aspect >>(see page 612)

  > ⓘ If you have SysML Plugin[123] installed, the predefined aspect diagrams already exist in your project. Predefined diagram aspects >>(see page 611)

- The **Use Aspects** option is enabled. How to enable the diagram aspects option >>(see page 610)

120 https://docs.nomagic.com/display/SYSMLP190SP4/SysML+Plugin+Documentation
121 https://docs.nomagic.com/display/NMDOC/Installing+plugins
122 https://docs.nomagic.com/download/attachments/55847477/diagram%20aspects.mdzip?api=v2&modificationDate=1483357535810&version=1
123 https://docs.nomagic.com/display/SYSMLP190SP4/SysML+Plugin+Documentation

Creating a new aspect diagram is similar to creating any other diagram. Every way to create a diagram is described here: Creating diagrams(see page 483). The difference is that aspect Composite Structure diagrams(see page 325) are automatically collected under the separate **Aspect Diagrams** group of the **Create Diagram** dialog(see page 483). If you have SysML plugin[124] installed, the Internal Block diagrams[125] with custom and predefined aspects are collected under the same **Aspect Diagrams** group.



124 https://docs.nomagic.com/display/SYSMLP190SP4/SysML+Plugin+Documentation
125 https://docs.nomagic.com/display/SYSMLP190SP4/SysML+Internal+Block+Diagram

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

The Aspect Diagram group, highlighted in orange, with predefined and custom UML and SysML aspect diagrams. The illustration displays concepts from SysML Plugin.

Sample model

The model used in the figures of this page is the **Diagram aspects** sample models that comes with SysML Plugin[126]. To open this sample properly you need to install the SysML plugin[127] in the MagicDraw and:

- Download diagram aspects.mdzip[128].
- Find in the modeling tool: *<modeling tool installation directory>\samples\SysML\diagram aspects.mdzip.*

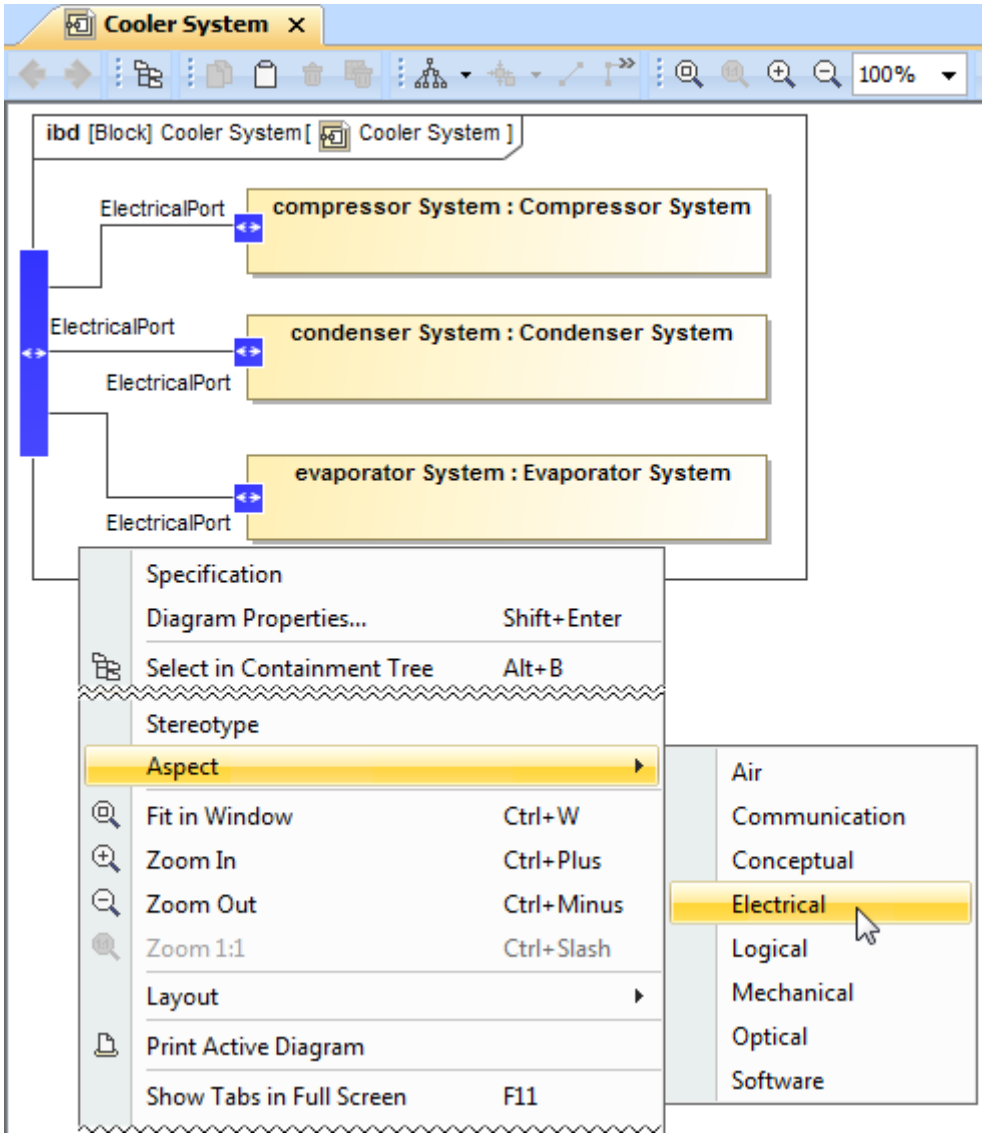
Applying or removing aspect for preexisting diagrams

You can apply custom or predefined aspects for preexisting diagrams. It is possible to apply multiple aspects for one diagram. The aspects can be applied for all diagrams except Activity(see page 311), State Machine(see page 308), and Sequence(see page 304) diagrams. The procedure below describes how to apply/remove an aspect for a diagram.

To apply/remove an aspect for a diagram

- From the diagram shortcut menu, point to **Aspect,** and choose an appropriate aspect from the list.

126 https://docs.nomagic.com/display/SYSMLP190SP4/SysML+Plugin+Documentation
127 https://docs.nomagic.com/display/NMDOC/Installing+plugins
128 https://docs.nomagic.com/download/attachments/55847477/diagram%20aspects.mdzip?api=v2&modificationDate=1483357535810&version=1



Applying an aspect for a diagram. The illustration displays concepts from SysML Plugin.

Sample model

The model used in the figures of this page is the **Diagram aspects** sample models that comes with SysML Plugin[129]. To open this sample properly you need to install the SysML plugin[130] in the MagicDraw and:

- Download diagram aspects.mdzip[131].
- Find in the modeling tool: *<modeling tool installation directory>\samples\SysML\diagram aspects.mdzip.*

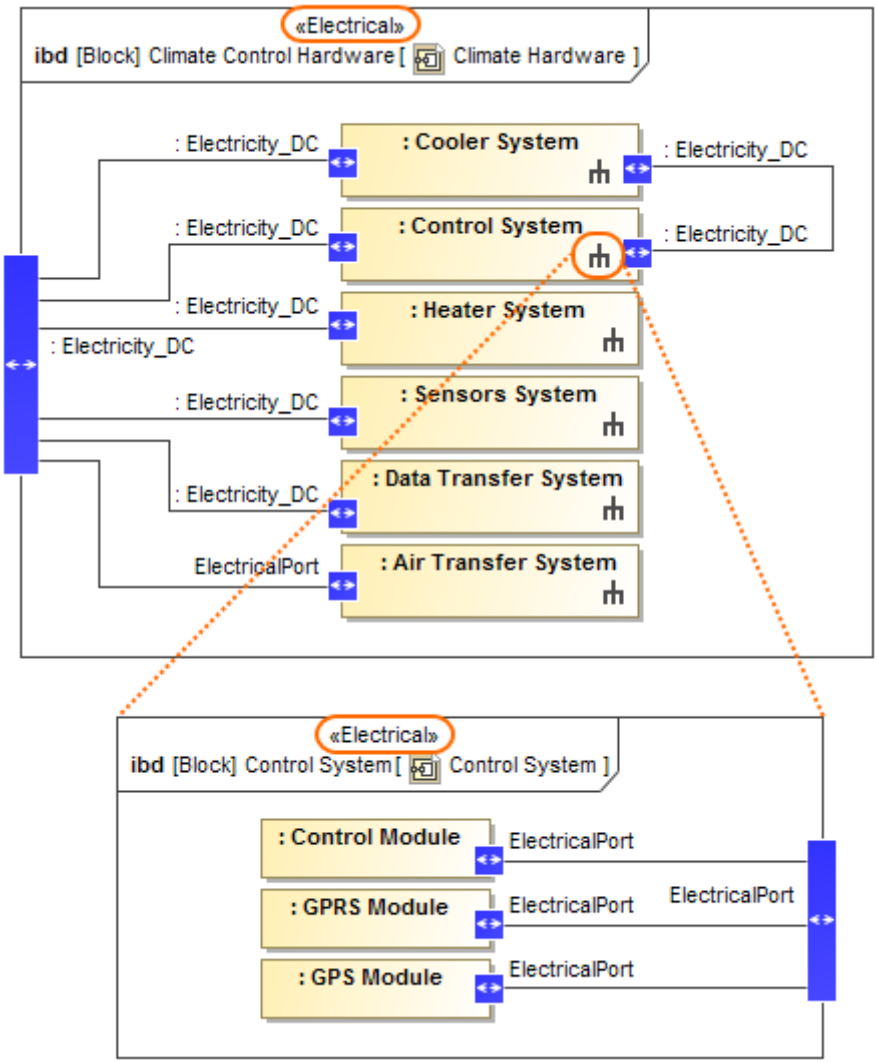
129 https://docs.nomagic.com/display/SYSMLP190SP4/SysML+Plugin+Documentation
130 https://docs.nomagic.com/display/NMDOC/Installing+plugins
13 https://docs.nomagic.com/download/attachments/55847477/diagram%20aspects.mdzip?api=v2&modificationDate=1483357535810&version=1

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Navigation among aspect diagrams

When elements have an internal structure, the rake icon(see page 984) appears on the element shape. If those elements belong to an aspect diagram, the rake icon navigates to the same aspect internal structure of the selected element. If there is no internal diagram of the same aspect, then any diagram of another aspect or diagram without aspect is opened.

The figure below demonstrates the *electrical* Internal Block diagram of the *Climate Control Hardware*. The rake icon on the part typed by *Control System* Block indicates that it contains an inner structure. There are four Internal Block diagrams for the *Control System* in the model. Although, after double-clicking on that part shape typed by *Control System* Block, the rake icon navigates exactly to the *electrical* Internal Block diagram of the *Control System.*



The navigation among aspect diagrams via rake icon. The illustration displays concepts from SysML Plugin.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Sample model

The model used in the figures of this page is the **Diagram aspects** sample models that comes with SysML Plugin[132]. To open this sample properly you need to install the SysML plugin[133] in the MagicDraw and:

- Download diagram aspects.mdzip[134].
- Find in the modeling tool: *<modeling tool installation directory>\samples\SysML\diagram aspects.mdzip.*

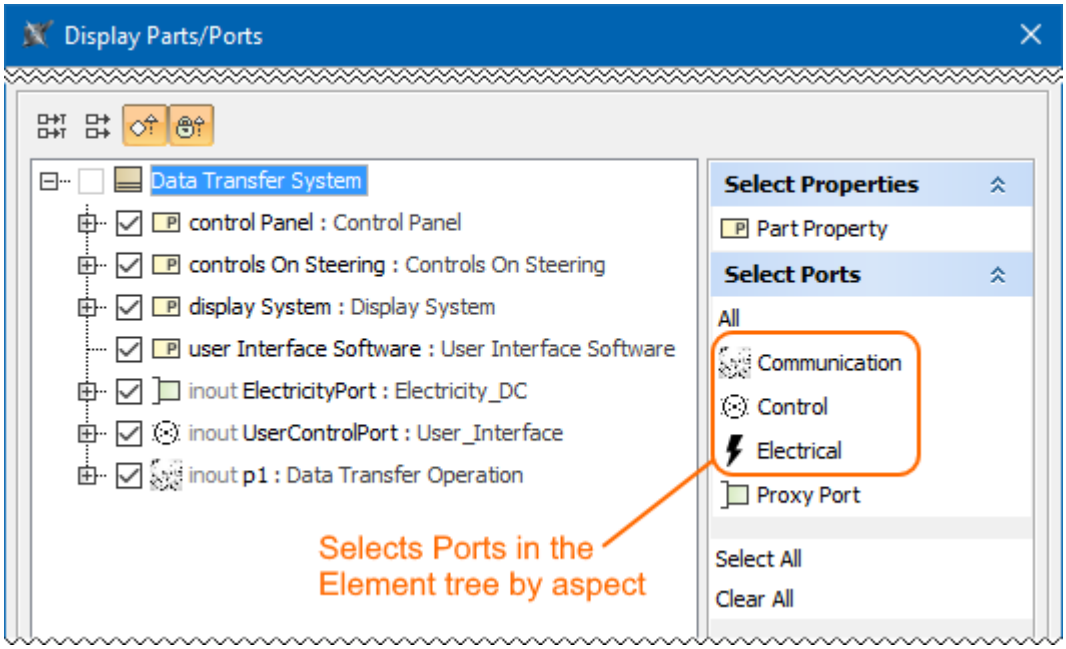
Displaying parts and ports according to aspects

You can select Parts(see page 804) and/or Ports(see page 809) by aspects when displaying them(see page 491) in a Composite Structure diagram(see page 325). Aspects are used when choosing which Parts and/or Ports you want to display on the diagram pane(see page 24).

All aspects that you have created in your project are collected, and can be used in the following ways:

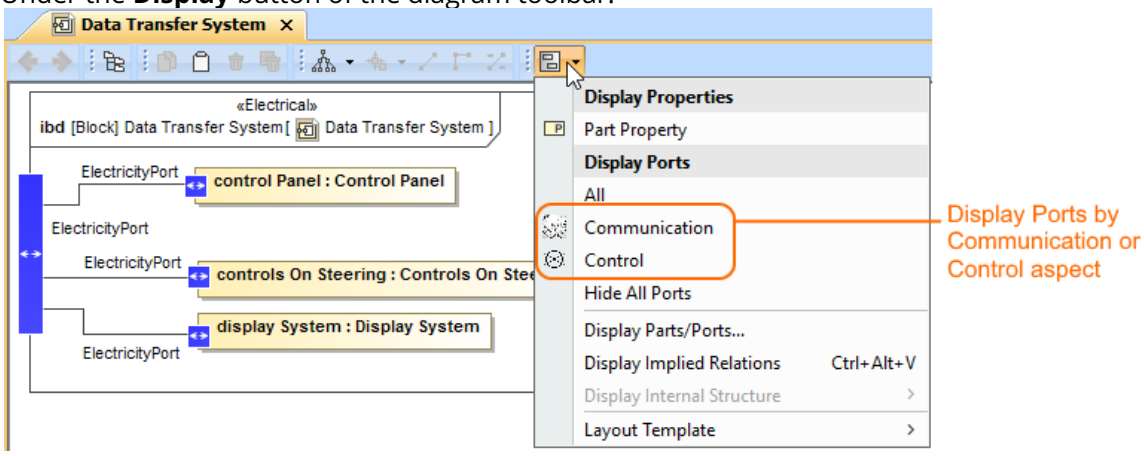
- In the **Display Parts/Ports** dialog > context-sensitive Panel. If you are modifying a pre-existing diagram with an applied aspect(s), only the matching parts/ports are pre-selected in the **Display Parts/Ports** dialog.

  ⓘ The parts/ports match according to following rules:
  - By aspect and part/port stereotype name.
  - By aspect and part/port type stereotype name.

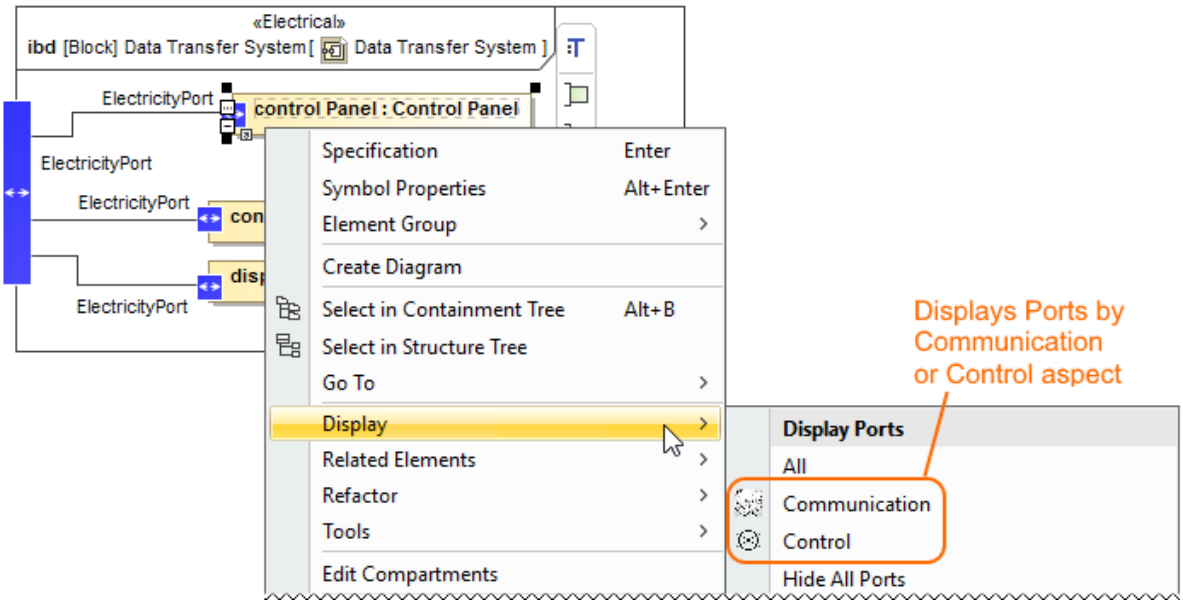


132 https://docs.nomagic.com/display/SYSMLP190SP4/SysML+Plugin+Documentation
133 https://docs.nomagic.com/display/NMDOC/Installing+plugins
134 https://docs.nomagic.com/download/attachments/55847477/diagram%20aspects.mdzip?api=v2&modificationDate=1483357535810&version=1

- Under the **Display** button of the diagram toolbar.

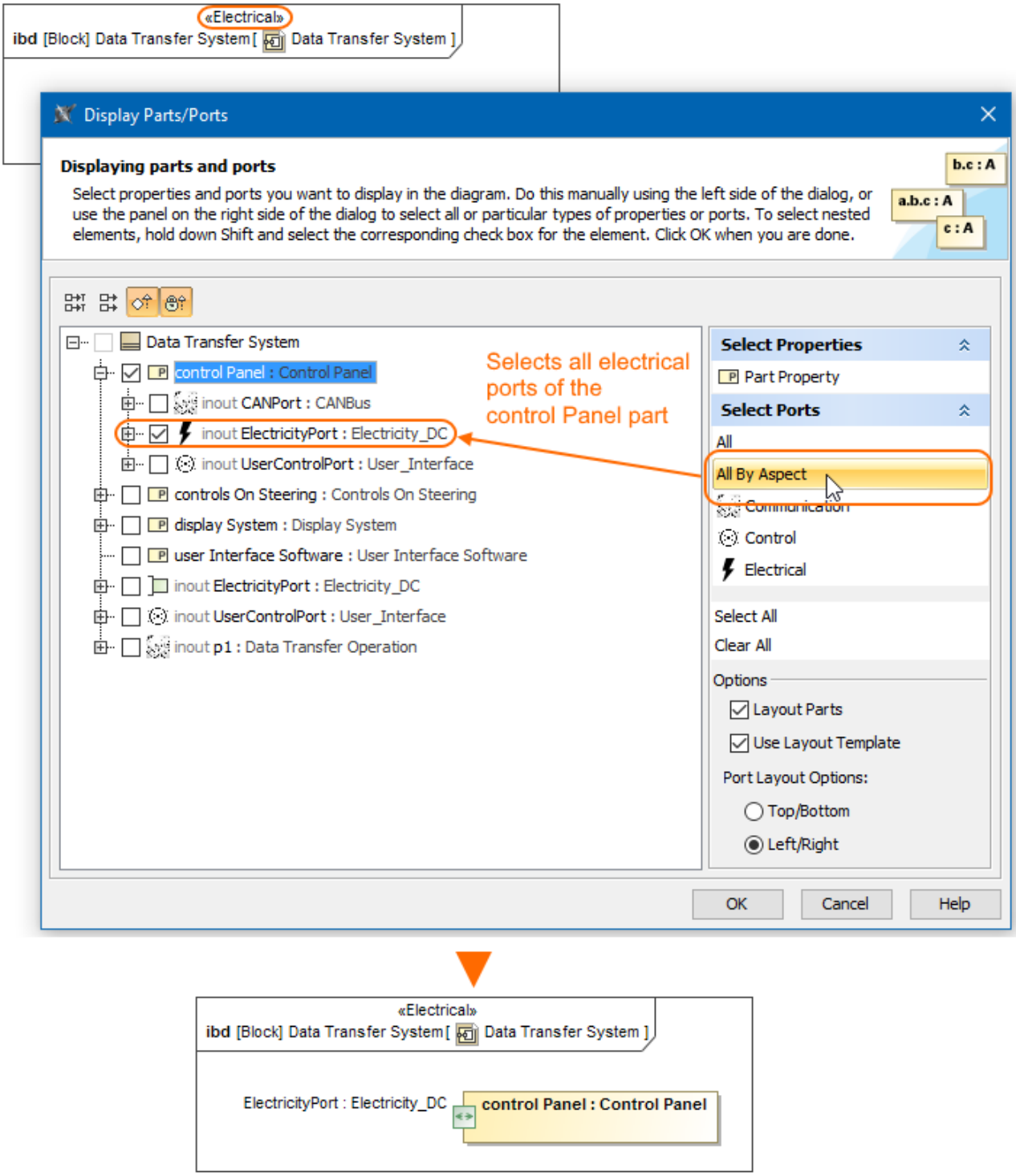


- Under the **Display** command group from the shortcut menu.



When displaying Parts/Ports in the aspect diagram, e.g., Electrical *Data Transfer System* Internal Block Diagram, you can display all the parts/ports of an appropriate aspect, e.g., Electrical, by using the **All By Aspect** button. This button can be found in the three places mentioned above in this section. After selecting this button, all parts or ports of the same aspect as the diagram's can be displayed. The figure below demonstrates the usage of this button in the **Display Parts/Ports** dialog when you are trying to display all electrical aspect ports of the selected *control Panel* part.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The All By Aspect button usage in the Display Parts/Ports dialog. The illustration displays concepts from SysML Plugin.

Sample model

The model used in the figures of this page is the **Diagram aspects** sample models that comes with SysML Plugin[135]. To open this sample properly you need to install the SysML plugin[136] in the MagicDraw and:

- Download diagram aspects.mdzip[137].
- Find in the modeling tool: *<modeling tool installation directory>\samples\SysML\diagram aspects.mdzip.*

Complete diagrams

On this page

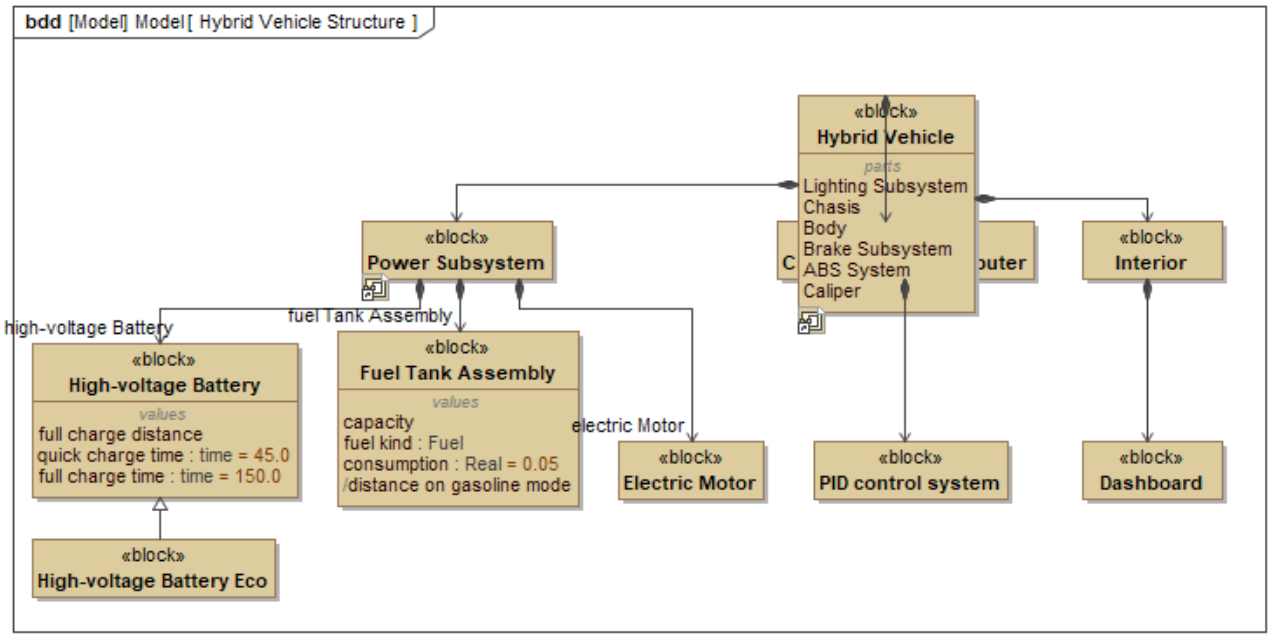
- Marking diagrams as complete(see page 622)
- Updating complete diagram layout(see page 624)
- Marking diagrams as incomplete(see page 625)
- Customizing complete diagram behavior(see page 626)

Marking diagrams as complete

When you finish working on a diagram, you expect that the diagram layout will not change. However, if the content of your model is changed, the diagram layout changes as well; shapes are automatically resized to fit new content and labels are automatically displayed on paths. This can cause symbols to overlap and paths to be left hanging in a diagram, as illustrated below. So you must go back to the finished diagram and fix its layout. To save time, you can mark diagrams as complete and prevent their layout from being automatically updated.

135 https://docs.nomagic.com/display/SYSMLP190SP4/SysML+Plugin+Documentation
136 https://docs.nomagic.com/display/NMDOC/Installing+plugins
137 https://docs.nomagic.com/download/attachments/55847477/diagram%20aspects.mdzip?api=v2&modificationDate=1483357535810&version=1

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



This diagram was distorted after updating a model. For comparison, see the figure below displaying the same diagram after model changes when the diagram is marked as complete.

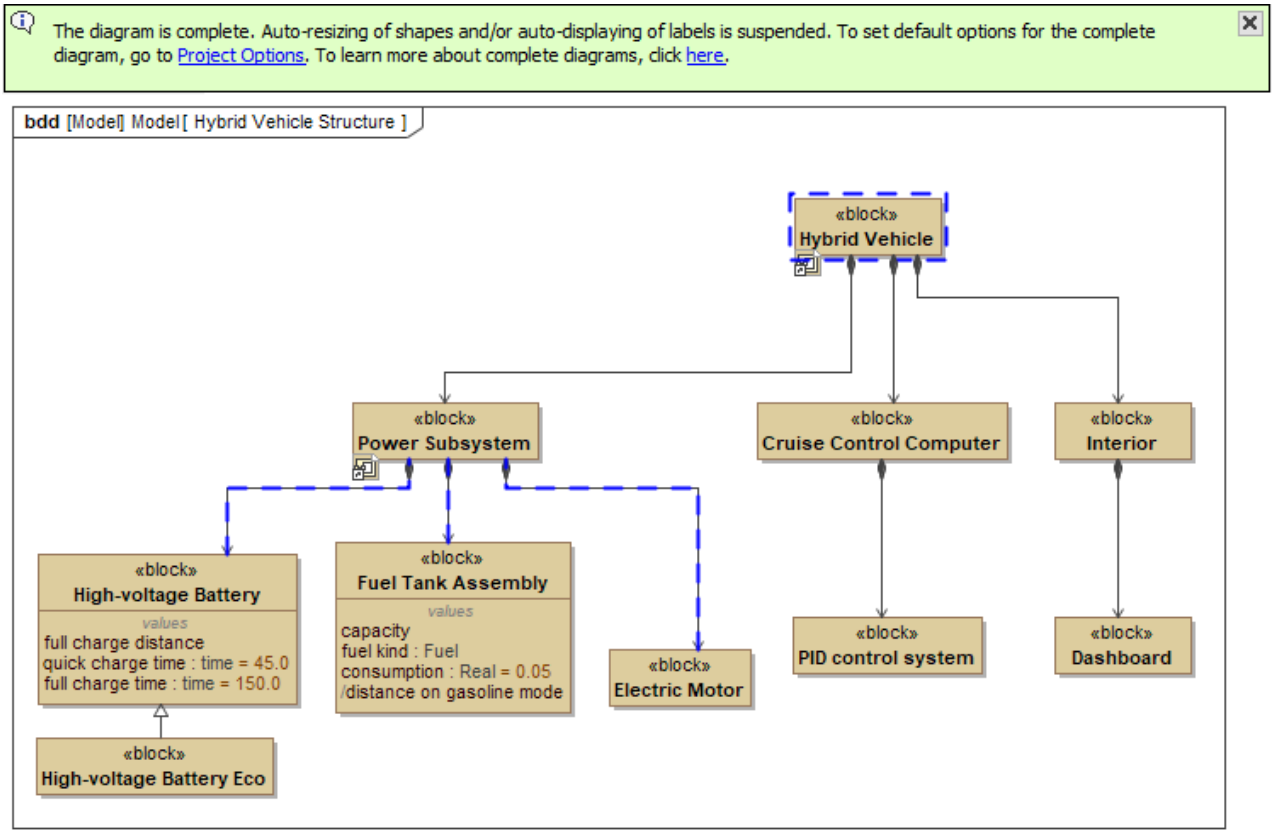
To mark a diagram as complete

- Right-click the diagram on the diagram pane or in the model browser and from the menu select **Complete**.

When a diagram is marked as complete, the following happens:

- The «complete» stereotype is applied to a diagram. This way you can easily indicate complete diagrams in the model browser without opening them.
- Auto-resizing of shapes and auto-displaying of labels is suspended. This means that the diagram layout is not automatically changed when the model content changes.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The figure displays an example of a complete diagram.

Alternative

You can suspend auto-resizing of shapes and auto-displaying of labels in a diagram without marking it as complete in one of the following ways:

- To suspend auto-resizing of shapes and auto-displaying of labels for individual symbols, open the **Symbol Properties** dialog(see page 975) of a symbol or a group of symbols, and set the **Suspend Auto-Resizing of Shapes** and/or **Suspend Auto-Displaying of Labels** properties to *true*.
- To suspend auto-resizing of shapes and auto-displaying of labels for all the diagram symbols, open the **Diagram Properties** dialog(see page 522) of a diagram, and set the **Suspend Auto-Resizing of Shapes** and/or **Suspend Auto-Displaying of Labels** properties to *true.*

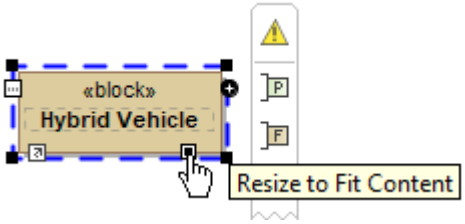
Updating complete diagram layout

When a diagram is marked as complete, the symbols with suspended content related to model changes are highlighted in blue (see the figure above). You can update these symbols manually, as described below.

To update shapes

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

1. In a complete diagram, select the highlighted shape you want to update.
2. Do one of the following:
   - Drag one of the selection handles on the corner of the shape to resize it. The suspended content is dynamically displayed after resizing the shape.
   - Click the smart manipulator on the bottom of the shape, as shown below.

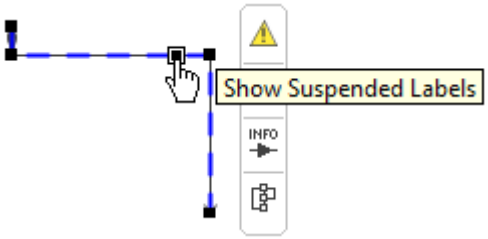


   - Click on the shape toolbar and select **Resize to Fit Content**.

> **Updating multiple shapes**
> To update multiple shapes in a complete diagram, select the highlighted shapes you want to update and click in the diagram toolbar.

To update paths

1. In a complete diagram, select the highlighted path you want to update.
2. Do one of the following:
   - Click the smart manipulator on the path, as shown below.



   - Click on the path toolbar and select **Show Suspended Labels**.

> **Updating multiple paths**
> To update multiple paths in a complete diagram, select the highlighted paths you want to update and click in the diagram toolbar.

Marking diagrams as incomplete

If you need to resume normal work on a diagram, mark it as incomplete, as described below.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To mark a diagram as incomplete

1. Right-click a diagram on the diagram pane or in the model browser and select the **Complete** option to clear its check box.
2. When the question dialog is displayed, do one of the following:
    - Select **Yes**, to continue suspending auto-resizing of shapes and/or auto-displaying of labels.
    - Select **No**, to resume auto-resizing of shapes and/or auto-displaying of labels.

If you select **No** in step 2, all diagram symbols are automatically resized to fit suspended content, and you can continue working on the diagram as usual. If you select **Yes**, auto-resizing of shapes and auto-displaying of labels in the diagram are still suspended, even though the diagram is not complete.

Customizing complete diagram behavior

By default, auto-resizing of shapes and auto-displaying of labels in a complete diagram are suspended. Also, the symbols with suspended content due to model changes are highlighted in blue. You can change this default behavior in the **Project Options** dialog.

To customize complete diagram behavior

1. In the main menu of your modeling tool, go to **Options** > **Project**.
2. On the left side of the **Project Options** dialog, expand the **General** node and select **Diagrams**.
3. On the right side of the dialog, specify the desired options in the **Complete Diagrams** group:
    - **Suspend Auto-Resizing of Shapes** - set to *true* to keep the original size of shapes while editing the model.
    - **Suspend Auto-Displaying of Labels** - set to *true* if you do not want to automatically show labels in a diagram while editing the model.
    - **Highlight Suspended Symbols** - set to *true* to highlight symbols that have suspended content related to model changes.

When you specify project options listed in step 3, they are applied to all diagrams that are marked as complete after specification.

Printing

With MagicDraw you can print:

- Active diagram
- Any diagram(s)
- All open diagrams
- Selected symbols of active diagram

You can specify the printing options or preview pages by using the following dialogs:

- Print Dialog
- Print Options Dialog

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Print Preview Window

To print an active diagram

1. Open a diagram you want to print.
2. Do one of the following:
    - In the main toolbar, click the Print Active Diagram button.
    - From the diagram shortcut menu, select **Print Active Diagram**.
    - From the **File** menu, select **Print**. On the Print dialog, **Print what** area, select **Active diagram** and click the **Print** button.

To print any diagram(s)

1. From the **File** main menu, select **Print**. The **Print** dialog opens.
2. In the **Print what** area, choose **Selected diagram(s)**. The **Select Diagram** dialog opens.
3. Select diagram(s) you want to print.
4. Click **OK**.
5. Click **Print**.

To print all open diagrams

1. From the **File** main menu, select the **Print** command. The **Print** dialog opens.
2. In the **Print what** area, select **All open diagram(s)**.
3. Click **Print**.

To print selected symbols of active diagram

1. On the diagram, select symbols you want to print.
2. Do one of the following:
    - From the selected symbols' shortcut menu, select **Print Selected**.
    - From the **File** menu, select **Print**. In the **Print** dialog, the **Print what** area, choose **Selected symbols of active diagram** and click the **Print** button.

Related pages

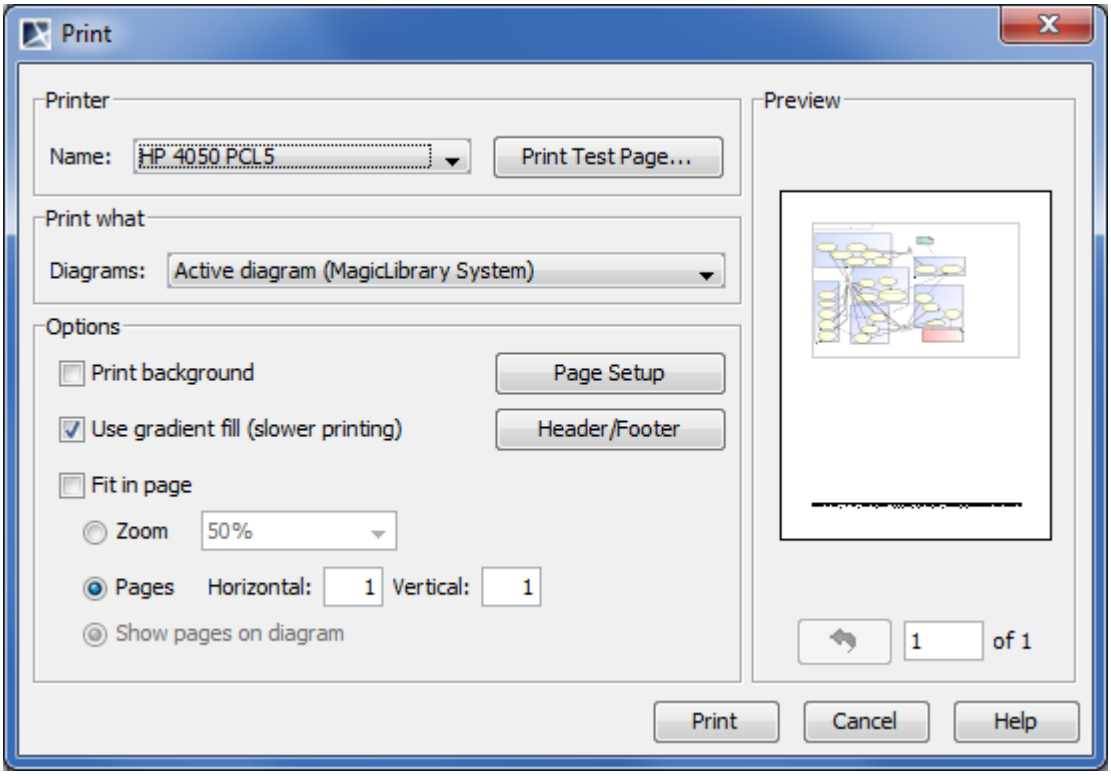
- Diagramming
  (see page 482)

Print dialog

To open the Print dialog, do one of the following

- From the **File** menu, select **Print**.
- Press CTRL+P.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

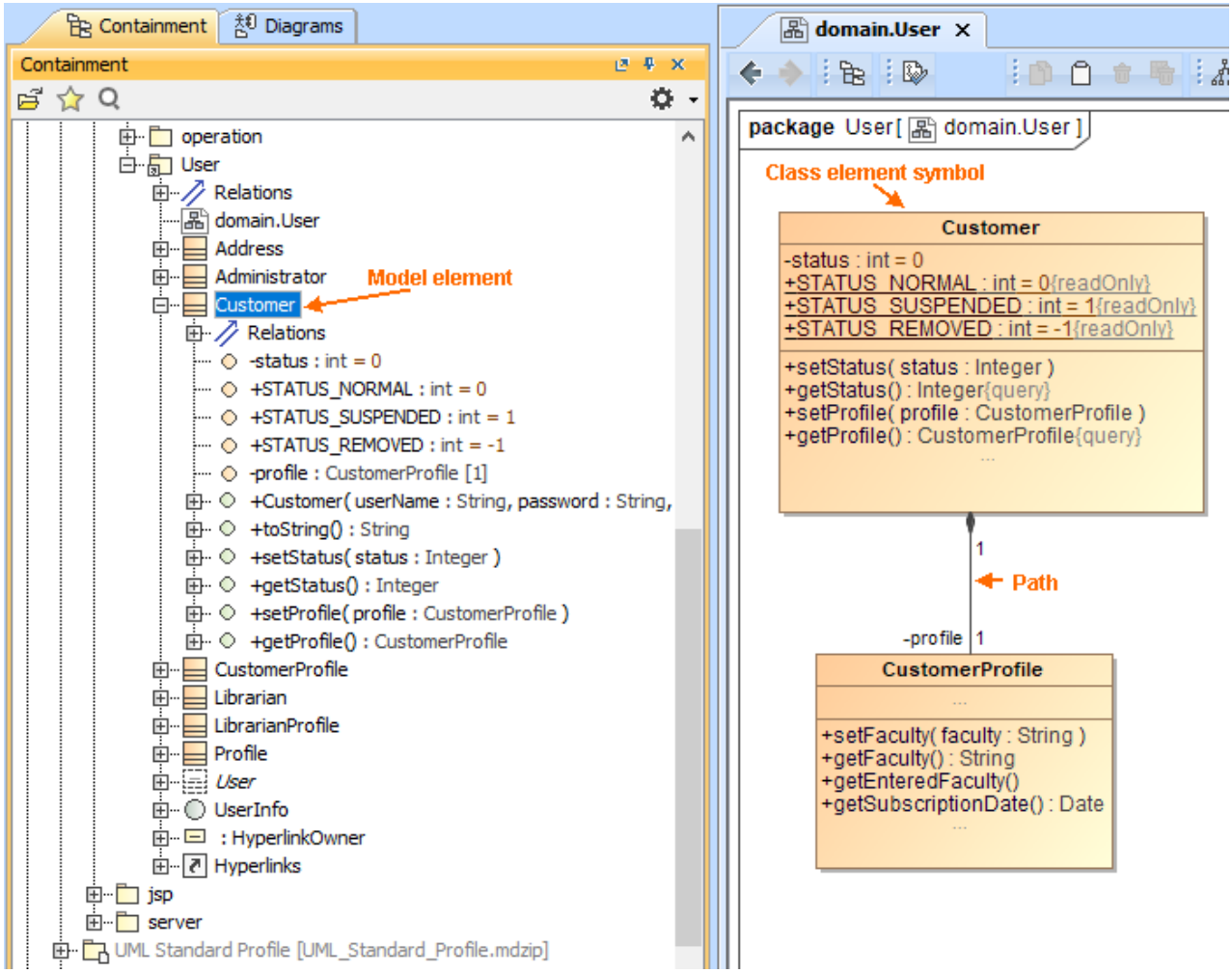


UML elements

Modeling tools developed by No Magic, Inc. fully support the latest UML standard elements. The child pages provide in-depth definitions for each model element used in creating your projects.

In the model, a model element is stored in the Model Browser and its symbol or path (for relationships) is represented on the diagram pane, as shown below.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Model elements and symbols.

You can represent several symbols or paths of the same element in the diagram.

Action

An Action is a named element that is the fundamental unit of an executable functionality. The execution of an Action represents some transformations or processing in the modeled system, be it a computer system or otherwise. An Action execution represents the run-time Behavior (see page 662)of executing an Action within a specific Behavior execution. As the Action is an abstract Class(see page 666), all Action executions will be the executions of a specific kind of Action. When the Action will be executed and what its actual inputs will be determined by the concrete Action and the Behaviors in which it is used.

To remove an action from a diagram by keeping incoming and outgoing control flows connected, do one of the following

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- On a diagram, select the Action and press Delete. Then, in the opened message, click **Yes**.
- On a diagram, from the Action shortcut menu, select **Refactor** and then **Remove**.

> - Flows can be joined only if the removed Action has one incoming and one outgoing Control Flows.
> - You can select more than one Action to remove only if the selected part has one incoming and one outgoing Control Flow.
> - In the **Environment Options** dialog(see page 65), in the **Diagram** options group, under the **Editing** category, you can select one of the **Join Control Flows** option values: *Ask to join control flows*, *Yes,* or *No*.

An Action can be specified by changing its property values in the Action Specification window(see page 913). Each property is described in the description area on this window.

Related pages

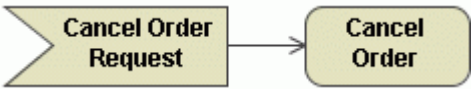
- Working with Actions(see page 641)
- Formatting Symbols(see page 975)
- Customizing Environment Options(see page 65)
- Specification Window(see page 913)
- Model Elements(see page 628)
- Activity diagram(see page 311)

Accept Event Action

On this page

- Using Time Event Shape(see page 632)
- Specifying the Is Unmarshall property(see page 632)

The Accept Event Action is an Action that waits for the occurrence of an Event (see page 708)that meets specified conditions. If an Accept Event Action has no incoming edges, then the Action starts when the containing activity or Structured Node starts, whichever most immediately contains the Action. In addition, an Accept Event Action with no incoming edges remains enabled after it accepts an Event. It does not terminate after accepting an Event and outputting a value, but continues to wait for other Events. An Accept Event Action that has no incoming edges and is contained by a Structured Node is terminated when its container is terminated.

Notation	Description
Accept Event Action 	The *Order Cancel Request* Accept Event Action is connected to *Cancel Order* with the Exception Handler relationship.

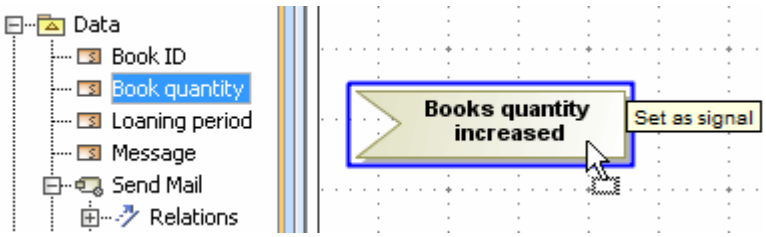
Notation	Description
Rotated Accept Event Action 	The symbol of Accept Event Action is rotated.
Accept Time Event Action 	The Time Event is connected to the Call Behavior Action.

Assigning signals

To assign a Signal using a drag-and-drop operation

To assign a Signal to an Accept Event Action, at least one Signal element should exist in the project.

1. Select the Signal in the Model Browser(see page 29).
2. Drag the Signal to the Accept Event Action as shown in the following picture.



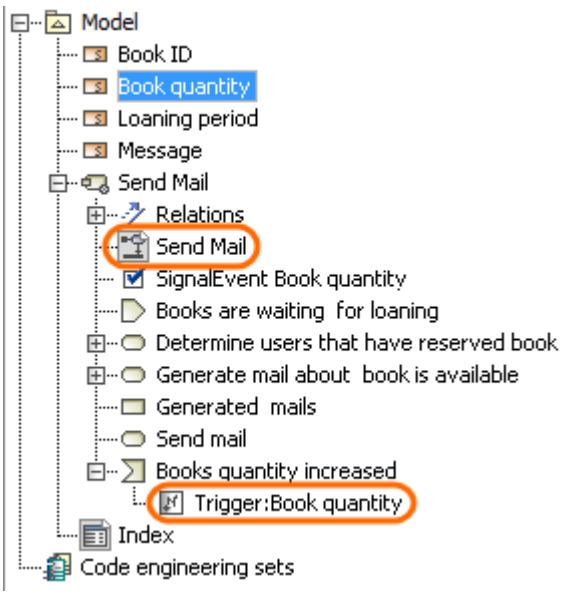
To assign a Signal using the Accept Event Action's shortcut menu

1. Right-click the Accept Event Action to open the shortcut menu.
2. On the shortcut menu, click Signal. Click to select the Signal that you wish to assign to the element, or click the **Create Signal** button.

Once the Signal is assigned to the Accept Event Action, a Signal Event and a Trigger for this Action are created automatically. If you change the Signal, the Signal Event and the Trigger will change accordingly

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

to a new Signal. The following figure shows the Signal Event and the Trigger positions in the Containment tree(see page 32).



Using Time Event Shape

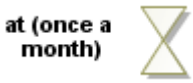
To draw the accept event action with the time event shape

- In the Activity diagram panel, select the **Time Event** button, and then click the diagram pane.

To change the position of the Accept Event Action name

1. Select the Accept Event Action shape on the diagram pane.

2. Click the **Rotate State** button on the upper-right side of the shape. The name of the Accept Event Action will appear on the opposite side of the shape.



You can specify for an Accept Event Action, whether there is a single Output Pin for the Event or multiple Output Pins for the attributes of the Event.

Specifying the Is Unmarshall property

To specify the **Is Unmarshall** property

1. Select the Accept Event Action, and open its Specification window(see page 913).
2. Select or clear the **Is Unmarshall** check box. If the value is set to:
    - *false*, then there is a single Output Pin for the Event, and a real-world instance of the Signal is placed on this Output Pin.
    - *true*, then there are multiple Output Pins for the attributes of the Event, and feature values of the Signal instance are placed on the corresponding Output Pins.

Related pages

- Elements multiple selection(see page 1032)
- Activity diagram(see page 311)
- Model Elements(see page 628)

Call Behavior Action

On this page

- Description(see page 633)
- Assigning a Behavior to the Call Behavior Action(see page 633)
- Creating a new diagram for the Call Behavior Action(see page 634)
- Changing the Name Display Mode on the Call Behavior Action symbol(see page 635)

Description

The Call Behavior Action invokes a Behavior(see page 662) directly rather than invoking a behavioral feature that, in turn, causes the Behavior. The argument values of the Action are available to the execution of the invoked Behavior. The execution of the Call Behavior Action waits until the execution of the invoked Behavior is completed and a result is returned on its Output Pin. In particular, the invoked Behavior can be an Activity.

Assigning a Behavior to the Call Behavior Action

To assign a Behavior to the Call Behavior Action, do one of the following

- On the Activity diagram(see page 311) palette(see page 24), click . An Activity will be created automatically on a Call Behavior Action creation(see page 311).
- Open the Specification window(see page 913) of the Call Behavior Action, and click the cell of the **Behavior** property value. Then click and, in the opened dialog, select the Behavior. Click **OK** when you are finished.
- On the Activity diagram, right-click the Call Behavior Action shape. On the shortcut menu, click **Behavior,** and then click to select a Behavior that you wish to assign on the element.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- On the Activity diagram, click a Call Behavior Action shape, and then, on the smart manipulator toolbar, click . In the opened list, select the Behavior.
- On the Activity diagram, click a Call Behavior Action shape, and then press Ctrl+T. In the opened list, select the Behavior.
- On the diagram, click a Call Behavior Action shape, then click the name area. Type the ":" colon, and then type the existing Behavior name. Press **Enter,** or click anywhere on the diagram. The Behavior is created and assigned.

Type the ":" colon, then press Ctrl+Space or Ctrl+Backspace and then, in the opened list, select the Behavior.

- Drag a Behavior from the Containment tree(see page 32) onto the Call Behavior Action shape on the diagram. The Behavior is assigned to the Call Behavior Action.

- Double click the Call Behavior Action with the assigned Behavior - the Behavior Specification window(see page 913) opens, or, if the assigned Behavior is a diagram, the diagram opens in the same diagram tab.
- The rake icon(see page 984) is displayed on the shape if the Call Behavior Action refers to another Activity diagram. The rake icon is not shown by default, therefore, you need to set the **Show Rake Icon** property value to *true* in the **Symbol Properties** dialog(see page 975).
- A parameter is created automatically for each newly created Pin on the Call Behavior Action. The parameter's name, type, direction, and multiplicity are synchronized with the Pin name, type, direction, and multiplicity.
  Note that a parameter for a pin is created automatically if:
  - the Call Behavior Action has the Behavior assigned.
  - the Behavior is editable.
  - the Behavior has only one usage, and it is not used in another project (in server projects).

Creating a new diagram for the Call Behavior Action

To create a new diagram for the Call Behavior Action, do one of the following

---

- On the Activity diagram(see page 311), click the Call Behavior Action shape, and then, on the smart manipulator toolbar, click .The new Activity diagram opens in the same diagram tab.

- The name of the new Activity diagram is the same as the name of the Call Behavior Action.
- If a Behavior was already assigned for the Call Behavior Action, a new Activity diagram under the Behavior will be created. This is valid if the Behavior is editable and the Behavior's type is Activity.
- If Pin(s) are created on the Call Behavior Action, then the parameters for these Pins are created in a newly created activity. Pins are synchronized with the created parameters, i.e., the Pin name, type, and multiplicity are synchronized with the parameter name, type, and multiplicity.
- If parameters are created for a new Activity, then Activity parameter Nodes are created in a new Activity diagram and displayed on the Activity diagram frame.

- On the Activity diagram, right-click the Call Behavior Action shape, point to **Create Diagram,** and then point to a diagram type.

Changing the Name Display Mode on the Call Behavior Action symbol

To change the Name Display Mode on the Call Behavior Action symbol

---

- Right-click the Call Behavior Action and select the **Symbol Properties** command.
- In the **Symbol Properties** dialog(see page 975), find the **Name Display Mode** property.
- Select one of the following value:
  - **Show Action Name** (default only for UPDM projects) - to show action name.
  - **Show Behavior Name** to show behavior name.
  - **Show Both** - to show action and behavior names.
  - **Show Both or Behavior Name** (default) - to show only the behavior name if the action name is not specified, or if the action name matches the behavior name.
- Click **Close**.
  The selected name is shown on the Call Behavior Action symbol.

Related pages

- Model Elements(see page 628)
- Behavior(see page 662)
- Action(see page 629)
- Activity diagram(see page 311)

Call Operation Action

The Call Operation Action transmits an operation call request to the target object, where it can cause the invocation of an associated Behavior(see page 662).

You can display an Action name and/or name of the Operation(see page 677) on the Call Operation Action shape. For example, if you have two Call Operation Action elements calling the same Operation, you can specify their names to distinguish the Actions from each other.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Notation	Description
The Call Operation Action with the *getOrder* operation 	In this example, the Call Operation Action has been assigned for the *getOrder* operation whose type is the *OrdersDB* class.
The Call Operation Action named *Get supplementary order* 	In this example, the Call Operation Action is named *Get supplementary order*. It has been assigned for *getOrder* operation whose type is the *OrdersDB* class.
The Call Operation Action with the hidden classified name 	The classified name of an operation is hidden from the Call Operation Action shape.

To assign an Operation for the Call Operation Action, do one of the following

- On the diagram, right-click the Call Operation Action, and, on the shortcut menu, click **Select Operation**. In the opened **Select Operation** dialog(see page 1031), select an Operation you need to assign to the element, or click the **Creation Mode** button to create a new Operation. Click **OK** when you are finished.
- In the Call Operation Action Specification window(see page 913), click to select the **Operation** property specification cell, and click button . In the opened **Select Operation** dialog(see page 1031), select the Operation you need to assign to the element, or click the **Creation Mode** button to create a new Operation. Click **OK** then you are finished.

MagicDraw allows you to display the Operation name and Class (see page 666)of the Operation on the Call Operation Action shape.

When the Operation is assigned to the Call Operation Action, there are two name/operation display options available:

Description	Example
If the Call Operation Action is not named, or the name of the Call Operation Action is the same as the assigned Operation name, the name of the Class is displayed under the Operation name.	

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Description	Example
If the Call Operation Action name differs from the assigned Operation name, then *<class of the operation>::<operation name>* is displayed under the Call Operation Action name.	

To show / hide the Operation name and Class of the Operation from the Call Operation Action shape

- Open the Call Operation Action **Symbol Properties** dialog(see page 975), and set the **Show Qualified Name for Operation** property value to *true* / false.

Related pages

- Model Elements(see page 628)
- Behavior(see page 662)
- Action(see page 629)
- Activity diagram(see page 311)

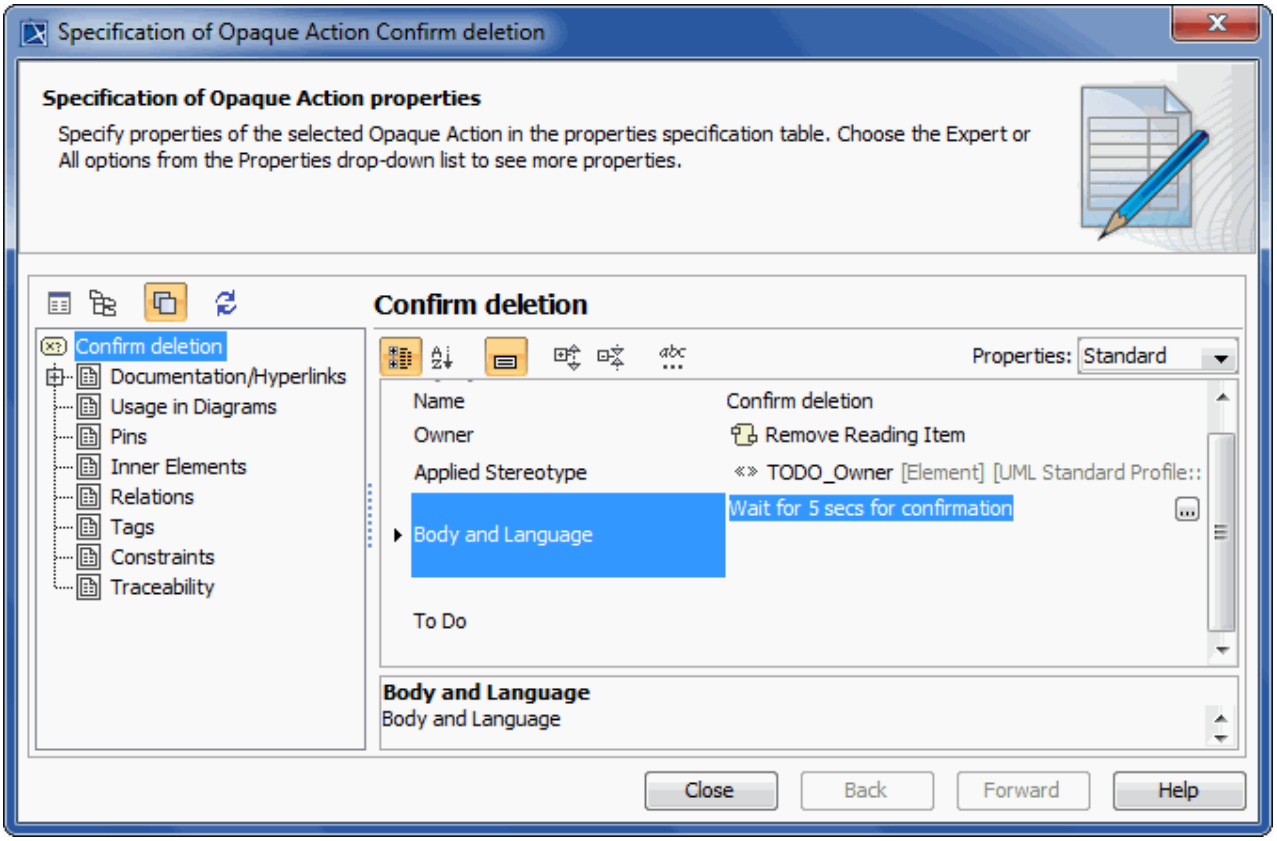
Opaque Action

The Opaque Action is introduced for implementation-specific actions, or for use as a temporary placeholder before other actions are chosen.

The Opaque Action has no specific notation.

There is an additional property named **Body and Language** in the Specification window(see page 913) of an Opaque Action.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Related pages

- Model Elements(see page 628)
- Behavior(see page 662)
- Action(see page 629)
- Activity diagram(see page 311)

Send Signal Action

The Send Signal Action creates a signal instance from its inputs and transmits it to the target object, where it can cause the start of a State Machine(see page 848) transition or the execution of an Activity. The argument values are available to the execution of associated Behaviors(see page 662). The requester continues the execution immediately. Any reply Message(see page 756) is ignored and is not transmitted to the requester. If the input is already a Signal instance, use the Send Signal Action.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Notation	Description
Send Signal Action 	This example describes an order process. 1. First, an order is created (the *Create order* Call Behavior Action(see page 633)). 2. Next, a Signal to fill the order request is sent to the warehouse (the *Fill order request* Send Signal Action). 3. Finally, an invoice is created (the *Create invoice* Call Behavior Action). The relationships are represented with Control Flow paths.

Assigning signals

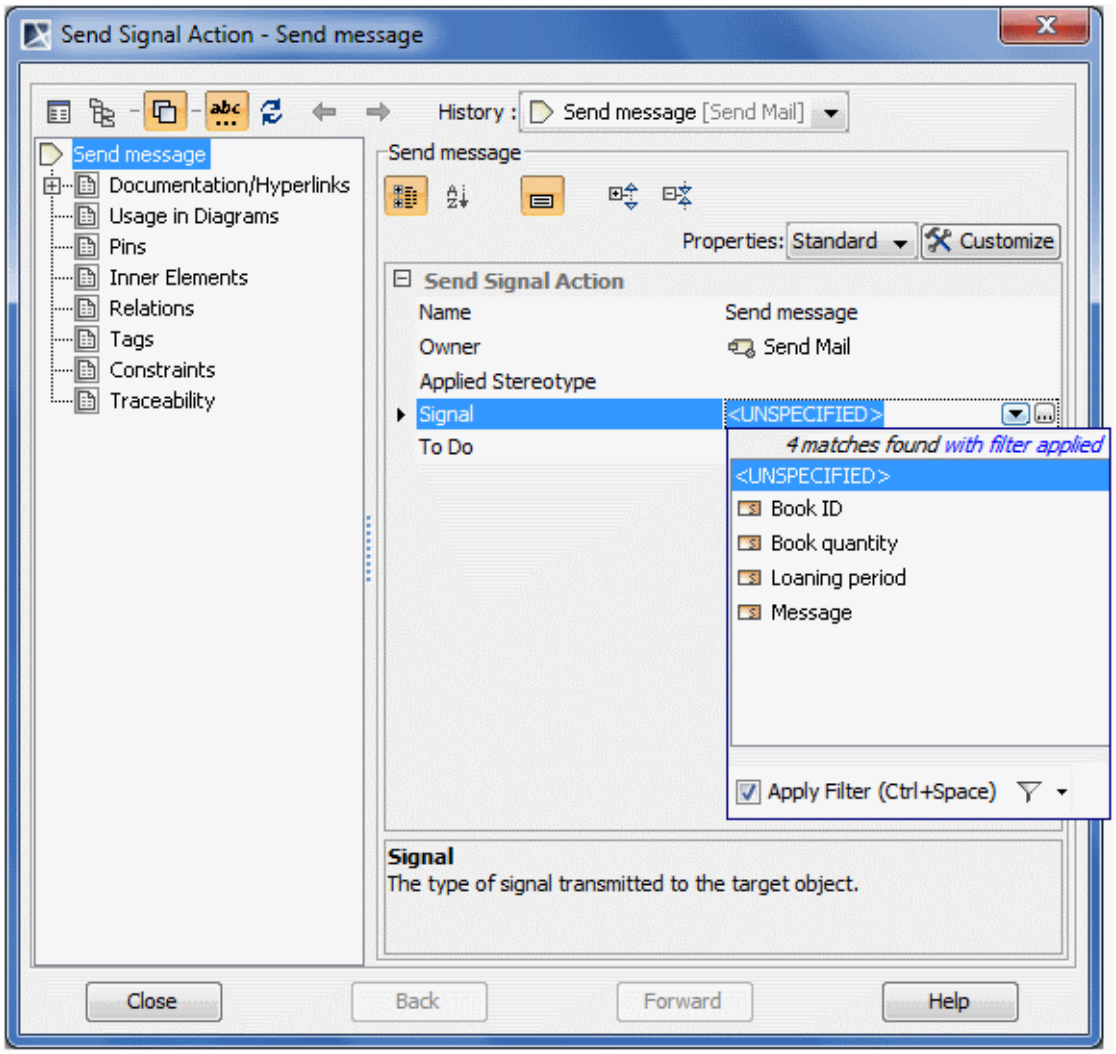
To assign a Signal to a Send Signal Action, you can use any of the following:

- the Send Signal Action's (see page 639) Specification window(see page 913)
- the drag-and-drop operation(see page 0)
- the Send Signal Action's shortcut menu(see page 640)
- type the Send Signal Action name(see page 641)

To assign a Signal via the Send Signal Action’s Specification window

1. Open the Specification window(see page 913) for the Send Signal Action.
2. In the **Signal** property value cell, do any of the following
    - Click the button. The **Select Signal** dialog opens. Select the Signal from the list, or create a new one(see page 1035).

- Click the button ▼ to open the list of Signals available in the project. Select the Signal from the list.

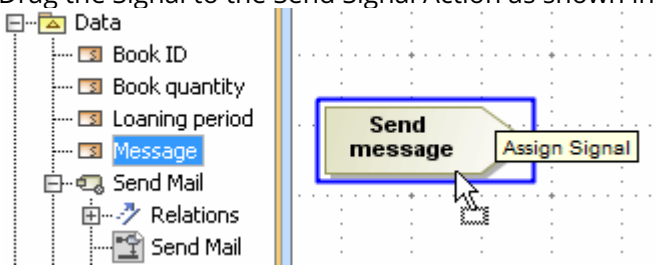


- Click **Close** once the Signal is selected.

To assign a signal using a drag-and-drop operation

---

1. Select the Signal in the Containment tree(see page 32).
2. Drag the Signal to the Send Signal Action as shown in the following picture.



To assign a signal using the send signal action shortcut menu

1. Right-click the Send Signal Action to open the shortcut menu(see page 638).
2. Click **Signal**. The list of Signals available in the project opens.
3. Select the Signal from the list, or click **New Signal** and create a new one(see page 1035).

To assign a signal by typing the Send Signal Action name

1. Select the Send Signal Action shape, click on it to activate the name edit mode.
2. Do one of the following:
   - if you want to use an existing Signal from the model, start typing an appropriate Signal name, and select it from the opened list.
   - if you want to create a new Signal in the model, type its name and press Enter.

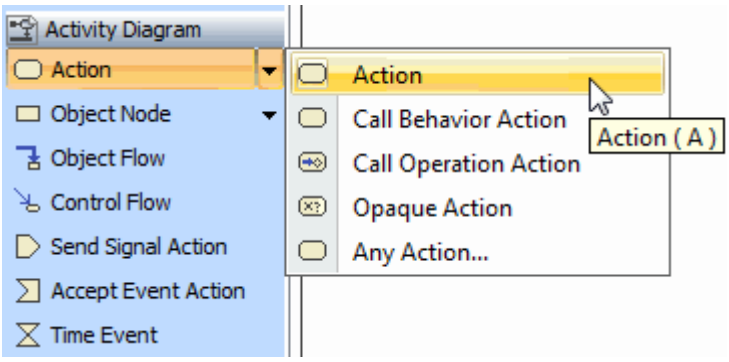
Related pages

- Action(see page 629)
- State Machine diagram(see page 308)
- Activity diagram(see page 311)
- Model Elements(see page 628)

Working with Actions

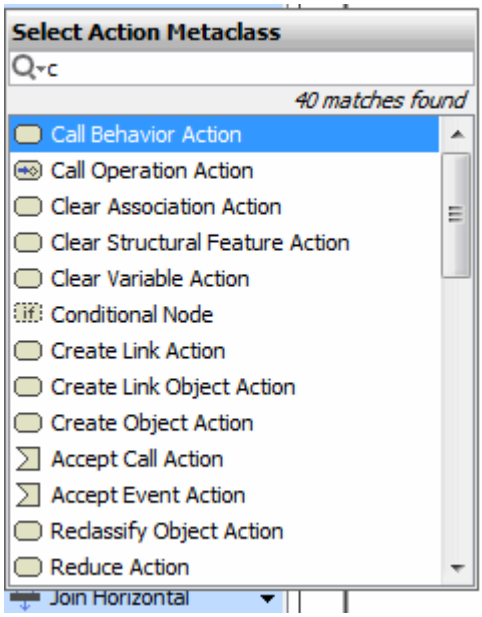
To create any action quickly

1. In the Activity diagram toolbar, right-click or expand the **Action** button. The menu opens.



2. Select the **Any Action** command. The **Select Action Metaclass** dialog opens.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



3. Select an Action metaclass from the list, or type the first letter of the metaclass in the Quick filter box(see page 918). Click **OK**. The Action is created.
4. Click the diagram pane. An action symbol is drawn.

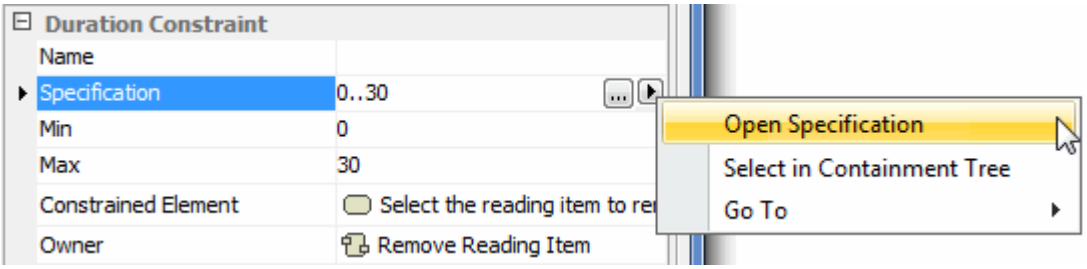
Applying duration constraint on the Action

You can create and apply a duration constraint on an Action that states that the output must occur after delay from the input.

To create and apply a duration constraint

---

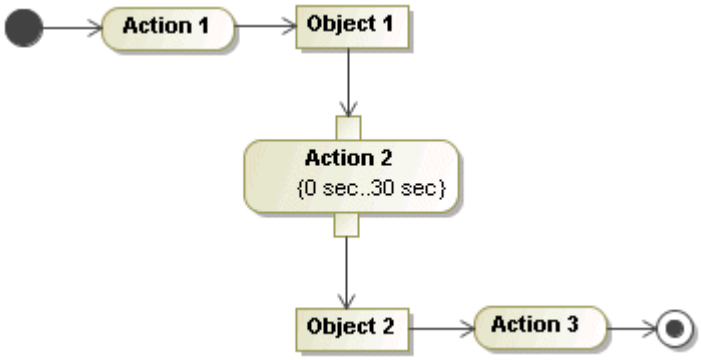
1. Select an Action and create the input and output Pins(see page 805) to specify the Events(see page 708).
2. Create a duration constraint for the Action using one of the following way:
   - From the diagram palette:
     i. In the diagram palette, under the Common category, expand the **Constraint** command, and select the Duration Constraint, Time Constraint, or Constraint button, and create it directly on the action shape.
   - In the Specification window:
     a. In the Call Behavior Action(see page 633) Specification window(see page 913), click the **Constraints** group.
     b. Click the **Apply** button. The **Select Constraint** dialog opens.
     c. Select the constraint storage place, and click the **Create** button. In the opened menu, select the **Duration Constraint** command. The **Duration Constraint** Specification window opens.
3. Specify a duration interval. Type the minimum and maximum duration to hold the Activity in the **Min** and **Max** property specification cells, e.g., 0 sec and 30 sec.
4. Assign the Events for the input and output pins:
   a. In the opened **Duration Constraint** Specification window, click the **Specification** property specification cell, and click the Show Shortcut Menu button . The following shortcut menu opens:



b. Select the **Open Specification** command. The **Duration Interval** Specification window opens.
c. Click the **Min** property specification cell, and and click the Show Shortcut Menu button . In the shortcut menu, select the **Open Specification** command. The **Duration** Specification window opens.
d. In the Event field, click the "..." button. The **Select Element** dialog opens. Select the activity input pin.
e. Repeat steps c and d for the Max property - select the action output pin as an event.

5. Apply the created duration constraint on the action.

The following sample depicts a fragment of the Activity diagram(see page 311) with a duration constraint applied on the Action.



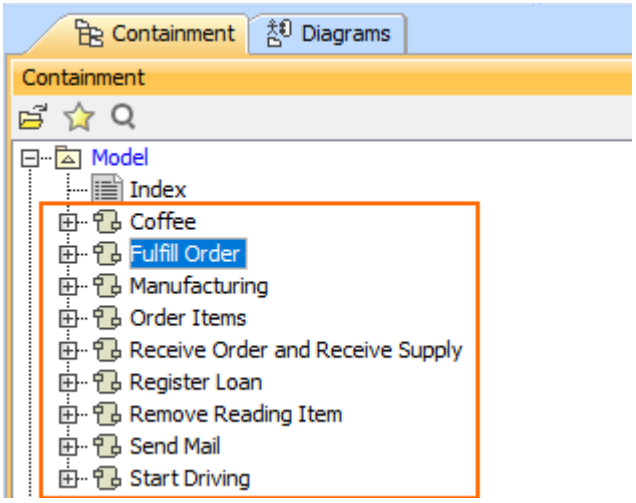
Related pages

- Model Elements(see page 628)
- Behavior(see page 662)
- Action(see page 629)
- Activity diagram(see page 311)

Activity

An Activity is a parameterized Behavior(see page 662) represented as a coordinated flow of Actions(see page 629). An Activity specifies the coordination of executions of subordinate behaviors, using a control and data flow model. The subordinate behaviors coordinated by these models may be initiated because other behaviors in the model finish executing, objects and data become available, or events occur external to the flow. The flow of execution is modeled as activity nodes connected by activity edges.

An Activity element is created in the Activity diagram.



An Activity can be specified by changing its property values in the Activity Specification window(see page 913). Each property is described in the description area on this window.

General specification pane properties

Property Name	Function
Read Only	If true, this activity must not make any changes to variables outside the activity or to objects. (This is an assertion, not an executable property. It may be used by an execution engine to optimize model execution. If the assertion is violated by the action, then the model is not formed correctly.) The default is false (an activity may make non-local changes).
Single Execution	If true, all invocations of the activity are handled by the same execution.
Node	Nodes coordinated by the activity.

In the Activity Specification window(see page 913), choose Variables(see page 645) from the property group list.

Related pages

- Activity diagram(see page 311)
- Behavior(see page 662)
- Action(see page 629)
- Specification window(see page 913)
- Variables(see page 645)

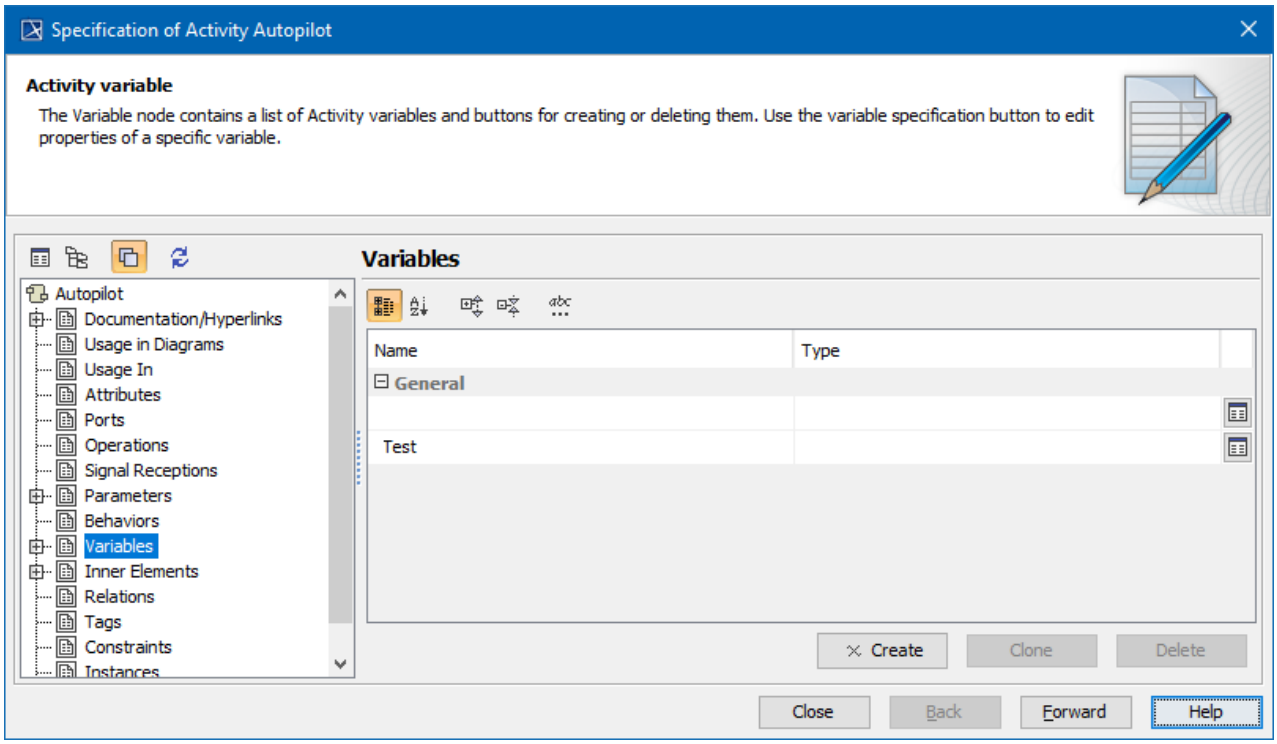
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Variables

Variables are elements that indirectly pass data between Actions(see page 629). A local variable stores values shared by the actions in a structured Activity(see page 643) group, but not accessible outside the group. The output of an action can be written to a variable and read for the input to a subsequent action, which is effectively an indirect data flow path. Because there is no predefined relationship between actions that read and write variables, these actions must be sequenced by control flows to prevent race conditions that may occur between actions that read or write the same variable.

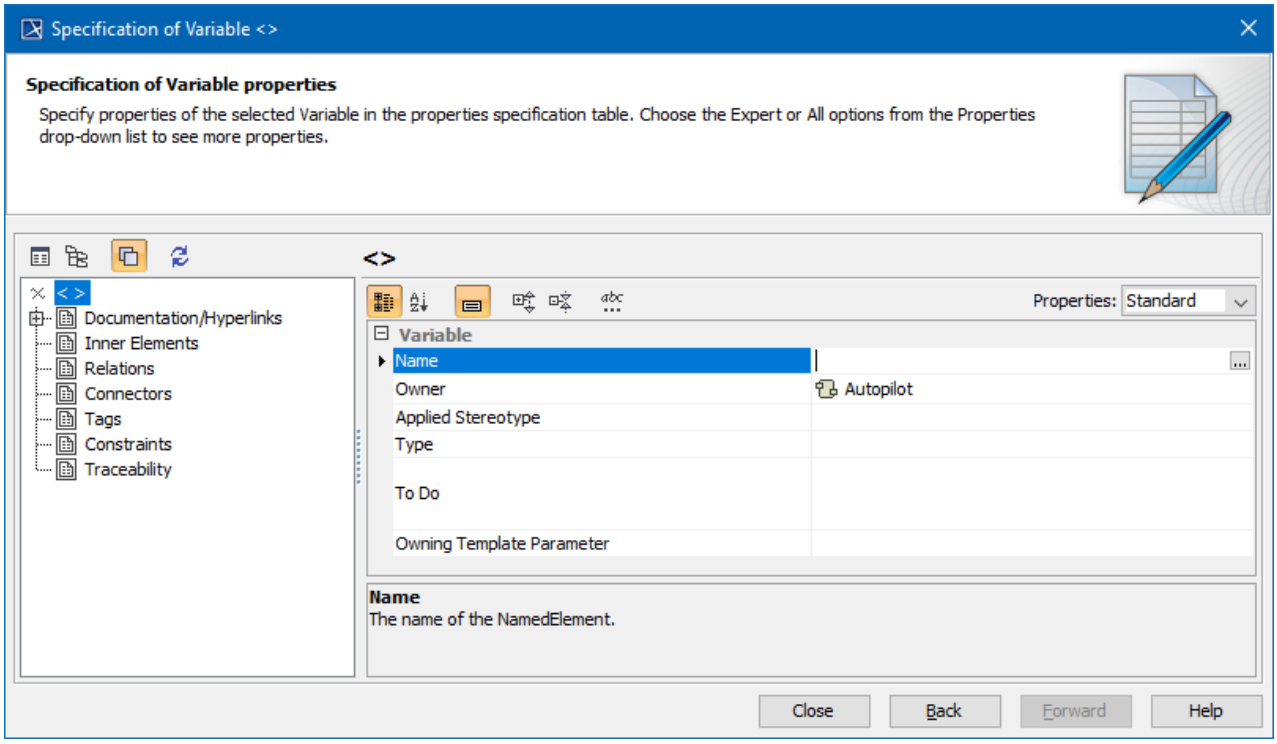
To create a Variable

1. Open Activity Specification window(see page 913).
2. Choose Variables from the property group list.
3. To create a new Variable click **Create**.
4. Specify the Variable in the Specification of Variable window.



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Creating variables in the Specification of Activity dialog



Specification dialog for a Variable

Activity Final Node

An Activity Final Node (symbol a circle surrounding a smaller solid circle) is a final node that stops all flows in an activity.

An Activity Final Node can be specified by changing its property values in the Activity Final Node Specification window(see page 913). Each property is described in the description area on this window.

Actor

Actors represent roles played by human users, external hardware, and other subjects. An Actor does not necessarily represent a specific physical entity but merely a particular “role” of some entities that is relevant to an action performed by the Actor. Typically, such actions are called Use Cases(see page 876). An Actor is represented as a “stick man” figure with the name below the figure.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

You can format the actor symbol properties in the Symbol Properties dialog(see page 975).

An Actor can be specified by changing its property values in the Actor Specification window(see page 913). Each property is described in the description area on this window.

Actors can be represented in other diagrams, such as Sequence(see page 304) or Activity diagrams(see page 311).

Actor usage in other diagrams

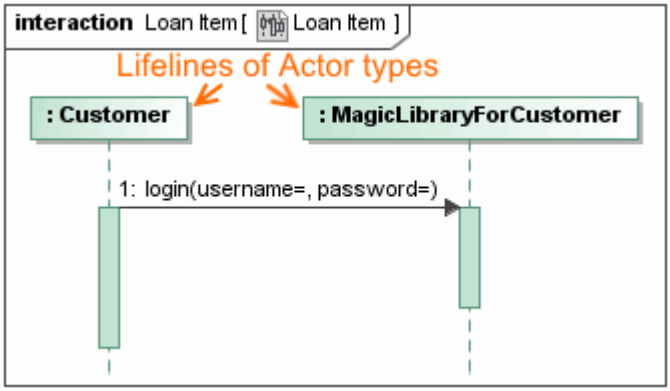
The actor can be used in other diagrams in various ways. Most common cases are as follows:

- Actors can be depicted as (see page 647) Lifelines (see page 746) in a Sequence diagram.(see page 647)
- Actors can be depicted as (see page 647) Swimlane (see page 861) headers in an Activity diagram(see page 647).

To create a Lifeline (see page 746)for an Actor in a Sequence diagram

---

1. Open the Sequence diagram wherein you want to create a lifeline.
2. Select an Actor in the Model Browser(see page 29).
3. Drag and drop the selected Actor on the Sequence diagram pane. The Lifeline is created. In the following figure, see an example wherein two Lifelines are depicted.

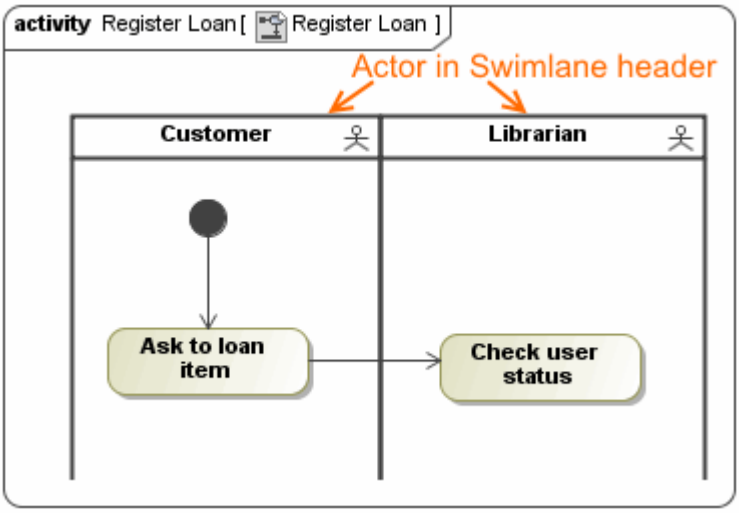


To create a Swimlanes(see page 861) header for an Actor in an Activity diagram

---

1. Open an Activity diagram.
2. Create a Swimlane.
3. Select an Actor in the Model Browser(see page 29).
4. Drag and drop the Actor on the Swimlane header in the Activity diagram. The Actor is assigned to the header of the Swimlane. See the example in the following figure.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



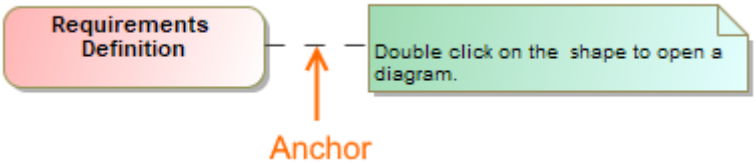
> You can see the usage of Actors in the example of Use Case diagram(see page 300) in *<MagicDraw installation directory>\samples\diagrams\use case diagram.mdzip*.

Related pages

- Model Elements(see page 628)
- Use Case(see page 876)
- Use Case diagram(see page 300)
- Sequence diagram(see page 304)
- Activity diagram(see page 311)

Anchor

Use an Anchor to relate an element to a note(see page 791) or comment(see page 683). By default, a newly created Anchor links an element with a note.



Example of an Anchor relating an Action to a Note.

You can format the anchor symbol properties in the Symbol Properties(see page 975) dialog.

To draw an anchor from a symbol

1. On the diagram's pane, select an symbol: class, signal, data type, etc.

2. In the smart manipulator toolbar appearing on the side of the symbol, click on the Anchor button.
3. If you drag the shadowed Anchor anywhere on the diagram pane, a new note appears anchored to the symbol.
4. Or you can click the note or comment to which you want to anchor the element.
5. The Anchor is created from the symbol.

To draw an anchor from the diagram pane

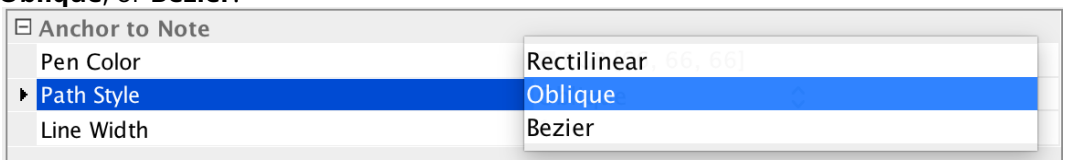
---

1. Click on the diagram's pane to create a new anchored note.
2. Drag it anywhere on the diagram pane and see a new anchored note appear.
3. Now, a new note is anchored to your diagram pane.

To change an anchor path style

---

1. On the diagram pane, select an anchor and open its shortcut menu.
2. Select **Symbol Properties**.
3. Select the **Path Style** proeprty, then choose one of the following path style: **Rectilinear**, **Oblique**, or **Bezier**.



Related pages

- UML elements(see page 628)
- Editing Property Values(see page 921)
- Specification window(see page 913)
- Note(see page 791)
- Comment(see page 683)
- Formatting symbols(see page 975)

Artifact

An Artifact represents a physical piece of information that is used or produced by a software development process. Examples of artifacts include models, source files, scripts, and binary executable files. An artifact may constitute the implementation of a deployable component.

To describe how an artifact is created, the model draws a Manifestations(see page 755) relationship, between the artifact and the elements used in its construction. All artifact manifestations are listed in the Artifact Specification window(see page 913), Manifestations pane. In this pane you may add new manifestations.

To be used, an artifact must be deployed, meaning, it must be placed in a location where it may be stored and used. The target of the deployment may be a node, such as an execution environment or a device, a property, or an instance specification. The relationship between the location and the artifact is defined by the deployment(see page 649) relationship.

Related pages

- UML elements(see page 628)
- Editing Property Values(see page 921)
- Specification window(see page 913)
- Manifestations(see page 755)
- Formatting symbols(see page 975)
- Component diagram(see page 318)
- Deployment diagram(see page 319)

Deployment

The Deployment Specification is a type of Artifact. The Deployment Specification is a general mechanism to parameterize a Deployment relationship.

To draw the Deployment Specification on the diagram pane

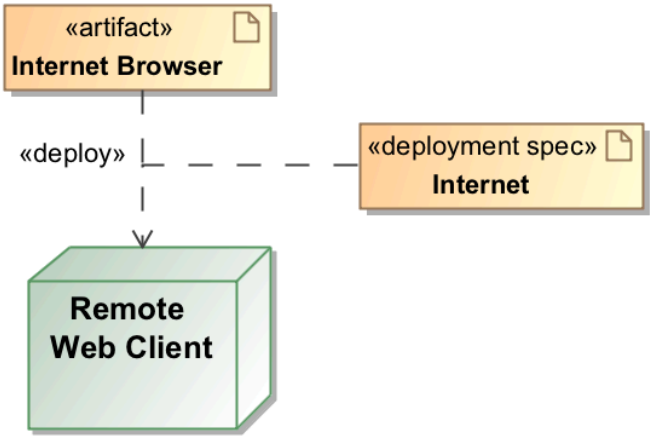
1. In the Component(see page 318) (or Deployment(see page 319)) diagram toolbar, click the Artifact[138] button group to access the drop-down menu.
2. In the opened list, select Deployment Specification and draw it on your diagram pane.

To specify the Deployment relationship

1. Create the Deployment Specification on the diagram pane.

138 https://docs.nomagic.com/display/CBMP190SP4/Artifacts

2. Between the Deployment relationship and Deployment Specification, draw the Dependency relationship. The Dependency is drawn without an arrow.



Related pages

- Model Elements(see page 628)
- Component diagram(see page 318)
- Deployment diagram(see page 319)
- Artifact[139]
- Working with Dependencies(see page 702)
- Dependency(see page 699)

Association

In the Class diagrams(see page 297), an Association represents the semantic relationship between two or more classifiers which specifies connections between their instances. An Association relationship is the most general of all relationships and the most semantically weak.

In the Use Case diagrams(see page 300), an Association represents the participation of an Actor (see page 646)in a Use Case(see page 876), for example, when instances of an Actor and instances of a Use Case communicate with each other. This is the only relationship between Actors and Use Cases. Sometimes an association relationship is called communication Association.

An Association is drawn as a solid path connecting two classifier symbols.

The communication path is a subclass of an Association. It specifies the relationship between nodes by defining the number of nodes that may be connected (multiplicity), and the nature of the connection, via the name of the path or a stereotype.

Related pages

- Working with Associations(see page 652)
-

139 https://docs.nomagic.com/display/CBMP190SP4/Artifacts

Association End(see page 654)
- Association Class(see page 660)
- Association in Use Case diagrams(see page 661)
- Communication Path(see page 662)

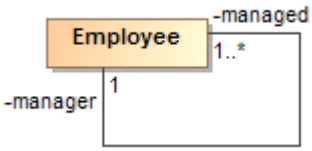
- Model Elements(see page 628)
- Class diagrams(see page 297)
- Use Case(see page 876)
- Use Case diagram(see page 300)
- Sequence diagram(see page 304)
- Activity diagram(see page 311)

Working with Associations

An association(see page 651) is a relationship between two classes. It is represented by a solid line. Though associations are bi-directional by default, they can also be unidirectional, where one class knows about the other class and the relationship but the other class does not. Sometimes an association links a class to itself. This association is called a recursive (reflexive) association. One example of this association is an employee who is managed by a manager who is also himself an employee.

To draw a recursive (reflexive) association

1. Select **Association**.
2. Click a class (**Employee** in this example) and hover over the class until you see a rectangle shape. Click the class again to create a recursive association.



Association Properties

You can specify association properties in the association Specification window(see page 913). The description of each property appears in the same window. The description area of the Specification window presents the descriptions.

To show the direction arrow near the association name

1. Do one of the following:
   - Right-click the association, and from the shortcut menu, select **Show Direction Arrow**.
   - Open the association Symbol properties(see page 975) dialog and set the **Show Direction Arrow** property to *true*.

This command is available in the Full Featured perspective(see page 67).

The default Direction Arrow direction is displayed according the path creation direction.

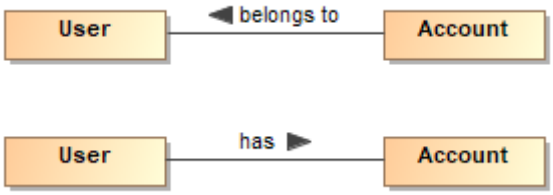
To change the **Direction Arrow** direction

1. Right-click the association.
2. From the shortcut menu, select **Reverse Direction Arrow**.

The Direction Arrow is a graphical representation most often used in top level domain class diagrams. It helps to read a diagram and explains diagram semantics, and has no meaning in a model.

Usually, you would use the Direction Arrow on a diagram where navigability is not yet defined. Direction Arrows are usually displayed for named associations. When you create more detailed diagrams with specified navigability, direction arrows and association names are not usually displayed in these types of diagrams.

As shown in the following figure, *User* and *Account* Classes are connected with the Association. A navigation arrow can be displayed to either side depending on the Association name. If the association name is *belongs to*, the Direction Arrow should point from the *Account* to the *User*. If Association name is *has*, the Direction Arrow should point from the *User* to the *Account*.



Examples of Direction Arrows

In the Class diagram(see page 297), you can add Attributes(see page 672) to an Association using an Association Class(see page 660). The Association Class is a simple Class (see page 666)with a dashed line connected to the Association.

Adding Association between Read-Only Classifiers

Adding a new Association always creates two roles or properties at both ends. They are owned by the attached Classifier by default. However, when one or both ends of the Association are not editable (for example, locked in a server project or stored in a read-only profile or used project), the properties will be owned by the Association itself. In this case, a warning appears informing you about any unexpected issue in a model creation.

Related pages

- Editing(see page 921) Property Values(see page 921)
- Specification window(see page 913)
- Customizing and Selecting Perspective(see page 67)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Association End

On this page

- Association end properties(see page 654)
- Editing the Association End properties(see page 654)
- Defining Association End qualifiers(see page 656)
- Defining Association End aggregation(see page 656)
- Showing Association End properties(see page 657)
- Converting Association End properties(see page 657)
- Association navigability(see page 658)
- Advancing actions: navigable owned Association Ends(see page 659)

Association Ends are represented by properties each of which is connected to the type of the End. When a property is an Association End, the value or values are related to the instance or instances at the other End(s) of the Association(see page 651).

An Association End is the connection between lines depicting an Association and the shape.

The following properties of the association end can be specified: name, association end type, visibility, multiplicity, qualifier, aggregation kind.

Association end properties

Name

The Association End has other name - role. A role indicates a role played by the Class(see page 666) in terms of an Association. The role name is placed at the Association End near the Class playing that role. The role name at the implementation level maps to the reference name of the opposite Class. Roles can have visibility (public, package, protected, and private).

Association End type

Changing the Association End type, changes the target of the Association or in other words the classifier to which the Association is connected.

Qualifier

A qualifier is an attribute or a list of attributes whose values serve to partition the set of instances associated with an instance across an Association. Qualifiers are attributes of an Association. It is represented as a small rectangle attached to the End of an Association path between the final path segment and the symbol of the classifier that it connects to. The qualifier rectangle is part of the Association path, not part of the classifier. The qualifier rectangle drags with the path segments. The qualifier is attached to the source end of the Association.

Editing the Association End properties

To change the Association End type, do one of the following

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- On the diagram, select the Association End and move it to the other target. The Association is connected to the other target and the type of the Association End (that is, property) is changed.
- In the Containment tree(see page 32), select the property which represents the Association End and drag it onto the available classifier (the Association target). If the Association is represented on the diagram pane, the Association is redrawn (connected) to the changed target automatically.
- In the property (that is, the Association End) Specification window(see page 913), change the **Type** property value.

To define the Association End visibility

1. Open the Association End Specification window.
2. From the **Visibility** list, select one of the visibility type.

The Association End multiplicity describes how many entities are participating at each Association End:

- 0 – zero and only zero.
- 1 – one and only one.
- 0..1 – zero or one.
- 0..* – from zero to any positive integer.
- 1..* – from one to any positive integer.
- * – any positive integer.

To place multiplicity values in Association path ends

- Open the shortcut menu of a selected Association End and click a desired multiplicity.
- Open Association's shortcut menu, point to one of a desired Association End (Role of <*class name*>), and then click the desired multiplicity value.
- Open the Association Specification window and, from the **Multiplicity** list, select or type the multiplicity value for the desired Association End.
- Perform the following steps:
    a. Open the Association End Specification window.
    b. In the **Multiplicity** property value cell, type or select from the list a multiplicity value.
- Edit directly on the diagram pane:
    a. Select multiplicity area and press F2 to switch it to edit mode.
    b. Press Ctrl + Spacebar or Ctrl + Backspace to see available suggestions.

  Choose one of the suggestions or type the multiplicity value.

To define an Association End name

- Open the shortcut menu of a selected Association End and click **Edit Name**. The Association End is marked for editing. Type or edit the name directly on the diagram pane.
- Open the Association's shortcut menu, point to one of a desired Association End (**Role of <*class name*>**), and then click **Edit Name**. The Association End is marked for editing. Type or edit the name directly on the diagram pane.
- Perform the following steps:
    a. Open the selected Association End Specification window.
    b. Type the Association End name in the **Name** property value cell.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Defining Association End qualifiers

A qualifier is an attribute or a list of attributes whose values serve to partition the set of instances associated with an instance across an Association. Qualifiers are attributes of an Association. It is represented as a small rectangle attached to the end of an Association path between the final path segment and the symbol of the classifier that it connects to. The qualifier rectangle is part of the Association path, not part of the classifier. The qualifier rectangle drags with the path segments. The qualifier is attached to the source end of the Association.

To add, edit, or remove a qualifier to/from an Association End

---

1. Open the Association End Specification window.
2. In the property group list, click the **Qualifiers** group and do one of the following:
   - To add a new qualifier, click the **Create** button. In the opened qualifier Specification window, define qualifier properties and click **Back** to return to the Association End Specification window or click **Close** to exit the Specification window.
   - To edit a qualifier, select a qualifier and edit its properties in the Association End Specification window or click the Open Specification button at the end of a qualifier row to edit qualifier properties in the qualifier Specification window.

     > Existing qualifiers are also displayed in the property group list under the **Qualifiers** group. Click the selected qualifier to open its Specification window, and define qualifier properties there.

   - To remove a qualifier, select a qualifier and click **Delete**.
3. After you have finished working with qualifiers, click **Close** to exit the Specification window.

Defining Association End aggregation

The shared or composite aggregation kind can be assigned to the Association End. A composite aggregation is represented as a filled diamond. A shared aggregation is represented as a hollow diamond.

To assign an aggregation kind to the association end

- Open Association's shortcut menu, point to one of a desired Association End (**Role of *<class name>***), and then select one of the following:
  - **None**. No aggregation is assigned to the selected Association End.
  - **Shared**.
  - **Composite**.
- On the diagram palette(see page 24), click the **Composition** or **Aggregation** button and draw an appropriate path on the diagram.

- Right click the Association path end and select **Shared** or **Composite** command from the shortcut menu.
- Perform the following steps:
  a. Open the selected Association or Association End Specification window.
  b. For the selected Association End, click **Aggregation** property value cell to select an aggregation kind (**None**, **Shared**, or **Composite**).

  > ⚠ By default, the **Aggregation** property is in the **All** properties display mode.

Showing Association End properties

If two classes are linked with an Association path, both Classes have an attribute of an opposite Class type. This property can be displayed on the Class shape as well as an Association link.

To show Association Ends as attributes on linked Class shapes

---

1. From the Class shortcut menu, select **Symbol Properties** or press Alt+Enter. The **Symbol Properties** dialog(see page 975) opens.
2. In the **Attribute** category, click the **Show Association End as Attributes** property value cell.
3. Click one of the following display modes:
   - **All**. Properties and Association paths will be displayed on the Class shape.
   - **Without Association Symbol**. If an Association symbol is deleted, the property will be displayed on the Class shape.
   - **Do Not Show**. Neither property, nor Association path will be displayed on the Class shape.

An Association End is defined as a property. It has attribute properties defined in the Specification window.

To show Association Ends as Ports on the shape

---

1. Select the shape on which you want to display Association Ends as Ports and open its **Symbol Properties** dialog(see page 975):
   - Right-click a shape to open the shortcut menu, and click **Symbol Properties**.
   - Press Alt+Enter.
2. In the **Symbol Properties** dialog, at the right-top corner, click to expand the menu and select the **All** mode.
3. Under the **Ports** category, click the **Show Association Ends as Ports** property cell and select *All*.

Converting Association End properties

To convert an Association role to an attribute

---

1. On a diagram pane, select an Association with a role.
2. From its shortcut menu, select the **Refactor** > **Convert To** > **Attribute(s)**.

The Association is converted to the attribute.

You can specify Association End properties in the Association End Specification window. In the same window, you can find the description of each property. Descriptions are presented in the description area of the Specification window.

To convert association ends to ports

1. Right- click an Association End(s).
2. On the shortcut menu, click **Refactor** > **Convert To** > **Port**.

Association navigability

The Association navigability indicates whether it is possible to traverse an Association within an expression of a classifier to obtain the object or a set of objects associated with instances. The navigability is shown as an arrow that can be attached to the end of the path to indicate that the navigation is supported toward the classifier attached to the arrow.

> ⚠ By default, an Association is navigable on both sides and its navigability is not visible.

A role indicates the role played by the Class in terms of an Association. The role name is placed at the Association End, near the Class playing that role. The role name at the implementation level maps to the reference name of the opposite Class. Roles can have visibility (public, package, protected, and private).

To change the Association navigability, do one of the following

- Open the Association End Specification window, turn on the Expert property display mode, and do one of the following:
  - Set the **Navigable** property value to *true* to mark the Association End as navigable.
  - Set the **Navigable** property value to *false* to mark the Association End as not navigable.
- From the Association shortcut menu, select **Role of *<class name>*** and then do one of the following:
  - Select **Navigable** to mark the Association End as navigable.
  - Click to clear **Navigable** to mark the Association End as not navigable.
- Open the Association End shortcut menu and do one of the following:
  - Select **Navigable** to mark the Association End as navigable.
  - Click to clear **Navigable** to mark the Association End as not navigable.

To display the Association navigability

- From both Association Ends shortcut menu, select **Show Navigability**.

In the following figure, the Association is navigable on both sides and its navigability is visible.

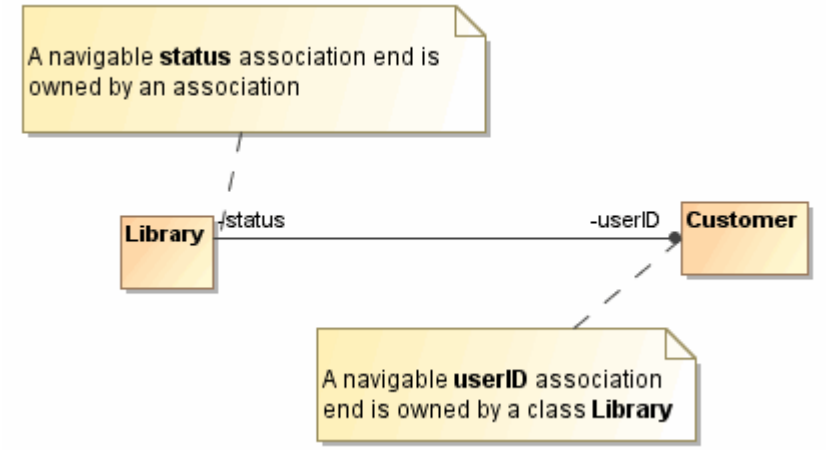
Advancing actions: navigable owned Association Ends

The navigability describes the need for an object to access another object by navigating across the link. The Association End can be owned by a classifier or an Association. The Association End owned by a classifier can be decorated with the dot. The absence of the dot signifies the ownership by the Association.

> ⚠ In MagicDraw, the dot notation is not enabled by default. Please pay attention to this before making decisions about the Association End ownership just from the model representation on a diagram.

To enable a dot notation

1. On the **Options** menu, click **Project**. The **Project Options** dialog(see page 214) opens.
2. In the **General** options list, select **General** and set the **Enable Dot Notation for Associations** value to *true*.



According to the UML 2.4.1 specification, the Association Ends owned by Classes and Associations is navigable. This improved functionality allows a proper management of the *navigableOwnedEnd* property for Associations:

- The ownership of an Association End can be changed manually.
- The navigability of Association Ends owned by the Associations can be changed keeping the ownership.

To change the ownership of an Association End, do one of the following

- On the Association End shortcut menu, click **Owned By** and then select the desired owner.
- In the Association End Specification window, click the **Owned By** property value cell and then select the desired owner from the list.
- From the Association shortcut menu, select the role of the desired Association End, click **Owned By** and then select the desired owner.
- In the Association Specification window, click the **Owned By** property value cell of the role with the desired Association End and then select the desired owner from the list.

Related pages

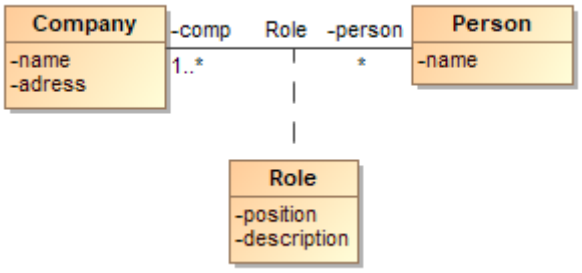
- Model Elements(see page 628)
- Class diagrams(see page 297)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Use Case(see page 876)
- Use Case diagram(see page 300)
- Sequence diagram(see page 304)
- Activity diagram(see page 311)

Association Class

An Association Class is a model element with both Association (see page 651)and Class(see page 666) properties. You can look at an Association Class as an Association that also has Class properties, or as a Class that also has Association properties. Additionally, it connects a set of classifiers and defines a set of features belonging only to the relationship and not to any of the classifiers.



Example of an association class.

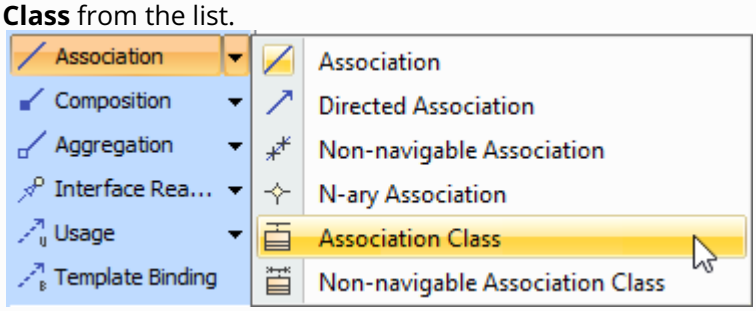
You can edit element property values(see page 921) in the Specification window(see page 913).

To draw an association class

1. Draw two Classes.
2. Click the **Association Class** button on the diagram palette.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Association Class button
If you do not see the **Association Class** button in the diagram palette, click the black arrow next to the **Association** button to see the list of associations. Select **Association Class** from the list.



3. On the diagram pane, click the first Class shape (path source).
4. Drag the path to the second Class (path destination) and drop it there.

If you need to model a relationship among several classes, you should use the N-ary Association. The N-ary association is drawn as a big diamond with all associations attached to its points. Every involved Class can have a role name and multiplicity.

To draw an N-ary association class

---

1. Draw several Classes on the diagram pane.
2. Draw the N-ary Association connector shape.

N-ary Association button
If you do not see the **N-ary Association** button in the diagram palette, click the black arrow next to the **Association** button to see the list of Associations. Select **N-ary Association** from that list. See the figure above.

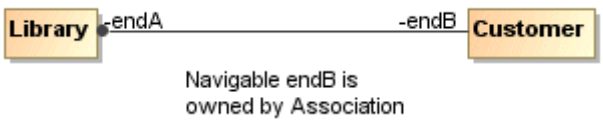
3. Connect all classes to the N-ary Association using an Association relationship.

Related pages

- Specification window(see page 913)
- Editing property values(see page 921)
- Formatting symbols(see page 975)

Association in Use Case diagrams

An Association shows the participation of an Actor (see page 646)in a Use Case(see page 876), for example, instances where an Actor and a Use Case communicate with each other. This is the only relationship between Actors and Use Cases. The association relationships are also known as communication associations.



Communication Path

A Communication path is a type of association between nodes in a deployment diagram that shows how the nodes exchange messages and signals.

When you specify communication paths, you can define the number of nodes that can be connected at each end. You can also use labels to identify the type of protocol or network that is used in the communication.

Behavior

A Behavior describes how the states of objects changes over time.

Behavior types are as follows:

- Activity(see page 643)
- Function Behavior(see page 718)
- Interaction(see page 739)
- Opaque Behavior(see page 799)
- Protocol State Machine(see page 310)
- State Machine(see page 308)

Behaviors are used:

- To specify an Entry, Do, and Exit activities(see page 849) of a State(see page 848).
- To specify an Effect(see page 874) of a Transition(see page 872).

The Behaviors are displayed in the compartments of the following elements:

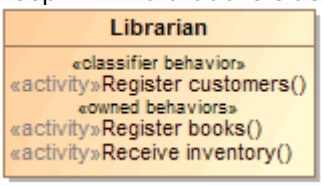
- Class(see page 666)
- Component(see page 685)
- Stereotype(see page 856)
- Node(see page 782)
- Actor(see page 646)
- Use Case(see page 876)
- Collaboration(see page 680)
- Interaction(see page 739)
- State Machine(see page 308)
- Activity(see page 311)
- Opaque Behavior(see page 799)
- Association Class(see page 651)

Behaviors are divided into two groups:

- classifier Behaviors and
- owned Behaviors.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Keep in mind that the **classifier** Behavior group can have only **one** Behavior.



Working With Behavior

Related Pages

- Formatting Symbols(see page 975)
- Customizing Environment Options(see page 65)
- Specification Window(see page 913)
- Model Elements(see page 628)

Related Diagrams

- Activity(see page 311)
- Sequence(see page 304)
- State Machine(see page 308)
- Protocol State Machine(see page 310)

Assigning Behaviors

See references on how to assign the behavior for the following elements:

- Action(see page 629)
- State(see page 848)
- Transition(see page 872)
- Operation(see page 677)

See the following procedures, on how to assign the behavior for the rest of the elements,

- Class(see page 666),
- Component(see page 685),
- Stereotype(see page 856),

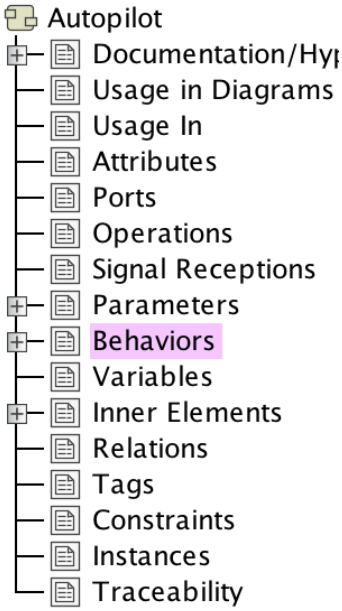
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Node(see page 782),
- Actor(see page 646),
- Use Case(see page 876),
- Collaboration(see page 680),
- Interaction(see page 739),
- State Machine(see page 308),
- Activity(see page 311),
- Opaque Behavior(see page 799), and
- Association Class(see page 651).

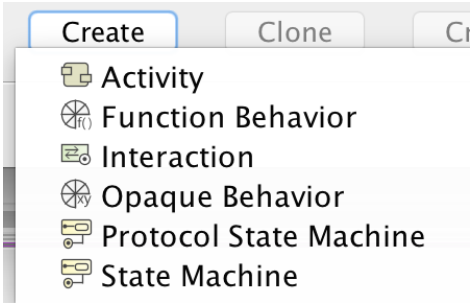
To create and assign a behavior

There are two ways to create and assign behaviors, so do one of the following:

- Create a behavior in the element's Specification Window.
  a. Open the element's Specification Window and click the **Behaviors** property group.



  b. Click the **Create** button and then from the menu, select the behavior type.



  c. In the behavior's Specification Window, specify the behavior and click **Close**.
- Right-click the element in the Model Browser(see page 29), on its shortcut menu, click **Create Diagram** and then click to create **Activity**, **Sequence**, **State Machine** or other behavior diagram.

To create an operation for the behavior

1. In the element's Specification Window, click **Behaviors** property group.
2. Click the behavior, then click the **Create Operation** button.

If the Specification property is defined already, then the **Create Operation** button is unavailable.

Related pages

- Behavior(see page 662)
- Assigning Behavior diagrams automatically(see page 665)
- Specification Window(see page 913)
- Model Elements(see page 628)

Related diagrams

- Activity(see page 311)
- Sequence(see page 304)
- State Machine(see page 308)
- Protocol State Machine(see page 310)

Assigning Behavior diagrams automatically

When assigning a behavior of an Activity, Interaction, Protocol State machine, or State Machine type for a state or transition, the corresponding behavior diagram is created automatically:

- an Activity diagram(see page 311) for an Activity behavior
- a Sequence diagram(see page 304) for Interaction behavior
- a Protocol State Machine diagram(see page 310) for Protocol State Machine behavior
- a State Machine diagram(see page 308) for State Machine behavior

Double-click the assigned behavior on a state shape to open the diagram. You can also open the behavior diagram from the Model Browser(see page 2310).

To disable an automatic diagram creating

1. On the **Options** menu, click **Environment**. The Environment Options dialog opens.
2. In **General** options, **Editing** category, set the **Create behavior diagram when the behavior type is selected for transition effect or state activities** value to *false* (the default value is *true*).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

You can specify behavior properties in the Specification window. In the same window, you can find the description of each property. Descriptions are presented in the description area of the Specification window.

More information
For more information about the Specification window usage, see Specification window(see page 913).

Related pages

- Specification Window(see page 913)
- Behavior(see page 662)
- Assigning Behaviors(see page 663)
- Editing property values(see page 921)
- Model Elements(see page 628)

Related diagrams

- Activity(see page 311)
- Sequence(see page 304)
- State Machine(see page 308)
- Protocol State Machine(see page 310)

Class

A Class is drawn as a solid-outline rectangle with three compartments separated by horizontal lines. The top name compartment holds the Class name and other general properties of the Class (including stereotype(see page 856)); the middle list compartment holds a list of properties; the bottom list compartment holds a list of operations. The property and operation compartments are optional, and you can suppress(see page 570) them.

A Class is a descriptor for a set of objects with similar structures, behaviors, and relationships. The model is concerned with describing the intention of the Class, which are the rules that define it. The run-time execution provides its extension that are its instances(see page 735).

A Class represents a concept within the system being modeled. It has a data structure, Behavior(see page 662), and relationships to other elements. The name of a Class has a scope within the package in which it is declared, and the name must be unique, among Class names, within its Package(see page 800).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Classes are declared in the Class diagrams(see page 297) and are used in most of other diagrams. UML provides a graphical notation for declaring and using these Classes as well as a textual notation for referencing Classes within the descriptions of other model elements.

What To Do With Classes

Related Pages

- Class diagram(see page 297)
- Class Diagram Wizard(see page 1347)
- Behavior(see page 662)
- Package(see page 800)
- Model Elements(see page 628)

Working with Classes

A Class(see page 666) can be defined as active (a border to the Class shape is added). An active Class specifies whether an object of the Class maintains its own thread of control.

A general information about working with shapes is provided in Diagramming(see page 482).

You can specify Class properties in the Class's Specification window. In the same window, you can find the description of each property. Descriptions are presented in the description area at the bottom of the Specification window.

More information

- For more information about the Specification window, take a look at the page Specification window(see page 913).
- For more information about specifying property values, peruse through the page Editing Property Values(see page 921).

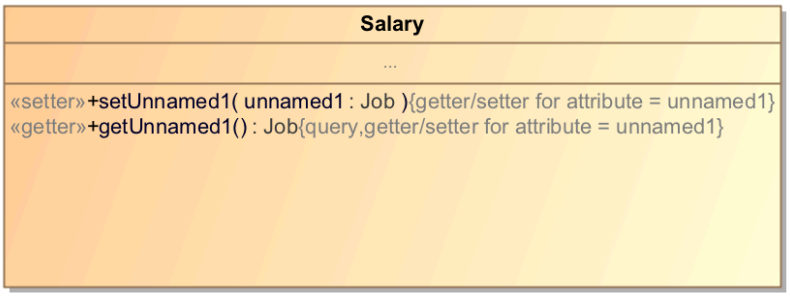
To insert an inner element in the selected class

1. Double-click the selected Class or select Specification from the Class's shortcut menu(see page 667). The Class's Specification window opens.
2. Click the **Inner Elements** tab and then click the **Create** button or press **Insert**.
3. Select the element you wish to add from the list. Click the selected element.
4. The corresponding Specification window opens. Define the Class, Use Case or Interface, and click **O K**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To generate operations for setting or getting private data to the selected class

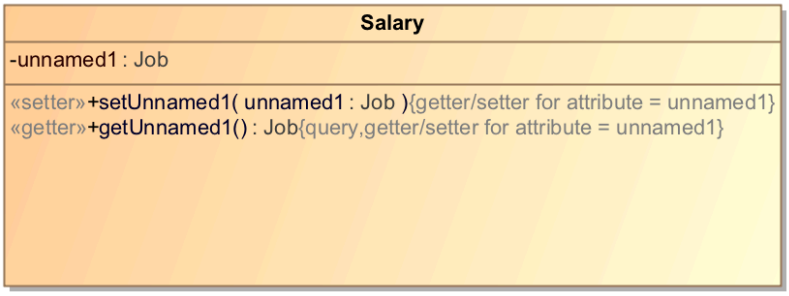
- From the Class's shortcut menu, select **Tools** and then **Create Setters/Getters**.



More Information
For a detailed description, read Creating Setters and Getters(see page 1154).

To control a list of operations and attributes that are visible on a diagram

1. Select **Edit Compartmen**t from the Class's shortcut menu.
2. The **Compartment Edit** dialog opens.



To define a class as abstract and/or active

1. Open the Class's Specification window.

2. In the **General** tab, select the **Is Abstract**, and/or **Is Active** check box to *true*.

To show members (attributes and operations) on the classifier shape according to the visibility

1. On the diagram pane, right-click the Class shape and then from the class's shortcut menu, select the **Symbol Properties**.
2. In the **Symbol Properties** dialog, change the **Show Members** property; the options are All, Only Public, or Not Private.

Related References

Related Pages

- Class diagram(see page 297)
- Class Diagram Wizard(see page 1347)
- Behavior(see page 662)
- Package(see page 800)
- Pattern Wizard(see page 1162)
- Editing Property Values(see page 921)
- Model Elements(see page 628)

Creating a Structured Class

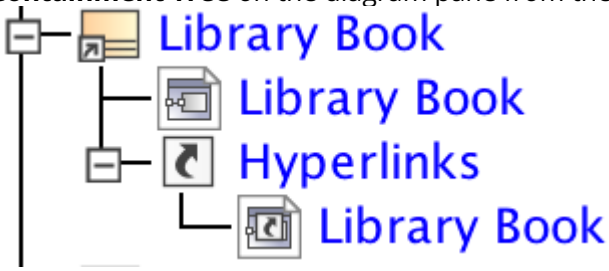
There is a way to create a piece of the model with a single click: Class(see page 666), with a Composite Structure diagram(see page 325) inside it and hyperlink them. It can be useful for architects and system engineers. The same applies to SysML Block and IBD.

To create a structured class

1. Create a Class diagram(see page 297).
2. In the class diagram toolbar, click the arrow next to the Class symbol to reveal a list of available elements. Select the Structured Class element.
3. Click on the Class diagram. The class element with a **hyperlink** to a Composite Structure diagram is created.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

4. Select the Class in the **Browser** to see its structure. To do this, choose **Select in Containment Tree** on the diagram pane from the Class shortcut menu.(see page 669)



In the Browser, you can see the created Class with the Composite Structure diagram inside it. The Class is linked with the diagram. It means that after double-clicking the diagram, or the Browser, the Composite Structure diagram will opened.

Hyperlinks
For more information about hyperlinks, peruse Defining Hyperlinks(see page 1008).

The names of the Class and the Composite Structure diagram are synchronized. Type a name for the Class, for example, *Library Book*, and the name of the diagram is automatically changed to *Library Book* and vice versa. This is synchronization of a diagram name and its context name.

Diagram Name and Context Name
For more information about diagram name and its context synchronization, see Diagram Name and its Context Name Synchronization(see page 523).

Related References

Related Pages

- Class diagram(see page 297)
- Class Diagram Wizard(see page 1347)
- Behavior(see page 662)
- Package(see page 800)
- Model Elements(see page 628)

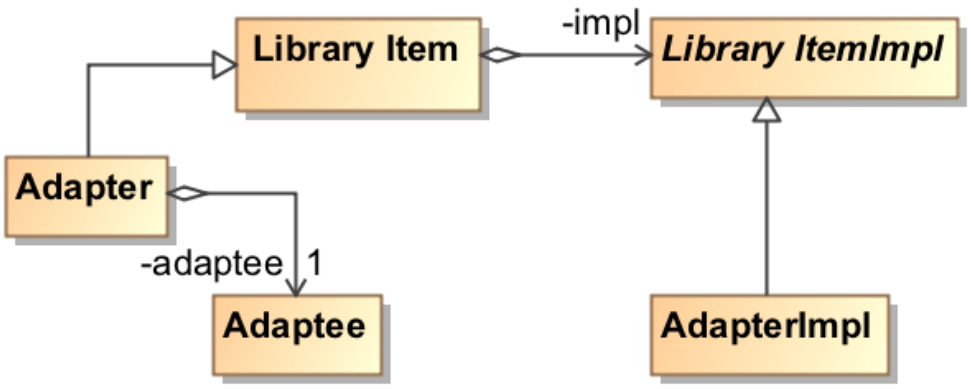
Design Patterns

You can create and edit the design patterns for the selected Class(see page 666). A detailed description of templates can be found in the **Design Patterns** of **Reusable Object-Oriented Software**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To create the design pattern for the selected class

1. From the Class's shortcut menu(see page 670), select **Tools**, and then select **Apply Pattern**.
2. The Pattern Wizard dialog opens. For a detailed description of this dialog, read Pattern(see page 1162) Wiz(see page 1162)ard(see page 1162).
3. Select the design pattern you want to apply and select the desired options. Click **OK**.



Related References

Related Pages

- Class diagram(see page 297)
- Class Diagram Wizard(see page 1347)
- Behavior(see page 662)
- Package(see page 800)
- Pattern Wizard(see page 1162)
- Model Elements(see page 628)

Creating Class Elements

A Class(see page 666) is drawn as a solid-outline rectangle with three compartments separated by horizontal lines. The top name compartment holds the class name and other general properties of the Class (including stereotype(see page 856)); the middle list compartment holds a list of properties; the bottom list compartment holds a list of operations. The property and operation compartments are optional, so you can suppress(see page 570) them.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To create a class element

1. On the left side of your model, find the available options under the Class diagram(see page 297) toolbar and click on Class.
2. You will see an arrow for a drop-down menu which you can click if you wish to create a different kind of Class element.
3. In any case, either
   a. Click on the element and then click anywhere on your diagram pane, or
   b. Drag and drop, the element you clicked on, anywhere on your diagram pane.

   At this point, you should have a Class element in your Class diagram, as shown in figure below.

Related References

Related Pages

- Model Elements(see page 628)

Attribute

On this Page

- New Attribute(see page 673)
- Attribute Name(see page 673)
- Attribute Type(see page 674)
- Additional Information About Attribute Type(see page 675)
- Attribute Visibility(see page 675)
- Attribute Multiplicity(see page 676)
- Attribute Changeability (see page 676)
- Convert Attribute To Association (see page 676)
- Convert Attribute To Port(see page 677)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

An attribute is a named property of a Class(see page 666) that describes a range of values that can be held by the instances of that Class. You can specify attribute properties in the attribute's Specification window. Specifying attribute properties include changing the name, defining the type, converting the attribute to a port, etc. In the same window, you can find the description of each property. Descriptions are presented in the description area of the Specification window at the bottom of the Specification window.

More information

- For more information about the Specification window usage, see Specification window(see page 913).
- For more information about specifying property values, see Editing Property Values(see page 921).

Adding New Attribute

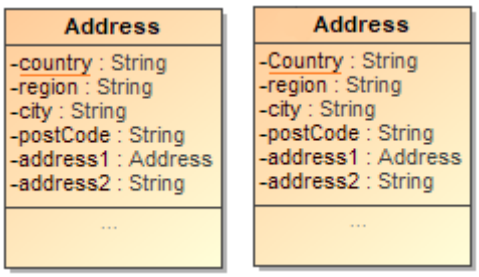
To create a new attribute, do one of the following:

- In the Containment tree, from the class shortcut menu, select **Create Element** > **Property**.
- In the class Specification window > **Attributes**, click the **Create** button.
- On the diagram pane, select the class and press **Ctrl+Alt+A**.
- On the diagram pane, select the class and click **Create Property**.
- On the diagram pane, select the class and click **Create Element** > **Property**.

Attribute Name

The attribute's name must be unique in the Class scope. Nevertheless, do one of the following instructions to change the attribute's name.

Warning
Illegal characters for attribute names are **( ) [ ] { } ! @ # % ; ', ; : ' " ?**

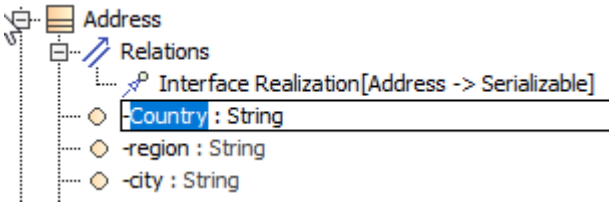


Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example of changing the attribute name from 'country' to 'Country'

To change an attribute name

- Click the Attribute in the selected Class on the diagram pane or in the **Browser** tree. Type a new name.



- Click on the Class block, double-click on the Attribute, and on the **Attribute** Specification window, change the name and click **Close**.

Attribute Type

Attribute Type
The attribute's type can be any of these: Class, Interface, or Primitive Class, such as *int* or *double*. You are allowed to define the attribute's type.

To define the type of an attribute

- Do one of the following:
  - Open the Specification window of an Attribute in your Class block, by right-clicking on the Attribute, and click on the cell of the **Type** property value. Then click the button. In the opened window, select the type. Click **OK**.
  - On the diagram pane, right-click the Attribute, and then point to **Type**. In the opened list, select the type or point to **New** to create a new type.
  - On the Class shape, click the Attribute, and then on the smart manipulator toolbar, click the button. In the opened list, select the desired type.
  - On the Class shape, click the Attribute, and then click the Attribute the second time. After the Attribute name, type a colon ":" and then type in the name of the new attribute type. Click **Enter** or click anywhere on the diagram. The type is now created and assigned.

Colon
Type the colon ":" , press **Ctrl+Space** or **Ctrl+Backspace**, and then in the open list, select the **type**.

- In the Containment tree(see page 693), select an element and drag it to the Attribute on the diagram.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Additional Information About Attribute Type

To add additional information about the type of an attribute

1. Open the **Attribute** Specification window, by right-clicking on the Attribute.
2. In the top right corner of the window, select **Expert** in the **Properties** drop-down menu.
3. In the **Type Modifier** list, select a particular sign in the:
    a. & - one class has a reference to other model elements.
    b. * - one class has a pointer to other model elements.
    c. [] - one class has an array of other model elements.

Attribute Visibility

Visibility Name	Function
Public '+'	An attribute can be accessed by any other elements.
Package '~'	An attribute can be accessed by elements from the same package.
Protected '#'	An attribute can be accessed from the inside of the selected class and classes derived from that class.
Private '-'	An attribute can be accessed only from inside of that class.

> **Visibility**
> The attribute visibility is shown at the attribute signature.

To set the attribute visibility

Do one of the following:

- Type '+', '~', '-', or '#' visibility marks just before an Attribute name directly on a diagram.
- Open the Attribute Specification window. From the **Visibility** drop-down list, select the desired item: public, package, protected, or private.

Attribute Scope

To set an attribute scope

1. Open the Attribute Specification window.
2. In the top right corner of the window, select **Expert** in the **Properties** drop-down menu.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

3. Select the **Is Static** check box to *true*.

Attribute Multiplicity

To set the attribute multiplicity

1. Open the Attribute Specification window.
2. In the top right corner of the window, select **Expert** in the **Properties** drop-down menu.
3. Select or set the multiplicity value for the **Multiplicity** field.

Attribute Changeability

The attribute changeability controls the access by operations on the class on the opposite end.

Name	Function
Is Read Only	When *false* - no restrictions on modifications. When *true* - the value can not be altered after the object is instantiated and its values initialized. No additional values can be added to a set.

To set the attribute changeability

1. Open the Attribute Specification window.
2. In the top right corner of the window, select **Expert** in the **Properties** drop-down menu.
3. Select the **Is Read Only** check box to *true*.

Convert Attribute To Association

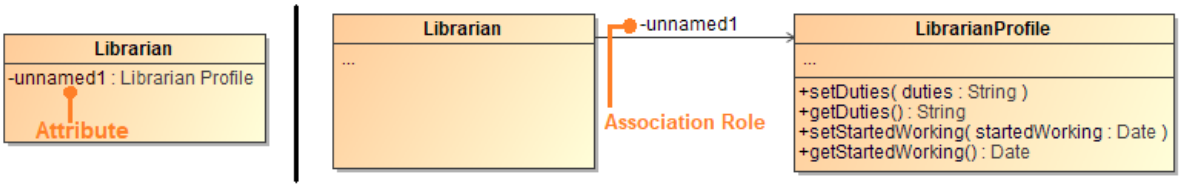
Converting an Attribute
Keep in mind a few things about an attribute when converting it to an association:
- the attribute has to have a **type** specified, and
- the attribute is **not** a member end of some association.

To convert an attribute to an association

1. On a diagram pane, select a Class with an Attribute, or select an Attribute itself.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

2. From the Class's or Attribute's shortcut menu, select the **Refactor** command, then **Convert To**, and **Association**.
3. On the diagram pane, the Attribute is represented as an **association** with a **role**[140].



Logitics
You can move attributes back to the class by dragging and dropping them on the class shape.

Convert Attribute To Port

To convert to a port

---

1. Right-click an Attribute or multiple Attributes.
2. On their shortcut menu, click **Refactor** > **Convert To** > **Port**.

Related Pages

- UML elements(see page 628)
- Class(see page 666)
- Class diagram(see page 297)
- Editing Property Values(see page 921)
- Specification window(see page 913)

Operation

Entries in the operation compartment are strings that show the operations defined on classes as well as those that are supplied by the classes. An operation is a service that can be requested to perform by an instance of the class. It has a name and a list of arguments.

Usually, class attributes are accessed through the operations. The operations are used to perform specific actions, such as system calls, utility functions, and queries. The operation signature provides all information needed to use that operation.

You can specify operation properties(see page 921) in the operation Specification window(see page 913). In the same window, you can find the description of each property. Descriptions are presented in the description area of the Specification window.

---

140 https://docs.nomagic.com/display/CBMP190SP4/Role

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To create a new operation, do one of the following

- On the diagram, select the class, click the smart manipulator(see page 567) > **Operation**.
- In the class Specification window > **Operations** property group, click the **Create** button.
- In the Containment tree, from the class shortcut menu, select **Create Element** > **Operation**.
- On the diagram, select the class and press the Ctrl+Alt+O.

To define the type of an operation, do one of the following

- Select an operation, type ":" and then specify the type. If you add a non-existent type, a new class is created for that type.

  After typing the ":", press Ctrl+Space or Ctrl+Backspace to get a list of suggested types.

- In the Operation Specification window, the **Type** property, select the operation type.

To quickly create a behavior or a diagram for an operation

1. From the operation shortcut menu, select **Create Method** and either **Behavior**, or **Diagram.**
2. Select one of either behaviors or diagrams you wish to create.

To add an operation parameter, do one of the following

- In the Operation Specification window, **Parameters** property group, click the **Create** button.
- On the diagram, select an operation and type a parameter text in parenthesis.
- Select an operation in the Containment tree and for the shortcut menu, select **Create Element** > **Parameter**.

Related Pages

- Model Elements(see page 628)
- Specification Window(see page 913)
- Parameter(see page 678)

Parameter

The **Parameter** Specification window defines an operation argument.

You can specify parameter(see page 921) properties in the parameter Specification window(see page 913), which also contains descriptions.

To add additional information about the return type of an operation

1. In the **Operation** Specification window, click on the **Properties** list, and select **Expert** mode. More properties for the operation are displayed.
2. In the **Type Modifier** property specification cell, select one of the following:
    - & - one class has a reference to another model element.
    - * - one class has a pointer to another model element.
    - [] - one class has an array of other model elements.

To define an operation as abstract, static, or query

---

1. In the **Operation** Specification window, click on the **Properties** list, and select **Expert** mode. More properties for the operation are displayed.
2. In the **General** property group, select one or all of the following
    - **Is Abstract**: the operation does not have an implementation; one must be supplied by a descendant.
    - **Is Static**: the values returned by the parameter have no duplicates.
    - **Is Query**: does not change the state of the system.

To set the operation visibility

- In the **Operation** Specification window, click on the **Visibility** property specification cell, and select one of the following:
    - **Public**: the operation can be accessed by any other object from the outside.
    - **Package**: the operation can be accessed by an element from the same package.
    - **Private**: the operation can be accessed only from that class.
    - **Protected**: the operation can be accessed from the inside of that class and the classes derived from that class.

To set an operation Concurrency

---

1. In the **Operation** Specification window, click on the **Properties** list, and select **Expert** mode. More properties for the operation are displayed.
2. In the Concurrency property specification cell, select one of the following:
    - **Sequential**: The callers must coordinate, so only one call to an Instance (on any sequential Operation) is made at a time. If simultaneous calls occur, the semantics and integrity of the system cannot be guaranteed.
    - **Guarded**: Multiple calls from concurrent threads may occur simultaneously to one Instance (on any guarded Operation), but only one is allowed to commence. The others are blocked until the performance of the first Operation is complete. It is the responsibility of the system designer to ensure that deadlocks do not occur due to simultaneous blocks. The Guarded Operations must perform correctly (or block themselves) if a simultaneous sequential Operation or guarded semantics cannot be claimed.
    - **Concurrent**: Multiple calls from concurrent threads can occur simultaneously to one Instance (on any concurrent Operation). All of them can proceed concurrently with the correct semantics. The Concurrent Operations must perform correctly in case a simultaneous sequential or the guarded Operation, or concurrent semantics cannot be claimed.

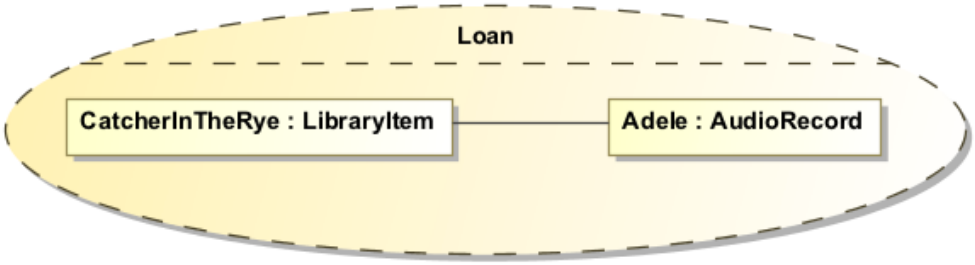
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related Pages

- Model Elements(see page 628)
- Specification Window(see page 913)

Collaboration

A Collaboration describes the structure of elements that implement a certain behavior. In the Collaboration, the elements are the properties connected with the connectors(see page 686).



Example of a collaboration

In this example, you can see the *Loan* Collaboration. The Collaboration contains two properties - *CatcherInTheRye* and *Adele*. These are the properties (attributes) and the *LibraryItem* and *AudioRecord* are the types of each property. The *CatcherInTheRye* and *Adele* are connected with the connector relationship.

More Information
For more information on how to assign a behavior to a Collaboration Use, read Assigning Behavior diagrams automatically(see page 665).

To create the property and connector in the collaboration

1. Create a Collaboration symbol Collaboration in a Class diagram(see page 297), you may name it *Loan*, for example.
2. On the Class diagram, select the Collaboration, right-click on it to open its shortcut menu(see page 2106). Select the **Symbol Properties** command.
3. In the **Symbol Properties** dialog, click to clear the **Suppress Structure** check box. Click **Close**.
4. On the diagram pane, or in the **Model Browser**, select the element and drag-and-drop it to the Collaboration symbol on the diagram pane.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

5. The new property is created. The type of the property is the element that was dragged and dropped.
6. To create a connector between properties, expand the Composite Structure diagram(see page 680) pallet and click the button.
7. Draw the connector relationship between properties.
8. Now you have the structure represented in the Collaboration.

Collaboration properties

You can format collaboration symbol properties in the **Symbol Properties** dialog.

> **More Information**
> For more information about symbol representation properties, see Formatting symbols(see page 975).

You can specify collaboration properties in the collaboration Specification window. In the same window, you can find the description of each property. Descriptions are presented in the description area of the Specification window.

> **More Information**
> For more information about the Specification window usage, see Specification window(see page 913).
> For more information about specifying property values, see Editing property values(see page 921).

Related Diagrams

- Class Diagram(see page 666)
- Composite Structure Diagram(see page 325)
- Activity(see page 311)
- Sequence(see page 304)
- State Machine(see page 308)
- Protocol State Machine(see page 310)
- Model Elements(see page 628)

Related Procedures

- Formatting Symbols(see page 975)
- Customizing Environment Options(see page 65)

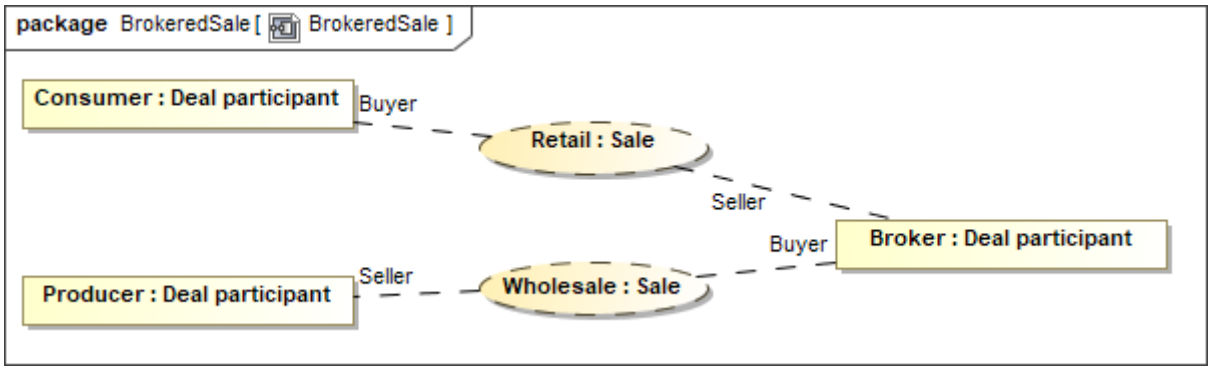
Related Refrences

- Specification Window(see page 913)
- Editing property values(see page 921)

Collaboration Use

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

A Collaboration Use represents the implementation of a pattern described by a Collaboration(see page 680) in a specific situation. In this context, the role binding(see page 699) means that the roles in the Collaboration are bound by concrete elements. A Collaboration can be used many times in different Collaboration Uses.



Example of multiple Collaboration Uses along with Parts

Model Tip
These instructions will work best if you have a Composite Structure diagram(see page 325) created already.

To assign a collaboration to a collaboration use

Before assigning a Collaboration to a Collaboration Use, you **must** have a Collaboration created in your model.

Collaboration
Read Collaboration(see page 680) on how to create a Collaboration in your model.

After creating a Collaboration, then do one of the following:

- On the Composite Structure diagram palette, click the Collaboration Use button Collaboration Use , and then in the opened dialog, select an existing Collaboration or click Creation Mode button Creation Mode to create a new Collaboration. Click **OK** when you are done.
- On the Composite Structure diagram pane, create a Collaboration Use, then click the Collaboration Use shape, and then on the smart manipulator toolbar, click the :T symbol. Select a Collaboration. Press **Enter** or click anywhere on the diagram pane. The Collaboration is now created and assigned.
- On the Composite Structure diagram pane, create a Collaboration Use, then click the Collaboration Use shape, and then press **Ctrl+T**. Type the name of the new Collaboration and press **Enter**, or click anywhere on the diagram pane. The Collaboration is now created and assigned.
- On the Composite Structure diagram pane, create a Collaboration Use, then click the Collaboration Use shape, after that click the name area. Type the **":"** colon and then type the

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Collaboration name. Press **Enter**, or click anywhere on the diagram pane. The Collaboration is now created and assigned.
- In the Containment tree, select the Collaboration and drag and drop it on the diagram pane. A Collaboration Use with the assigned Collaboration is created.
- On the Composite Structure diagram pane, open the Specification window of the Collaboration Use and click the cell of the **Type** property value. In the opened dialog, click ···, and select the Collaboration. Click **OK** when you are done.

Collaboration use properties

You can format Collaboration Use symbol properties in the **Symbol Properties** dialog.

> **Formatting Symbols**
> For more information about symbol representation properties, see Formatting symbols(see page 975).

You can specify Collaboration Use properties in the Collaboration Use's **Specification window**. In the same window, you can find the description of each property. Descriptions are presented in the description area of the Specification window.

> **Specification Window and Property Values**
> For more information about the Specification window usage, see Specification window(see page 913).
> For more information about specifying property values, see Editing property values(see page 921).

Related pages

- Creating a relationship(see page 558)

Related Diagrams

- Composite Structure Diagram(see page 325)
- Activity(see page 311)
- Sequence(see page 304)
- State Machine(see page 308)
- Protocol State Machine(see page 310)
- Collaboration(see page 680)
- Model Elements(see page 628)

Comment

A **Comment** is a UML element used to specify various remarks of an element in the model. In the diagram palette, under **Common**, the Comment command appears under the Note command. You can display text in the Comment as plain or HTML text.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

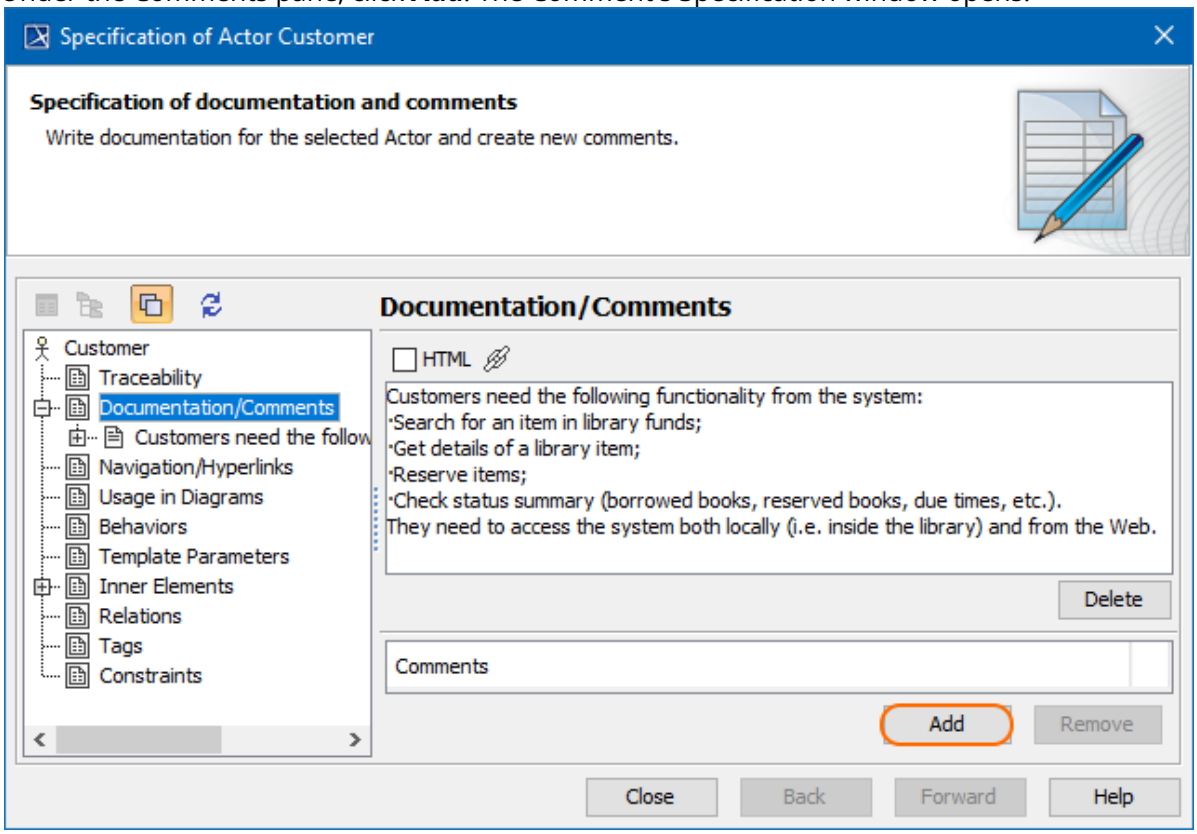
Example of a comment.

You can format the Comment's symbol properties in the **Symbol Properties** dialog(see page 975).

You can specify a Comment in the Comment's Specification window(see page 913), which also displays the description of each Comment property in the description area of the Specification window.

To create a comment in the element Specification window

1. Open the Specification window of the selected element.
2. In the panel on the left, select the **Documentation/Comments** property group.
3. Under the Comments pane, click **Add**. The Comment's Specification window opens.



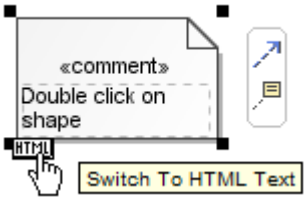
4. Type the Comment content as the **Body** property value and click **OK**.
5. Close the Specification window.

> To view the **Documentation/Comments** symbol on the diagram pane, expand the **Documentation/Comments** property group and drag the selected documentation/comment on the diagram pane.

To switch a comment to HTML or Plain Text

Do one of the following:

- On the diagram pane, select the Comment symbol. Click the button that appears on the lower-left corner of the shape:
  - The **HTML** button - to switch the text to the HTML text.
    The **Plain** button - to switch the text to the Plain text.



HTML text
You can edit the HTML text in the comment using the HTML editor(see page 1055).

To display an anchored element documentation in a comment

1. Anchor(see page 648) a Comment to a symbol on the diagram pane.
2. Select the Comment, on the diagram pane. Open its shortcut menu.
3. Select the **Retrieve Documentation** command.
   The anchored symbol's documentation is displayed on the Comment's symbol.

Retrieve Documentation
You can enable the **Retrieve Documentation** option **only** if you already have documentation in the Comment's Specification window. Read the first set of instructions on this page to see how to create the comment in the element's Specification window.

Edit
You can edit element documentation directly on the Comment symbol.

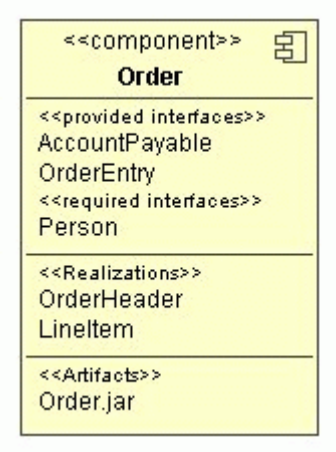
Component

A Component represents all kinds of elements that make up the system. A Component can always be considered as an autonomous unit within a system or subsystem. It is a module having the following features:

- A Component is a replaceable and independent part of the system performing a specific action.
- A Component acts in a context of a well-defined architecture.
- Components interact among each other using Interfaces(see page 741).

Components provides compartments for listing its provided and required Interfaces, realizations, and artifacts.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



A Component can be specified by changing its property values in the Component Specification window(see page 913). Each property is described in the description area on this window.

To show/hide the interfaces, realizations, and artifacts on the Component's shape, do one of the following

- On the diagram, right-click the Component's shape and then from the shortcut menu, select **Symbol Properties**. In the **Symbol Properties** dialog(see page 975), set the **Suppress Interfaces**, **Suppress Realizations**, or **Suppress Artifacts** property values to *true* or *false*.
- On the diagram, click the Component's shape and then click the Compartments button ⊡. Then in the menu, click to select or clear the **Interfaces**, **Realizations**, or **Artifacts** check box.

Related Pages

- Model Elements(see page 628)
- Class diagrams(see page 297)
- Use Case diagram(see page 300)
- Sequence diagram(see page 304)
- Activity diagram(see page 311)
- Component diagram(see page 318)

Component realization

A Component Realization concept is specified in the Components package to (optionally) define the Classifier(see page 461). The Classifier defines the contract offered by a component in terms of its provided and required interfaces. The component forms an abstraction from these various Classifiers.

Connector

A Connector specifies a link that enables communication between two or more instances. This link can be an instance of an association, or it can represent the possibility of the instances being able to

communicate because their identities are known by virtue of being passed on as parameters, held in variables or slots, or because the communicating instances are the same instance.

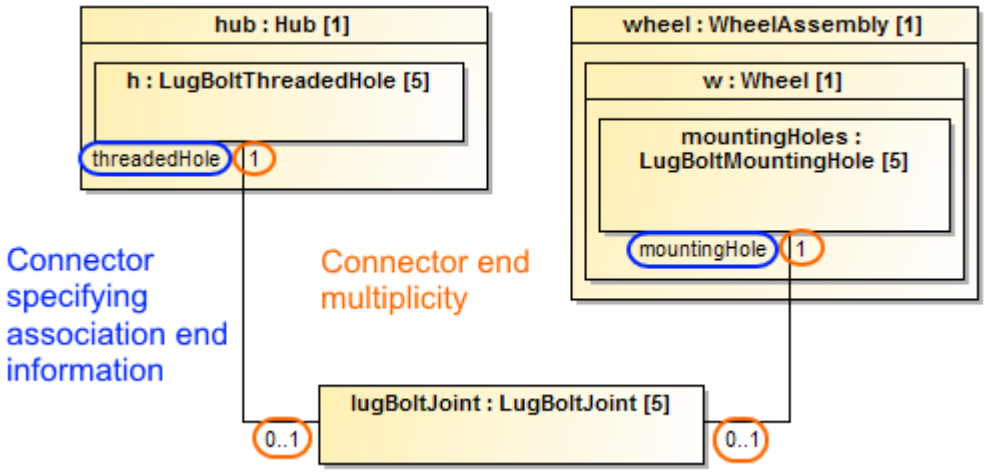
The link can be realized by something as simple as a pointer or by something as complex as a network connection. In contrast to the associations, which specify the links between any instance of the associated classifiers, the connectors specify the links between instances playing the connected parts only.

Each connector can be attached to two or more connectable elements, each representing a set of instances.

You can specify connector and connector end properties in the connector and connector end Specification window. In the same window, you can find the description of each property. Descriptions are presented in the description area of the Specification window(see page 913).

Drawing a connector end

A Connector has at least two ends, each representing the participation of instances of the classifiers typing the connectable elements attached to the end. The set of Connector ends is ordered. The connector end information can be displayed on the diagram.



Selecting a Port automatically when drawing a Connector

To select a Port automatically when drawing a Connector from a Port to a Part

1. Select a Port on the diagram pane.
2. On the smart manipulator toolbar, click and select Part shape. The **Select Port** menu appears.

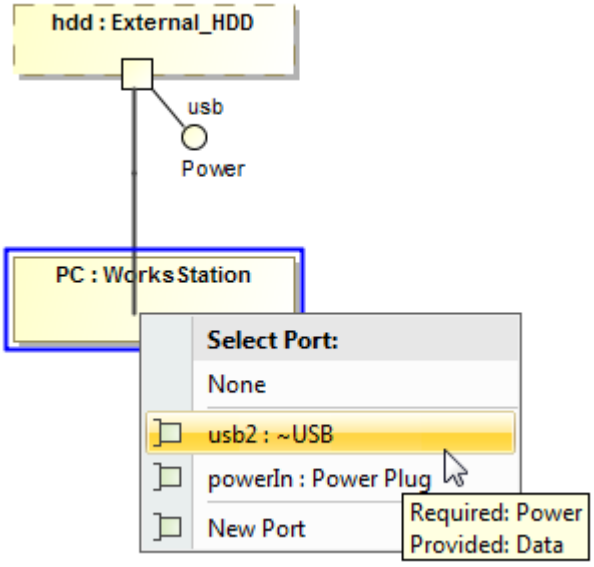
   The menu appears if a new Port can be created on the Part or the Part has hidden Ports.

3. Select one of the following command:

- **None** - if you want to connect connector straight to the part.

- On a port in the list. Hidden ports of the part are listed. The selected port will be represented on the diagram and the connector will be connected to it.
- **New Port** - to create a new port with the same name, type and multiplicity as the port from which the connector is drawn.
- **New Nested Port** - to create a new nested port. Note that the command exists only if the connector is drawn from the nested port.

You can draw a connector to a Port too and the menu appears. As a result, the nested Port is created.



Hiding Connector type

You can hide the type name of the represented Connector on the diagram pane.

To hide the type name of Connector

1. Select a single or any number of Connectors on the diagram pane.
2. Right-click to open the shortcut menu.
3. From the shortcut menu, select Symbol Properties.
4. In the **Symbol Properties** dialog, set the **Show Type** option value to *false.*
   The name of the Connector type is hidden and no longer visible in the Connector label.

On this page

- Drawing a connector end(see page 687)
- Selecting a Port automatically when drawing a Connector(see page 687)
- Hiding Connector type(see page 688)

Related pages

- Editing property values(see page 921)
- Setting an Information Flow(see page 732)
- Reversing Information Flow direction(see page 946)

- Model Elements(see page 628)

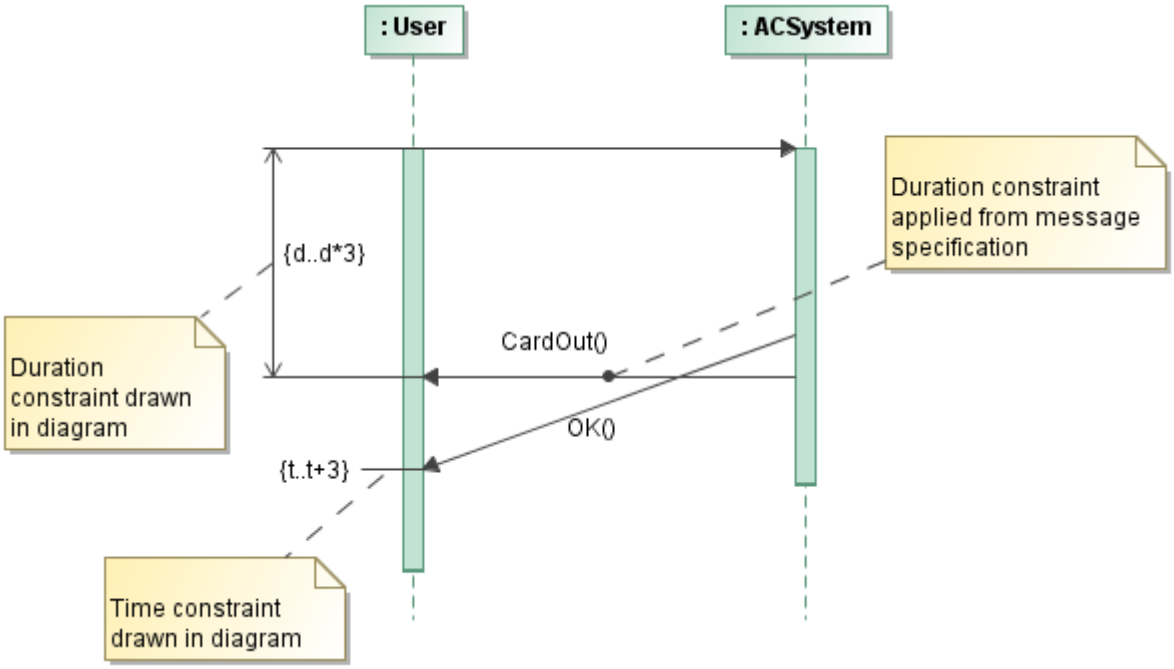
Constraint

A Constraint represents additional semantic information attached to the constrained elements. It is an assertion that indicates a restriction that must be satisfied by a correct design of the system. The constrained elements are those elements required to evaluate the constraint specification. In addition, the context of the Constraint can be accessed, and can be used as the namespace for interpreting names used in the specification.

A Constraint is represented as a linguistic, enclosed in braces ({}), statement in some formal (OCL, C++, and other), or a natural language. There are 14 standard constraints in UML such as association, global, and parameter. You can also define your own constraints.

You can create the following constraints:

- Time Constraint. It specifies the combination of min and max timing interval values.
- Duration Constraint. It defines a value specification that indicates the temporal distance between two time instants.
- Constraint. It represents an additional semantic information attached to the constrained elements.
- Constraint Link(see page 691). It is a graphical representation of a constraint element.



Constraints for validation
Constraints can also be used to define validation rules(see page 1306) for your model.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related pages

Related Pages

- Model Elements(see page 628)
- Class diagrams(see page 297)
- Use Case(see page 876)
- Use Case diagram(see page 300)
- Sequence diagram(see page 304)
- Activity diagram(see page 311)

Working with Constraints

You can edit constraints in the **Constraints** property group of the containing element Specification window, in the constraint's Specification window, or double click the selected constraint and edit it on the diagram pane directly.

To see the constraint expression, make sure the **Show Constraint** value is set to true, in the **Symbol Properties** dialog(see page 975).

In this section, see the following procedures:

- Defining a new constraint(see page 690)
- Editing a duration interval of the constraint(see page 691)
- Applying an existing constraint(see page 691)

Defining a new constraint

To define a new constraint, do one of the following

- In the Containment Tree, from the element shortcut menu, select **Create Element** > **Constraint**.
- In the Specification window(see page 913), select the **Inner Elements** property group and click the **Create** button (select Constraint from the list, if needed). Specify the constraint in the constraint Specification window(see page 913).
- In the Specification window(see page 913), select the **Constraints** property group and click the **Create** button. In the menu, select a duration, interaction, time, or simple constraint. The constraint Specification window(see page 913) is displayed.
- In the diagram palette, under the Common category, expand the **Constraint** command and select the Duration Constraint, Time Constraint, or Constraint button and create it directly on the action shape.

If a constraint is displayed in the **Constraints** property group of the element Specification window(see page 913), it means this constraint is applied for the element. If it is displayed in the **Inner Elements** list, this constraint is only owned by the element.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Editing a duration interval of the constraint

To edit the duration interval of the constraint

1. Right-click the element shape and select Specification.
2. In the Specification window, select the **Constraints** property group.
3. Edit the duration interval on the left side of the Specification window in the Specification column.

Applying an existing constraint

To apply a constraint to an element

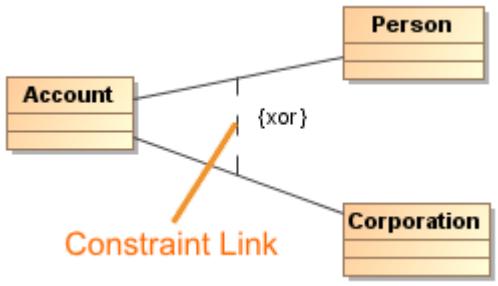
1. In the chosen element's Specification window(see page 913), select the **Constraints** property group and click the **Apply** button. The **Select Constraint** dialog opens.
2. Select a constraint or create a new one.
3. Click **OK** when you are done.

Related Pages

- Model Elements(see page 628)
- Class diagrams(see page 297)
- Use Case(see page 876)
- Use Case diagram(see page 300)
- Sequence diagram(see page 304)
- Activity diagram(see page 311)
- Applying duration constraint on the Action(see page 641)

Constraint Link

A Constraint Link is a graphical representation of a constraint element. The Constraint Link is shown on the diagram pane as a dashed arrow from one shape to another. You can assign a particular constraint to the Constraint Link. The constraint assigned to the Constraint Link is labeled in braces: {} . See an example of the constraint representation in the following figure.



On the Constraint Link, you can display the constraint direction arrow. The direction of the arrow represents a relevant information within the constraint. The client (the tail of the arrow) is mapped to the first position and the supplier (arrowhead) is mapped to the second position in the constraint.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

You can format (see page 975)the constraint symbol properties in the Symbol Properties dialog(see page 975).

After applying the constraint to the Constraint Link, you can specify (see page 921)the constraint in the Constraint specification window. In the same window, you can find the description of all constraint properties. Descriptions are presented in the description area of the Specification window(see page 913).

To draw a Constraint Link

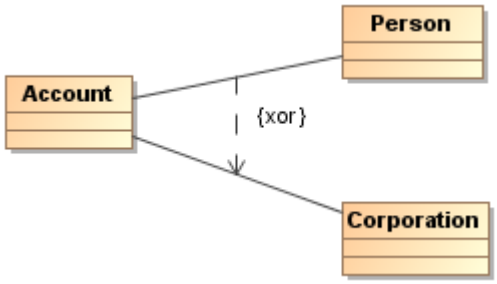
1. In the Class, Use Case, Sequence, or Activity diagram palette(see page 24), under the Common category, expand the **Constraint** command and select the Constraint Link.
2. Draw a Constraint Link between desired shapes.

To add a constraint expression to the Constraint Link

- From the Constraint Link shortcut menu, choose the **Select Constraint** and do one of the following:
  - Select **Create Constraint** to create a new constraint.
  - Choose a constraint from the list.

To display a Constraint Link direction arrow

1. Select the Constraint Link and from the shortcut menu, select **Show Arrow**.
   The direction of the Constraint Link will be displayed. See an example in the following figure.

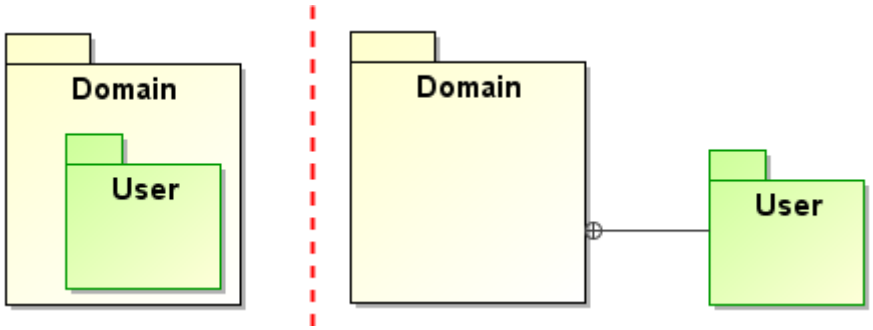


Related Pages

- Model Elements(see page 628)
- Class diagrams(see page 297)
- Use Case(see page 876)
- Use Case diagram(see page 300)
- Sequence diagram(see page 304)
- Activity diagram(see page 311)
- Applying duration constraint on the Action(see page 641)

Containment

The Containment relationship represents the containment of elements on the diagram pane. A Containment path replaces a Nesting representation(see page 692). For example, instead of drawing one package nested into another, you can draw a containment relationship from one package to another (see the figure below). This example shows that the Containment relationship is a graphical representation as well as a nesting of elements.



Example of package nesting (on the left) and package nesting using containment relationship (on the right)

In the example above, the Containment end with a circle plus symbol shows a containing element. The *User* package is a contained element and the *Domain* package is a containing element: the owner of the *User* package is the *Domain* package.

You can format containment symbol properties in the **Symbol Properties** dialog(see page 975).

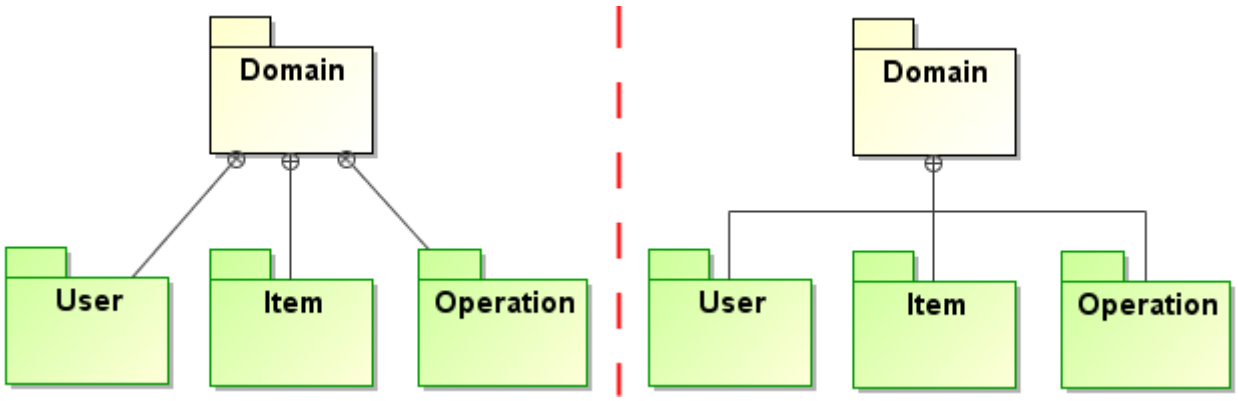
Related pages

- Generating reports from the Containment tree(see page 1590)
- Customizing Environment Options(see page 65)
- Class Diagram(see page 666)
- Composite Structure Diagram(see page 325)
- Use Case Diagram(see page 300)
- Component Diagram(see page 318)
- Object Diagram(see page 299)
- Package Diagram(see page 321)
- Model Elements[141]

Containment Tree

You can create the Containment tree when drawing more than one Containment relationship.

141 https://docs.nomagic.com/display/MD190SP4/Model+elements

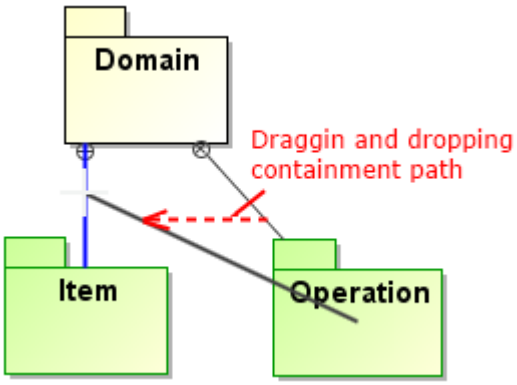


Example of several containment relationships (on the left) and example of containment tree (on the right)

To group containment paths into a tree

1. Draw a Containment path from each contained element to the containing element.
2. Now, do one of the following:

- Through the diagram pane.
  a. Select the Containment relationship.
  b. Click and hold the little square (the end of the Containment path) next to the Containment end with the circle plus symbol.
  c. Drag and drop the selected Containment end on the other Containment path.
  d. While dragging, the Containment path turns blue, indicating an allowable connection. Containment paths merge together.

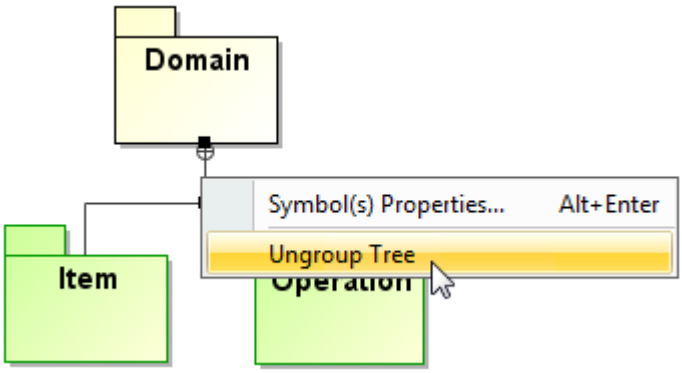


- Through the shortcut menu.
  a. Open the shortcut menu(see page 693) of the containing element, in the example above: *Domain*.
  b. Click **Refactor** > **Make Sub Tree**.

To ungroup a containment tree

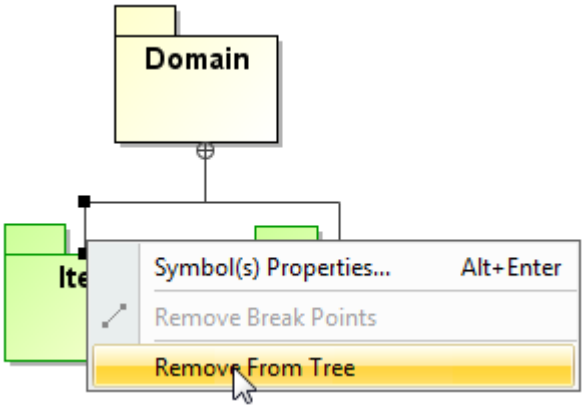
1. Right-click the Containment tree's end with the circle plus symbol pointing to a containing element.

2. In the opened shortcut menu, click the **Ungroup Tree** command.



To separate the containment path from the containment tree

- Select the containment path you want to remove from the tree. Open its shortcut menu and click the **Remove From Tree** command. The contained element is added to the containing element separately.



- Drag one Containment path to another element:
  a. On the diagram pane, select the vertical part of the particular Containment path.
  b. Click and hold the upper little square.
  c. Drag and drop end of the Containment path to the other containing element.
  d. The Containment path is removed from the Containment tree and the owner of the containing element has been changed.

Related pages

- Generating reports from the Containment tree(see page 1590)
- Customizing Environment Options(see page 65)
- Containment(see page 692)

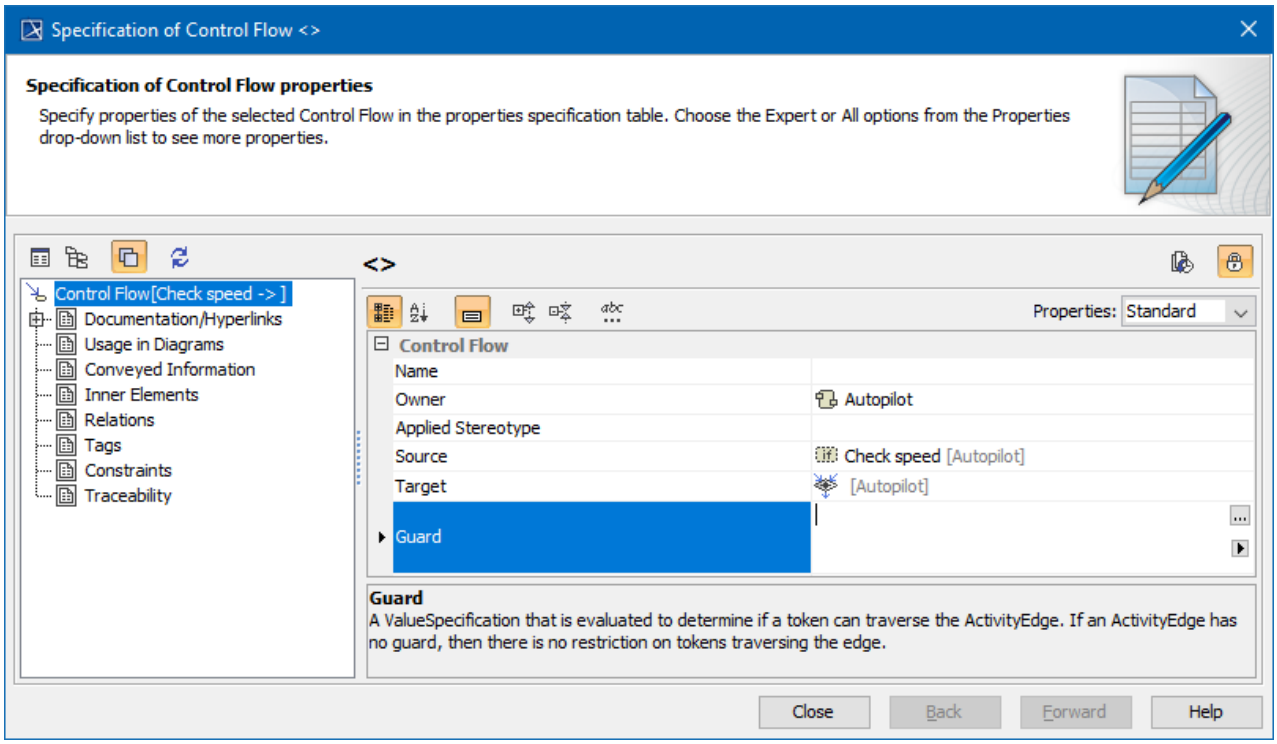
Control Flow

A Control flow is an edge that starts an Action(see page 629) node after the previous one is finished.

You can specify a Control flow by changing its property values in the Control flow Specification window(see page 913). Each property is described in the description area on this window.

You can format the Control flow symbol properties in the Symbol Properties dialog(see page 975).

A Control flow can have guards (constraints that provide fine-grained control over the firing of the Control Flow). To specify guards, choose the Guard box in the Control Flow dialog box.



Specification of the Control Flow, Guard property

Set the Control Flow symbol properties in the Symbol Properties dialog(see page 695).

Data Type

A Data Type is a type whose values have no identity; that is, they are pure values. It is a classifier and inherits the general features of the classifier: visibility, generalizable element properties, and operations.

MagicDraw provides the following predefined data types: boolean, byte, char, date, double, float, int, Integer, Real, long, short, void, and String.

You can also create Enumeration (see page 697)or Primitive (see page 698)Data Types.

To create a new Data Type, including Enumeration or Primitive

- From the Browser(see page 29), select **New** from the Class(see page 666) or **Create Element** from the Package(see page 800), Subsystem(see page 860), or Model(see page 780) shortcut menu, and then select Data Type, Enumeration, or Primitive.
- In the Class, Package, Subsystem, and Model Specification widows(see page 913), the **Inner Elements** property group, click **Create** and select a Data Type.

A Data Type can be specified by changing its property values in the Data Type Specification window(see page 913). Each property is described in the description area on this window.

Related pages

Related Pages

- Model Elements(see page 628)
- Class diagrams(see page 297)
- Use Case(see page 876)
- Use Case diagram(see page 300)
- Sequence diagram(see page 304)
- Activity diagram(see page 311)

Enumeration

Enumeration defines a kind of data type whose range is a list of predefined values called Enumeration Literals. An Enumeration can contain Operations, but they must be pure functions (this is the rule for all data type elements).

An Enumeration can be specified by changing its property values in the Enumeration Specification window(see page 913). Each property is described in the description area on this window.

Enumeration Literals

Enumeration Literal defines an element of the run-time extension of an Enumeration data type. It has no relevant substructure, therefore, it is atomic.

To create an Enumeration Literal, do one of the following

- Select the Enumeration shape, click the Create Element button ⊕, and then select **Enumeration Literal**.
- In the Containment tree, right-click the Enumeration and select **Create Element** > **Enumeration Literal**.

Enumeration Attributes

To create an Attribute for the Enumeration, do one of the following

- Select the Enumeration shape, click the Create Element button ⊕, and then select **Property**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- In the Containment tree, right-click the Enumeration and select **Create Element** > **Property**.

As you can see, the Attribute is by default read-only.

Related Pages

- Model Elements(see page 628)
- Class diagrams(see page 297)
- Use Case(see page 876)
- Use Case diagram(see page 300)
- Sequence diagram(see page 304)
- Activity diagram(see page 311)
- Data Type(see page 696)

Primitive

A Primitive defines a predefined Data Type(see page 696) without possessing any relevant UML substructure; that is, it has no UML parts. A Primitive data type can have an algebra as well as operations defined outside of UML (for example, mathematically). The Primitive data types used in UML include Integer, Unlimited Integer, Real, and String.

Related Pages

- Model Elements(see page 628)
- Class diagrams(see page 297)
- Use Case(see page 876)
- Use Case diagram(see page 300)
- Sequence diagram(see page 304)
- Activity diagram(see page 311)
- Data Type(see page 696)

Decision and Merge

A Decision Node in an Activity diagram(see page 311) is used much like a choice or junction(see page 846) point in the State diagrams(see page 308). Decision nodes allow you to separate the Activity Edges. Merge nodes allow you to combine the Activity Edges together.

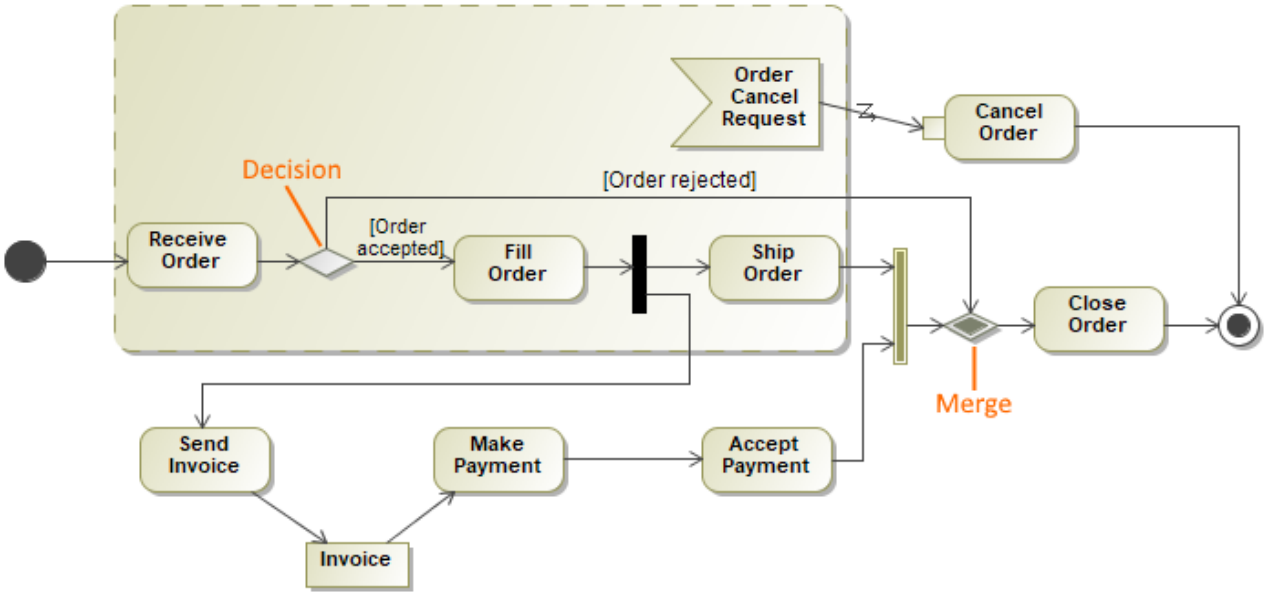
Decision Nodes are created using guard conditions. They help protect Activity Edges that depend on guard conditions.

1. The symbol used for the Decision Node is a large diamond shape. It can have one or two incoming Activity Edges, and at least one outgoing Activity Edge

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

2. The symbol used for the Merge Node is a large diamond shape. It can have exactly one outgoing Activity Edge but may have multiple incoming Activity Edges.

The Decision and Merge are represented by the same symbol. Although, you can set the **Use different Fork/Join and Decision/Merge notations** project option(see page 2221) to draw different notations for the **Decision** and **Merge**.



Activity diagram illustrates that when an order cancellation request is made (only while filling, receiving or shipping orders) - the Cancel Order behavior is invoked.

Related pages

- Activity diagram(see page 311)
- Interaction Overview diagram(see page 332)
- Activity diagram(see page 311)
- Junction(see page 846)
- Model Elements[142]

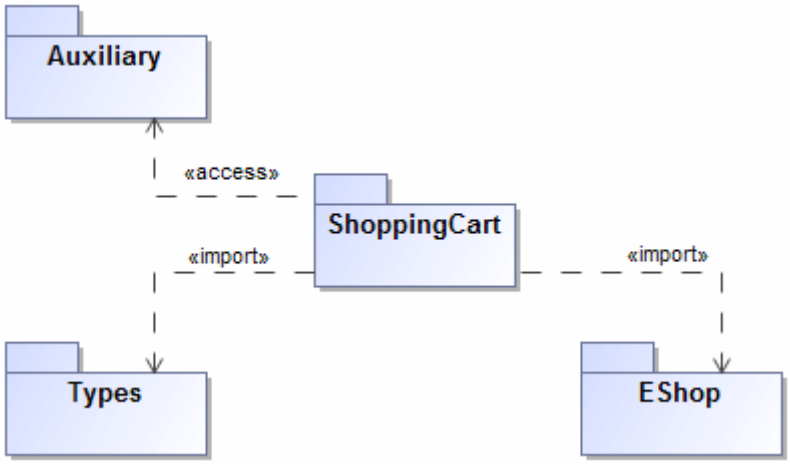
Dependency

A dependency is a relationship signifying that a model element[143] requires other model elements for its specification or implementation. The client element is either semantically or structurally dependent on the definition of the supplier element(s).

A dependency is shown as a dashed arrow between classes or packages. The client element (at the tail of the arrow) depends on the supplier element (at the arrowhead). You can label the arrow with an optional stereotype and an individual name.

142 https://docs.nomagic.com/display/MD190SP4/Model+elements
143 https://docs.nomagic.com/display/MD190SP4/Model+elements

You can also draw a dependency between a Class(see page 666) and other Class elements, such as attributes(see page 672) and operations(see page 677).



Example of dependency relationships: «access» and «import».

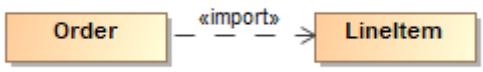
Dependency, abstraction(see page 1211), and usage relationships(see page 1192) are defined in the dialog of the same structure. They differ from one another only by the corresponding Specification name. You can specify a dependency by changing its property values in the dependency Specification window(see page 913). Each property is described in the description area on this window.

Editing property values
For more information about specifying property values, see Editing property values(see page 921).

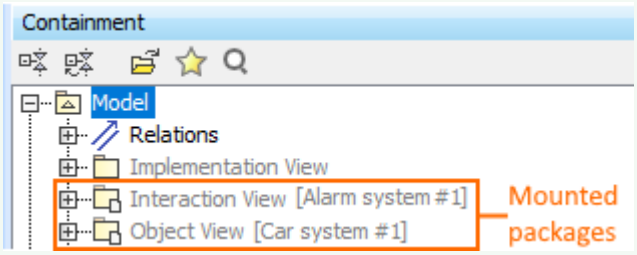
There are several kinds of dependencies:

Dependency kind	Description
Template binding	Represents a relationship between a templatable element and a template. A template binding specifies the substitutions of actual parameters for the formal parameters of the template. You can specify template binding dependency properties and find each property's description in the template binding dependency Specification window. Descriptions are presented in the description area of the Specification window.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Dependency kind	Description
Abstraction	Relates two elements or sets of elements that represent the same concept at different levels of abstraction or from different viewpoints. In the metamodel, an abstraction is a dependency in which there is a mapping between the supplier and the client. Define an abstraction relationship in the Abstraction Specification window.
Usage	A relationship in which one element requires another element (or set of elements) for its full implementation or operation(see page 677). In the metamodel, a usage is a dependency in which the client requires the presence of the supplier. Define a usage relationship in the Usage Specification window.
Package merge	A directed relationship between two packages(see page 800), indicating that the contents of the two packages are to be combined. It has a dependency relationship with the applied stereotype *«merge»*. Define a merge relationship in the Dependency Specification window.
Element import	A directed relationship between an importing namespace and a packageable element. The name of the packageable element or its alias is added to the namespace of the importing namespace. It has a dependency relationship with the applied stereotype *«import»*. Define an import relationship in the Dependency Specification window. To draw the Element Import link, select the Element Import path to draw in the Class diagram toolbar from the Abstraction group. 
Access	Shows that elements can only be accessed from a package(see page 800), and it cannot be referenced.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Dependency kind	Description
Deployment	The allocation of a deployment target to an artifact or artifact instance.
Role Binding	A relationship in a Composite Structure diagram(see page 325). Role Binding is used to connect Parts(see page 804) with Collaboration Use(see page 681).
Mount	A mount dependency represents a former location of a shared package; that is, a location where the package was mounted in a local project or a Teamwork Server project. A mount dependency is created: • After you export a project from Teamwork Server to Teamwork Cloud; or • When a local project is added to Teamwork Cloud. Mounted packages in the Containment tree are adorned with an icon and have their project usage name in brackets, as shown in the following figure:  The purpose of establishing the mount dependency is to ensure the same containment structure during project transformation to Teamwork Cloud, to maintain the correct scope in utilities like smart packages, validation rules, tables etc.

Related pages

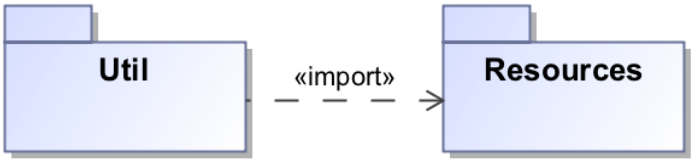
- Working with dependencies(see page 702)
- Package import(see page 704)
- Abstraction(see page 707)

Working with dependencies

Element Import

To draw the Element Import dependency

1. Open or create the Package diagram(see page 321).
2. In the diagram, select or create at least two packages. When using created packages, simply drag and drop the packages onto your diagram pane and name them as you wish.
3. On the Package diagram palette, expand the **Package Merge** button Package Merge and select the **Element Import** button Element Import.

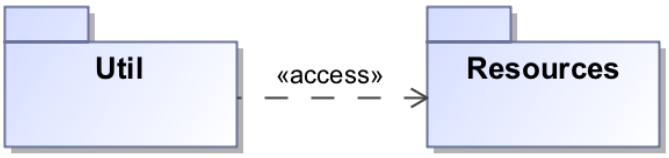


Access

An Access relationship shows that elements can only be accessed from a package, and it cannot be referenced.

To draw the Access dependency

1. Open or create the Package diagram.
2. On the Package diagram palette, expand the **Package Merge** button Package Merge and select the **Package Import** button Package Import.
3. Draw the dependency on your diagram pane between two packages.
4. Select the **Package Import** dependency, open its Specification window(see page 913), and set the **Visibility** property to *private*.



Deployment

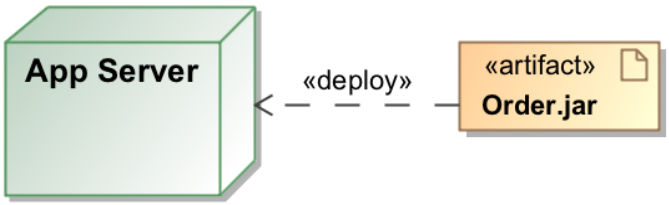
To draw a deployment relationship

1. Create a Deployment diagram(see page 319).
2. On the Deployment diagram pane, create a Node(see page 782) and an Artifact[144] by dragging and dropping each one onto the diagram pane.

144 https://docs.nomagic.com/display/CBMP190SP4/Artifacts

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

3. Click the **Deployment** button Deployment and draw a deployment link from the node to the artifact.



To display the deployed artifacts on the node instance shape

Do one of the following:

- On the diagram pane, select the node instance symbol, then click the **Compartments** button (usually it is displayed on the top left corner of the symbol). In the opened menu, select the **Deployed Elements**.
- On the diagram pane, right-click the node instance and from the shortcut menu, select **Symbol Properties**. Then in the **Symbol Properties** dialog, set the **Suppress Deployed Elements** property value to *false*.

On this page

- Element Import(see page 702)
- Access(see page 703)
- Deployment(see page 703)

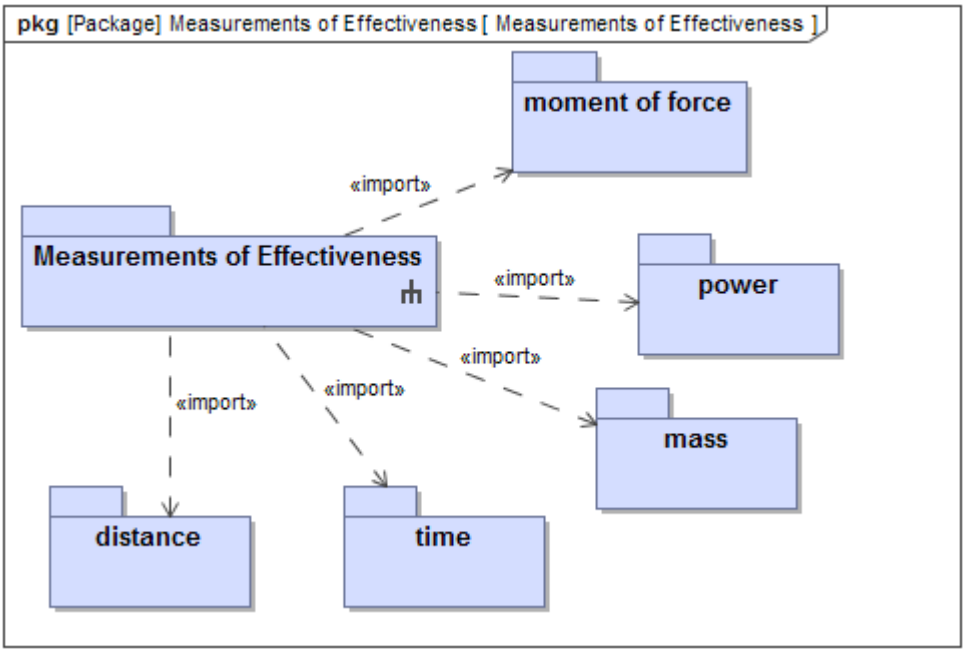
Package import

In UML, package import is defined as a directed relationship between an importing namespace and a package whose members are imported into that namespace. The visibility of a package import is either public or private. The predefined keywords are «import» for a public package import, and «access» for a private package import.

A package import is displayed as a dashed arrow with an open arrowhead aiming from the importing namespace towards the imported package.

Package import applies to an entire package, and all the model elements of the target package are imported into the source namespace. The source package can access subpackages and their imported members of the target package to an infinite depth. The elements of an imported package are also accessible to the subpackages of the importing package.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

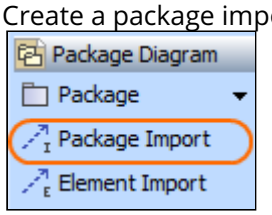


An example of the package import.

Restricting type selection list to imported elements

To restrict the type selection list to imported elements

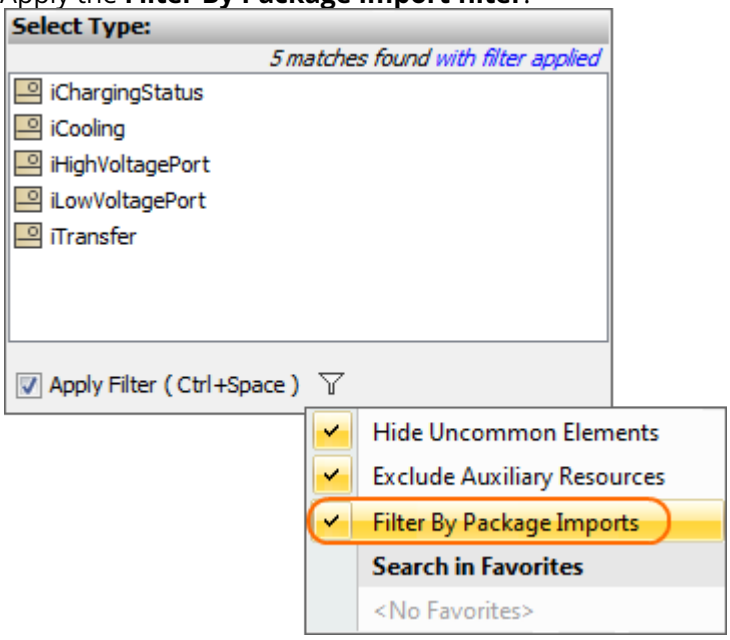
1. Create a package diagram.
2. Drag the package with typed elements, as well as packages with types, on the diagram pane.
3. Create a package import between them by clicking **Package Import** in the diagram palette.



To apply the filter

1. Click a typed element to open the **Select Type** dialog.

2. Apply the **Filter By Package Import filter**.



As a result, the type selection list is restricted to imported elements only.

> ⓘ There are more useful filters(see page 1035) you can use for the same purpose.

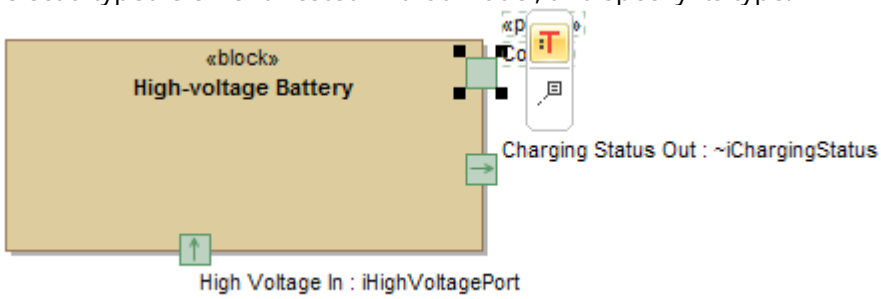
Different abstraction levels

You can also use a Model instead of a package, in which case, the elements cannot be accessed from outside, and you can only select the types from packages that are owned within the Model element.

To only show elements from the desired abstraction level

---

1. Right-click the package to which you want to restrict your type selection, and select **Refactor** > **Convert To** > **Model**.
2. Select a typed element nested in that model, and specify its type.

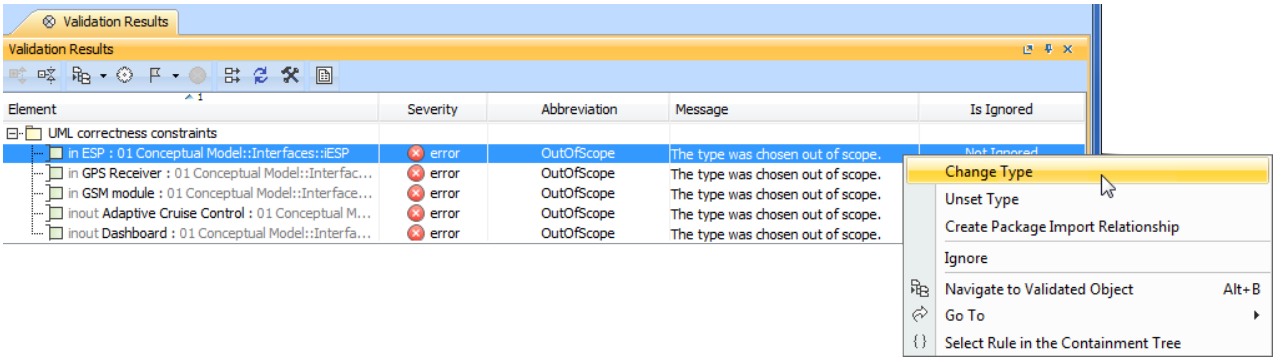


3. Apply **Filter By Package Imports** to only list the elements that belong to the same abstraction level.

Validation

Use active validation to evaluate which element types are out of the scope specified by package imports.

1. Go to **Analyze** > **Validation** > **Validate.**
2. In the **Validation** dialog**,** select **UML correctness constraints** from the **Validation Suite** drop-down list.
3. Click **Validate**.
4. In the **Validation Results** panel, right-click the element you want to change or unset type for, and select **Change Type**, **Unset Type**, or **Create Package Import Relationship** accordingly.

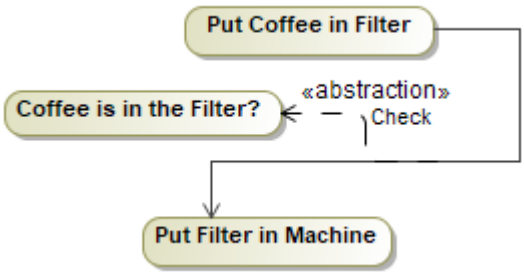


On this page

- Restricting type selection list to imported elements(see page 705)
- Different abstraction levels(see page 706)
- Validation(see page 707)

Abstraction

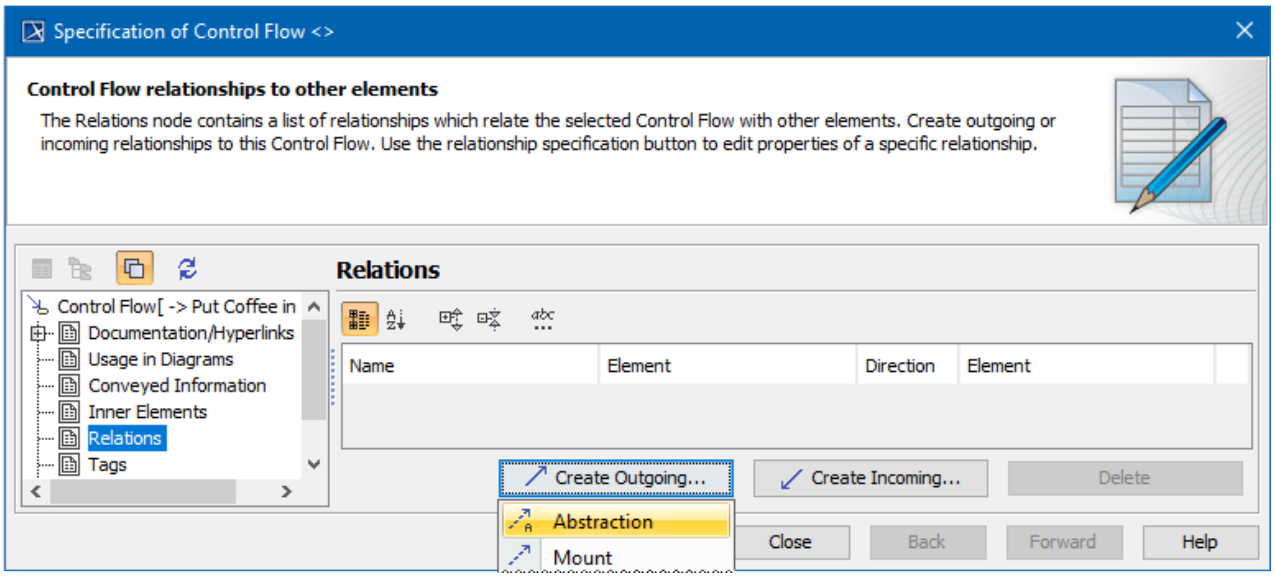
An Abstraction is a Dependency(see page 699) relationship that relates two elements or sets of elements that represent the same concept at different levels of abstraction or from different viewpoints.



An Abstraction can be specified by changing its property values in the Abstraction Specification window(see page 913). Each property is described in the description area in the Specification window.

You may see between which elements Abstraction is drawn in the Abstraction dialog, Source and Target boxes.

Set the Abstraction symbol properties in the Symbol Properties dialog(see page 975).



Creating Abstraction relationship

Device

A Device is a physical computational resource, such as a piece of hardware (a desktop computer, processor, or server), or a human work unit (a department or team). A device is commonly composed of other devices. Composition may be modeled by placing the contained device in the node icon representing the containing device.

A Device can be specified by changing its property values in the Device Specification window(see page 913). Each property is described in the description area on this window.

The Device Device Specification window consists of the following specific specification panes:

- Nested Nodes: lists nodes defined (nested) within the device.
- Deployed Artifacts: an artifact or artifact instance that has been deployed to a deployment target.

You can format the Device symbol properties in the Symbol Properties dialog(see page 975).

Event

An event is the specification of some occurrence that may potentially trigger effects by an object, that is, an event shows what should happen to change a particular state (see page 848)in a system. There are the following event types:

- Any Receive Event(see page 709)

- Call Event(see page 709)
- Change Event(see page 710)
- Signal Event(see page 710)
- Time Event(see page 710)

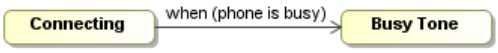
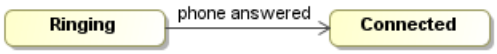
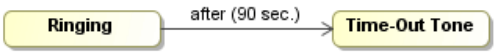
Events are important in diagrams which represent a behavior (see page 662)of a system. These diagrams are listed in the following table.

Diagram name	Event usage description
Activity diagram(see page 311)	To specify an event type for the Accept Event Action(see page 630).
State Machine diagram(see page 308)	To specify a event type for the transition(see page 872), or transition to self.
Protocol State Machine diagram(see page 310)	To specify an event type for the protocol transition, or protocol transition to self.

When specifying an event type for a transition, you can type the command straight on the transition path(see page 873) on the diagram pane. The same assignment is valid for a transition to self, protocol transition, and protocol transition to self.

Event types, their functions, samples, and command syntax are described in the following table.

Name	Function and Sample	Command syntax
Any Receive Event	A trigger for an AnyReceiveEvent is triggered by the receipt of any message (see page 756)that is not explicitly handled by any related trigger.	*all*
Call Event	A call event specifies the receipt by an object of a message (see page 756)invoking a call of an operation(see page 677): In this example, the call event type is specified with the *create()* operation.	*<operation ()>*

Name	Function and Sample	Command syntax
Change Event	A change event specifies a change in the system configuration that makes a condition true:  In this example, the change event type is specified, and its Change expression property is specified as *phone is busy*.	*when (<expression>)*
Signal Event	A signal event represents the receipt of an asynchronous signal instance. A signal event may, for example, cause a state machine to trigger a transition:  In this example, the signal event type is specified, and its Signal property is specified as *phone answered*.	*<signal name>*
Time Event	A time event specifies a point in time. At the specified time, the event occurs. There are two possible types of event occurrences: at the relative time and at the absolute time:  In this example, the relative time event is specified, and its When property is specified as *90 sec*. For more information, see Specifying a time for a time event(see page 711).	• *after (<time>)* - an event occurrence at the relative time • *at (<time>)* - an event occurrence at the absolute time

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related pages

- Activity diagram(see page 311)
- State Machine diagram(see page 308)
- Protocol State Machine diagram(see page 310)
- Model Elements(see page 628)

Specifying a time for a time event

A time event specifies an instance in time by an expression. The expression might be absolute or it might be relative to some other point in time. Relative time events must always be used in the context of a trigger and the starting point is the time at which the trigger becomes active.

A relative time trigger is specified with the keyword “after” followed by an expression that evaluates to a time value, such as “after (5 seconds).” An absolute time trigger is specified with the keyword “at” followed by an expression that evaluates to a time value, such as “Jan. 1, 2012, Noon.”

You can change the time expression by changing the **Is Relative** property value.

To specify a time for a time event

---

1. Open the selected Transition(see page 872)Specification window(see page 913) or Accept Event Action (see page 630)Specification window having the **TimeEvent** type assigned.
2. In the **Trigger** category, do one of the following:

   If you do not see the **Event Type** property, click the button **+** near the **Trigger** category to expand its content.

   - To specify the event occurrence at the absolute time, set **Is Relative** property value to *false*.The time event is specified as absolute, and the command syntax changes from *at ()* to *after ()*.
   - To specify the event occurrence at the relative time, set **Is Relative** property value to *true*.The time event is specified as relative, and the command syntax changes from *after()* to *at ()*.

Signal Event

A signal may be sent as the action of a state in a state machine or as a message in an interaction.

A signal is a named object sent asynchronously by one object and received by another.

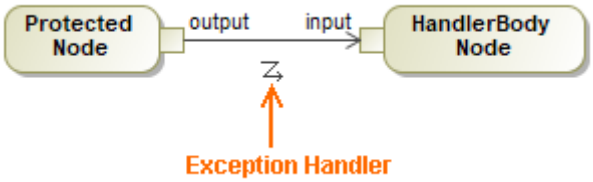
In the Signal Specification window(see page 913), you can modify the current element and add various specifications to it.

Exception Handler

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

An Exception Handler element specifies a body to execute if the specified exception occurs during the execution of the protected node.

The Exception Handler may be drawn from an Output Pin to an Input Pin or from an Action(see page 629) to an Input Pin, as shown in the figure below.



Representation of an Exception Handler.

Related pages

- Formatting symbols(see page 975)
- Specification window(see page 913)
- Model Elements[145]
- Editing property values(see page 921)
- Class diagram(see page 297)
- Action(see page 629)

Execution Environment

An Execution Environment is a type of node that defines a set of services that support software. For example, an execution environment in the processor on a server might provide the operating system-level services needed to support a database application installed in that environment. Some typical stereotypes for an execution environment include <*OS*>, <*workflow engine*>, <*database system*>, and <*J2EE container*>.

Specify the Execution Environment element in the Execution Environment Specification window(see page 913).

The Execution Environment dialog box contains the following specific panes:

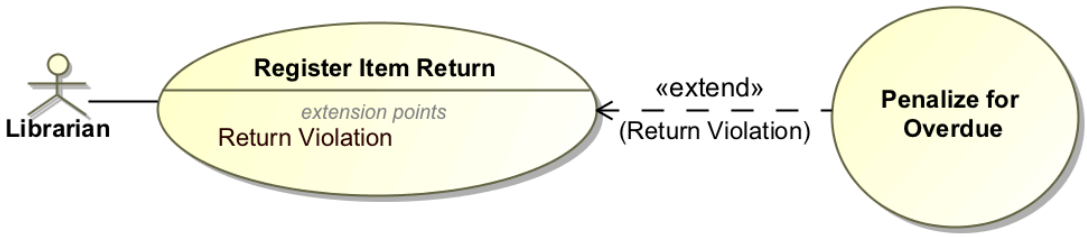
- Nested Nodes: lists nodes defined (nested) in the Execution Environment.
- Deployed Artifacts: an artifact or artifact instance that has been deployed to a deployment target.

You can format the Execution Environment symbol properties in the Symbol Properties dialog(see page 975).

145 https://docs.nomagic.com/display/MD190SP4/Model+elements

Extend

An Extend Use Case indicates how and when the behavior(see page 662) defined in the extending Use Case can be inserted into the behavior defined in the extended Use Case. The Extend relationship is a relationship for an extending Use Case to the extended Use Case. An Extend relationship can have a condition. If the condition is true, then the Extend action(see page 629) is performed.



Example of Extend relationship

The extended Use Case owns one or more specific extension points(see page 877). An extension point identifies a point in the behavior of a Use Case where that behavior can be extended.

In the preceding example, the *Penalize for Overdue* Use Case extends the *Register Item Return* Use Case. The *Register Item Return* Use Case owns the *Return Violation* extension point.

You can format the extend symbol properties in the Symbol Properties(see page 975) dialog.

You can specify Extend properties in the Extend Specification window(see page 913). You can find the description of each property In the same window. Descriptions are presented in the description area of the Specification window.

Editing property values
For more information about specifying property values, see Editing property values(see page 921).

Related pages

- Use Case Diagram(see page 300)
- Formatting Symbols(see page 975)
- Model Elements(see page 628)
- Specification Window(see page 913)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related Resources

Example of the Use Case diagram in *<MagicDraw installation directory>\samples\diagrams\use case diagram.mdzip.*

Extension

An Extension is used to indicate that the Properties(see page 2308) of a Metaclass(see page 780) are extended through a Stereotype(see page 856), and provides the flexibility to add (and later remove) stereotypes to a Class(see page 666).

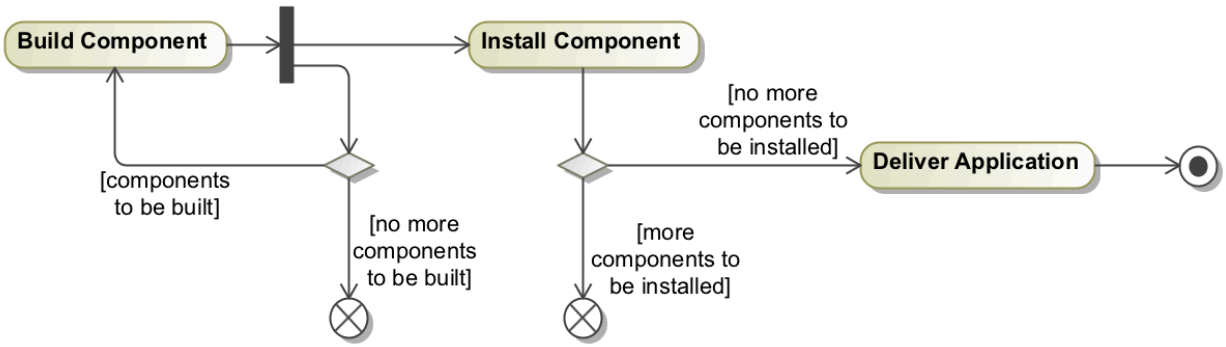
An Extension is a kind of Association(see page 651). One end of the Extension is an ordinary Property that ties the Extension to a Class. The other end is an ExtensionEnd tying the Extension to a stereotype that extends the Class.

You can specify an Extension by changing its property values in the Extension Specification window(see page 913). Each property is described in the description area on this window.

You can format the Extension symbol properties in the Symbol Properties dialog(see page 714).

Flow Final Node

The Flow Final Node terminates a flow and destroys all tokens that arrive at it. It has no effect on other flows in the activity.



Example of flow final node element notation

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related Pages

- Note(see page 791)
- Formatting symbols(see page 975)
- Specification window(see page 913)
- Model Elements(see page 628)
- Editing property values(see page 921)
- Decision and Merge(see page 698)
- Model Merge(see page 1083)
- Activity diagram(see page 311)

Fragment

A fragment is an abstract notion of the most general interaction unit. It is a piece of an interaction. Each interaction fragment is conceptually similar to an interaction by itself.

Our modeling tool represents twelve kinds of fragments: Alternatives, Loop, Option, Parallel, Break, Negative, Critical Region, Consider, Ignore, Weak Sequencing, Strict Sequencing, and Assertion.

Related References

Related Pages

- Model Elements(see page 628)
- Sequence diagram(see page 304)
- Alternatives(see page 889)

Alternative Fragment

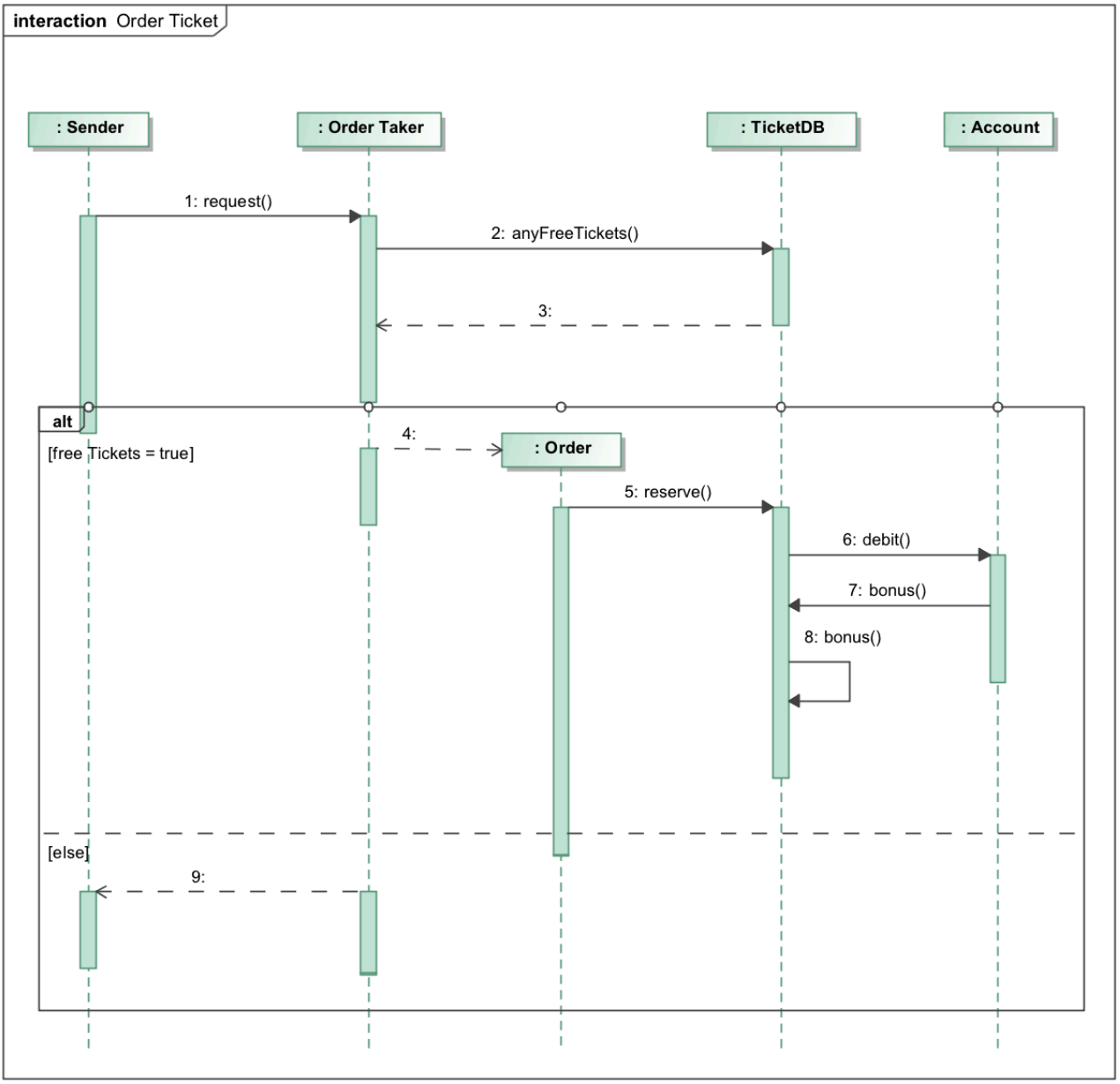
Our modeling tool represents twelve kinds of fragments: Alternatives, Loop, Option, Parallel, Break, Negative, Critical Region, Consider, Ignore, Weak Sequencing, Strict Sequencing, and Assertion.

The alternative fragment models *if...then...else* constructions.

To draw an Alternative Fragment

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- In the Sequence diagram(see page 304) pallet, select the Alternatives element to draw.



Related Pages

- Formatting symbols(see page 975)
- Specification window(see page 913)
- Model Elements(see page 628)
- Fragment(see page 715)
- Combined Fragment(see page 717)

Combined Fragment

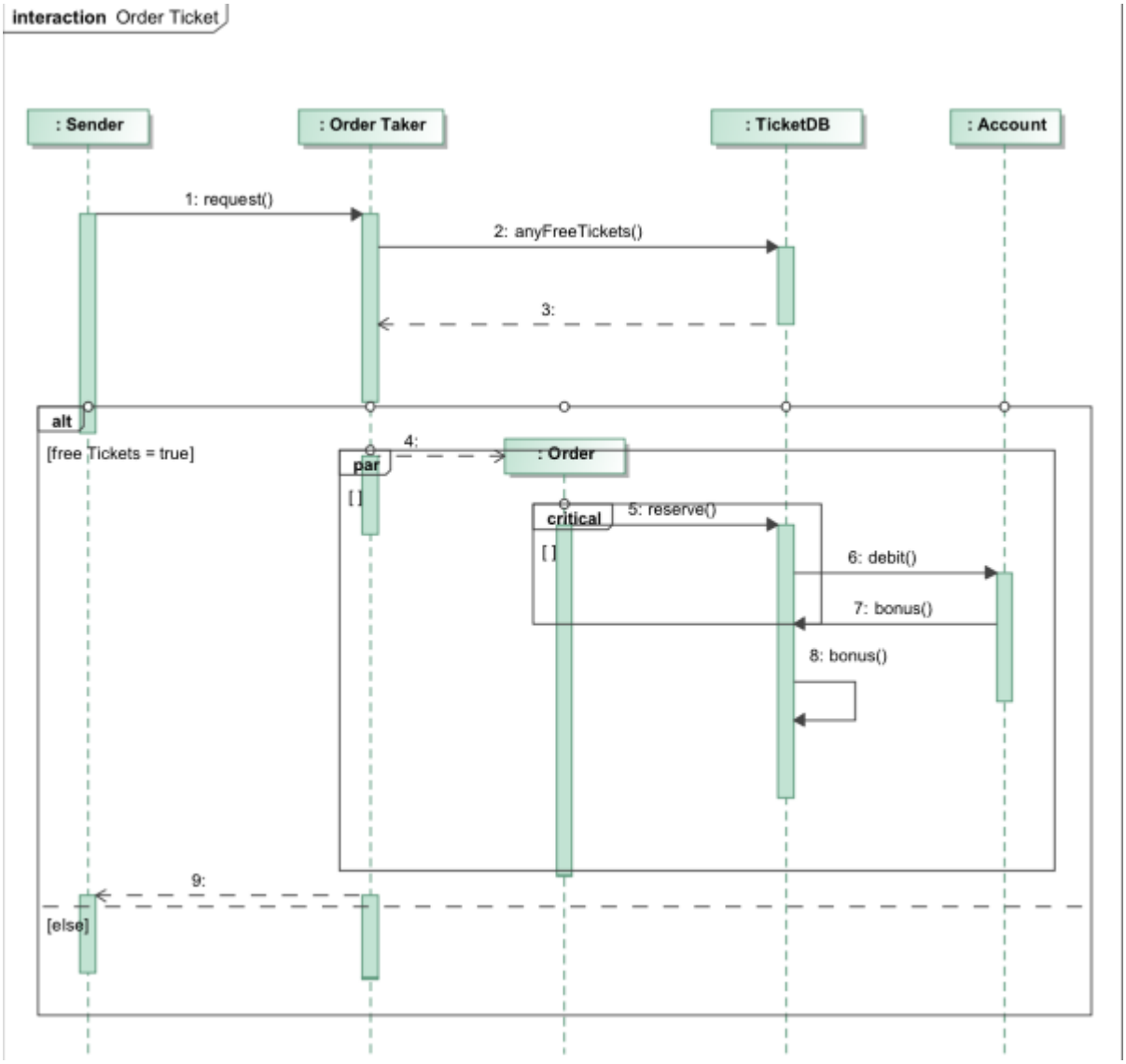
Our modeling tool represents twelve kinds of fragments: Alternatives, Loop, Option, Parallel, Break, Negative, Critical Region, Consider, Ignore, Weak Sequencing, Strict Sequencing, and Assertion.

The UML Combined Fragment element allows the expressions of interaction fragments to be defined in the Sequence diagram(see page 304). The Combined Fragments provide a means to define particular conditions and subprocesses for any sections of lifelines in the Sequence diagram by specifying an area where the conditions or subprocesses apply. Using the Combined Fragment, a fragment of a Sequence diagram can be separated.

To draw a Combined Fragment

1. In the Sequence diagram toolbar, select the Alternatives element to draw.
2. From the Combined Fragment shortcut menu, select the **Covered Lifelines** command and in the Covered Lifelines dialog, select the lifelines to display.
3. In the Sequence diagram toolbar, in the **Options** group select the **Parallel** Combined Fragment to draw.
4. Draw the **Critical Region** Combined Fragment from the Sequence diagram toolbar, in the **Option** button group.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Related pages

- Formatting symbols(see page 975)
- Specification window(see page 913)
- Model Elements(see page 628)
- Sequence diagram(see page 304)
- Alternative Fragment(see page 715)
- Fragment(see page 715)

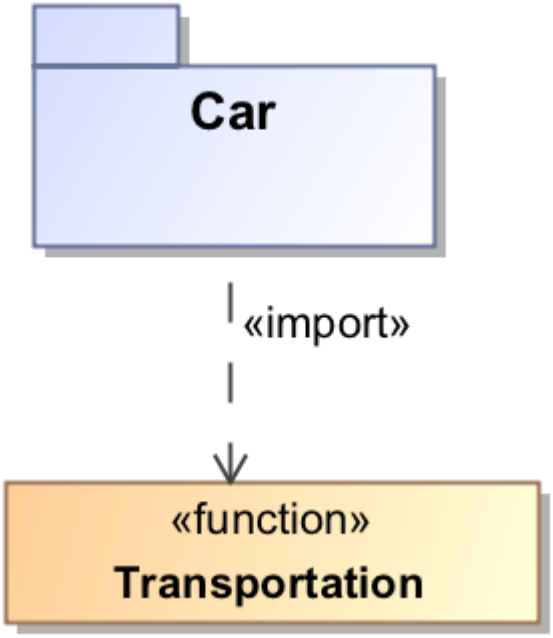
Function Behavior

The function behavior allows modeling external functions that take only inputs and produce outputs. It has no effect on the specified system.

To create a function behavior

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

1. In the **Model Browser**, right click any root package in your model, in the figure below the root package is *Car*.
2. In the shortcut menu(see page 718), select **Create Element > Function Behavior** .



Example of a package importing a function.

Related Pages

- Formatting symbols(see page 975)
- Specification window(see page 913)
- Model Elements(see page 628)
- Sequence diagram(see page 304)
- Behavior(see page 662)
- Function(see page 718)

Gate

MagicDraw allows for displaying the messages (see page 756)when leaving or entering a sequence diagram(see page 304), interaction use(see page 740), or combined fragment(see page 717). The Gate is a connection point for representing a message(see page 756) from the outside to the current sequence diagram(see page 304), interaction use(see page 740), or combined fragment(see page 717).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Gates can be used in three cases:

- For passing and returning arguments to InteractionUse(see page 740), which calls some Interaction.
- For displaying "exceptions" as messages(see page 756) that stops an interaction execution and leaves it.
- For "calling" sequence blocks represented as CombinedFragments(see page 717).

Gate has no notation. Gates are created as message(see page 756) ends when drawing messages(see page 756) to/from a diagram frame, an interaction use(see page 740), or a combined fragment(see page 717).

To create a formal gate, do one of the following

- Draw a call, send, create, or delete a message(see page 756) from the diagram frame.
- Draw a reply message(see page 756) to the diagram frame.
- Draw a call, send, create, or delete a message(see page 756) from the combined fragment(see page 717) (inside combined fragment(see page 717)).

- You can view the created gates of message(see page 756) in the Message Specification window(see page 913). In the **Send** Event (see page 708)list box you can see a formal gate. The **Receive Event** (see page 708)lists the actual gate.
- The Gate uses text from the message(see page 756) as an identification name. For example, a message name or a message operation.

To draw create an actual gate of the formal gate, do one of the following

---

- Draw a call, send, create, or delete a message(see page 756) to the interaction use(see page 740), which refers to the diagram with the formal gates. The **Select Formal Gate** dialog opens.
- Draw a reply message(see page 756) from the interaction use(see page 740), which refers to the diagram with the formal gates. The **Select Formal Gate** dialog opens.
- Draw a call, send, create, or delete a message(see page 756) to the combined fragment(see page 717) (outside combined fragment(see page 717)). The **Select Formal Gate** dialog appears.

You can also view the formal and actual gates in the gates Interaction Specification window(see page 913), Interaction Use dialog, Combined Fragment dialog, and the Actual Gates and Formal Gates panes.

To select a formal gate for the actual gate

---

1. Do one of the following:
   - Draw a message.
   - From the message shortcut menu, choose **Select Formal Gate**.
2. In the open dialog, select one of the listed formal gates and click **OK**. An actual gate is created.

The **Select Formal Gates** command is available only if there are formal gates.

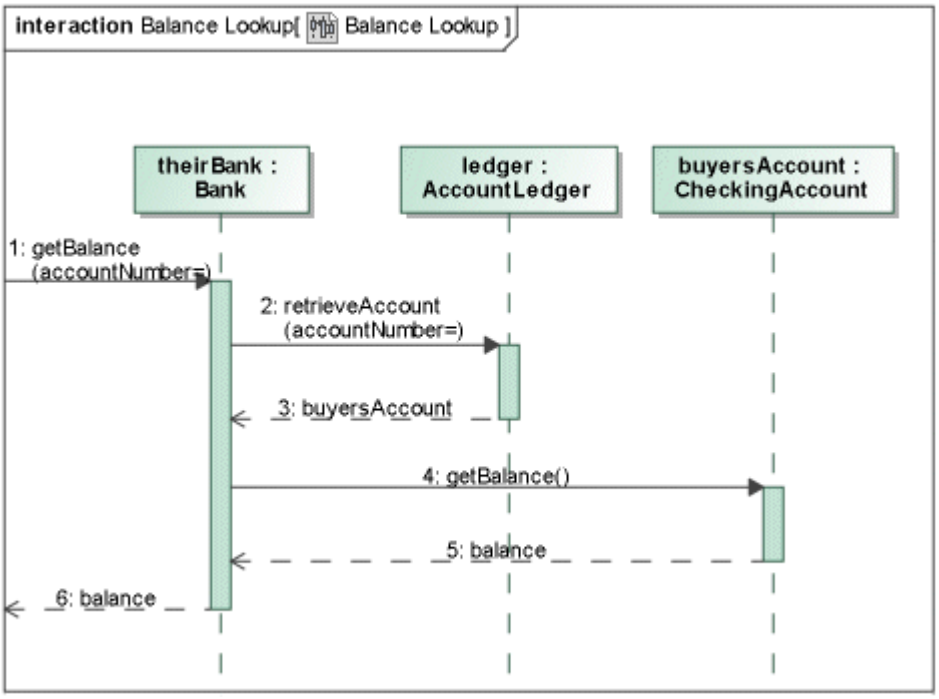
Related pages

Related Pages

- Model Elements(see page 628)
- Class diagrams(see page 297)
- Use Case(see page 876)
- Use Case diagram(see page 300)
- Sequence diagram(see page 304)
- Activity diagram
  (see page 311)

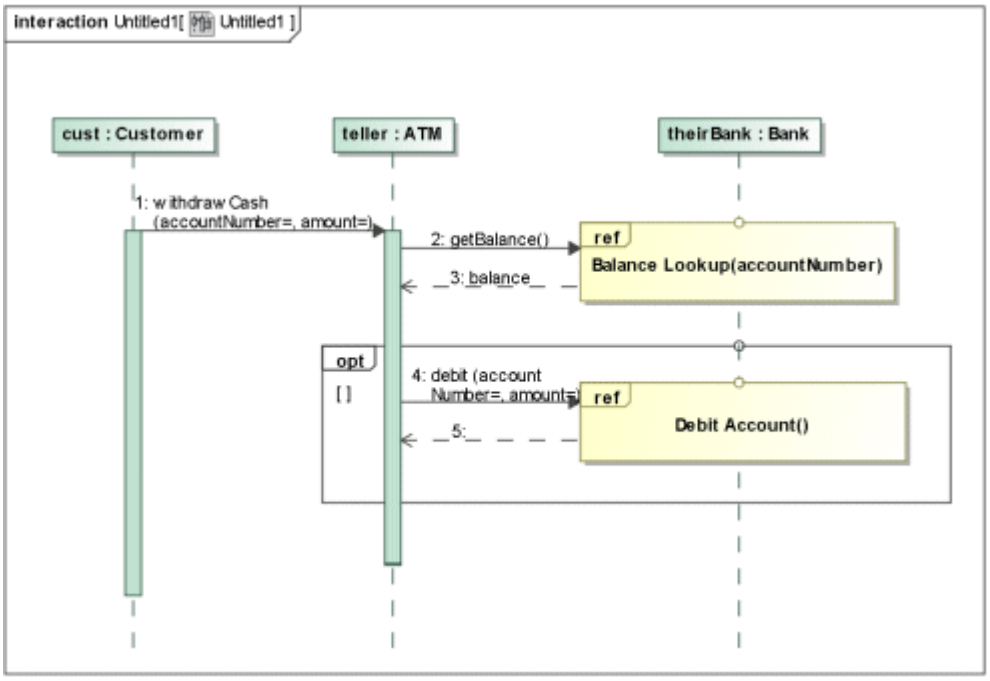
The formal gate and actual gate usage in the sequence diagram

See the following figure where the *getBalance* message is drawn from the diagram frame to the *theirBank* lifeline. The *getBalance* message has a gate.



See the following figure where the actual gate is presented. The Balance Lookup interaction use refers to the Balance Lookup sequence diagram. The *getBalance* message (see the 2nd message) has selected the formal gate and automatically repeats the data of the *getBalance* message from the Balance Lookup diagram.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

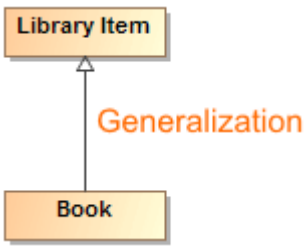


Related Pages

- Model Elements(see page 628)
- Class diagrams(see page 297)
- Use Case(see page 876)
- Use Case diagram(see page 300)
- Sequence diagram(see page 304)
- Activity diagram(see page 311)

Generalization

A generalization is a taxonomic relationship between a more general classifier and a more specific classifier. Each instance of the specific classifier is also an indirect instance of the general classifier. Thus, the specific classifier inherits the features of the more general classifier.



Generalizations can be grouped to the generalization tree(see page 723).
Generalization may be assigned to the generalization set(see page 725).

You can format the generalization symbol properties in the **Symbol Properties** dialog(see page 975).

A generalization can be specified by changing its property values in the generalization Specification window(see page 913). Each property is described in the description area on this window.

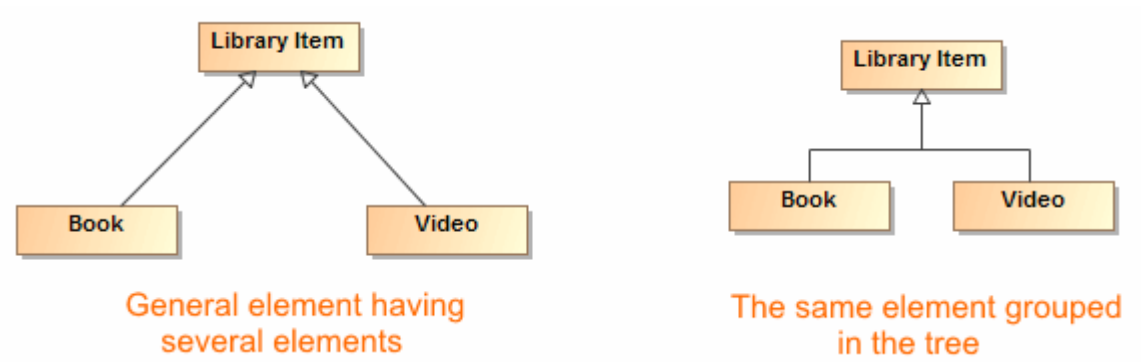
Related pages

- Class diagram(see page 297)
- Use Case diagram(see page 300)
- Component diagram(see page 318)
- Deployment diagram(see page 319)
- Profile diagram(see page 324)
- Model Elements(see page 628)

Sample Model
You can see a generalization relationship sample in *<MagicDraw installation directory>\samples\case studies\Magic Library.mdzip.*

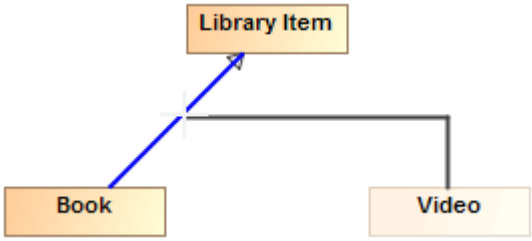
Generalization tree

Usually the general element has more than one specific elements and sometimes specific elements may have their specific elements - the diagram with these generalizations may become messy. To display the generalization relationships and specific elements orderly, the generalization tree can be created.

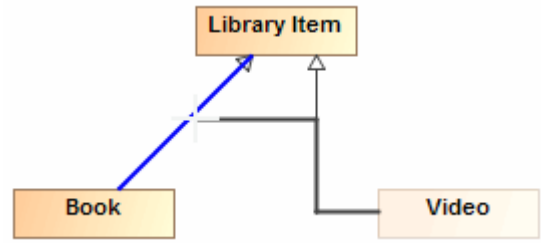


To group generalization paths into a tree, do one of the following

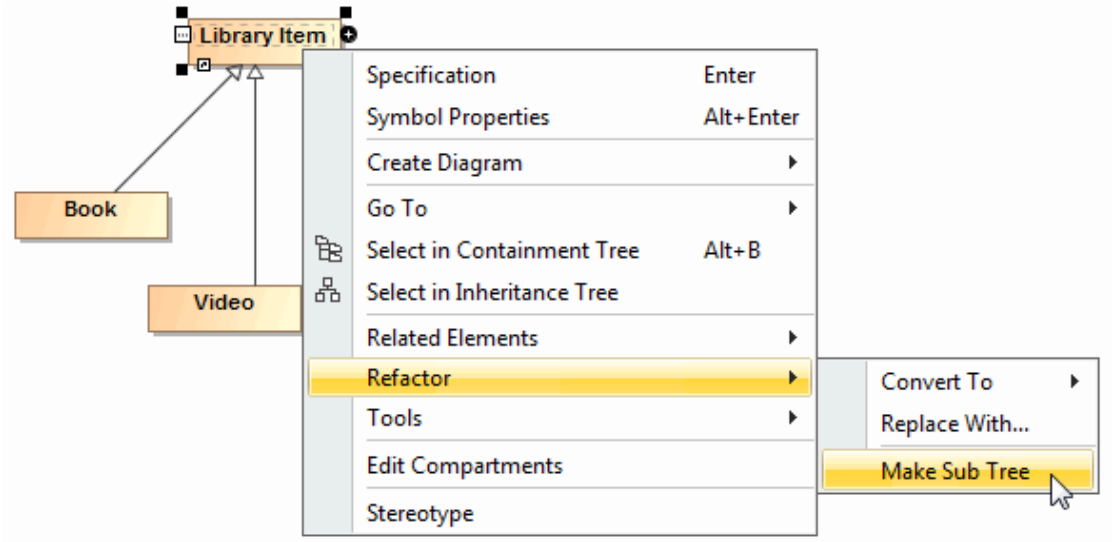
- When drawing generalizations, connect each following generalization to the first one.



- If all generalizations are drawn separately, do on of the following:
  - Select the generalization and drag it to the other generalization.



  - Select the generalization element and from its shortcut menu select **Refactor** > **Make Sub Tree**.



  - Select all elements you want to add to the tree and, on the diagram toolbar, click the Quick Diagram Layout (Ctrl + Q) button and select one of the tree form layout.

To ungroup a generalization tree, do one of the following

- Right click any relationship in the generalization tree. On the opened shortcut menu, click **Ungroup tree**.
- Select the end of a concrete generalization relationship and drag it to other element.

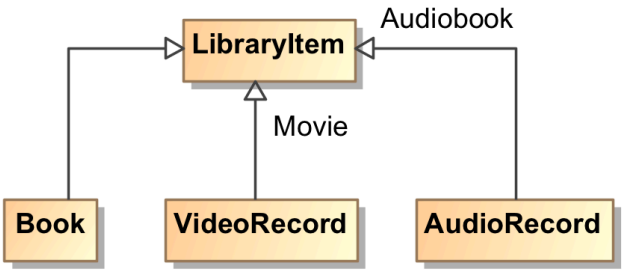
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related Pages

- Generalization(see page 722)
- Formatting Symbols(see page 975)
- Specification Window(see page 913)
- Generalization Set(see page 725)
- Model Elements(see page 628)

Generalization set

A generalization set is a packageable element whose instances define collections of subsets of generalization relationships. Each generalization set defines a particular set of generalization relationships that describe the way which a general classifier may be divided using specific subtypes.



Example of *Movie* and *Audiobook* generalization sets

You can format the generalization symbol properties in the **Symbol Properties** dialog. For more information on how to open the generalization set's Symbol Properties dialog, scroll down to read on how to open the generalization set's Symbol Properties dialog(see page 730).

More Information
For more information about symbol representation properties, see Formatting symbols(see page 975).

You can specify generalization set properties in the Generalization Set's Specification window. For more information on how to open the Generalization Set's Specification window, scroll down to read on how to open that Specification window(see page 729). In the same window, you can find the description of each Generalization Set property in the description area of the Specification Window.

More Information
For more information about the Specification window usage, see Specification window(see page 913).
For more information about specifying property values, see Editing property values(see page 921).

Note
For a generalization set, use the name of a selected generalization set (not the relationship path) to open the **Symbol Properties** dialog or Specification window.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

On this page

- Creating a new generalization set(see page 726)
- Assigning a generalization set for a generalization(see page 728)
- Opening the Generalization Set Specification window and Symbol Properties dialog(see page 729)

Related Pages

- Model Elements(see page 628)
- Specification Window(see page 913)
- Class diagram(see page 297)

Creating a new generalization set

You can create a generalization set using a generalization Specification window or a command from the generalization shortcut menu(see page 725).

To create a new generalization set using the Specification window

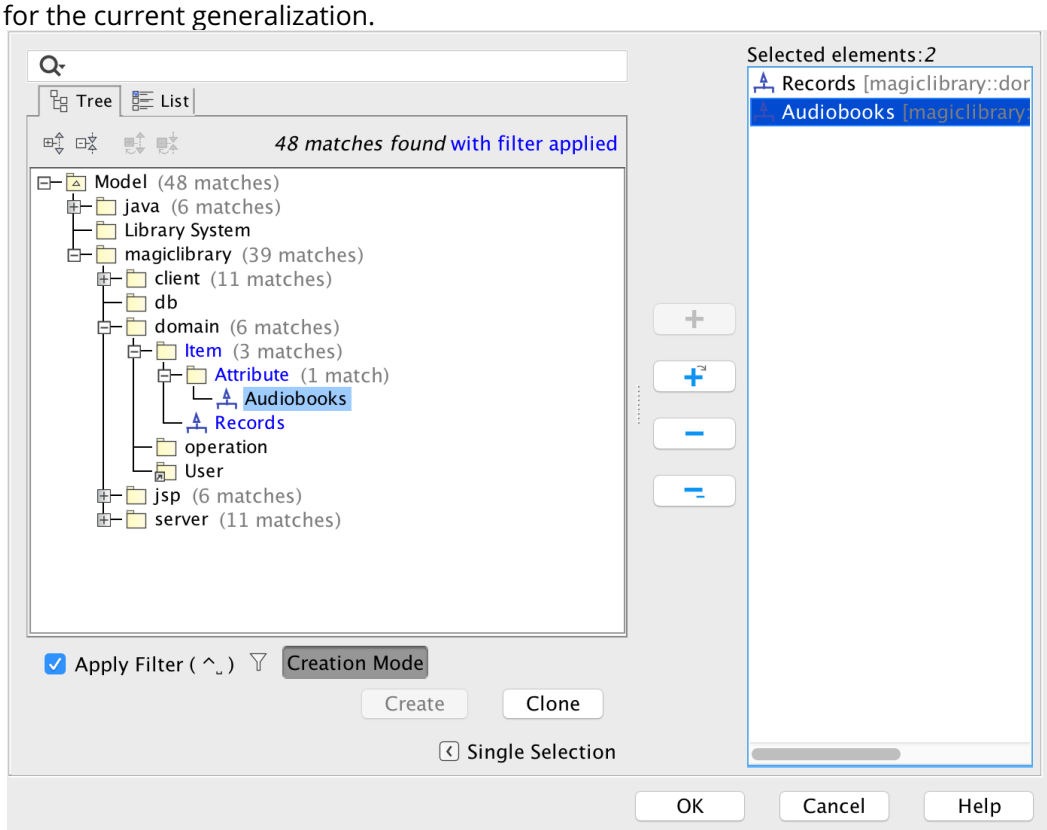
1. Select the generalization and open its Specification window.
2. In the **Generalization Set** property value cell, clock the Edit button . The Select Generalization Set dialog opens.

   For more information about working with the elements Selection dialog, see Selecting elements(see page 1031).

3. Select a generalization set owner and click the **Create** button. If this command is unavailable, turn on the **Creation Mode**.
4. In the opened list, click **Generalization Set**. The Generalization Set Specification window opens.
5. Type the generalization set name and set other properties you need. Click **Close** when you are done.
6. The created generalization set is displayed in the model tree or list and in the **Selected elements** area of the **Select Generalization Set** dialog.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

7. Select the created generalization set and click **OK**. The generalization set is created and assigned for the current generalization.



> **Selected Elements**
> In the **Generalization Set** property value cell, several generalization sets can be listed, but only the first generalization set name is displayed on the diagram pane near the generalization.

To create a new generalization set from the generalization shortcut menu

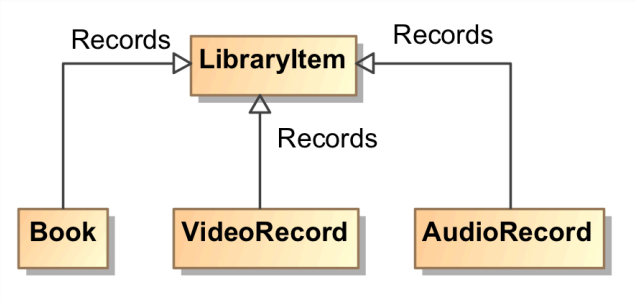
---

1. On the diagram pane, select a generalization and open its shortcut menu.
2. In the shortcut menu, click **Generalization Set**. The Select Generalization Set dialog opens.
3. If it's already there, select the generalization. Otherwise, click the **Create Generalization Set** button.
4. The Select Owner dialog opens.
5. In the dialog, select an existing or create a new generalization set owner and click **OK** when you are done.
6. The Generalization Specification window opens. Type the generalization set name and set other properties you need.
7. Click **Close** when you are done.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Several Generalizations
To create a generalization set for several generalizations at a time

1. Select one generalization, then hold down the Shift key and select the other generalizations you desire.
2. While hovering your mouse over one of the generalizations, right click to show the shortcut menu and select **Generalization Set.**
3. Repeat steps 2-7 from the aforementioned set of instructions.



Assigning a generalization set for a generalization

Assigning a Generalization Set
Before assigning a generalization set, make sure you have at least one generalization set in your model.

To assign a generalization set for a generalization using the Specification window

1. Select a generalization and open its Specification window.
2. In the **Generalization Set** property value cell, click the Edit button . The Select Generalization Set dialog opens.
3. Select the desired generalization set and click **OK**. The selected generalization set is displayed in the **Generalization Set** property value cell.

   For more information about working with the elements Selection dialog, read Selecting elements(see page 1031).

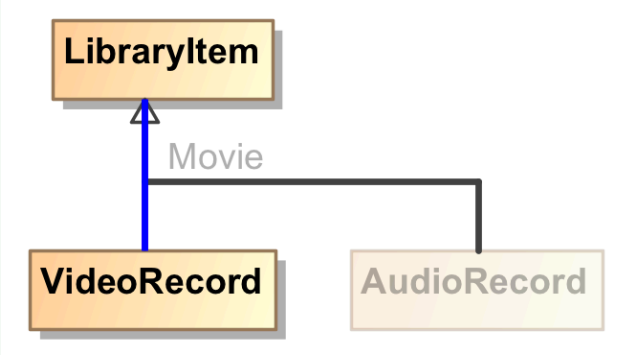
4. Close the Specification window.

To assign a generalization set for a generalization using the shortcut menu

1. On the diagram pane, select a generalization and open its shortcut menu
2. Click **Generalization Set**. In the opened Select Generalization Set dialog, select the generalization set you need. If you do not see your desired Generalized Set then you can can create a new generalization set(see page 727) from the dialog.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Another Alternative to Assigning Generalization Sets
Draw or move a generalization line to a generalization set tree or to a generalization that belongs to a generalization set (see the following figure). The newly created generalization is assigned to the same generalization set.



To assign a generalization to a particular generalization set

1. Open the Generalization Set Specification window.
2. In the **Generalization** property value cell, click the Edit button . The Select Generalization dialog opens.
3. From the list of generalizations, select all you need and click **Apply**.

You can select generalizations that belong to other sets. Selecting such a generalization removes it from the previous set and adds it to the current one.

Opening the Generalization Set Specification window and Symbol Properties dialog

To open its Specification window

Do one of the following:

- On the diagram pane, double click on the generalization set name revealing the Generalization Set Specification window.
- Invoking the Generalization Set Specification window from the diagram pane shortcut menu
  a. On the diagram pane, right click the generalization set name revealing its shortcut menu.
  b. Select the **Specification** command opening the Generalization Set Specification window.
- Opening the Generalization Set Specification window from the Generalization Specification window.
  a. Open the Generalization Specification window by double-clicking on the Generalization arrow.
  b. In the Generalization Specification window, select the **Generalization Set** property cell.
  c. Select the generalization set you desire.
  d. Right-click to open the generalization set shortcut menu. Select **Open Specification**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

e. The Generalization Set Specification window opens.

To open its Symbol Properties dialog

- On the diagram pane, select the generalization set name and from its shortcut menu, select **Symbol Properties**. The Symbol Properties dialog opens.

Grouping generalizations to generalization set trees

> **Make Generalization Set Tree command**
> The Make Generalization Set Tree command is available if the general element has at least two generalizations and at least two generalizations has assigned the same generalization set.

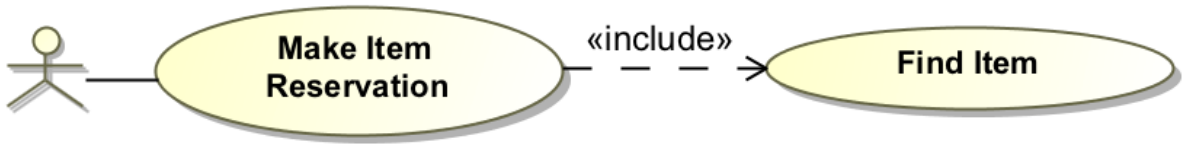
To group the generalizations to the generalization set trees

1. Select a general element on the diagram pane and invoke its shortcut menu.
2. Click **Refactor** > **Make Generalization Set Tree**.

Include

The Include relationship is used when there are common parts of the behavior among two or more Use Cases. Each common part is then extracted to a separate Use Case, to be included by all base Use Cases having this part in common. The Include relationship is represented by a base Use Case to an inclusion Use Case.

An Include relationship between Use Cases indicates that an instance of the including Use Case also contains the behavior as specified by the included Use Case.



Example of Include relationship

In the example above, the *Register Item Reservation* Use Case is a base Use Case. It includes the *Find Item* Use Case. The *Make Item Reservation* Use Case is the including Use Case and the *Find Item* Use Case is the included Use Case.

To use the Include relationship

1. Create two or more Use Cases by dragging and dropping the Use Case element Use Case onto your diagram pane.
2. Click on the Include relationship Include and connect it between two Use Cases.
3. The result should look like the aforementioned example.

You can format comment symbol properties in the **Symbol Properties** dialog.

More Information
For more information about symbol representation properties, see Formatting symbols(see page 975).

You can specify a comment in the Include Specification Window. In the same window, you can find the description of each Include property in the description area of the Specification Window.

More Information
For more information about the Specification window usage, see Specification window(see page 913).
For more information about specifying property values, see Editing property values(see page 921).

Related Pages

- Formatting symbols(see page 975)
- Specification window(see page 913)
- Model Elements(see page 628)
- Sequence diagram(see page 304)
- Use Case diagram(see page 300)
- Behavior(see page 662)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Information Flow

An Information Flow specifies that one or more information items circulate from its sources to its targets. Information flows require an “information channel” for transmitting information items from the source to the destination.

An information channel is represented in various ways, depending on the nature of its sources and targets. It can be represented by connectors, links, associations, or dependencies. For example, if the source and destination are parts in composite structure diagrams such as a collaboration, the information channel is likely to be represented by a connector between them. Or, if the source and target are objects (which are a kind of Instance Specification), they can be represented by a link that joins the two, and so on. The Information Flow and the information item notation are added. You can draw them using the Information Flows toolbar in the Class diagram(see page 297) or Composite Structure diagram(see page 325).

Setting an Information Flow

Information Flows can be set on Associations (see page 651)in the Class Diagram and on Connectors (see page 686)in the Composite Structure Diagrams.

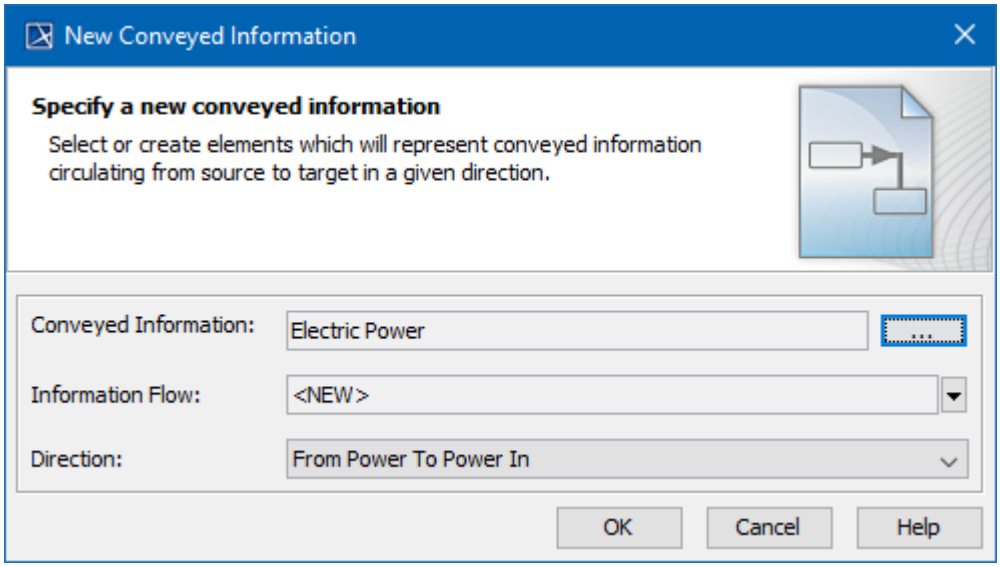
To set an Information Flow

1. Create Classes(see page 666) (Parts(see page 804)) and an Association(see page 651) (Connector(see page 686)) relationship between them.

   Make sure the properties enabling display of Conveyed Information on the relationships are selected.
   To display Conveyed Information

   a. Right-click the Association (Connector) and choose the Symbol Properties(see page 975) command.
   b. In the open dialog, change the properties display mode(see page 917) to **Expert** or **All**.
   c. Set the **Show Conveyed Information A** and **Show Conveyed Information B** check boxes to *true*.

2. Specify a Conveyed Information in the following ways:
   - Drag the Class or Information Item on an Association (Connector) and select the **Set as Conveyed Information** command.
   - Select an Association (Connector) relationship and from the smart manipulator toolbar click .
3. In the **New Conveyed Information** dialog, select **Conveyed Information**, **Information Flow**, **Direction** and click **OK**.



If a new Information Flow is set on a Connector(see page 686) that is represented in other diagrams, the new Conveyed Information is displayed only on the diagram on which you set it. You can reverse the direction of the Information Flow(see page 732) on Connectors(see page 686). How to reverse direction >>(see page 946)

An Information Flow is set.

Conveyed information
One Information Flow may have several Conveyed Information items.

Related pages

- Reversing Information Flow direction(see page 946)
- Information Item(see page 733)
- Model Elements(see page 628)

Information Item

The Information Flows(see page 732) package provides mechanisms for specifying the exchange of information between entities in a system at a high level of abstraction(see page 1211). Information Flows describe the circulation of information in a system in a general manner. They do not specify the nature of the information or the mechanisms by which this information is conveyed (message passing, signal, common data store, parameter of operation, etc.). Also, they do not specify sequences or any control conditions. It is intended that representation and realization links(see page 750) will be able to specify which model element implements the specified information flow, and how the information will be conveyed. while modeling in detail.

An Information Item is an abstraction of all information that can be exchanged between objects. It is a kind of classifier(see page 507) intended to represent information in a very abstract way, one which cannot be instantiated.

In a classifier, the Information Item can be represented as a name inside a rectangle. The black triangle icon at the top of the rectangle indicates that it is an Information Item.

Information Items, or any conveyed classifiers, can be displayed on any relationships.

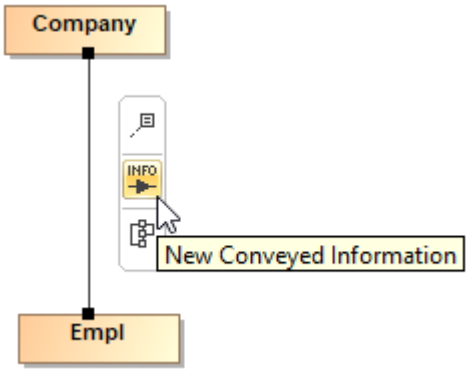
Example of an Information Item named Wage

One purpose of Information Items is to enable you to define preliminary models before making detailed modeling decisions on types or structures. Another purpose of Information Items and Information Flows is to abstract complex models by using a less specific more general representation of the information exchanged between entities of a system.

To create an Information Item

---

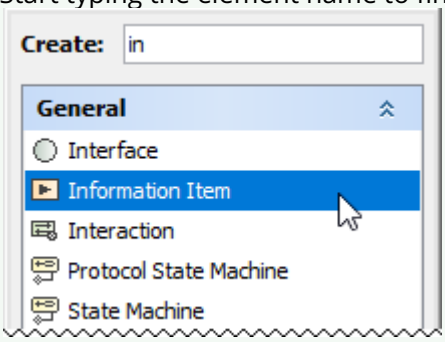
1. In a model or diagram, select a relationship or create a new one.
2. Set a relationship as an Information Flow, using one of the following methods:
   - Click a relationship. The smart manipulator opens. Select the **New Conveyed Information** command.



   - Open the relationship Specification window(see page 913). Select the **Conveyed Information** property group and click the **Add** button.
3. The **New Conveyed Information** dialog opens. Click ... next to the **Conveyed Information** text box.
4. The Select Conveyed Information(see page 1031) dialog opens. Click the Creation Mode button to enable the creation of new elements.
5. Select the package where you want to create a new element.
6. Click the **Create** button and select **Information Item** element in the list.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Quick search
Start typing the element name to find the element quicker.



7. The Specification window of the newly created element opens. Type the element name and specify other properties according to your needs.
8. After you finish, close all the dialogs one by one. The Information Item element is created.

Related pages

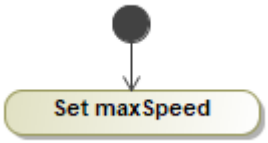
- Formatting symbols(see page 975)
- Specification window(see page 913)
- Model Elements(see page 628)
- Information Flow(see page 732)

Initial Node

An Initial node is a control node where flow starts when the Activity(see page 643) is invoked. An Activity may have more than one Initial node.

You can specify an Initial node by changing its property values in the Initial node Specification window(see page 913). Each property is described in the description area on this window.

You can format Initial Node symbol properties in the Symbol Properties dialog(see page 975).



Instance

On this page

- Saving Instance values to the default property values of the Class (Block)(see page 737)

An instance specification specifies the existence of an entity in a modeled system and completely or partially describes the entity.

The description may include:

- The classification of an entity by one or more classifiers of which the entity is an instance. If the o nly classifier specified is abstract, then the instance specification only partially describes the entit y.
- A kind of instance based on its classifier or classifiers. For example, an instance specification who se classifier is a class describes an object of that class, while an instance specification whose classifier is an association describes a link of that association.
- A specification of structural features values of the entity. Not all structural features of all classifiers of the instance specification need to be represented by slots, in which case the insta nce specification is a partial description.
- A specification of how to compute, derive, or construct the instance (optional).

MagicDraw allows you to create the instances of classifiers such as class(see page 666), interface(see page 741), enumeration(see page 697), use case(see page 876), actor(see page 646), node(see page 782), component(see page 685), artifact(see page 755).

In the Component (or Deployment) diagram Node Instance, Component Instance, Artifact Instance elements are the same Instance Specification elements with an assigned component, node or artifact.

The instances are shown using a rectangle by underlining the name string of an instance element. The instance of an actor (see page 646)is shown as an actor “stick man” figure with the actor’s name below the symbol.

You can specify instance properties in the instance Specification window(see page 913). In the same window, you can find a description of each property. Descriptions are presented in the description area of the Specification window. If you need to handle a huge amount of instance specifications, use the instance table(see page 459).

To assign a classifier to an instance, do one of the following

---

- On the diagram, click an instance shape, and then on the smart manipulator toolbar, click . In the open list, select the classifier.
- On the diagram, click an instance shape, and then press **Ctrl+T**. In the open list, select the classifier.
- On the diagram, click an instance shape, then click the name area. Type the “:” and then type the classifier name. Press **Enter** or click anywhere on the diagram. The classifier is created and assigned.

  > Type the “:” colon, then press Ctrl+Space or Ctrl+Backspace and then in the open list, select the classifier.

- In the Containment tree, select a classifier and drag it to the instance shape.

Use the Node Instance button to create the Instance Specification with assigned Node and the Node Instance will have a Node shape.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To assign Instance Specification as Default Value

- Drag Instance Specification element on Classifier property on a diagram.

> Drag-and-drop feature is available only if Instance Specification classifiers are compatible with the Property type and if Property is editable.

To set the initial value to an attribute of the assigned classifier

1. Open instance Specification window.
2. In the **Slots** property group, select a property and click the **Create Value** button.
3. Type the needed value in the field on the right.

To show slots of the assigned classifier

1. Select an instance shape and click the Compartments button ⊡.
2. In the menu, select **Slots**.

> By default slots of the classifier are suppressed.

To display slot type on the instance symbol

1. Open instance Symbol Properties(see page 975) dialog.
2. Set the **Show Slot Type** property to one of the following:

- None
- Name
- Qualified name

To display specification value on the Instance Specification symbol

1. Open instance Symbol Properties(see page 975) dialog.
2. Select the **Show Specification Value** check box.

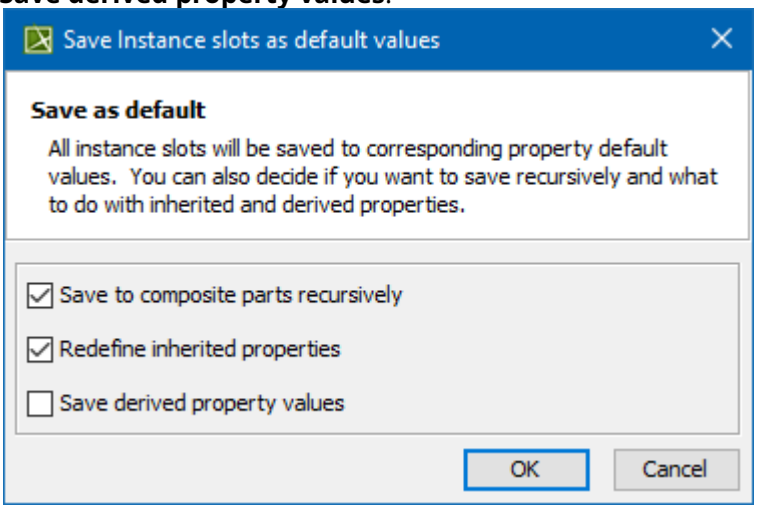
Saving Instance values to the default property values of the Class (Block)

When you have a Class (Block) and an Instance of that Class (Block) which contains some particular values, you can save those values as default to the Values property of the Class (Block).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

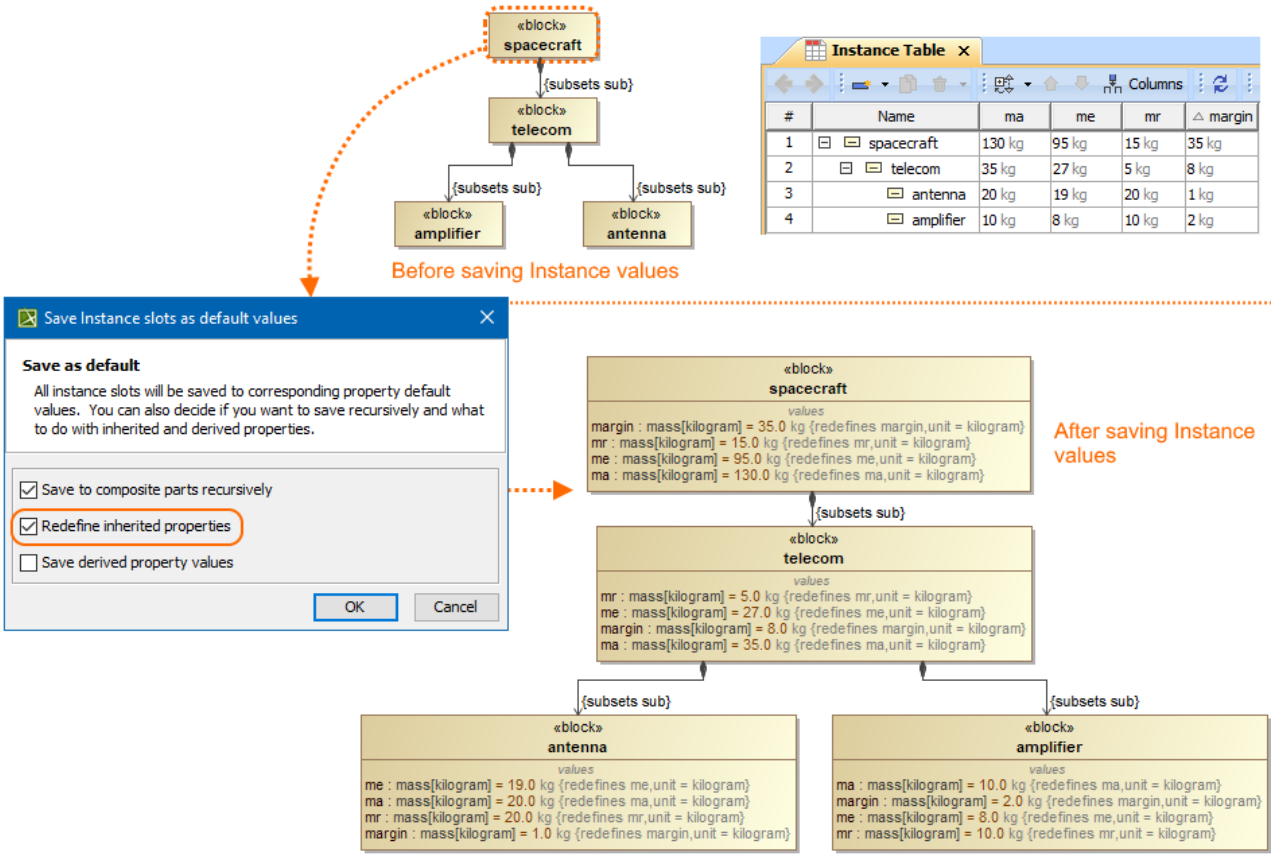
To save the Instance values as default

1. Do one of the following:
    - On the diagram pane, select an Instance.
    - In the Containment tree, select an Instance.
    - In the Instance table, select an Instance.
2. Right-click an Instance and from the shortcut menu, select **Tools** > **Save to Default Value(s)**.
3. In the open dialog, select the needed options:
    - **Save to composite parts recursively**.
    - **Redefine inherited properties**.
    - **Save derived property values**.



4. Click **OK**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



An example of Instance property values saved as default in the Block value properties. The "Save to composite parts recursively" option was selected.

Related pages

- Model Elements(see page 628)
- Class(see page 666)
- Use Case(see page 876)
- Use Case diagram(see page 300)
- Sequence diagram(see page 304)
- Activity diagram(see page 311)
- Class diagram(see page 297)

Interaction

An Interaction is a unit of Behavior(see page 662) focusing on the observable exchange of information between connectable elements, a specialization of Interaction Fragment and of Behavior.

An Interaction originates in a modeling tool while creating Communication(see page 739) and Sequence(see page 739) diagrams. When you double click on an Interaction, a diagram is opened. In the element dialog box, you can modify the current element and add various specifications to it. For more information about element dialog boxes and panes of the element dialog boxes, click here(see page 739).

To open the **Interaction** dialog box, choose the **Specification** command from the Interaction shortcut menu.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

In the **Interaction** dialog box, the **Messages** pane lists Messages contained in this Interaction.

Interaction operand

An Interaction operand represents an Interaction with an optional guard expression. The Interaction operand is executed only if the guard expression tests true at runtime. The object that starts the interaction requires all the information necessary to make the decision about whether to proceed. The absence of any guard condition means that the interaction is always executed.
In the Interaction operand Specification window(see page 913), you can modify the current element and add various specifications to it.

Interaction Use

Interactions are units of behavior of an enclosing Classifier. They focus on the passing of information with Messages between the Connectable Elements of the Classifier.

A reference to the interaction can be created.

You can specify interaction use properties(see page 921) in the Specification window(see page 913). In the same window, you can find the description of each property. Descriptions are presented in the description area of the Specification window(see page 913).

To add a reference to an interaction use, do one of the following

- Open the Specification window(see page 913) of the interaction use and click the cell of the **Refers To** property value. Then click ⊡ and in the open dialog, select the element. Click **OK** when you are done.
- On the diagram, right-click the interaction use shape, and then point to **Refers To**. Then select an element you wish to refere to.
- On the diagram, click an interaction use shape, and then press **Ctrl+T**. Then select an element you wish to refere to.
- On the diagram, click an interaction use shape, and then on the smart manipulator toolbar, click . Then select an element you wish to refere to.
- In the Containment tree, select an element and drag it to the interaction use shape.

To add an actual gate(see page 719)

1. Add a reference to the diagram, from which frame the formal message is created.
2. Draw an actual message with the selected formal gate(see page 719) to the current interaction use.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

The rake icon is displayed on the shape if the interaction use refers to another sequence diagram. The rake icon is not shown by default therefore you need to set the Show Rake Icon(see page 984) property value to true in the Symbol Properties(see page 975) dialog.

Related Pages

- Model Elements(see page 628)
- Class diagrams(see page 297)
- Use Case(see page 876)
- Use Case diagram(see page 300)
- Sequence diagram(see page 304)
- Activity diagram(see page 311)

Interface

An interface is a specifier for the externally-visible operations of a class, component, or other classifiers (including subsystems) without a specification of the internal structure. Each interface often specifies only a limited part of the behavior of an actual class.

The set of interfaces realized by a classifier is its provided interfaces, which represent the obligations that instances of that classifier have to their clients. They describe the services that the instances of that classifier offer to their clients.

You can specify interface properties(see page 921) in the interface Specification window(see page 913). In the same window, you can find the description of each property. Descriptions are presented in the description area of the Specification window(see page 913).

Related pages

Related Pages

- Model Elements(see page 628)
- Class diagrams(see page 297)
- Use Case(see page 876)
- Use Case diagram(see page 300)
- Sequence diagram(see page 304)
- Activity diagram
  (see page 311)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Provided and Required Interfaces

The set of interfaces realized by a classifier is its provided interfaces, which represent the obligations that instances of that classifier have to their clients. They describe the services that the instances of that classifier offer to their clients.

The interfaces may also be used to specify required interfaces, which are specified by a usage dependency between the classifier and the corresponding interfaces. Required interfaces specify services that a classifier needs in order to perform its function and fulfill its own obligations to its clients.

To draw a Provided Interface

---

1. In the class diagram, create a class.
2. Select the class shape and on the smart manipulator toolbar, click the Interface Realization button and then click on the diagram. The provided interface is created.
3. Select the provided interface shape and click the Suppress Operations button.

To draw a Required Interface

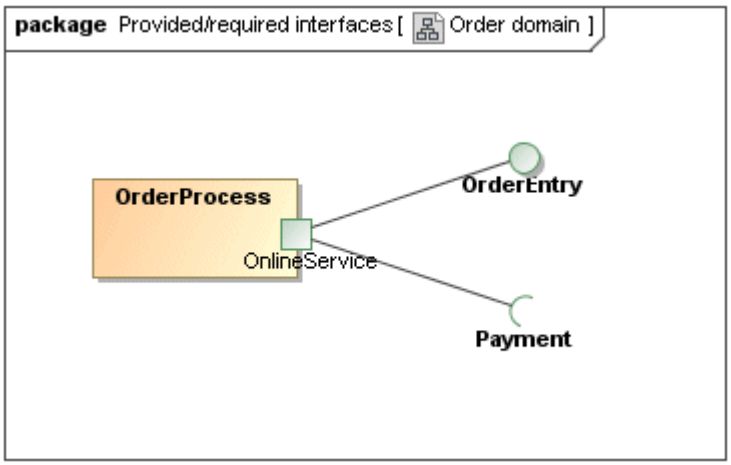
---

1. In the class diagram, create a class.
2. Select the class shape and on the smart manipulator toolbar, click the Usage button and then click on the diagram. The required interface is created.
3. Click the required interface shape and then click the Suppress Operations button.

Provided and Required Interfaces in the Composite Structure diagram

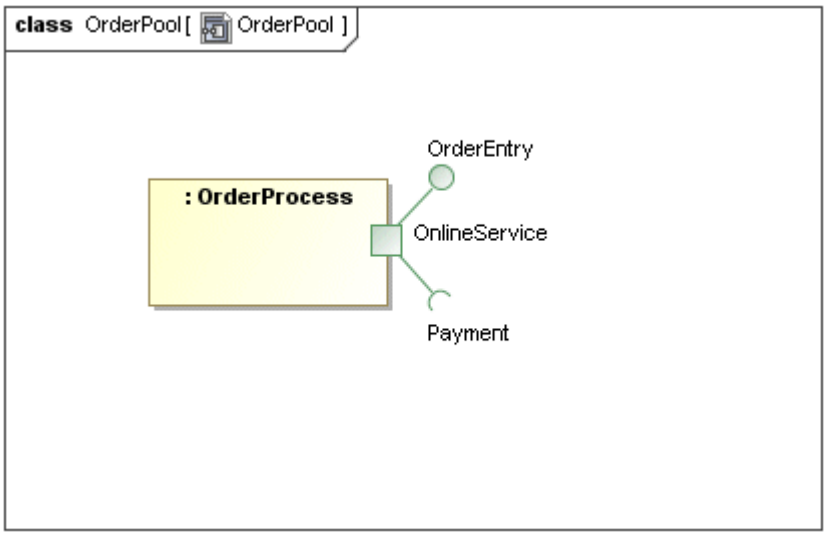
Information about provided and required interfaces is crucial in the assembly stage of complex internal structures. It helps to decide where connectors should be attached.
Provided and required interfaces are valuable parts of the UML Composite Structure Diagram and SysML Internal Block Diagram.

A provided interface is shown using the "lollipop" notation attached to the port, and a required interface is shown using the "fork" notation attached to the port (see the following figure).



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

In the Composite Structure diagram, you cannot draw provided and required interfaces themselves, but, with the new functionality of MagicDraw, you can display a preexisting port with the required and provided interfaces as images.



Lollipop and fork symbols in the Composite Structure diagram are implemented as small attachments to a Port symbol similar to name a label. It is not the same as the independent stand alone notation of the interface, it is only part of the port symbol. It is important for Composite Structure diagrams in which real Interfaces (as Classifiers) cannot be used. It is an optional notation as a port does not display provided or required interfaces by default.

To display provided/required interfaces in the Composite Structure diagram

---

1. Select the port shape that has the provided/required interfaces you want to display.
2. Do one of the following:
   - From the port shortcut menu, select **Display** > **Display Provided/Required Interfaces**.
   - On the diagram toolbar, click

   and select **Display Provided/Required Interfaces.**

Ports can provide or require many interfaces, therefore, you can choose which ones to display or hide. Use the **Edit Compartment** dialog to manage the visibility of these interfaces.

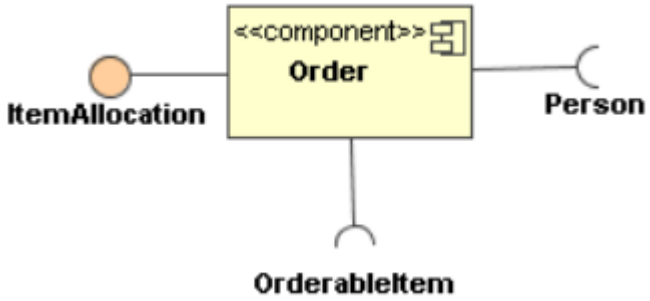
Provided/required interfaces in the Component diagram

A component specifies a formal contract of the services that it provides to its clients and those that it requires from other components or services in the system in terms of its provided and required interfaces.

The required and provided interfaces may optionally be organized through ports.

To add and manage the provided and required interfaces quickly, in the **Component** Specification window, select the **Provided/Required Interfaces** property group.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



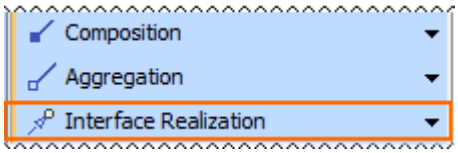
Interface realization

An Interface realization is a specialized Realization(see page 818) relationship between a Classifier Behavior[146] and an Interface(see page 741), signifying that the realizing classifier conforms to the contract specified by the Interface.

A classifier that implements an interface specifies instances conforming to the interface and its ancestors. A classifier may implement a number of interfaces.

In the Interface realization Specification window(see page 913), you can modify the current element and add various specifications to it.

To open the Interface Realization dialog box, choose the Specification command from the Realization shortcut menu.



In the Interface Realization Specification window(see page 913), the Implementing Classifier property references the classifier that realizes the Interface to which it points.

Internal Transition

In all other cases, the Action(see page 629) label identifies the event that triggered the corresponding Action expression. These events are called Internal Transitions and are semantically equivalent to self transitions except that the State(see page 848) is neither exited nor re-entered. This means that the corresponding exit and entry actions are not performed.

> **Fork And Join**
> For more information about defining transitions, read Fork and Join(see page 847).

Specify the Internal Transition in the Transition's Specification window(see page 913).

> **Fork And Join**
> For a more detailed description of this dialog, read Transition(see page 872).

146 https://docs.nomagic.com/display/CST190/Classifier+Behavior+property

To create and specify an Internal Transition

In your project, if you have a State Machine diagram(see page 308) created, do the following:

1. Double-click the State or select Specification from the State's shortcut menu(see page 744). The State's Specification Window opens.
2. Click the **Internal Transitions** group.
3. Click the **Create** button. The Transition's Specification window opens.
4. Specify an **Internal Transition**.
5. Click Close.

To remove an Internal Transition

1. Double-click the State or select Specification from the State's shortcut menu. The State's Specification Window opens.
2. Click the **Internal Transitions** group.
3. The Transition's Specification window opens.
4. Click **Delete**. Then, Click **Close**.

Related Pages

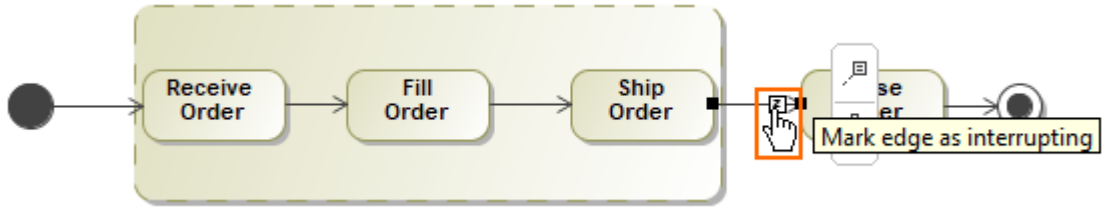
- Formatting symbols(see page 975)
- Specification window(see page 913)
- Model Elements(see page 628)
- Action(see page 629)
- Event(see page 708)
- State Machine diagram(see page 308)
- Fork and Join(see page 847)
- Transition(see page 872)

Interruptible activity region

An Interruptable region contains activity nodes. When a token leaves an interruptable region via edges designated by the region as interrupting edges, all tokens and behaviours in the region are terminated.

Specify the Interruptible Activity Region in the Interruptible Activity Region Specification window(see page 913).

To make an edge as interrupting, simply select an edge and click the "lightning" button, as shown in the image bellow.



Mark edge as interrupting

Lifeline

A lifeline represents an individual participant in the Interaction. The lifeline represents only one interacting entity, symbolized by a rectangle.

The following steps show you how to work with a lifeline:

- Assigning a type to a lifeline(see page 746)
- Showing/hiding a base classifier(see page 747)
- Displaying lifelines(see page 747)

Assigning a type to a lifeline

To assign a type (classifier) to a lifeline

Do one of the following:

- In the lifeline Specification window, in the Type property value, click the **...** button, and in the open dialog, select the type. When you are finished, click **OK**.
- On the diagram, from the lifeline shortcut menu, select **Type**. In the open list, select the type.
- On the diagram, click a lifeline, and then, on the smart manipulator toolbar, click T. In the open list, select the type.
- On the diagram, click a lifeline, and then press Ctrl+T. In the open list, select the type.
- On the diagram, select a lifeline, and in the name area, type the ":" and the name of the type. Press Enter or click anywhere on the diagram. The type is created and assigned.

  ⓘ Type the ":", press Ctrl+Space, or Ctrl+Backspace. In the open list, select the type.

- In the Containment tree, select an element and drag it to the lifeline shape.

Showing/hiding a base classifier

To hide/show a base classifier

- On the diagram, right-click the lifeline. From the shortcut menu, select the **Show Classifier** check box.

Displaying lifelines

You can display lifelines on the Sequence or Communication diagram in the following ways:

- By manually dragging lifeline elements from the Containment tree.
- By automatically using the **Display Lifelines** dialog.

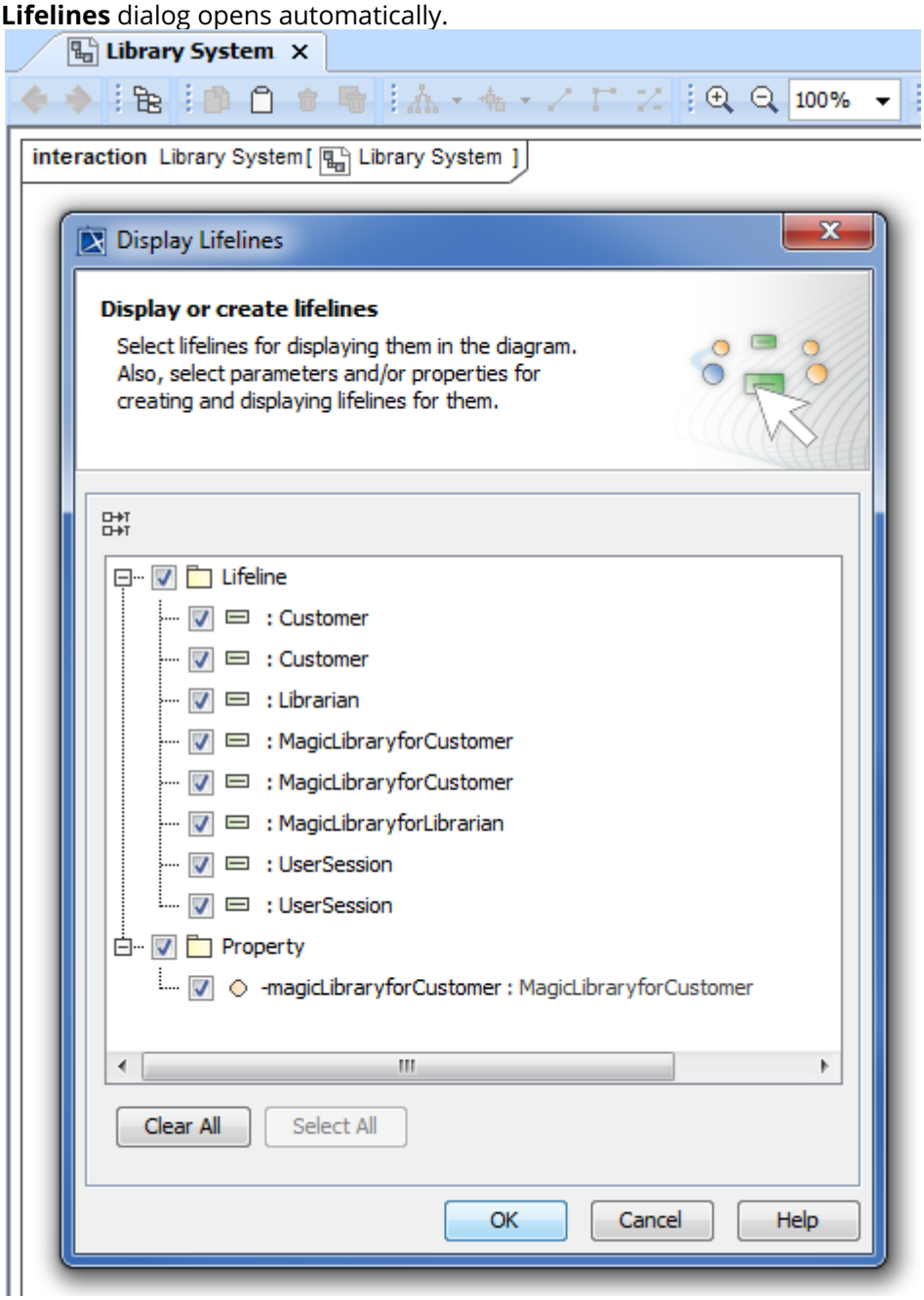
There are two situations in which you can use the **Display Lifelines** dialog to automatically display lifelines:

- When creating a new diagram(see page 747).
- When modifying a pre-existing diagram(see page 748).

To display lifelines when creating a new diagram

---

1. When creating a new Sequence or Communication diagram for an existing classifier, the **Display Lifelines** dialog opens automatically.



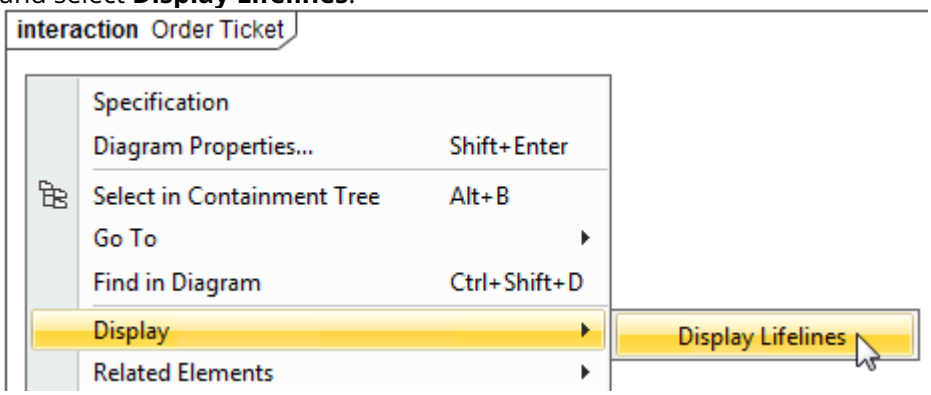
2. In the opened **Display Lifelines** dialog, select the lifelines you want to display, and click **OK**. Selected lifelines are displayed on the diagram pane.

To display lifelines in a preexisting diagram

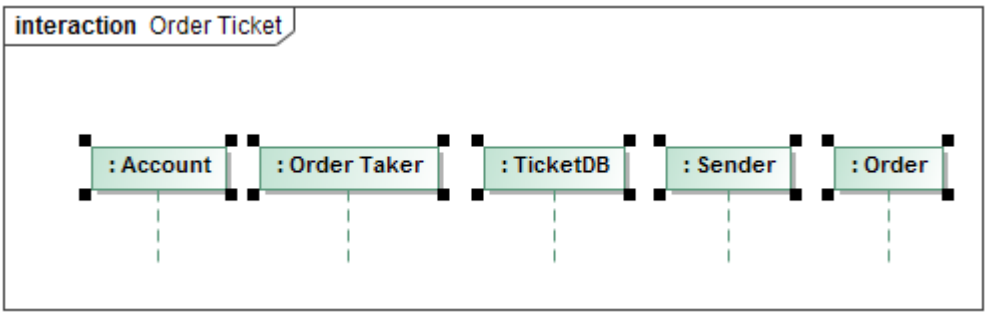
---

1. Select the diagram pane(see page 24).
2. Open the **Display Lifelines** dialog by performing one of the following steps:
   - From the diagram shortcut menu, select **Display** > **Display Lifelines**.
   - On the diagram toolbar(see page 24), click

and select **Display Lifelines**.



3. In the dialog, select the lifelines you want to display, and click **OK**.
   Selected lifelines are displayed on the diagram pane.



Related pages

- Sequence diagram(see page 304)
- Communication diagram(see page 302)
- Creating diagrams(see page 483)
- Model Elements(see page 628)

Sample model

The models used in the figures of this page is the **Communication diagram** and **Sequence diagram** sample models that comes with MagicDraw. To open this sample do one of the following:

- Download*:*
  - communication diagram.mdzip[147].
  - sequence diagram.mdzip[148].
- Find in modeling tool:
  - *<modeling tool installation directory>\samples\diagrams\communication diagram.mdzip.*
  - *<modeling tool installation directory>\samples\diagrams\sequence diagram.mdzip.*

---

147 https://docs.nomagic.com/download/attachments/55847477/communication%20diagram.mdzip?api=v2&modificationDate=1479988632454&version=1
148 https://docs.nomagic.com/download/attachments/55847477/sequence%20diagram.mdzip?api=v2&modificationDate=1479988618002&version=1

Link

An instance of an association is called a Link. A Link is a tuple with one value for each end of the association, where each value is an instance of the type of the end. Links can be represented in the Relationship Map(see page 364) and Dependency Matrix(see page 334).

(Insert Screenshot)

With our modeling tool, you can display on diagrams the following:

- Links created between Instance specifications (the Display Paths(see page 513) command on the Instance specification shortcut menu(see page 750)).
- Instance specifications at the other end of a Link (the Display Related Elements(see page 508) command on the Instance specification shortcut menu).

Related Pages

- Object diagram(see page 299)
- Class diagram(see page 297)
- Relationship Map(see page 364)
- Dependency Matrix(see page 334)
- Model Elements(see page 628)

Related References

Working with Links

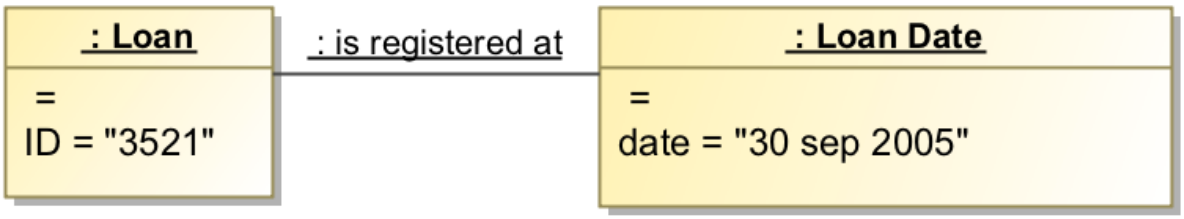
To create a link

1. Create an Object diagram(see page 299).
2. In the diagram pane, create two Instance shapes by dragging and dropping the element onto the diagram pane. Select any Classifier you would like.
3. Draw a **Link** between the two Instance shapes.
4. The Select Association dialog opens.

   > **Select Association dialog**
   > The Select Association dialog displays associations that are created between the classifiers of the Link.

5. Select the appropriate association and click **OK**.
6. The Create Slots dialog opens. Select the appropriate slots and click **OK**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

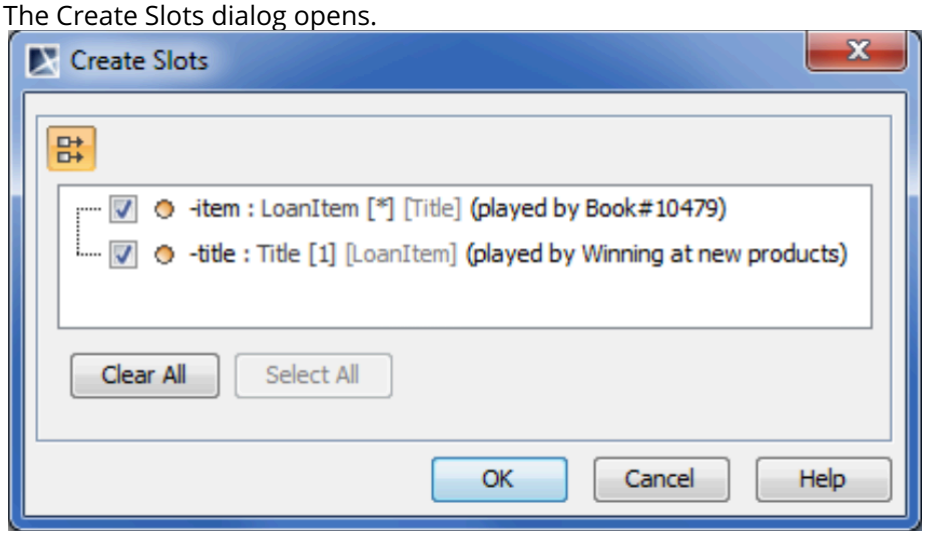


Example of a link between two Instance shapes.

To create slots on a link

---

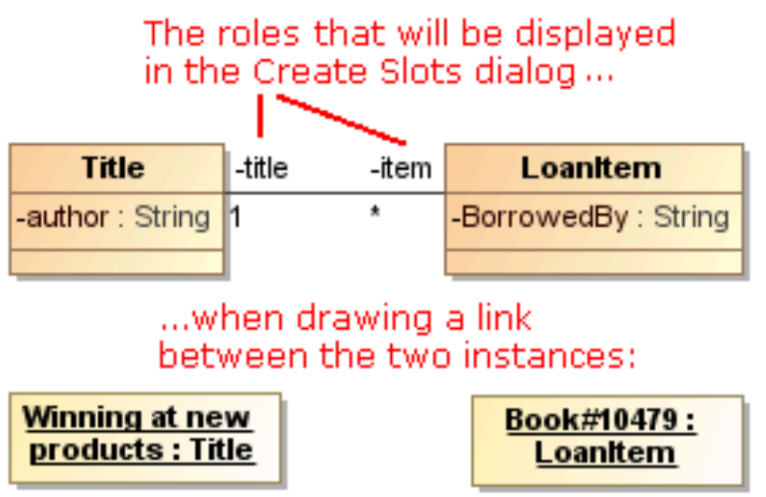
1. In the Object diagram's diagram pane, create two Instance shapes by dragging and dropping the element onto the diagram pane. Select any Classifier you would like.
2. Draw a **Link** between the two Instance shapes. The Select Associations dialog opens.
3. In the Select Association dialog, select the appropriate associations then click **OK**.
4. The Create Slots dialog opens.



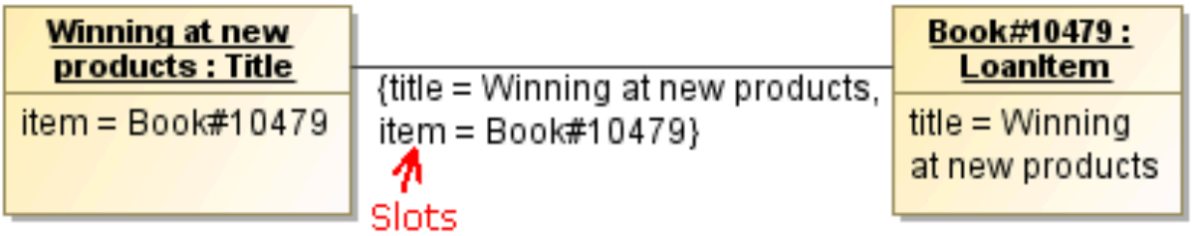
5. Select the slots.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Create Slots dialog
The Create Slots dialog displays the roles of the association that is created between the classifiers of the Link.



6. Click **OK.**



Automatic Creation of Slots on Links

- The slots on the Link are created automatically only if the **Automatically Create Slots of Link** option is set to *true*. For more information, read the instructions on how to turn on automatic creation of slots in Links further on in this page.
- The slots on a Link are created automatically only when the Link references the association. For more information read on how to create a Link at the top of this page.

To turn on automatic creation of slots in links

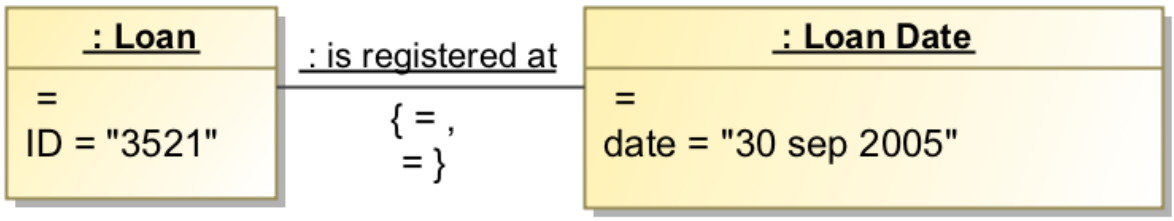
1. In the window, click **Options** > **Environment**. The Environment Options dialog opens.
2. In the **General** options group, under the **Editing** subgroup, set the **Automatically Create Slots in Link** option value to *true*.

To display slot values on a link

1. Right-click on a Link. Select **Symbol Properties**. The Symbol Properties dialog opens.
2. Deselect the **Suppress Slots** check box, so it says *false*.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

3. The result should look something like the figure below.



Suppress Slots Default Value

- The default value of the **Suppress Slots** option is *true*, that is, slots are not displayed on a Link.
- Slots on a Link are displayed in the following form: *{< slot1 name > = < slot values >, < slot2 name > = <slot values> , ... }*

Edit Compartment

You can select to display or hide an individual slot in the Compartment Edit dialog.
To open the Compartment Edit dialog

- Select the link on a diagram pane and on its shortcut menu(see page 750), click **Edit Compartment** > **Slots.**

To change a link end

1. On a diagram pane, select an end of a Link and move it to the other target.
2. The Link is connected to the other target.

Related Pages

- Model Elements(see page 628)
- Classifiers(see page 507)
- Specification Window(see page 913)
- Link(see page 750)

Related References

- Object Diagram(see page 299)
- Active Validation Rules for Links(see page 754)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Active Validation Rules for Links

Active Validation instantly checks the accuracy, completeness, and correctness of a model, displays errors in the model, and suggests solutions.

There are two active validation rules for links:

- Instance Specification type mismatch(see page 754)
- Missing slots in links(see page 754)

Instance Specification type mismatch

The *Instance Specification type mismatch* validation rule allows for updating slot values when referenced association end types are incompatible with connected instance specification types.

Missing slots in links

The *Missing slots on links* validation rule are more useful than migrating projects from the MagicDraw 17.0.2 version or earlier. The validation rule detects links that have related associations but do not have slots created in them. For each incorrect link, the warning is displayed. To resolve the warning, there is the **Create slots for this link** resolution action. After choosing that action, slots are created in the link.

> ⚠ The validation rule is available only if the **Automatically Create Slots in Link** option is set to *true* in the Environment Options dialog.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

On this Page

- Instance Specification type mismatch(see page 754)
- Missing slots in links(see page 754)

Related Pages

- Link(see page 750)
- Working with Links(see page 750)
- Model Elements(see page 628)
- Object diagram(see page 299)

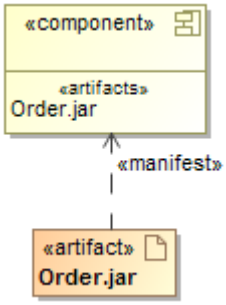
Manifestations

An artifact embodies or manifests a number of model elements. It owns the manifestations, each representing the utilization of a packageable element.

To create the manifestations, simply draw the Manifestation link from an artifact to a component.

To display the manifested artifacts on the component shape

- On the diagram pane, select the component shape, click the Compartments button, and from the list, select **Artifacts**.
- In the component **Symbol Properties** dialog, cancel the selection of the **Suppress Artifacts** check box.



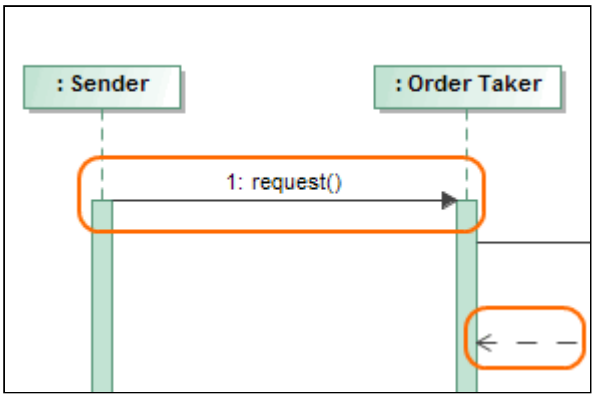
Related pages

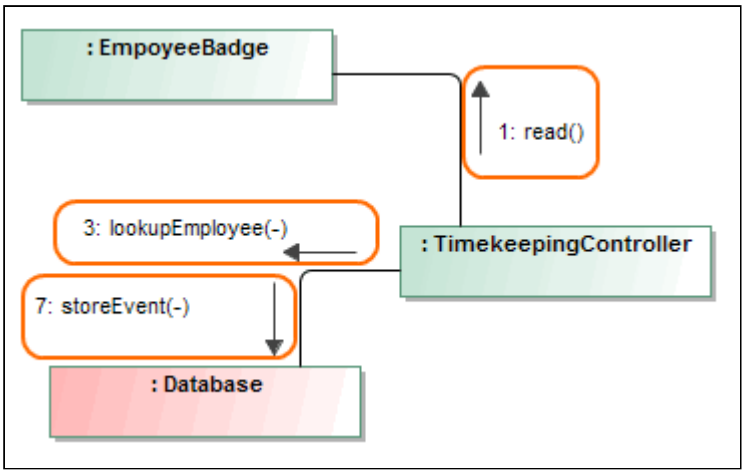
- Model Elements[149]
- Component diagram(see page 318)
- Specification Window(see page 913)
- Artifact[150]
- Compartment(see page 569)

Message

A Message defines a particular communication between Lifelines(see page 746) of an Interaction(see page 740), such as raising a Signal, invoking an Operation(see page 677), or creating or destroying an Instance. Messages specify the kind of communication, the sender, and the receiver.

A Message is represented by arrows between the lifelines. The style of the Message line and arrowhead reflect the types of the Message.



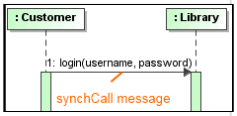
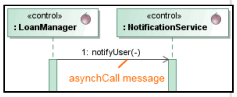
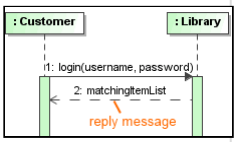
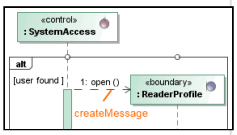
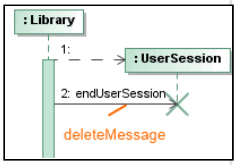


You can see the description of a selected property in the description area of the Specification window(see page 913). To see descriptions, be sure the **Show Description** option is selected.

149 https://docs.nomagic.com/display/MD190SP4/Model+elements
150 https://docs.nomagic.com/display/CBMP190SP4/Artifacts

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Message sorts

Message Sort	Description	Example
synchCall	The Message was generated by a synchronous call to an Operation. Synchronous Messages can overtake each other. When a Message represents an Operation, the arguments of the Message must correspond to the parameters of the Operation.	
asynchCall	Asynchronous means that the caller continues immediately after the call. asynchCall Messages do not have reply Messages.	
asynchSignal	The Message was generated by an asynchronous send Action. The argument of the Message must correspond to the attributes of the Signal.	
reply	A reply Message to an Operation call.	
createMessage	The Message designates the creation of another Lifeline object. No other Messages on a given Lifeline in an Interaction operand may appear above a Lifeline.	
deleteMessage	The Message designates the termination of another Lifeline.	

Message syntax

The Message name is displayed on the message path on the diagram pane. The syntax of the message name is as follows:

<messageident> ::= [<attribute> '='] <signal or operation name> ['(' [<argument> ','

<argument>] ')'] [':' <return value>]*

<argument> ::= ([<parameter name> '='] <argument-value>) / '-'

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example
v=fo(P1="a", P2=True):12

Common actions with Messages

To set an action type for a Message

Do one of the following:

- In the Message Specification window(see page 913), click the **Message Sort** property value cell and select the action type from the drop-down list.
- On the diagram pane, right-click the Message and select a desired action type from the shortcut menu.

To show / hide Message numbers

---

1. Right-click the diagram pane to open its shortcut menu.
2. Then do one of the following:

- In a Sequence (see page 304)or Time diagram, select / clear **Show Message Numbers**.
- In a Communication diagram(see page 302), select / clear **Numbering** > **Show Message Numbers**.

Related pages

- Assigning Operations to Messages(see page 758)
- Message name parsing(see page 760)
- Assigning Signals to Messages(see page 768)
- Creating Signal receptions for Messages(see page 769)
- Messages in Sequence and Time Diagrams(see page 771)
- Messages in Communication diagrams(see page 774)

- Model Elements(see page 628)
- Specification Window(see page 913)
- Lifelines(see page 746)
- Interaction(see page 740)
- Operation(see page 677)

Assigning Operations to Messages

You can assign Operations (see page 677)only to Call (synchronous and asynchronous) Messages. Only one Operation can be assigned to a Message.

To assign an Operation to a Message

- Assign an Operation to a Message in the Message Specification window(see page 913):

  1. Draw a call Message between two Lifelines (see page 746)or select an existing Message on a diagram pane.
  2. Open the Message Specification window.
  3. Click the **Signature (operation)** property value cell and then select an Operation in the drop-down list.

  > - If the **Signature (operation)** drop-down list is empty, check if the Lifeline the current Message is drawn to has a type assigned, and if this type has a specified Operation.
  > - In the **Signature (operation)** drop-down list, both Operations and signals of your project are listed. Make sure you have selected an Operation. If you assign a signal to a call Message, the Message will automatically be converted to a send Message.

- Type an Operation name right on the Message on the diagram pane.
- On the diagram, right-click the Message and on the shortcut menu, click **Operation**. Then click to select the Operation that you want to apply on the element, or click the **Create Operation** button.

If an Operation you are assigning to a Message has default parameter values, these default parameter values are assigned to the Message arguments. However, if the types of the Operation parameter and the Message argument do not match, the default parameter value is assigned to the Message argument.

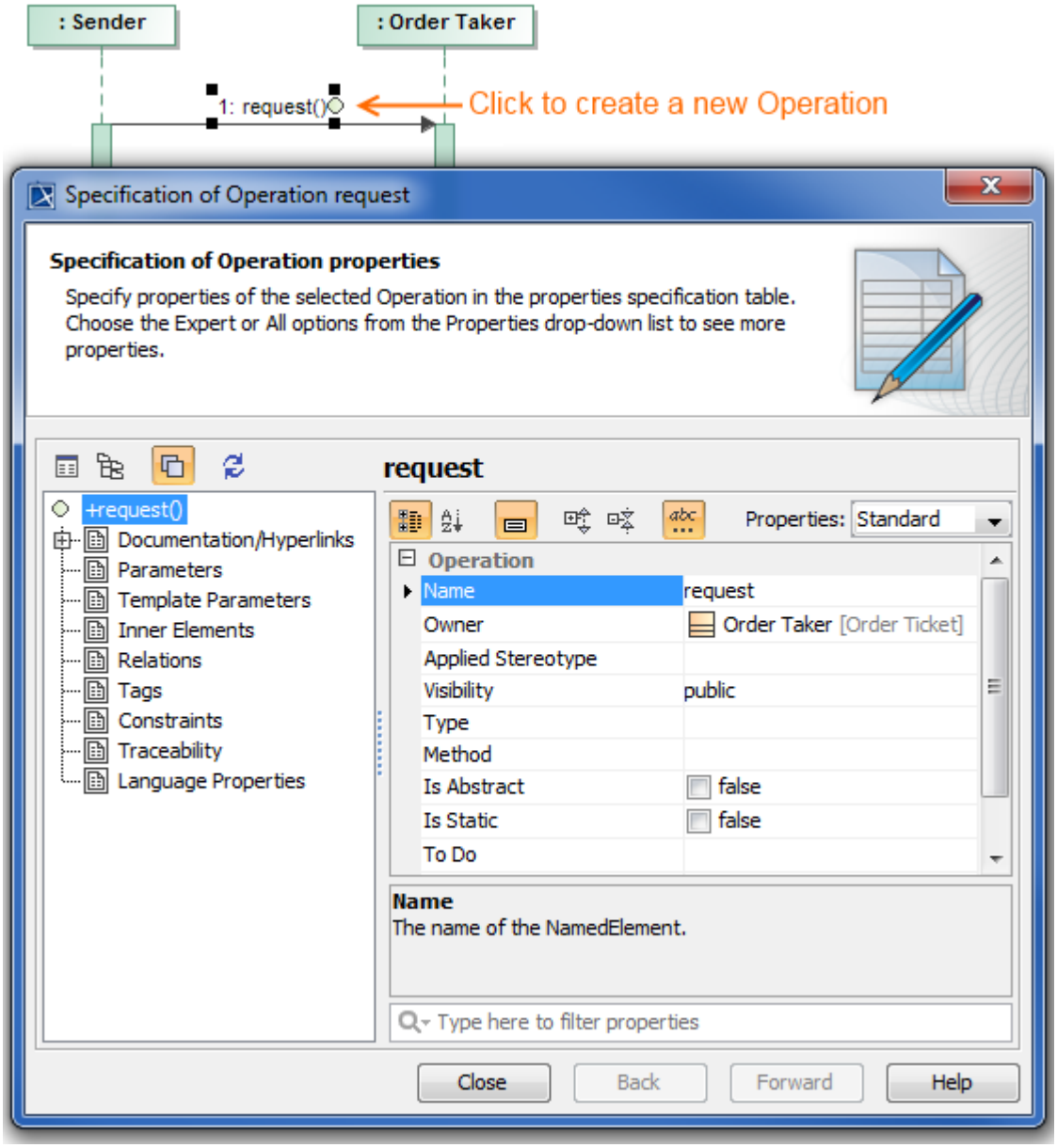
> Names of operation parameters are displayed in message signature by default. However, you can change this behavior by opening the **Symbol Properties** dialog (right-click a Message and select **Symbol Properties**) and setting the **Show Attribute/Parameter Names** property to **true** to display them or **false** to hide them.

To create a new Operation for a Message

> You can create a new Operation only if the Lifeline the Message is drawn to has a type assigned.

1. Draw a Call Message between two Lifelines or select an existing Message on a diagram pane.
2. Do one of the following:
   - Click the **Create New Operation According to This Message** button (a small green circle at the end of the Message name) and define the Operation property values in the Operation Specification window.
   - On the Message shortcut menu, click **Operation** and then click the **Create Operation** button. Select the owner and click **OK**. In the Operation Specification window, define the Operation and click **Close**.
   - Type an Operation name right on a Message on the diagram.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Related pages

- Model Elements(see page 628)
- Specification Window(see page 913)
- Lifelines(see page 746)
- Interaction(see page 740)
- Operation(see page 677)
- Message(see page 756)

Message name parsing

Arguments, parameters, and return values specified on a Message in a Sequence diagram(see page 304) can be parsed and new Operations (see page 677)can be created. Parsing and mapping Message names to real model elements (operations, parameters, etc.) allows for easy transition from prototype

Sequence diagrams to complete Sequence diagrams representing actual Interactions between system parts.

Message name parsing

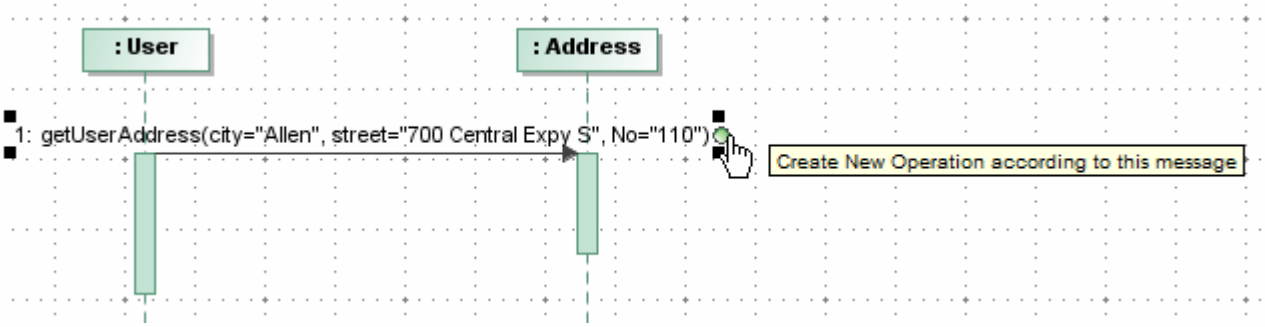
On a Message name area on a diagram pane, you can type the name of a new Operation together with arguments, parameters, and return value. After you click on the button near the Message name, the new Operation will be created, together with its specifications.

Message syntax

The whole Message name should be written according to Message syntax rules(see page 756).

Message Parsing notification message

If any Message name syntax errors are found, users are warned and given suggestions for fixing these mistakes. The Message Parsing notification message opens, showing the concrete error(see page 0).



In the preceding example, you can see that the following appears on the Message name:

getUserAddress(city="Allen", street="700 Central Expy S", No="110"). After you click the Create New Operation (this message button), the new *getUserAddress* Operation will be created. The following Message arguments will be created - *Allen*, *700 Central Expy S*, and *110*, together with the corresponding parameters - *city*, *street*, *No*.

To create a new operation from the diagram pane

1. Create a call Message (asynchCall or synchCall) between two Lifelines.

   > The Lifeline that the call Message is connected to must have a type specified.The preceding figure shows that the *Address* type is assigned to the second Lifeline.

2. On the diagram pane, on the message, type the message name according to the syntax rules(see page 756).
3. After you have typed the Message name, exit the edit mode by clicking on the diagram pane.
4. Select the Message label again. The Create New Operation according to this message button is displayed next to the Message name.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

5. Click the Create New Operation according to this message button . A new Operation is created and represented in the message Specification window(see page 913), the **Signature (operation)** property cell.

> Note that a new Operation is created only after you click the Create New Operation according to this message button . If you do not click this button, the Message label will be treated as a Message name.
> If the syntax is written incorrectly, the Message Parsing window(see page 0) opens.

Parsing rules

The following table describes Message name samples and parsing results:

Purpose	Message name typed on the diagram pane	Action after typing on the message label	Result
Create a new Operation	*login ()*	Click the Create New Operation according to this message button .	The new operation is created: *login ()* 1. The Operation is referenced in the Message Specification window(see page 913), the **Signature (operation)** property cell. 2. For the Lifeline type, the login operation is created.
	After click on the Create New Operation according to this message button , a new Operation is always created. A new Operation is created even if an Operation with the same name already exists.		

Purpose	Message name typed on the diagram pane	Action after typing on the message label	Result
Create a new Operation with an argument	*login ("John")*	Click the Create New Operation according to this message button .	The new operation is created: *login (unnamed1="John")* 1. The Operation is referenced in the Message Specification window(see page 913), the **Signature (operation)** property cell. 2. In the message Specification window > the **Arguments** property group, the following argument is created: *John*. 3. For the Lifeline type, the *login* Operation is created. 4. The *login* Operation's parameter is created and synchronized(see page 1064) with the message argument.

Different kinds of arguments are supported. For the following arguments the following value specifications are created:

John - Opaque Expression

12345 - Literal Integer

True - Literal Boolean

"John" - Literal String (always when quotation-marks are used).

For more information about value specification, see Assigning value specification as property value(see page 930).

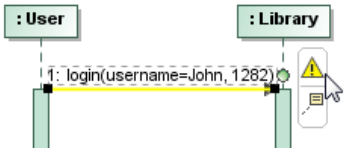
Purpose	Message name typed on the diagram pane	Action after typing on the message label	Result
Create a new Operation with a parameter and an argument	*login (username="John")*	Click the Create New Operation according to this message button .	The new operation is created: *login (username="John")* 1. The Operation is referenced in the Message Specification window(see page 913), the **Signature (operation)** property cell. 2. In the Message Specification window > the **Arguments** property group, the following argument is created: *John*. 3. For the Lifeline type, the *login* operation is created. 4. The *login* operation's parameter is created and synchronized(see page 1064) with the Message argument.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Purpose	Message name typed on the diagram pane	Action after typing on the message label	Result
Create a new Operation with a return parameter	*login () : true*	Click the Create New Operation according to this message button .	The new operation is created: *login () : true* 1. The Operation is listed in the Message Specification window(see page 913), the **Signature (operation)** property cell. 2. In the Message Specification window > the **Arguments** property group the following argument is created: *true*. 3. For the classifier assigned to the Lifeline, the *login* Operation is created, together with the parameter. 4. The *login* Operation's parameter with the return direction is created. The Operation's parameter is synchronized(see page 1064) with the Message argument.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

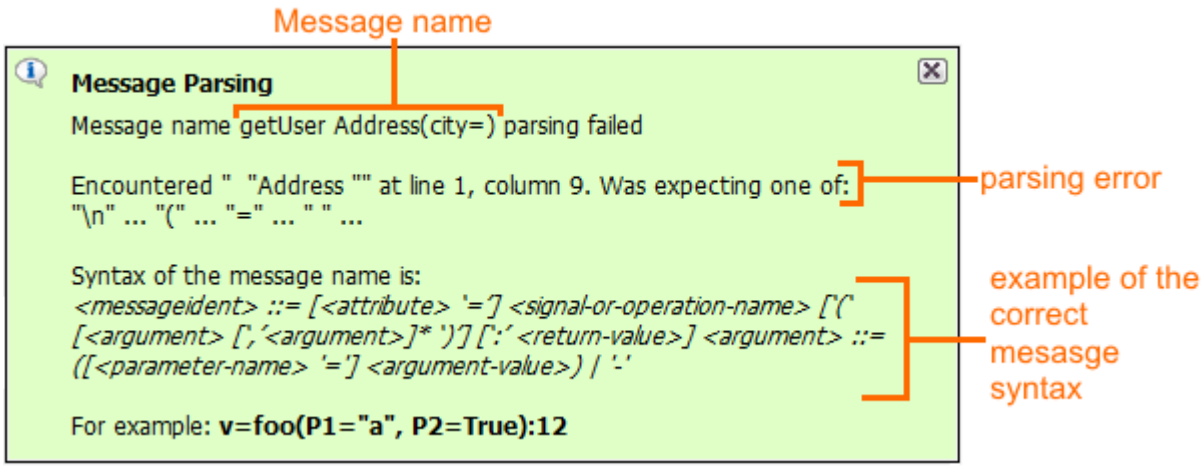
Purpose	Message name typed on the diagram pane	Action after typing on the message label	Result
Create a new Operation for the Message together with a reply Message	*status = login ()*	Click the Create New Operation according to this message button . 	The new operation is created: *status = login ()* 1. The Operation is listed in the message Specification window(see page 913), the **Signature (operation)** property cell. 2. In the Message Specification window > the **Reply message** property cell, the replay Message is created. (The **Replay message** property cell is displayed in Expert mode only.) 3. The *status* argument is created in the replay Message.

Purpose	Message name typed on the diagram pane	Action after typing on the message label	Result
Create a new argument for the Message If this is valid, the argument is added straight from the Messa ge label for the already-existing Operation.	Before was: *login (username=John)* Add the argument *1282*: *login (username=John, 1284)*	Exit the Message label edit mode by clicking the diagram pane.	A new argument is created and the Message is highlighted with yellow color; that is, the validation warning appears informing you that a parameter is missing and the Message argument is not synchronized with the operation parameter.  To resolve the validation warning: 1. Click the Message. The Message smart manipulator toolbar displays. 2. Click the button to synchronize the Message argument with the Operation parameter.
Update the argument	Before was: *login (John)* Now change the argument to: *login (Joseph)*	Exit the message label edit mode by clicking the diagram pane.	The argument is updated.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Notification message about incorrect syntax

If the Message syntax is incorrect, you will receive the following notification message at the bottom right corner of the MagicDraw window.



In the first line of the Message Parsing notification message, you will see an incorrect message name. In the example, the incorrect Message name is *getUser Address(city=)*. Due to the incorrect Message name, parsing cannot be executed and a new Operation cannot be created. In the second paragraph of the Message Parsing notification message, you will see the explanation of which syntax is expected. Next, you can see an example of the correct Message syntax.

Related pages

- Model Elements(see page 628)
- Specification Window(see page 913)
- Lifelines(see page 746)
- Interaction(see page 740)
- Operation(see page 677)
- Message(see page 756)

Assigning Signals to Messages

You can assign a Signal only to Send or Call (synchronous or asynchronous) Messages. Only one Signal can be assigned to a Message.

> ⓘ If a Signal is assigned to a Call Message, the Message is automatically converted to a Send Message.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To assign a Signal to a Message by dragging

1. Select a Signal in the Containment tree.
2. Drag the selected Signal to a desired Message.

To assign a Signal to a Message by typing Message name

- Click the Message to type the Message name on the diagram. A Signal is automatically created with the same name as Message name.

To assign a Signal to a Message from the Message Specification window

1. Draw a Send Message between two Lifelines or select an existing one on a diagram pane.
2. Open the Message Specification window.
3. Click the **Signature (signal)** property value cell and then select a Signal in the drop-down list.

> Names of signal attributes are displayed in the message signature by default. However, you can change this behavior by opening the **Symbol Properties** dialog (right-click a Message and select **Symbol Properties**) and setting the **Show Attribute/Parameter Names** property to **true** to display them or **false** to hide them.

Related pages

- Model Elements(see page 628)
- Specification Window(see page 913)
- Lifelines(see page 746)
- Interaction(see page 740)
- Operation(see page 677)
- Message(see page 756)

Creating Signal receptions for Messages

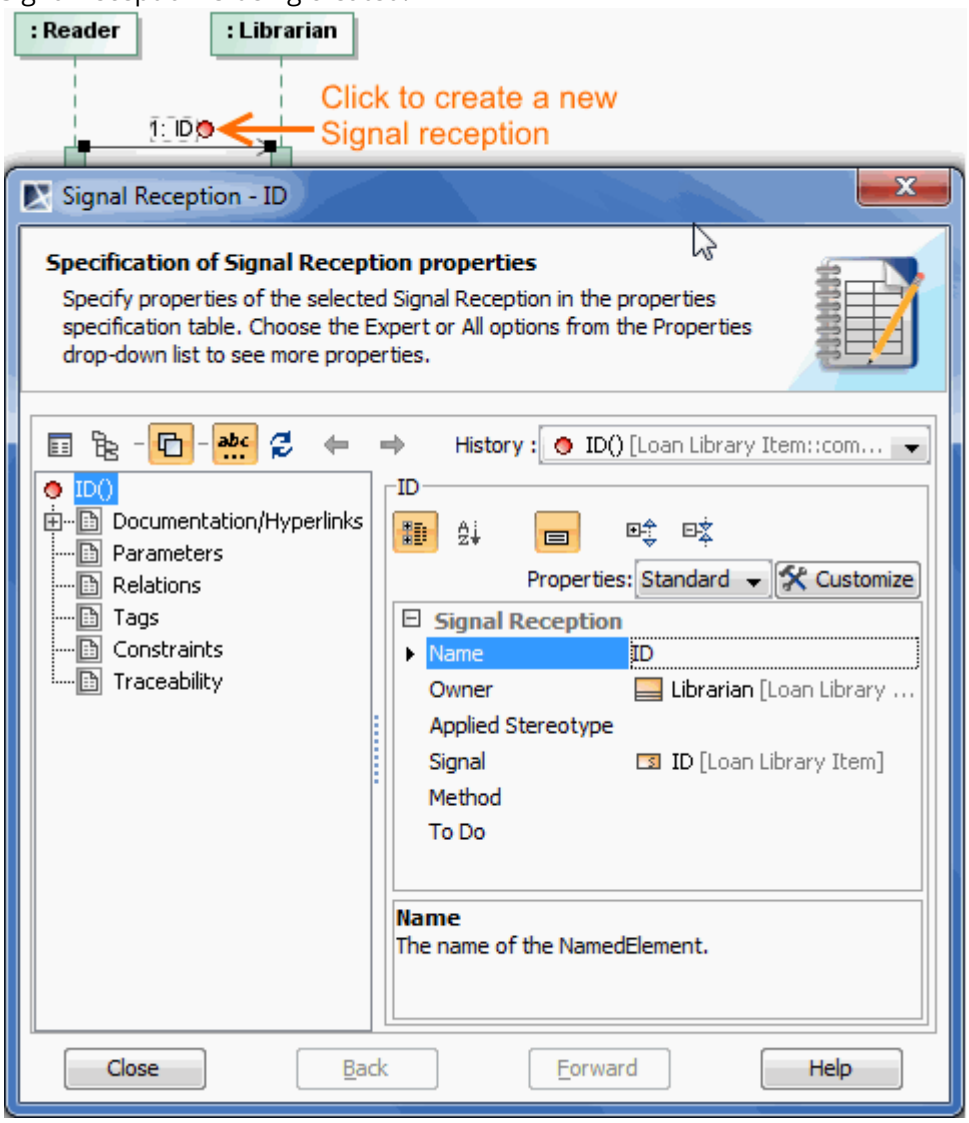
Assigning a Signal reception to a Message is very similar to the procedure of assigning an Operation to a Message(see page 758).

> These two conditions must be satisfied when creating a new Signal reception:
> - At least one Signal must exist in your project.
> - A possible Signal reception receiver (an activation to which the Message points) must have a type assigned.

To create a new signal reception for the message

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

1. Draw a send Message between the Lifelines(see page 746) or select an existing one on a diagram pane.
2. Assign a Signal to the Message(see page 768).
3. Do one of the following:
    - Click the button in the shape of a small red circle at the end of the Message name. Define the Signal reception property values in the Signal reception Specification window(see page 913). The default name of the Signal reception is the name of the Message for which the Signal reception is being created.



    - From the Message shortcut menu, select the **Create New Signal Reception** (the command with the small red circle icon, as shown above). Define Signal reception property values in the Signal reception Specification window(see page 913). The default name of the Signal reception is the name of the Message for which the Signal reception is being created.

Related pages

- Model Elements(see page 628)
- Specification Window(see page 913)

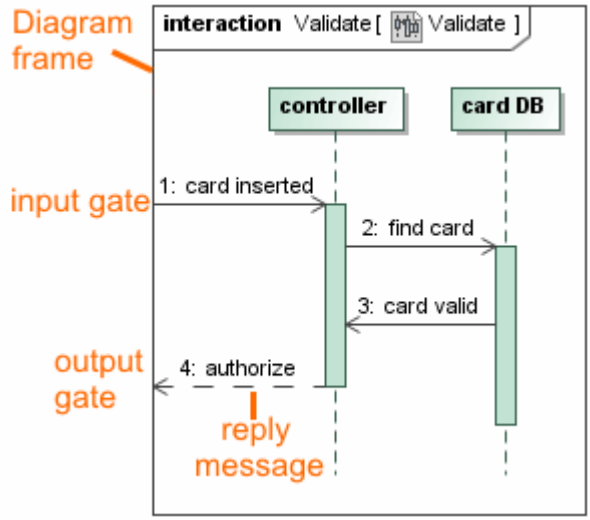
- Lifelines(see page 746)
- Interaction(see page 740)
- Operation(see page 677)
- Message(see page 756)

Messages in Sequence and Time Diagrams

Messages allow displaying an interaction between objects. A Message is labeled with either the message name or the assigned Operation (see page 677)(signal) name and its arguments.

Drawing Message from/to Lifeline to/from diagram frame

Outgoing and incoming messages communicate with the outside environment by connecting to the border gates (the diagram frame).



In the example above, you can see the Message *card inserted* that is drawn from the diagram frame border to the Lifeline *controller*. The Message *authorize* is created from the Lifeline *controller* to the diagram frame.

> For a Message to be correct, its receive end should be known. However, we never know who will connect to the gate from the outside. Thus, we are using the reply Message when creating the Message from the Lifeline to the diagram frame (see Message *authorize* in the figure).

Creating nested activation

Nested activations allow you to model:

- A parallel execution of operations that belong to a single Class(see page 666).
- Callback messages.

The nested activation can be created between at least two messages that point to the same activation.

Nested activations can be created for the following messages:

1. Synchronous Call Message (synchCall)
2. Asynchronous Call Message (asynchCall)
3. Asynchronous Signal Message (asynchSignal)

To create a nested activation

> ⚠ Ensure that you have at least two messages pointing to the same activation in your model.

1. Select any subsequent Message.
2. Do one of the following:
   - From the Message shortcut menu, select **Create Nested Activation**.
   - On the Message smart manipulator toolbar, click the Create Nested Activation button.

The Message will be connected to the nested activation.

To merge a nested activation with a parent activation

---

1. Select a Message that has a nested activation.
2. Do one of the following:
   - From the Message shortcut menu, select **Reduce Nesting Level**.
   - On the Message smart manipulator toolbar, click the Reduce Nested Activation button.

The Message will be connected to the parent activation.

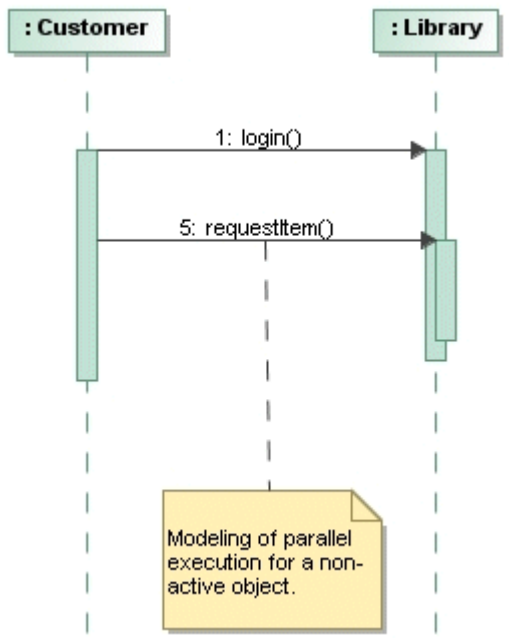
Nested activations can be used in the following cases:

- To model parallel executions for a non-active Lifeline(see page 772)
- To model parallel executions for an active Lifeline(see page 0)
- To model a callback Message(see page 773)

To model parallel executions for a non-active Lifeline

> ⚠ A non-active Lifeline is one that has a non-active Class as a type assigned. The non-active Class is the one whose Is Active property is set to false. This property is available in the Expert mode.

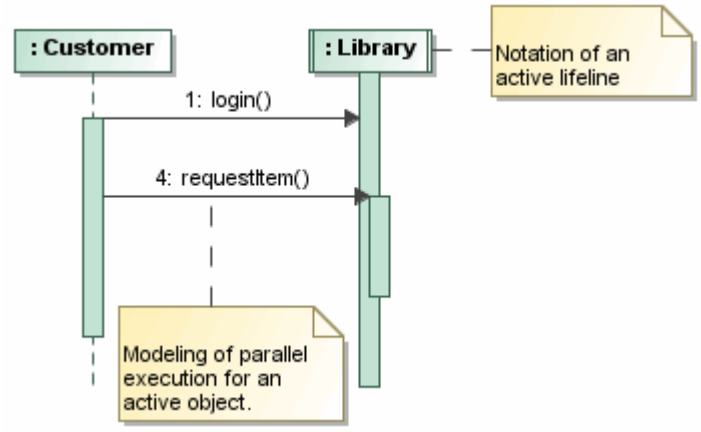
The figure below shows the creation of a simple nested activation, described in the procedure To create a nested activation(see page 772).



To model parallel executions for an active Lifeline

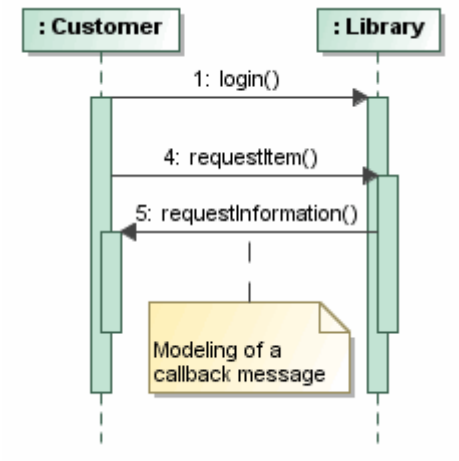
An active Lifeline is one that has an active Class as a type assigned. The active Class is the one whose Is Active property is set to true. This property is available in the Expert mode.

1. Select a Lifeline and from its shortcut menu, select **Show Entire Activation**. All activations of the selected Lifeline become all-in-one.
2. Create an outgoing Message or select an existing Message pointed to an active object. Be sure this is not the first one for the outgoing activation.
3. From the Message shortcut menu, select **Create Nested Activation**.



To model a callback Message

1. Create or select an incoming Message that is modeled as a callback Message.
2. From the Message shortcut menu, select **Create Nested Activation**.



Related pages

- Model Elements(see page 628)
- Specification Window(see page 913)
- Lifelines(see page 746)
- Interaction(see page 740)
- Operation(see page 677)
- Message(see page 756)

Messages in Communication diagrams

Messages in Communication diagrams(see page 302) can only be depicted on connectors; thus, you should draw a connector before assigning a Message to it.

The following steps show you how to work with messages:

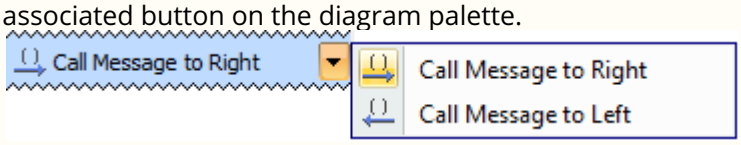
- Assigning a message to a connector(see page 774)
- Setting/removing the advanced numbering of messages(see page 775)
- Changing current message numbering(see page 775)
- Displaying messages(see page 776)

Assigning a message to a connector

To assign a message to a connector

1. On the diagram palette, click the button corresponding to the desired Message type.

2. Click a desired connector on the diagram pane. A Message arrow will be placed on the selected connector.

> A Message flow has two directions: right and left. Choose one of them by clicking the associated button on the diagram palette.
>
> 
>

Setting/removing the advanced numbering of messages

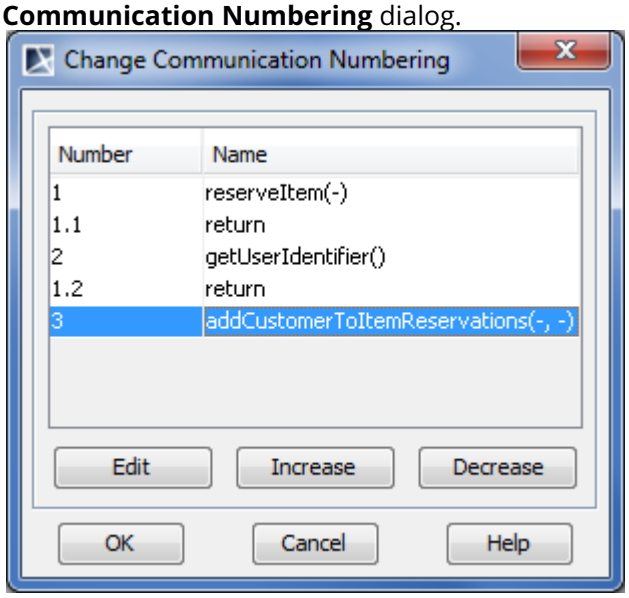
To set/remove the advanced numbering of messages

1. From the diagram shortcut menu, select **Numbering**.
2. Select to set, or clear to remove the **Use Advanced Numbering**.

Changing current message numbering

To change current message numbering

1. From the diagram shortcut menu, select **Numbering** > **Change Numbering**.
2. Increase, decrease, and / or change the level of numbering in the **Change Communication Numbering** dialog.

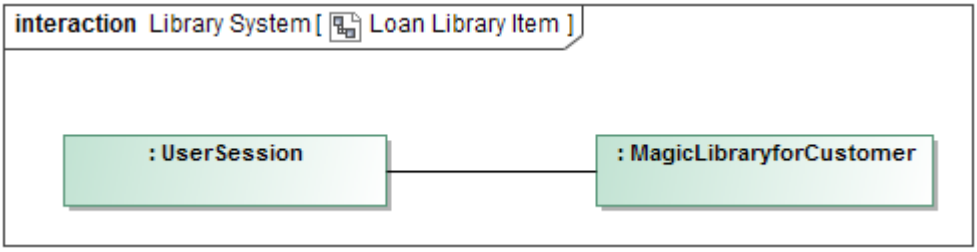


> If the **Edit** button is inactive, remove the automatic advanced numbering(see page 775) of Messages.

Displaying messages

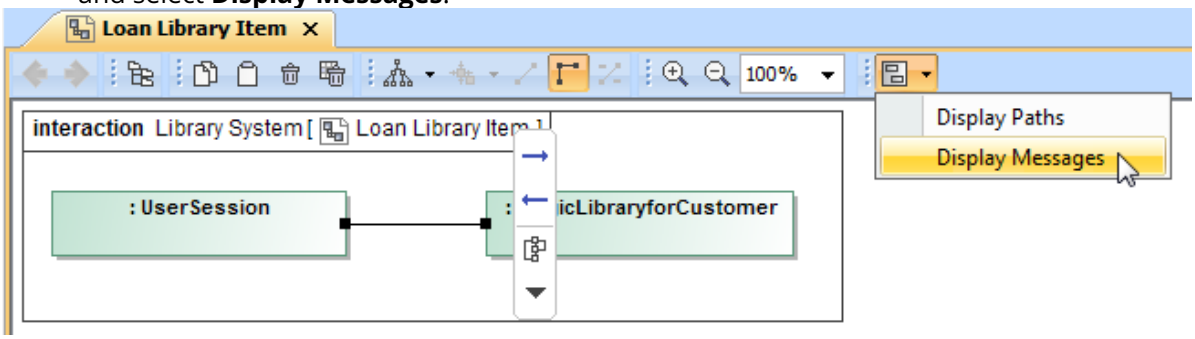
If you want to display messages on connectors in the Communication diagram you must first implement the following conditions (see the figure below):

- Messages are assigned to connectors. How to assign a message to connector >(see page 774)
- The lifeline shapes are already displayed on the diagram pane. How to display lifelines >(see page 746)
- The paths between lifelines are displayed on the diagram pane. How to display paths >(see page 513)



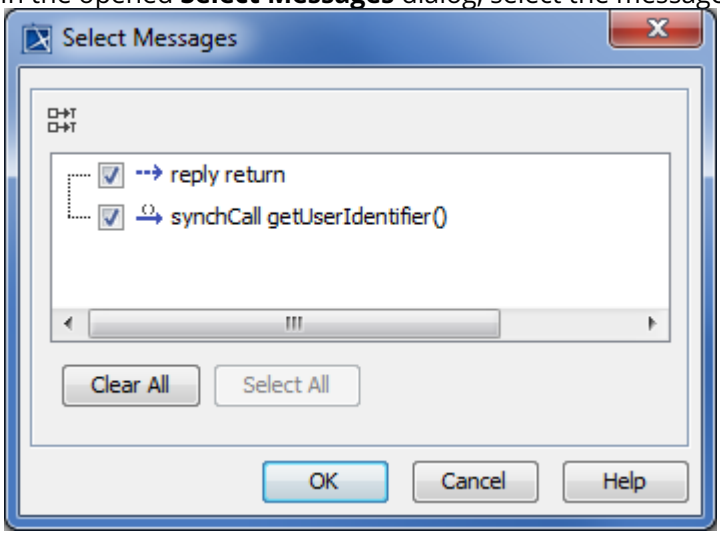
To display messages

1. Select the connector between lifelines.
2. Perform one of the following actions:
   - Right-click the connector to open the shortcut menu, and select **Display** > **Display Messages**.
   - On the diagram toolbar, click , and select **Display Messages**.

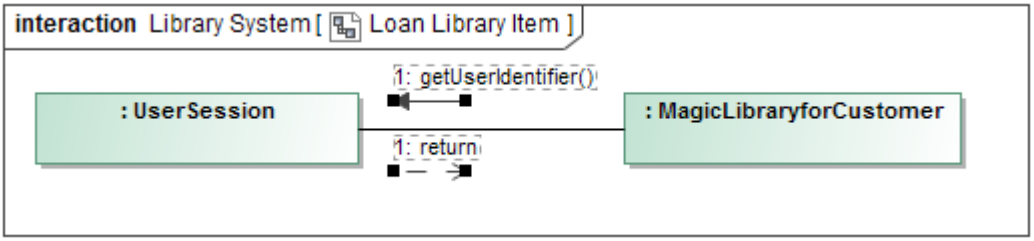


Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

3. In the opened **Select Messages** dialog, select the messages you want to display, and click **OK**.



Messages of the selected connector are displayed on the diagram pane.



Related pages

- Model Elements(see page 628)
- Specification Window(see page 913)
- Lifelines(see page 746)
- Interaction(see page 740)
- Operation(see page 677)
- Message(see page 756)

Sample model

The model used in the figures of this page is the **Communication diagram** sample model that comes with MagicDraw. To open this sample do one of the following:

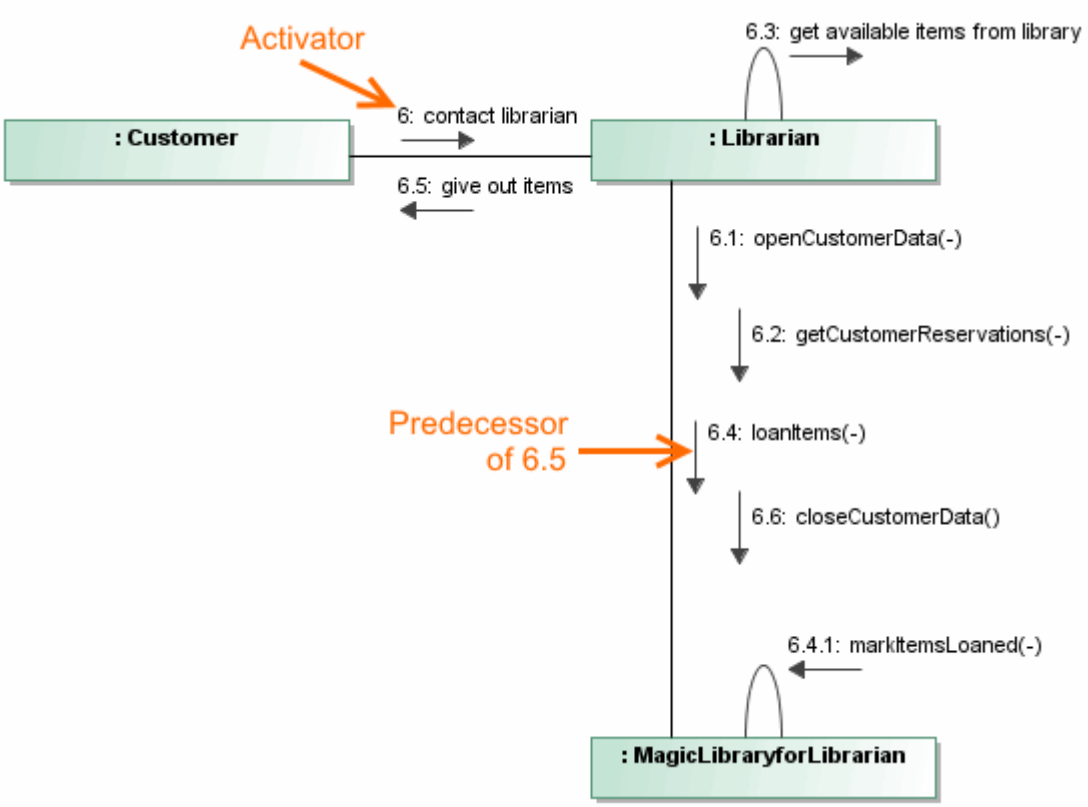
- Download communication diagram.mdzip[151].
- Find in modeling tool *<modeling tool installation directory>\samples\diagrams\communication diagram.mdzip.*

Activators and predecessors

15 https://docs.nomagic.com/download/attachments/55847477/communication%20diagram.mdzip?api=v2&modificationDate=1479988632454&version=1

The predecessor denotes the set of Messages. The completion of these Messages enables the execution of the current Message. The predecessor means that the execution of a Message is not enabled until all the communications with sequence numbers appearing in the list have occurred. Therefore, the list of predecessors represents a synchronization of threads. The Message corresponding to the numerically preceding sequence number is an implicit predecessor and does not need to be explicitly listed.

All of the sequence numbers with the same prefix form a sequence. The numerical predecessor is the one in which the final term is one less. That is, number 6.4 is the predecessor of 6.5, where the number "6" is an activator (see the example in the following figure).

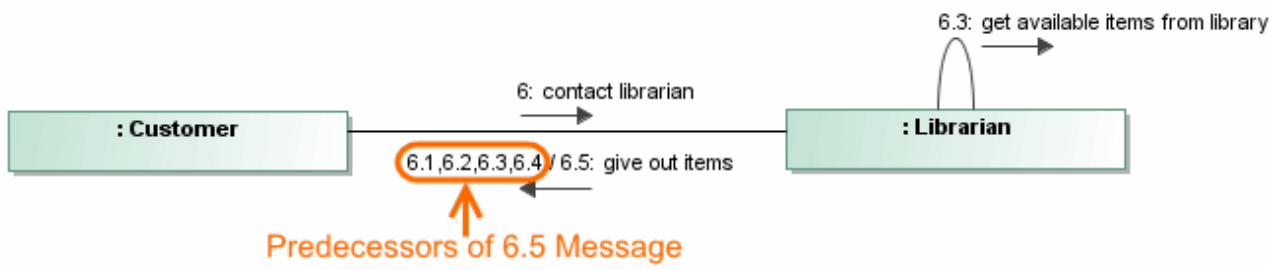


To show predecessors beside the Message number

Make sure the **Use Advanced Numbering** is selected(see page 774).

- From the Message shortcut menu, select **Show Predecessors**.
- In the Message **Symbol Properties** dialog(see page 975), set **Show Predecessors** to *true*.

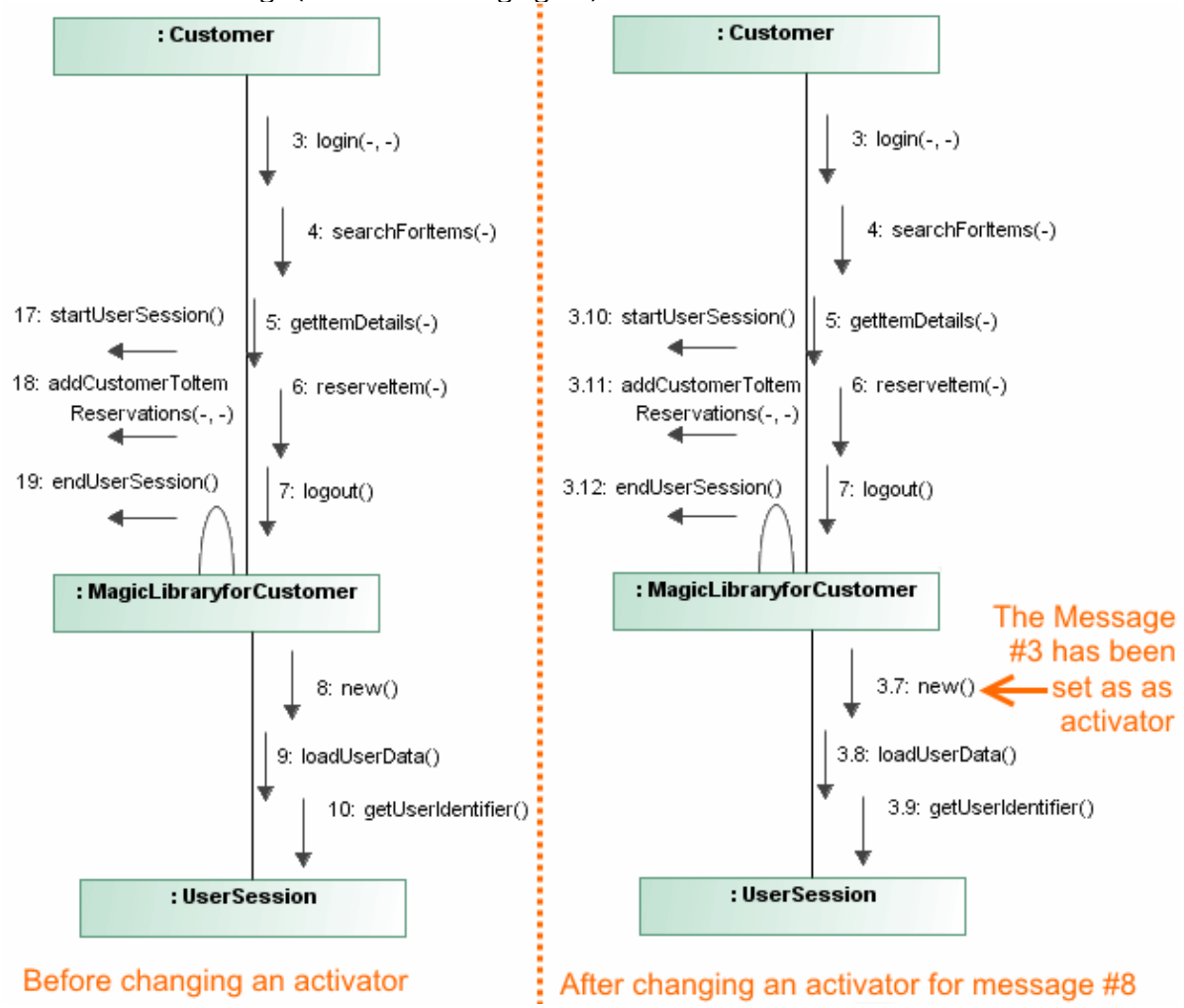
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



To change an activator number of Messages

Make sure the **Use Advanced Numbering** is selected(see page 774).

1. From the Message shortcut menu, select **Activator**.
2. Select the activator you want to assign to the selected Message. Numbering of the selected Message and subsequent Messages decreases by one level. The first level number is the number of an activator Message (see the following figure).



If you change the activator number to a predecessor Message, this number will change for subsequent predecessors.

Related pages

- Model Elements(see page 628)
- Specification Window(see page 913)
- Lifelines(see page 746)
- Interaction(see page 740)
- Operation(see page 677)
- Message(see page 756)

Message Occurrence Specification

A Message Occurrence Specification specifies the occurrence of message events, such as sending and receiving signals or invoking or receiving operation calls. A Message Occurrence Specification is a kind of Messages(see page 2688) end. Messages are generated either by synchronous operation calls or asynchronous signal sends. They are received by the execution of the corresponding accept event actions.
In the Message Specification window(see page 913), you can modify an element and add various specifications to it.

Metaclass

The abstract syntax of UML is specified using a UML model called the UML metamodel. Classes in a metamodel are called metaclasses. The detailed descriptions of each metaclass are provided in the UML specification.pdf[152].

The metaclass icon in the Model Browser(see page 29) is . Its shape is displayed with <metaclass> stereotype.

All metaclasses in our modeling tools are stored in the **Auxiliary Resources**, expand the **UML Standard Profile** Package > **UML 2 Metamodel** Package.

To show the auxiliary resources

- In the Model Browser(see page 29), click and enable the **Show Auxiliary Resources** option.

Model

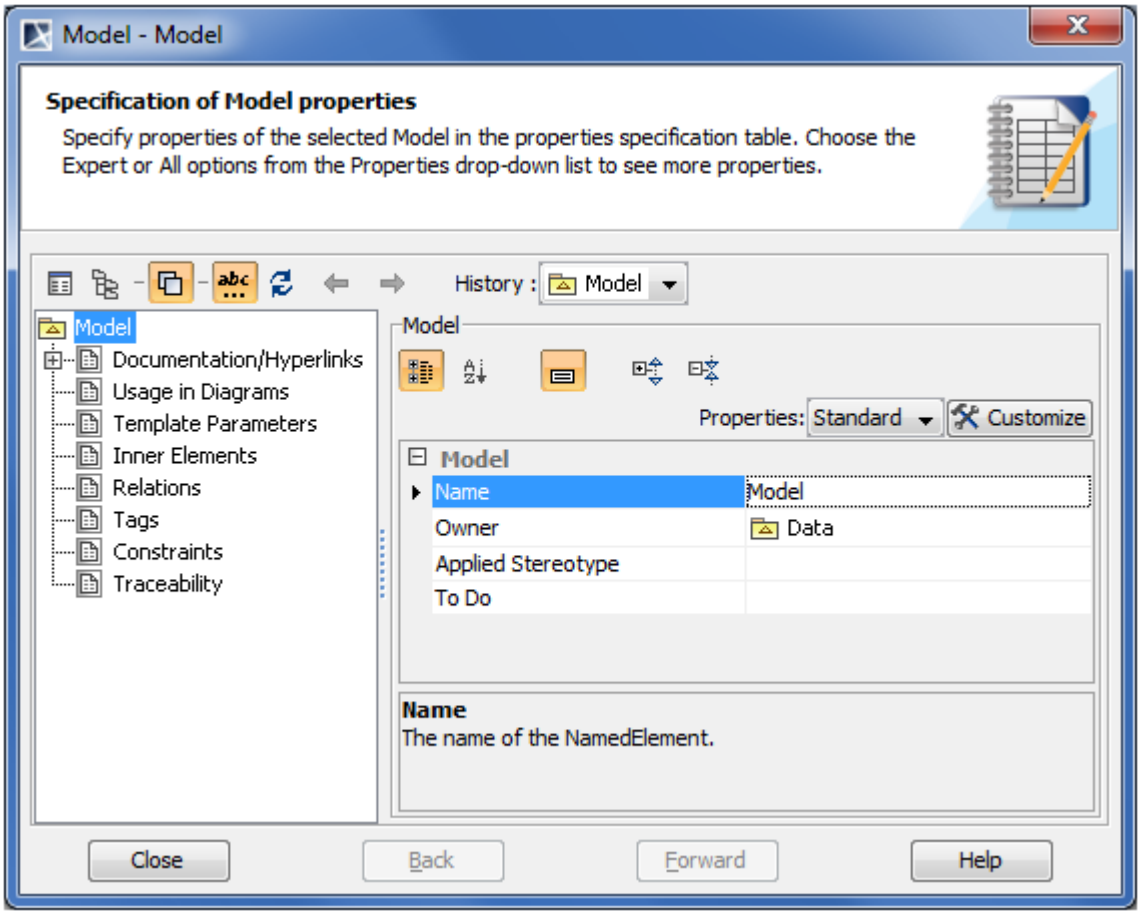
A model contains a (hierarchical) set of elements that describes the physical system being modeled. It can also contain a set of elements that represent the environment of the system (typically Actors (see

25 https://docs.nomagic.com/download/attachments/55843078/UML%20specification.pdf?api=v2&modificationDate=1504698722367&version=1

page 646)together with their interrelationships), such as Associations (see page 651)and Dependencies(see page 699).

A model is presented as a package (see page 800)with a small triangle in the upper right corner of the large rectangle. The triangle can be shown in the tab.

The model is defined as a package; that is, it has package properties in the Model Specification window(see page 913), as shown below.



Related Pages

- Object diagram(see page 299)
- Class diagram(see page 297)
- Dependency(see page 699)
- Actors(see page 646)
- Package(see page 800)
- Associations(see page 651)
- Model Elements(see page 628)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Node

Any computer or device that is relevant to an implemented system can be shown as a Node. A Node symbol is a three-dimensional square with a name on it. Devices in a system are typically represented with a stereotype that specifies the device type. The nodes can be represented as types and as instances.

You can specify Node properties in the Node **Specification** window, where you can also read descriptions of each property. Descriptions are presented in the description area of the **Specification** window.

Related References

Related Pages

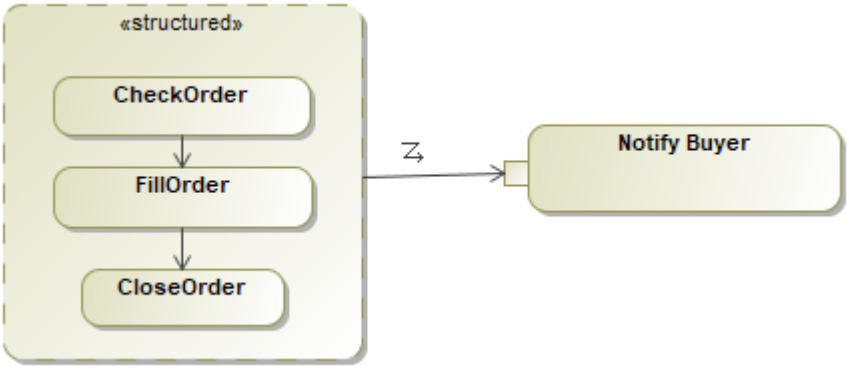
- Model Elements(see page 628)
- Specification Window(see page 913)

Structured Activity Node

A Structured activity node is an executable activity node that may have an expansion into subordinate nodes as an activity group. The subordinate nodes must belong to only one structured activity node, although they may be nested.

Specify the Structured Activity Node element in the Structured Activity Node Specification window(see page 913).

To open the Structured Activity Node dialog box, in the Activity diagram draw a Structured Activity Node element. From its shortcut menu, choose **Specification**.



A variable defined in the scope of the structured activity node. It has no value and may not be accessed outside the node.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

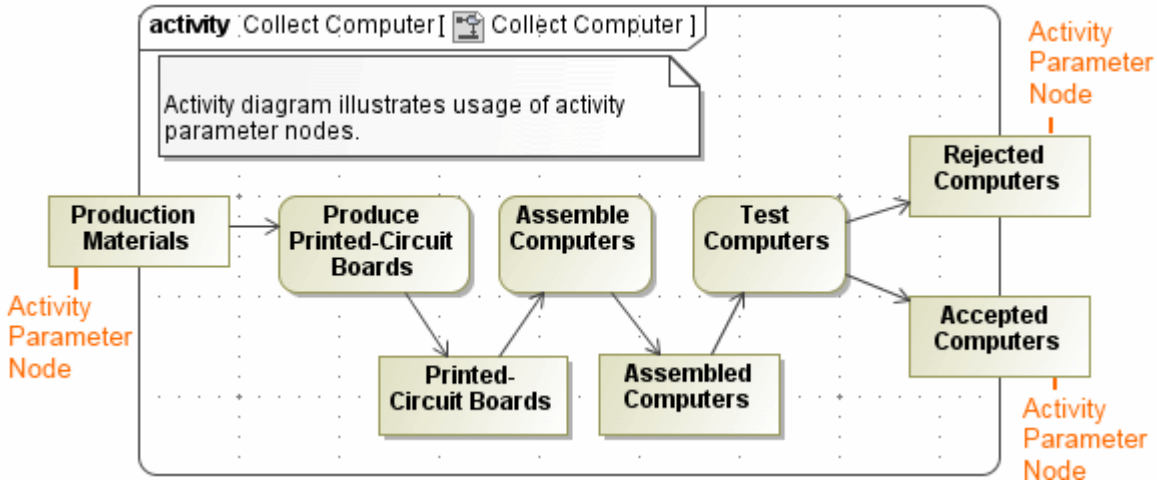
Activity Parameter Node

On this page

- Description(see page 783)
- Creating an Activity Parameter Node(see page 783)
- Specifying Activity Parameter Node type(see page 784)
- Setting the direction of an Activity Parameter Node(see page 785)
- Displaying Activity Parameter Nodes(see page 785)
- Displaying or hiding the direction of an Activity Parameter Node(see page 788)

Description

Activity Parameter Nodes are object nodes at the beginning and end of the flows that are used to accept inputs and provide outputs from activities.



Creating an Activity Parameter Node

To create an Activity Parameter Node

---

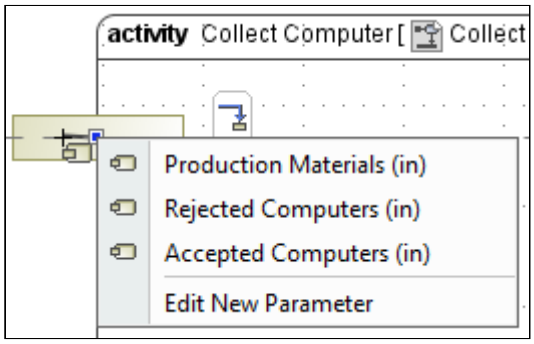
1. In the Activity diagram palette, click the arrow next to the **Object Node** button to expand the list, then select **Activity Parameter Node**.
2. Click the Activity diagram frame. The Activity Parameter Node is created, and its shortcut menu opens.
3. Select **Edit New Parameter** to specify the parameter. The **Parameter** dialog opens.
4. Specify the parameter, and click **Close.**

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

The parameter for the Activity Parameter Node is created automatically.
The parameter name, type, and multiplicity are synchronized with the pin or Object Node from which/ to which the Object Flow was created. The direction of the parameter is set according to the Object Flow direction, otherwise you can set it manually by simply right-clicking the activity parameter, and selecting the preferred direction.

When creating an Activity Parameter Node on a diagram frame, click on the diagram frame to open the menu. In the menu, the following Activity Parameter Nodes are listed:

- parameters that are not displayed on the diagram frame.
- parameters that are owned by the same activity as the activity diagram.
- parameters that have assigned parameters that are also owned by the same activity.



If a parameter with an *inout* direction is selected for the Activity Parameter Node creation, then two Activity Parameter Nodes are created on the diagram frame.

> ⓘ You can also create an Activity Parameter Node by drawing an Object Flow directly to or from the Activity diagram frame.

Specifying Activity Parameter Node type

After the Activity Parameter Node is created, you can specify it's type. Use one of the following way:

- Using the **Specify Type** button.(see page 784)
- Using the Specification window.(see page 785)

To specify Activity Parameter Node type by using the **Specify Type** button

---

1. Select the Activity Parameter Node shape.
2. Click T on the smart manipulator toolbar(see page 568).
3. Do one of the following:
   - Select an existing element from the **Select Type** menu.
   - Create new type:
     a. Type a name.
     b. Press Enter.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

c. In the **Create <type name> As** dialog, select element kind.
The type is specified for a Activity Parameter Node.

To specify Activity Parameter Node type by using the Specification window

---

1. Right-click the Activity Parameter Node and select the **Specification** command.
2. In the Specification window(see page 913), select the **Type** property value cell.
3. Do one of the following:
   - Select an existing element from the list.
   - Click ... and in the **Select Element** dialog do the following:
     - Select an existing elements from the **Tree** or **List** tab.
     - create new element by switching on the **Creation Mode** and clicking the **Create** button.
4. Click **Close**.
   The type is specified for a Activity Parameter Node.

Setting the direction of an Activity Parameter Node

To set the direction of an Activity Parameter Node

---

1. Right-click the Activity Parameter Node on the diagram, and select **Specification**.
2. In the **Parameter** property group, click to expand the **Direction** options drop-down menu.
3. Select the preferred direction, and click **Close**.

Displaying Activity Parameter Nodes

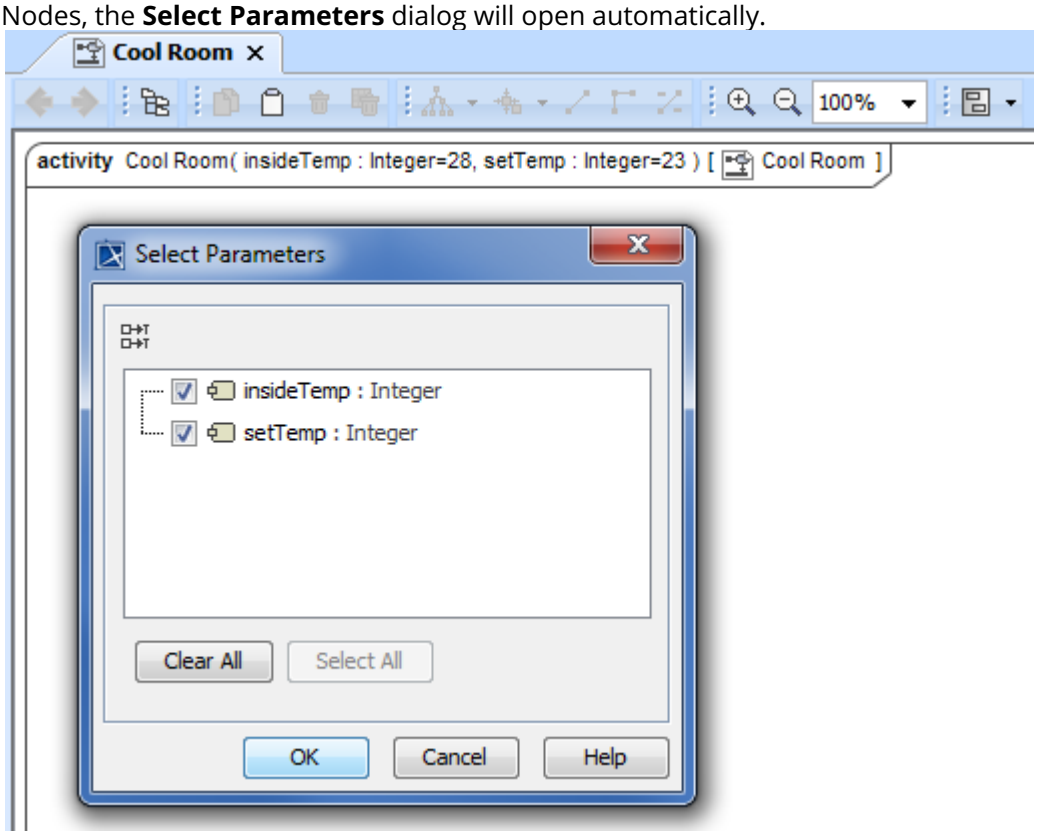
You can display Activity Parameter Nodes on the frame of the Activity diagram(see page 311) when you are:

- Creating a new Activity diagram(see page 785).
- Modifying a pre-existing Activity diagram(see page 786).

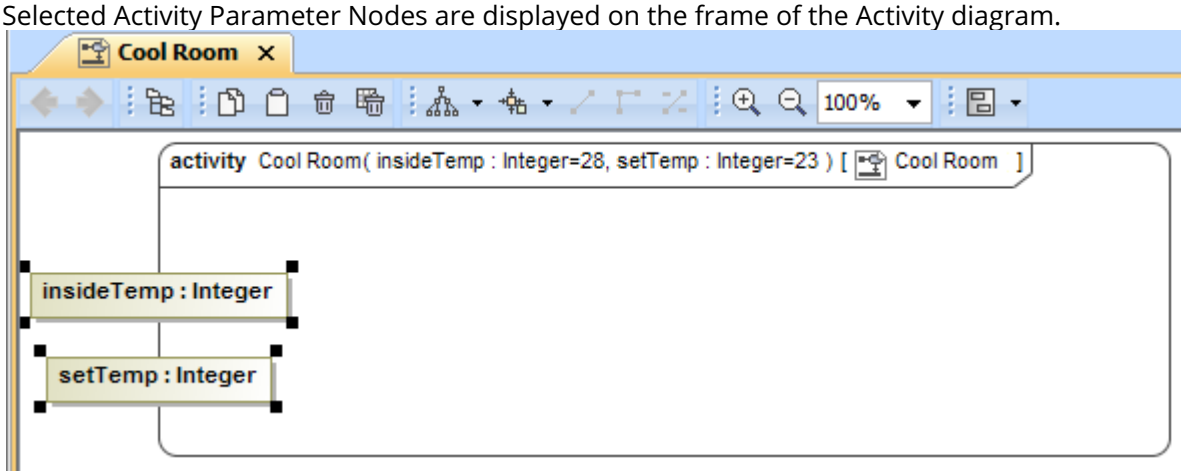
To display an Activity Parameter Nodes when creating a new Activity diagram

---

1. When creating a new Activity diagram for an Activity element that contains Activity Parameter Nodes, the **Select Parameters** dialog will open automatically.



2. In the opened **Select Parameters** dialog, select the parameters that you want to display, and click **OK**.
Selected Activity Parameter Nodes are displayed on the frame of the Activity diagram.



> If you want all Activity Parameter Nodes to be displayed automatically when creating a new Activity diagram, go to **Options** > **Project** on the main menu, and set the **Display All Activity Parameter Nodes in Diagrams** option to *true.*

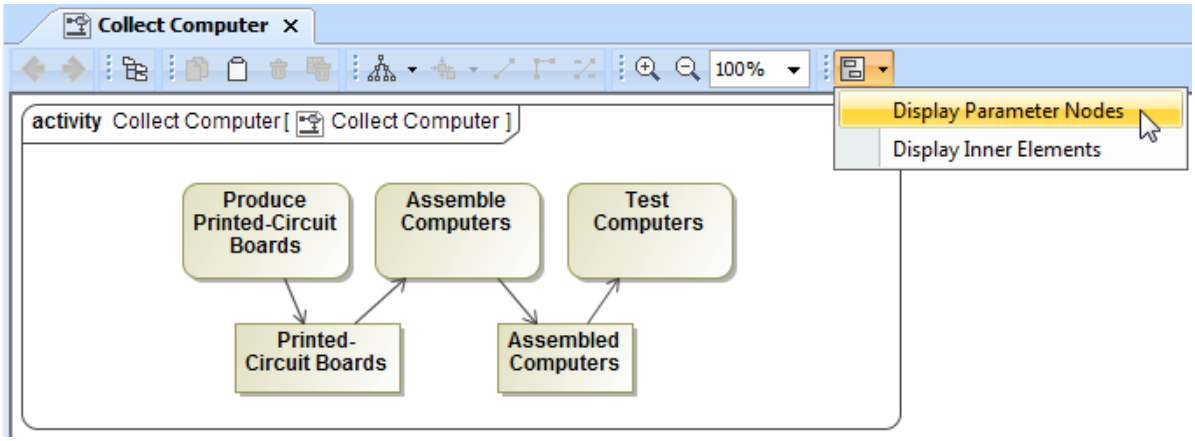
To display Activity Parameter Nodes on the frame in the pre-existing Activity diagram

---

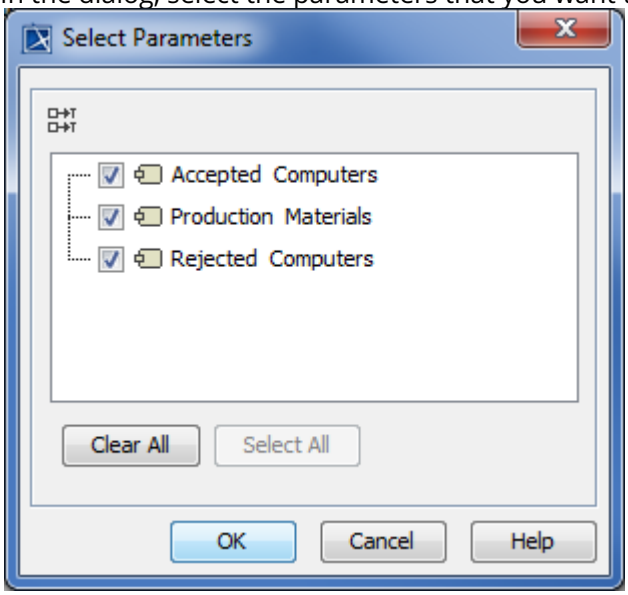
1. Select the Activity diagram pane.

2. Open the **Select Parameters** dialog by performing one of the following steps:
   - From the shortcut menu, select **Display** > **Display Parameter Nodes**.
   - On the diagram toolbar, click

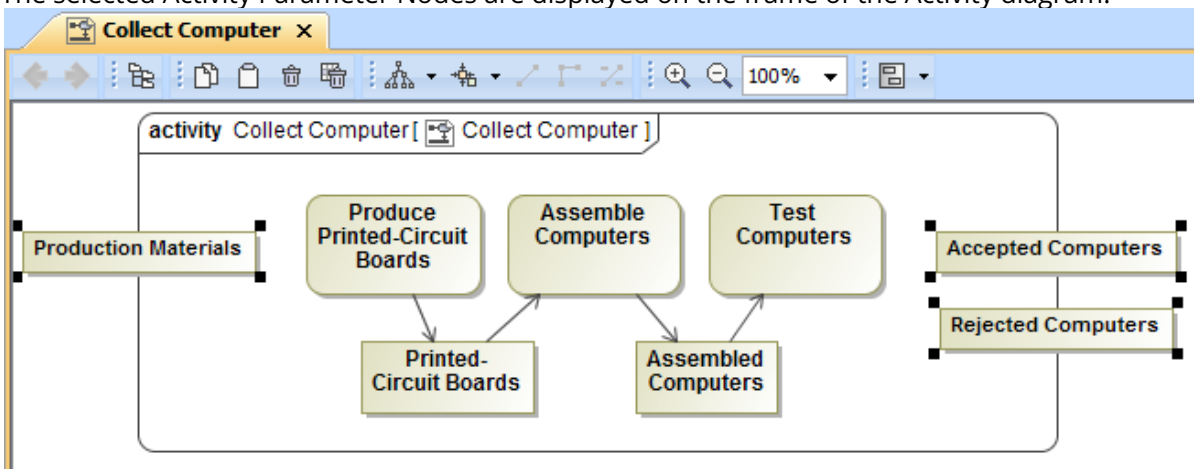
   and select **Display Parameter Nodes.**



3. In the dialog, select the parameters that you want to display, and click **OK**.



The selected Activity Parameter Nodes are displayed on the frame of the Activity diagram.

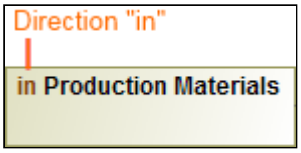


Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Displaying or hiding the direction of an Activity Parameter Node

To display or hide the direction of an Activity Parameter Node

1. Right-click the Activity Parameter Node on the diagram, and select **Symbol Properties**.
2. In the **Symbol Properties** dialog, click to select or clear the **Show Direction** check box to display or hide the direction.
3. Click **Close**.



- You can see the direction of the parameter displayed in the **Containment** tree before choosing the parameter name.
- If an Activity Parameter Node is associated with the parameter of an *inout* direction, then the direction to the Activity Parameter Node is set according to the connected flows. If an outgoing flow is connected to the newly created Activity Parameter Node, then this Activity Parameter Node is set as an input node, and the 'in' parameter direction is used.

Related pages

- Activity diagram(see page 311)
- Node(see page 782)
- Expansion Region and Expansion Nodes(see page 788)
- If, Loop and Sequence Conditional Nodes(see page 789)
- Structured Activity Node(see page 782)
- Model Elements(see page 628)

Sample model

The model used in the figures of this page is the **Activity diagram** sample model that comes with MagicDraw. To open this sample do one of the following:

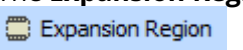
- Download activity diagram.mdzip[153].
- Find in modeling tool *<modeling tool installation directory>\samples\diagrams\activity diagram.mdzip.*

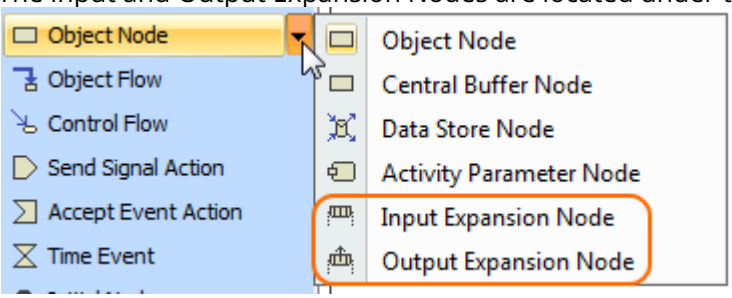
Expansion Region and Expansion Nodes

The Expansion Region and Expansion Nodes can be drawn in the Activity diagram(see page 311).

You can create those elements using the diagram palette(see page 24):

[153] https://docs.nomagic.com/download/attachments/55847477/activity%20diagram.mdzip?api=v2&modificationDate=1479890703683&version=1

- The **Expansion Region** button.
  

- The Input and Output Expansion Nodes are located under the **Object Node** button group.
  

To display Expansion Nodes on the Expansion Region border:

---

1. Select the Expansion Region shape of the Expansion Nodes you want to display.
2. Perform one of the following steps:
   - From the shortcut menu, select **Display** > **Display Expansion Nodes**.
   - On the diagram toolbar, click

   and select **Display Expansion Nodes.**

Related Pages

- Model Elements(see page 628)
- Specification Window(see page 913)

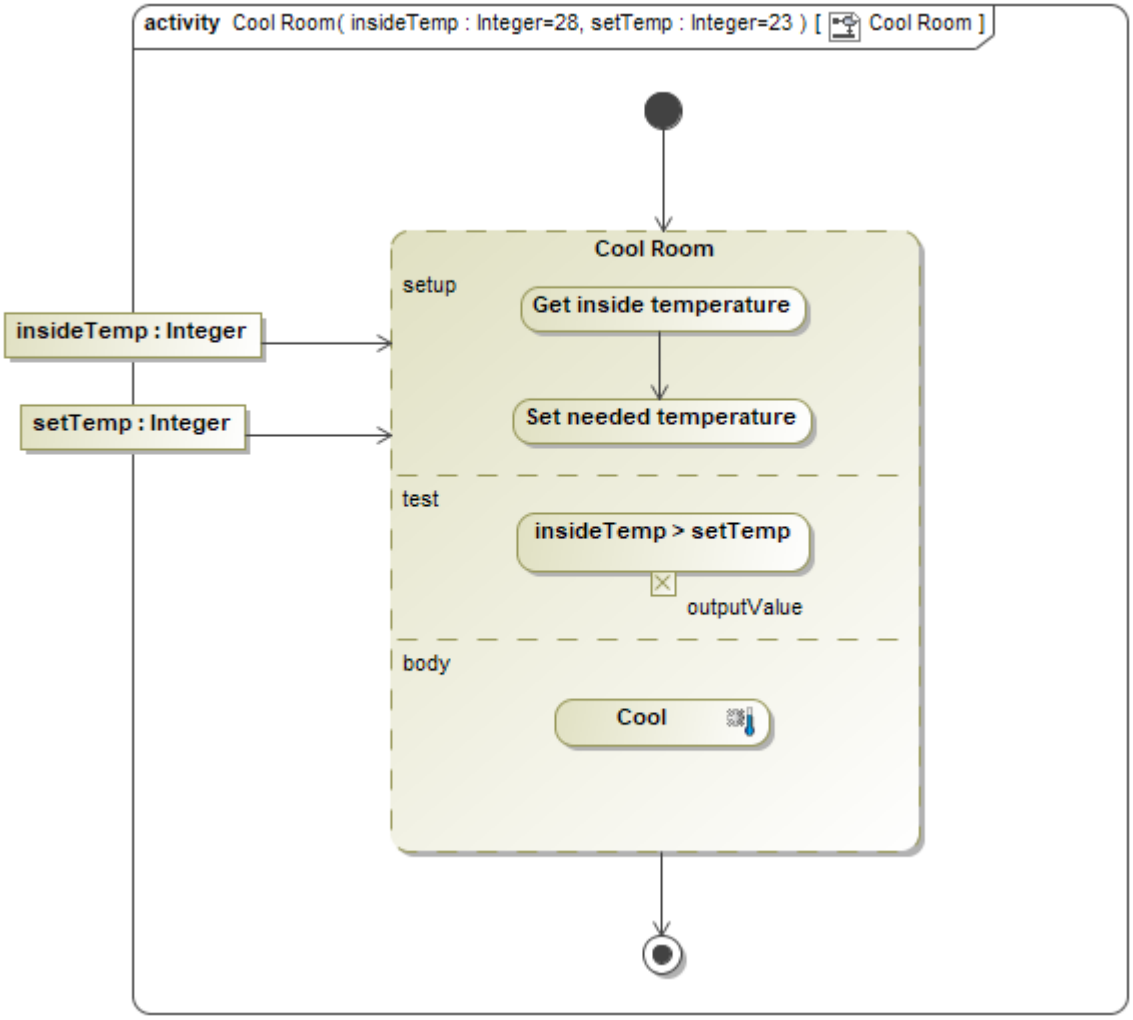
If, Loop and Sequence Conditional Nodes

Loop Node

A loop node is a constructed activity node representing a loop with setup, test, and body sections.

Each section is a well-nested subregion of an activity whose nodes follow any predecessors of the loop and precede any successors of the loop. The test section may precede or follow the body section. The setup section is executed once on entry to the loop, and the test and body sections are executed repeatedly until the test produces a false value. The results of the final execution of the test or body are available after execution of the loop is completed.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Sequence Node

A sequence node is a structured activity node that executes its actions in order.

When the sequence node is enabled, its executable nodes are executed in the order specified. When combined with flows, actions must also satisfy their control and data flow inputs before starting execution.

Conditional Node

A conditional node is a structured activity node representing an exclusive choice among a number of alternatives.

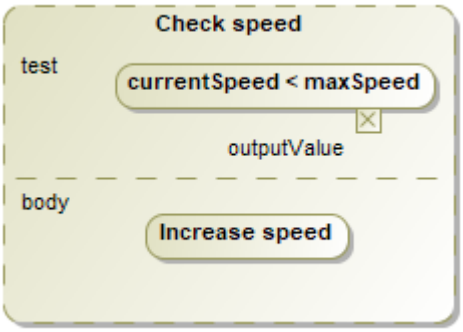
A conditional node consists of one or more clauses, each consisting of a test section and a body section. When the conditional node begins execution, the test sections of the clauses are executed.

- If one or more test sections yield a true value, one of the corresponding body sections will be executed.
- If more than one test section yields a true value, only one body section will be executed.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

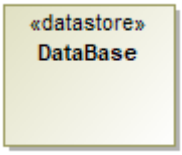
- The choice is non-deterministic, unless the test sequence of clauses is specified. If no test section yields a true value, then no body section is executed; this may be a semantic error if output values are expected from the conditional node.

In general, a test section may be executed in any order, including simultaneously if the underlying execution architecture supports it. The result may therefore be nondeterministic if more than one test section can be true concurrently. To enforce ordering of an evaluation, sequencing constraints may be specified among clauses.



Data store node

A data store keeps all tokens that enter it, copying them when they are chosen to move downstream. Incoming tokens containing a particular object replace any tokens in the object node containing that object.



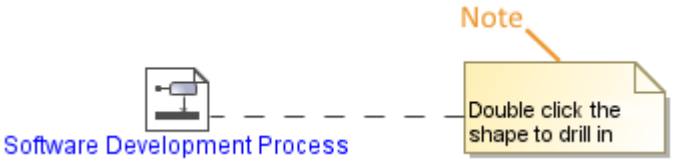
You can format Data store node symbol properties in the Symbol Properties dialog(see page 975).

You can specify a Data store node by changing its property values in the Data store node Specification window(see page 913). Each property is described in the description area on this window.

Note

A note is a graphical symbol containing textual information. It is used to add any explanatory information needed for your element or diagram. A note is usually connected to the element symbol using an anchor line, as shown below.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



In the note, you can also display and edit the elements the note is anchored to, such as documentation, constraint, and tagged value, as well as element properties.

Text in the note can be displayed as plain or HTML text.

You can format the note symbol properties in the Symbol Properties dialog(see page 975).

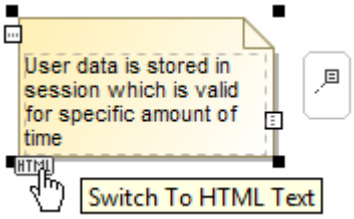
The Note anchored to the diagram frame(see page 523) can represent the information of the diagram context element (for example, Activity(see page 643) or State Machine(see page 842)).

To switch the representation to the diagram information

1. Right-click the Note.
2. In the shortcut menu clear the selection of **Represent Diagram Context**.

To switch the note text between HTML and Plain text modes

- On the diagram pane, select the note symbol and click either HTML or Plain button.



You can edit the HTML text in the note using the HTML editor(see page 1055).

To display the element documentation on a note

1. Draw a note to an element whose documentation you want to display.
2. From the note shortcut menu, select **Text Display Mode** > **Show Documentation**.

> ⓘ You can edit element documentation directly on the note symbol.

To display element properties, constraints, and/or tagged values on a note symbol

To see compartment editing results, make sure element properties, constraints, and/or tagged values are displayed on the note symbol.

1. Draw a note to an element whose documentation you want to display.
2. Select a note and click the Compartments button on the left.
3. In the list, click to select the following:
    - **Element Properties**
    - **Constraints**
    - **Tagged Values**

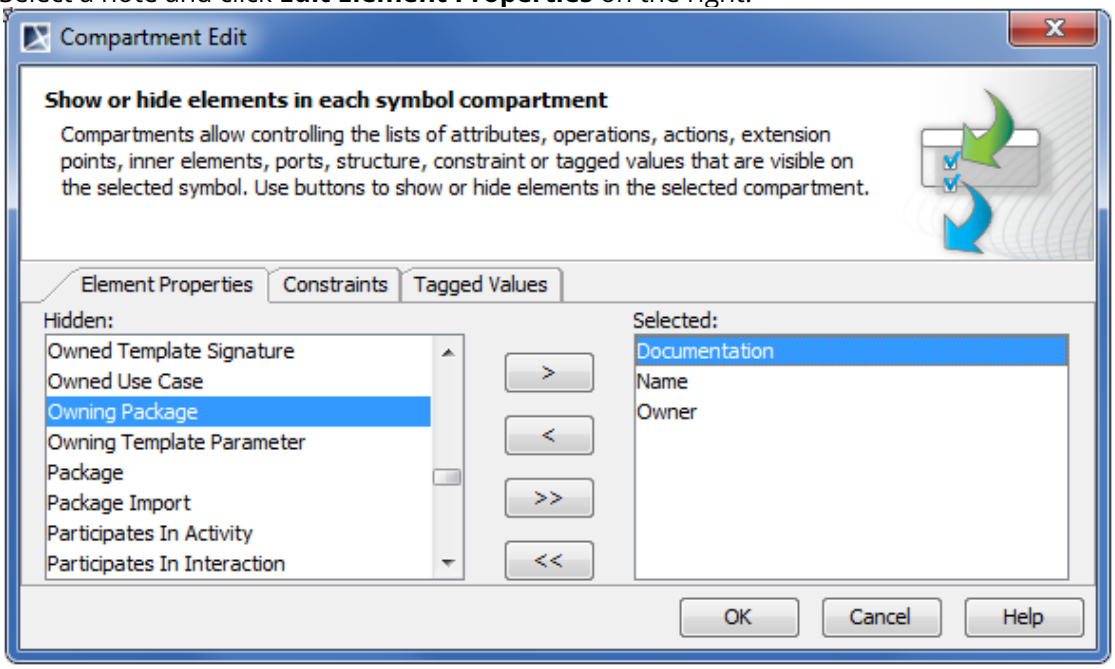
Element properties, constraints, and / or tagged values are displayed on the note symbol in the separate compartments.

You can choose which element properties, constraints, or tagged values to display on the note symbol by editing an appropriate compartment.

To edit a compartment on a note symbol

---

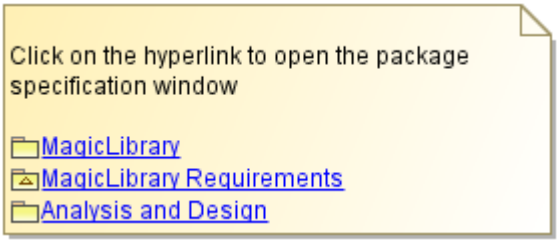
1. Open the **Compartment Edit** dialog by doing one of the following:
    - From the note shortcut menu, select **Edit Compartments**.
    - Select a note and click **Edit Element Properties** on the right.



2. Move elements between Hidden and Selected lists using the **>**, **<**, **>>**, or << **buttons**. Only items from the selected list will be displayed in the compartment on the note's symbol.
3. Click **OK** after you are done.

To add a hyperlink to an element on a note symbol

- Select one or more elements in the **Containment tree** and drag them to the note symbol on the diagram pane.



After the hyperlinks on the note are created, the text format of the note changes to HTML.

Realted pages

- Model Elements(see page 628)
- Specification Window(see page 913)
- Diagram frame(see page 523)

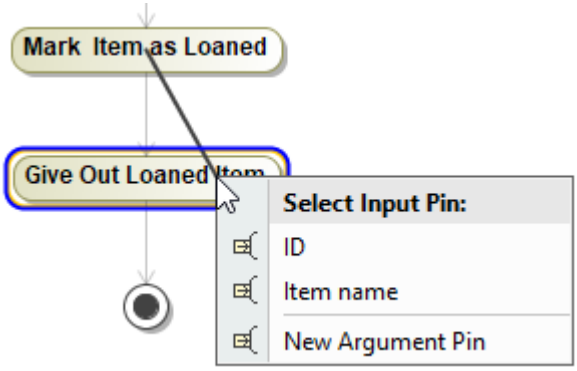
Object Flow

You can view Object Flows that are connected via Pins in the Action's Specification window(see page 913), as well as the **Display Related Elements** and **Display Paths** commands are available from the Action's shortcut menu(see page 794).

To select output/input pin automatically when drawing an Object Flow

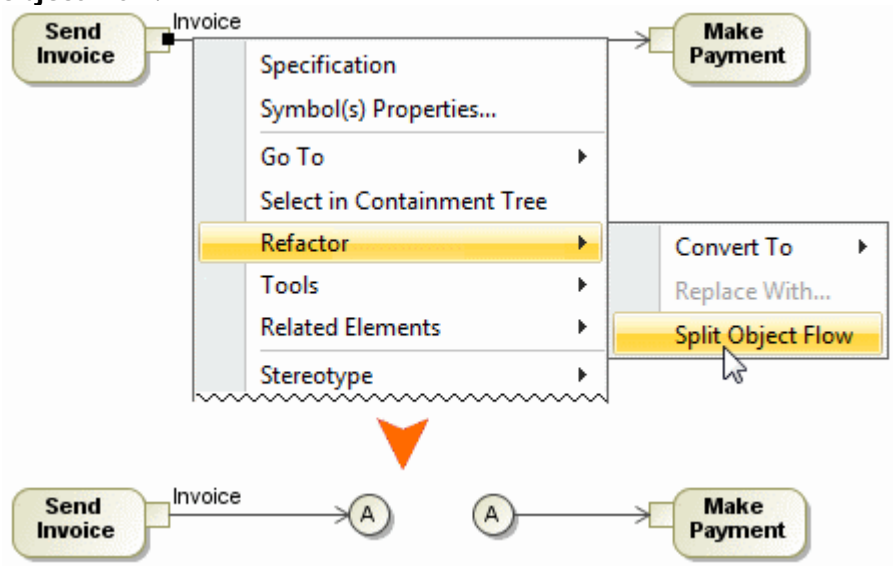
- When drawing an Object Flow, our modeling tool automatically selects Output or Input Pins, creates a new Pin, under certain conditions, or you can choose a Pin from the list of options. For example, a list of available options to select input Pins opens when drawing Object Flow to the Action that has more than one hidden Input Pin.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



To split Object Flows

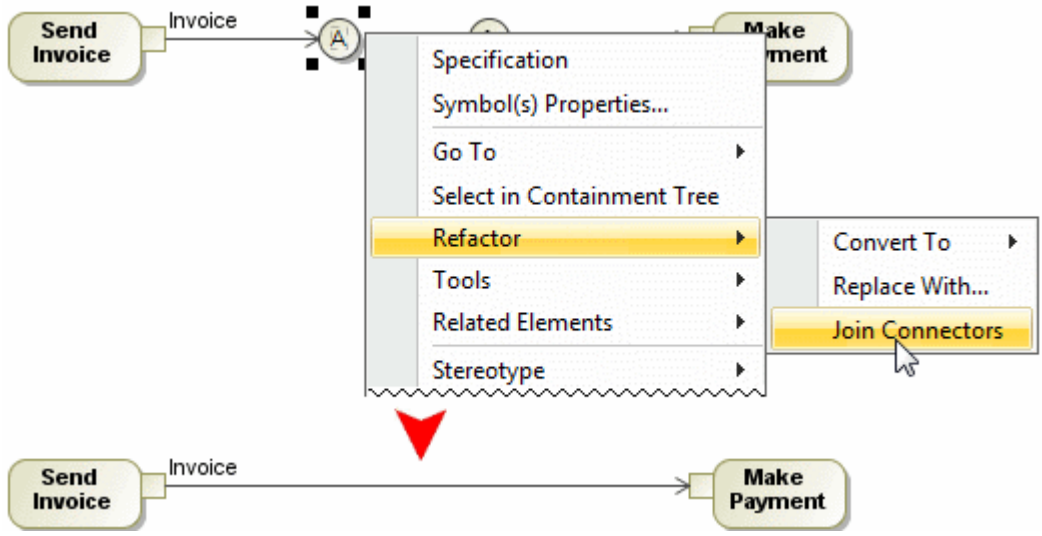
- From an Object Flow shortcut menu on a diagram pane, select **Refactor** and then select **Split Object Flow**.



- The **Split Object Flow** command splits Object Flow into two connectors that are marked with an *A* letter.
- Double-click the connector to select the other connector on the diagram pane.
- Splitting of object flows helps to navigate in a complicated diagram.
- For more information about Refactoring, read Refactoring(see page 936).

To join Object Flow connectors

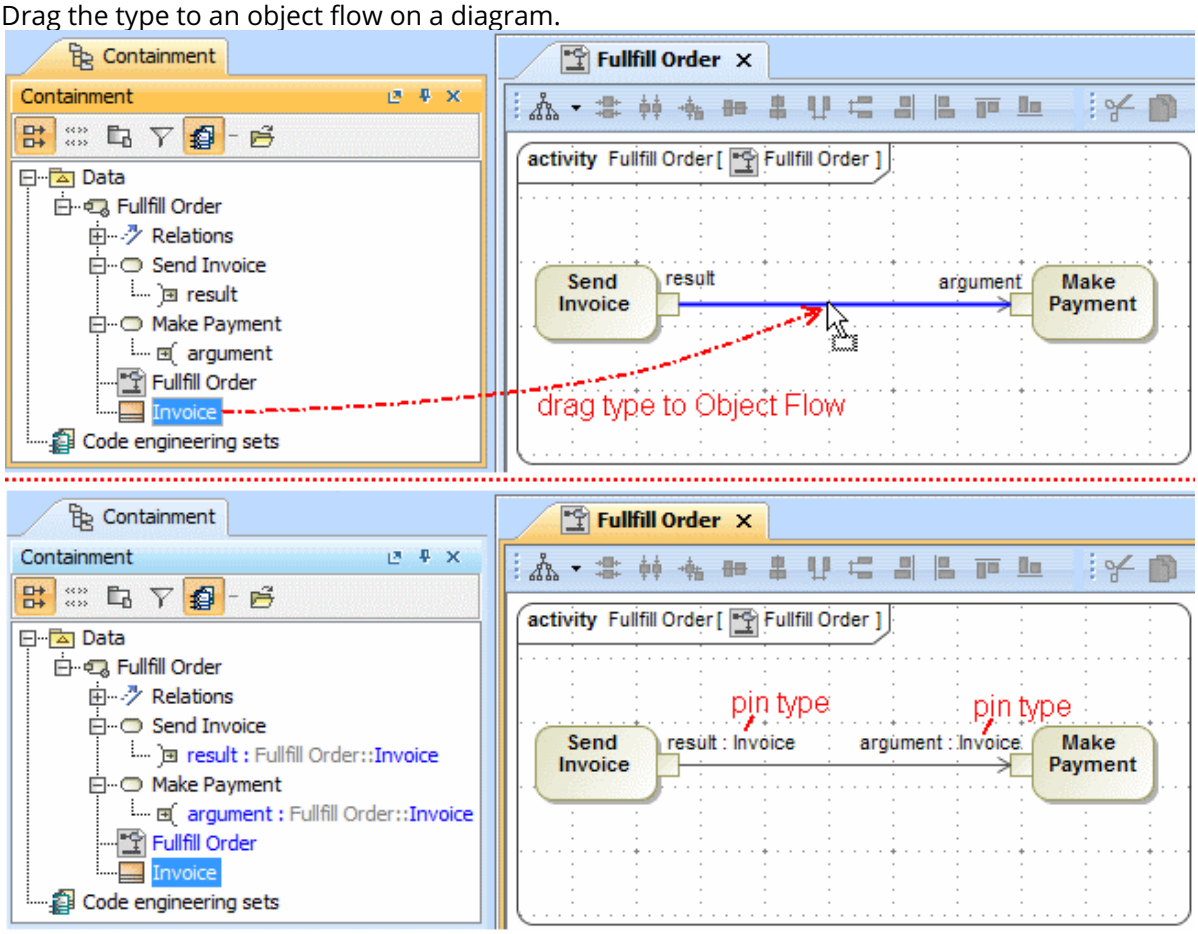
- From a connector's shortcut menu, select **Refactor** and then select **Join Connectors**.



To set type of Pins by using a drag-and-drop operation

---

1. In the Model Browser(see page 29), select a type of pins.
2. Drag the type to an object flow on a diagram.

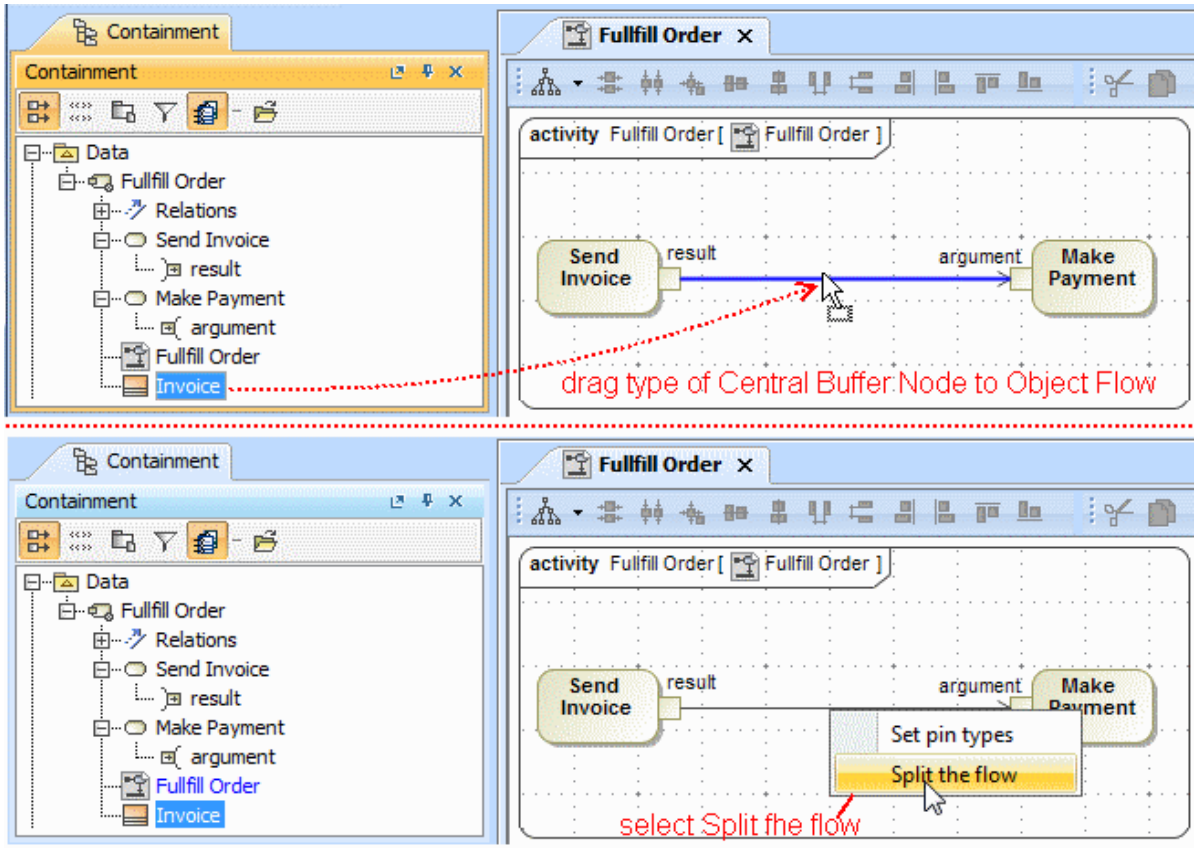


Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

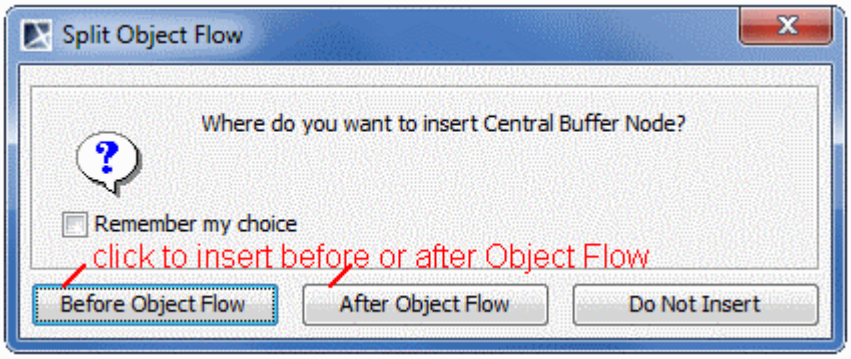
- If type of pins is set already, after drag-and-drop type will be changed to the new one.
- To display a type of a pin on a diagram, from the pin shape shortcut menu, select **Symbol Properties** and then select the **Show Type** check box.

To create Central Buffer Nodes on Object Flows by using a drag-and-drop operation

1. In the Model Browser(see page 29), select a type of a Central Buffer Node, and drag it to an Object Flow path on a diagram pane.
2. Wait for a few seconds and drop it.
3. Select the **Split the flow** command.

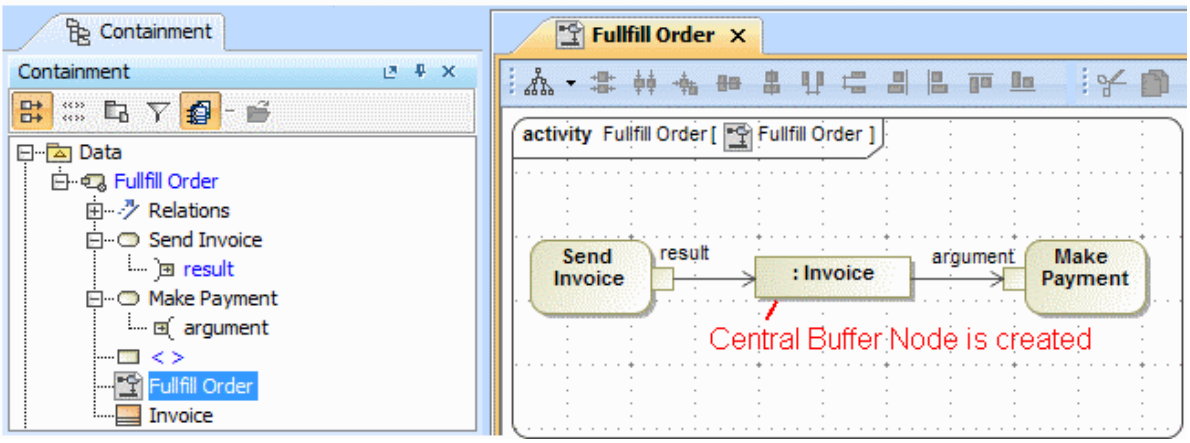


4. The **Split Object Flow** message opens.



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

5. Select to insert the Central Buffer Node before or after the Object Flow.



Related pages

- Activity diagram(see page 311)

Object Node

An Object Node is an Activity node indicating that an instance of a particular classifier, possibly in a particular State, may be available at a particular point in the Activity. It can be used in a variety of ways (depending on where the objects are flowing from and to), as described in the semantics section.

> **More Information**
> For more information about working with symbols, please see Diagramming(see page 482).

> As UML2 does not support generic Object Node, our modeling tool creates the particular Central Buffer Node element.

You can specify Object Node properties in the Central Buffer Node Specification window(see page 913). You can find the description of each property(see page 921) in the same window. Descriptions are presented in the description area of the Specification window.

To set a classifier to an object node

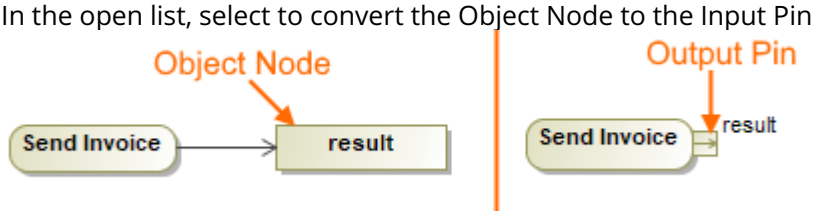
1. Double-click the Object Node or select **Specification** from the shape's shortcut menu(see page 798). The Central Buffer Node Specification window opens.
2. Select the classifier you wish to assign to an Object Node from the **Type** drop-down list.

To assign a state or final state to an object node

1. Click the button in the Central Buffer Node Specification window, **In State** value property. The **Select Elements**(see page 1031) dialog opens.
2. Select a State from the existing model elements, or click **Create** to create a new one. The State Specification window opens. Specify a new state, to be assigned to an Object Node.

To convert an object node to a pin

1. Select an Object Node on the diagram pane.
2. From the Object Node's shortcut menu, select the **Refactor** command, then select **Convert To**.
3. In the open list, select to convert the Object Node to the Input Pin or Output Pin.



Related pages

- Activity diagram(see page 311)
- Class diagram(see page 297)
- Dependency(see page 699)
- Action(see page 629)
- Package(see page 800)
- Associations(see page 651)
- Model Elements(see page 628)
- Exception Handler(see page 711)

Opaque Behavior

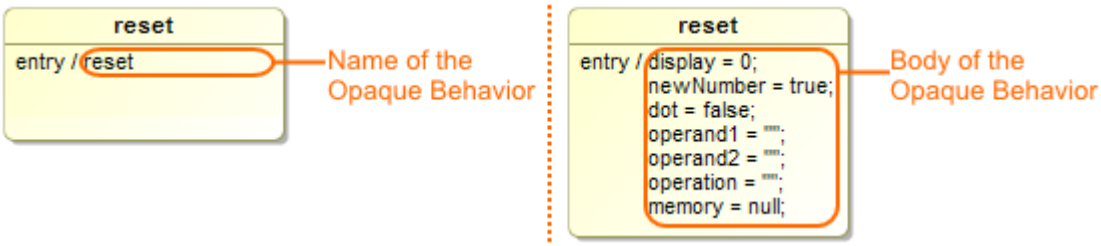
A behavior with implementation-specific semantics. The opaque behavior is introduced for implementation-specific behavior or for use as a place-holder before one of the other behaviors is chosen.

To create an opaque behavior

1. In the Containment tree, select a package and from the shortcut menu, select **Create Element** > **Opaque Behavior**.
2. Enter a name for the new opaque behavior.
3. In the Specification window(see page 913) of the opaque behavior, click the cell of the **Body and Language** property value, and then click . The **Body and Language** dialog opens.
4. From the **Language** drop-down list, select a language.
5. In the **Body** area, define the expression(see page 1227).

To change an opaque behavior display mode on state or transition shapes

1. In the State or Transition shape shortcut menu, select Symbol Properties(see page 975).
2. In the **Opaque Behavior Display Mode** property value cell, select either **Body** or **Name**.



To create a symbol of an opaque behavior

- In the Containment tree, select the opaque behavior and drag it to the diagram pane.

Related References

Related Pages

- Model Elements(see page 628)
- Specification Window(see page 913)

Package

A package groups classes and other model elements together. All types of UML model elements can be organized into packages. Each diagram must be owned by one package and the packages themselves can be nested within other packages. Subsystems and models are special kinds of packages.

The packages can have dependency, generalization, realize, containment, and association relationships. These relationships are usually derived from the relationships between the classes inside those packages.

Related References

Related Pages

- Model Elements(see page 628)
- Specification Window(see page 913)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Working with packages

You can specify package properties(see page 921) in the package Specification window(see page 913). In the same window, you can find the description of each property. Descriptions are displayed in the description area of the Specification window.

> You can conveniently import packages(see page 704) to find the elements you're looking for in the type selection list.

The procedures below illustrate how to work with packages:

- Adding inner elements(see page 801)
- Changing the package header position(see page 801)
- Showing the list of elements assigned to a package on its shape(see page 801)
- Displaying inner elements inside package shape(see page 802)

Adding inner elements

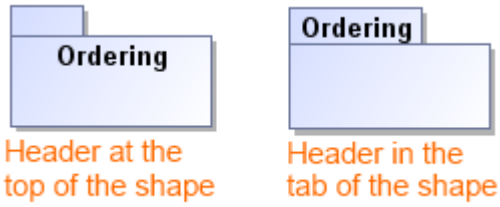
To add inner elements to the selected package

1. In the **Package** Specification window, select the **Inner Elements** property group.
2. Click the **Create** button, and select an element you wish to add. The selected element Specification window will open.
3. Define the properties you need.
4. Click **Back** to return to **Package** Specification window.

Changing the package header position

To change the package header position

- In the **Symbol Properties** dialog, set the Header Position property to:
  - **Top** to place a package header at the top of a package shape.
  - **In Tab** to place a package header in a package tab.

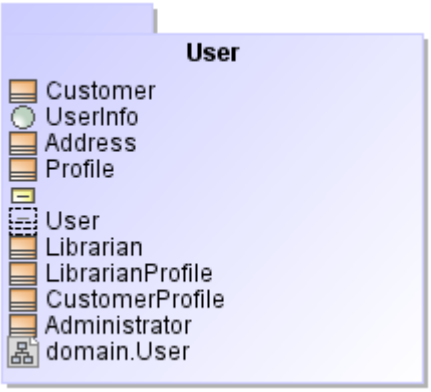


Showing the list of elements assigned to a package on its shape

To show the list of elements assigned to a package on the package shape

- From the selected package shortcut menu, choose **Show Inner Elements List**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



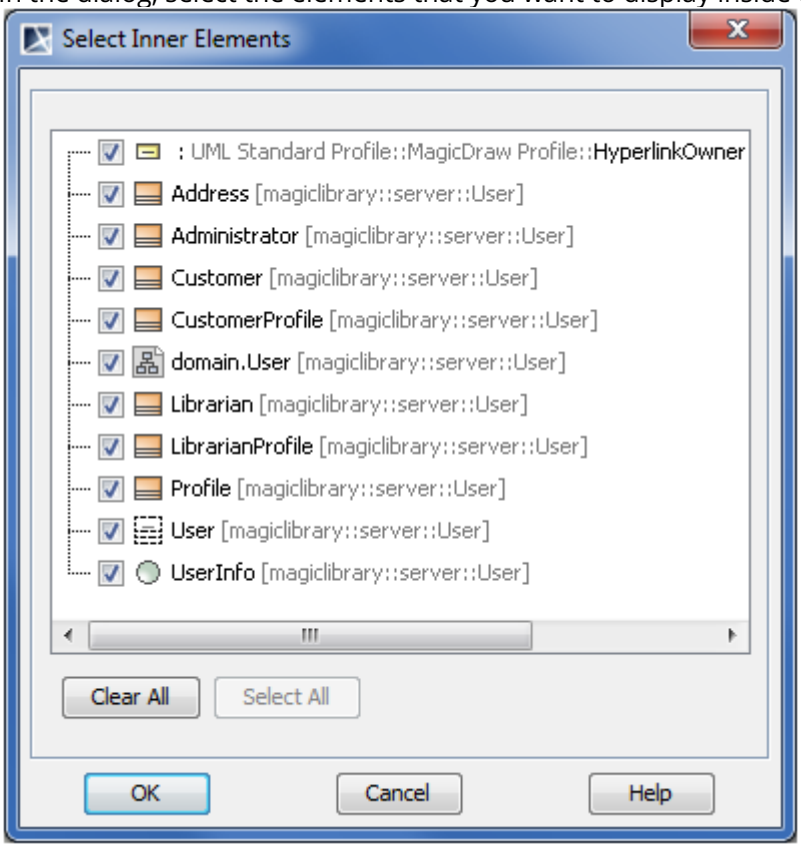
Displaying inner elements inside package shape

To display the inner elements of a package in a diagram

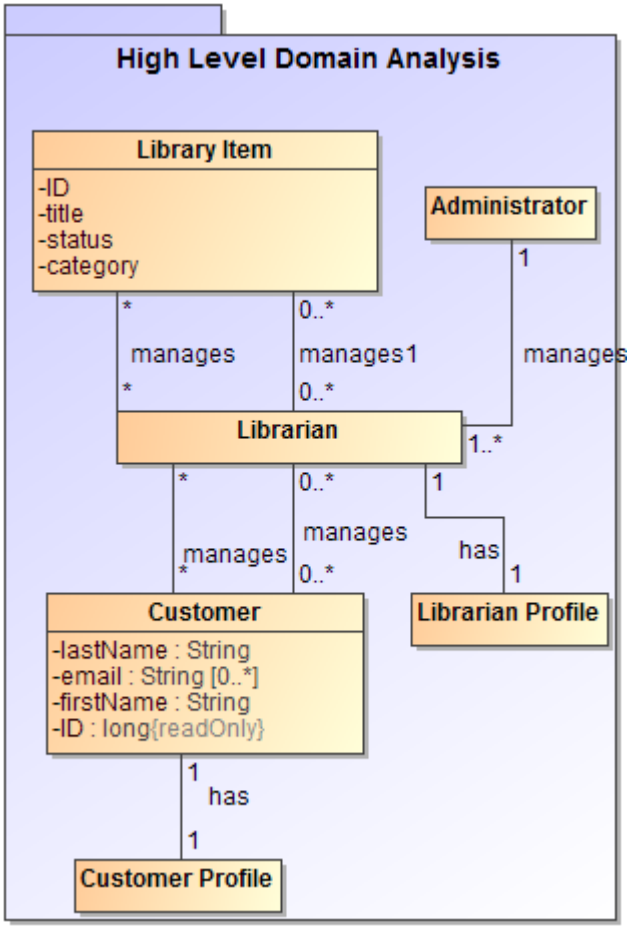
---

1. Select a single package shape on the diagram pane.
2. Open the **Select Inner Elements** dialog by doing one of the following:
   - From its shortcut menu, select **Display** > **Display Inner Elements**.
   - On the diagram toolbar, click

   and select **Display Inner Elements.**
3. In the dialog, select the elements that you want to display inside the package shape.



4. Click **OK**.
   The selected elements are displayed on the shape of the selected package.



> The rake icon is displayed on the shape if the package refers to another package diagrams. The rake icon isn't shown by default, therefore, you need to set the Show Rake Icon(see page 984) property value to *true* in the Symbol Properties(see page 975) dialog.

Related Pages

- Model Elements(see page 628)
- Specification Window(see page 913)

Package Merge

A package merge is a directed relationship between two packages indicating that the contents of the two packages are to be combined. It is very similar to a Generalization(see page 722), in that the source element conceptually adds the characteristics of the target element to its own characteristics, resulting in an element that combines the characteristics of both.

Specify the Package Merge in the Package Merge Specification window(see page 913).

You can format the Package Merge symbol properties in the Symbol Properties dialog(see page 975).

Part

On this page

- Creating a part(see page 804)
- Creating a new composite structure diagram for the type of the part(see page 804)
- Defining a type of the part(see page 804)
- Hiding the multiplicity(see page 805)

Creating a part

To create a part, do one of the following

- In the **Composite Structure** diagram palette, select **Part** and click on the diagram.
- In the Containment tree, select an element, which can be defined as a part type, and drag it to the composite structure diagram.

Creating a new composite structure diagram for the type of the part

To create a new composite structure diagram for the type of the part, do one of the following

- Select a part and from the smart manipulator toolbar, choose the New Composite Structure Diagram button.

  - A new diagram is created under the part type.
  - If type was not assigned for the selected part then a new type is created automatically.
  - The created diagram is named after the part type.

- From the part shape's shortcut menu, select Create Diagram and then diagram from the list. This accelerates the creation of composite structure diagram (the Create Diagram menu item is available only if the type is specified for the part).

Defining a type of the part

To define a type of a part, do one of the following

- On a diagram pane, draw a new part and in the opened Select Type list, select a type.

  The **Select Type** list is not displayed if the **Type Selection Mode** is disabled. How to turn on the Type Selection Mode >>(see page 1036)

- On a diagram, select a part and press Ctrl+T (for MAC OS X press Cmd+T) and in the opened Select Type list, select a type.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- On a diagram, select a part and from the smart manipulator toolbar, select the Specify Type button. In the Select Type dialog, select a type.
- On a diagram, select a part and then type the name of the type after a ":". The type will be created with the specified name.
- In the part Specification window(see page 913), **Type** property specification cell, select a type.

> When creating a port on a part, if a part does not have a specified type, a new type for the part is created automatically.

> The rake icon is displayed on the shape if the interaction use refers to another composite structure diagram. The rake icon is not shown by default therefore you need to set the Show Rake Icon(see page 984) property value to true in the Symbol Properties(see page 975) dialog.

Hiding the multiplicity

To hide the multiplicity next to the part's type

1. Right-click the part shape and select **Symbol Properties**.
2. In the **Symbol Properties** dialog, set the **Show Multiplicity** property value to *false*.
3. Click **Close**.

Related pages

- Displaying parts and ports(see page 491)
- Model Elements(see page 628)
- Specification Window(see page 913)

Pin

On this page

- Description(see page 805)
- Assigning Pins(see page 806)
- Specifying Pin type(see page 807)
- Displaying Pins(see page 807)
- Showing Pin's type(see page 808)
- Converting a Pin(see page 808)
- Navigating from a Pin to the associated parameter(see page 808)

Description

A Pin represents a connection point for the input and output values of an action. There is one input Pin for each input parameter, and one output pin for each output parameter.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Using MagicDraw, you can create the following types of Pins:

- Input Pin. An input pin is a pin that holds input values to be consumed by an action.
- Output Pin.
- Value Pin.
- Action Input Pin.

- When a new action is created that must have mandatory Pins, they are automatically added to that action on the diagram pane, as well as in the Containment tree.
- As of MagicDraw 17.0.4 version, the *Pins Displaying* validation (see page 1306)suite has been introduced. It includes the *Action Containing Hidden Pins* validation rule that detects all hidden pins for the concrete action symbol.
- When deleting the last Pin symbol from the diagram, the Pin element is deleted from the project. This is valid when the Display All Pins in Diagrams option value is true.
- For a send signal action, a Pin is created, but the symbol of the Pin is not represented on the diagram pane.
- You can navigate to the associated parameter from the Pin shortcut menu. From the Pin shortcut menu, select Go To and choose a parameter.

You can format the pin symbol properties in the Symbol Properties dialog(see page 975).

You can specify pin properties(see page 921) in the pin Specification window(see page 913). In the Specification window(see page 913), you can find the description of each property, which are located in the description area of the Specification window.(see page 913)

Assigning Pins

To assign Pins for an action on the diagram pane

1. In the activity diagram, draw an action.
2. In the activity diagram palette, select a Pin you want to create.
3. On the diagram pane, click the action. The Pin for this action is created.

To assign Pins for an action in the action Specification window(see page 913)

1. In the action Specification window(see page 913), in **Pins** property group, select one of the following:
   - Input Pin. Select Argument, and, in the property specification cell, click the Add button. Select one of the preferred pins from the list.
   - Output Pin. Select Result, and in the property specification cell, click the Add button.
2. In the opened Pin Specification window(see page 913), type the Pin name.
3. Click **Close** or **Back** to return to the action Specification window(see page 913).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Specifying Pin type

After the Pin is assigned to an action, you can specify it's type. Use one of the following way:

- Using the **Specify Type** button.(see page 807)
- Using the Specification window.(see page 807)

To specify Pin type by using the **Specify Type** button

---

1. Select the Pin shape.
2. Click T on the smart manipulator toolbar(see page 568).
3. Do one of the following:
   - Select an existing element from the **Select Type** menu.
   - Create new type:
     a. Type a name.
     b. Press Enter.
     c. In the **Create <type name> As** dialog, select element kind.

   The type is specified for a Pin.

To specify Pin type by using the Specification window

---

1. Right-click the Pin and select the **Specification** command.
2. In the Specification window(see page 913), select the **Type** property value cell.
3. Do one of the following:
   - Select an existing element from the list.
   - Click ... and in the **Select Element** dialog do the following:
     - Select an existing elements from the **Tree** or **List** tab.
     - create new element by switching on the **Creation Mode** and clicking the **Create** button.
4. Click **Close**.
   The type is specified for a Pin.

Displaying Pins

To display Pins on shapes

---

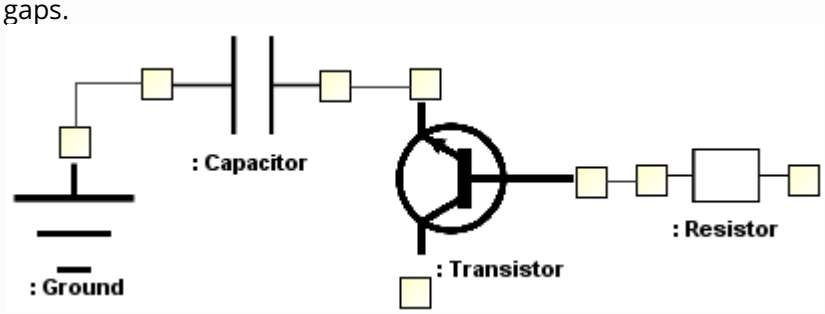
1. On the diagram pane, select any number of shapes on which you want to display Pins.
2. Open the **Select Pins** dialog by performing one of the following steps:
   - From the shortcut menu, select **Display** > **Display Pins**.
   - On the diagram toolbar, click

   and select **Display Pins.**
3. In the dialog, select the Pins you want to display.
4. In the **Layout options** area, specify the Pins' layout on the shape:
   - **Top/Bottom** - displays Pins on the shape from its top to bottom.
   - **Left/Right** - displays Pins on the shape from its left to right.
5. When you are finished, click **OK**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

If you want all Pins to be displayed automatically when creating a new Activity diagram, go to **Options** > **Project** on the main menu, and set the **Display All Pins in Diagrams** option to *true*.

If you have an element displayed as an image, you can connect a Pin to that element with no gaps.



Showing Pin's type

To show the Pin's type, perform one of the following steps

- From the Pin shortcut menu, select **Show Type**.
- In the Pin Symbol Properties dialog(see page 975), set the **Show Type** property to true.

Converting a Pin

To convert an input Pin to an output Pin and vice versa

1. From the Pin shortcut menu, select **Refactor** > **Convert To**.
2. From the opened list, select one of the following:
    - Input Pin (Output Pin)
    - Action Input Pin
    - Value Pin

To convert a Pin(see page 937) to an object node

1. From the Pin shortcut menu, select **Refactor** > **Convert To**.
2. From the opened list, select **Object Node**.

Navigating from a Pin to the associated parameter

To navigate from a Pin to the associated parameter

- From the Pin shortcut menu, select **Go To,** and choose a parameter.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related pages

- Model Elements(see page 628)
- Specification Window(see page 913)

Port

On this page

- Description(see page 809)
- Creating Port(see page 809)
- Defining Port type(see page 810)
- Displaying Port type(see page 810)
- Displaying Ports on shapes(see page 810)
- Creating a Port on a composite structure diagram frame(see page 811)
- Displaying Port in compartment(see page 813)
- Changing position of Port label(see page 813)
- Specifying he Provided/Required Interfaces for a Port(see page 814)
- Converting a Port to an Association(see page 815)
- Displaying a Port name in a signature on a shape(see page 815)
- Hiding the multiplicity(see page 815)

Description

A port is a property of a classifier that specifies a distinct interaction point between that classifier and its environment, or between the (behavior of the) classifier and its internal parts. Ports are connected to the properties of the classifier by connectors through which requests can be made to invoke the behavioral features of the classifier.

A Port can specify the services a classifier provides (offers) to its environment as well as the services that a classifier expects (requires) from its environment. It has the ability to specify that any requests arriving at this port are handled.

The Class model element and Component model elements can have any number of Ports.

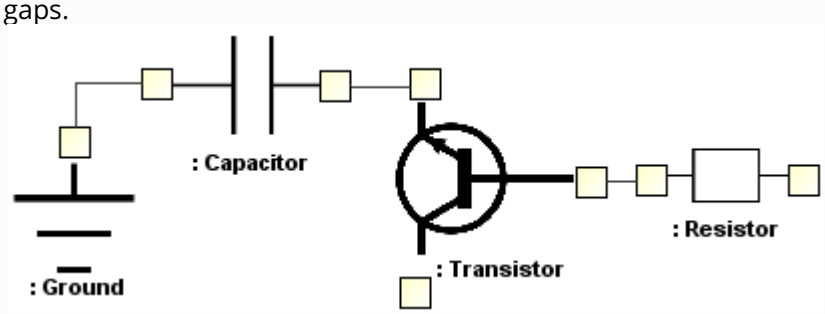
You can specify port properties(see page 921) in the port Specification window(see page 913). In the same window, you can find the description of each property. Descriptions are presented in the description area of the Specification window(see page 913).

Creating Port

To create a port, do one of the following

1. From the diagram palette, select port and on the diagram pane, click the element.
2. Select an element and from the smart manipulator toolbar, choose the Port button.
3. In the element Specification window(see page 913), **Port** property group, click the **Create** button.

ⓘ If you have an element displayed as image, you can connect a port to that element with no gaps.



Defining Port type

To define a Port type, do one of the following

- On a diagram, select a port and press Ctrl+T (press Cmd+T for MAC OS X). Select a type from the list.
- On a diagram, select a port and in the smart manipulator toolbar, click the Specify Type button. Select a type from the list.
- In the port Specification window(see page 913), **Type** property specification cell, select a type.

Displaying Port type

To show the Port type, do one of the following

- Select a port, and from the shortcut menu, choose **Show Type**.
- In the port Symbol Properties dialog(see page 975), set the **Show Type** property to true.

Displaying Ports on shapes

To display Ports on shapes

1. Right-click the shape, select **Display** > **Display Parts/Ports**. The **Display Parts/Ports** dialog opens.
2. Select ports you want to display on the shape:
   - On the left side of the dialog, select ports manually.
   - On the right side of the dialog, use the panel to select all or particular types of properties or ports.
3. Click **OK** when you are done.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- The port type near the port symbol is displayed if port has referenced type, but has no name.
- The port type near the port symbol is not displayed if type is provided or required interface and the interface is displayed on the diagram.
- When a part symbol is created, all its ports are displayed on the diagram.

Creating a Port on a composite structure diagram frame

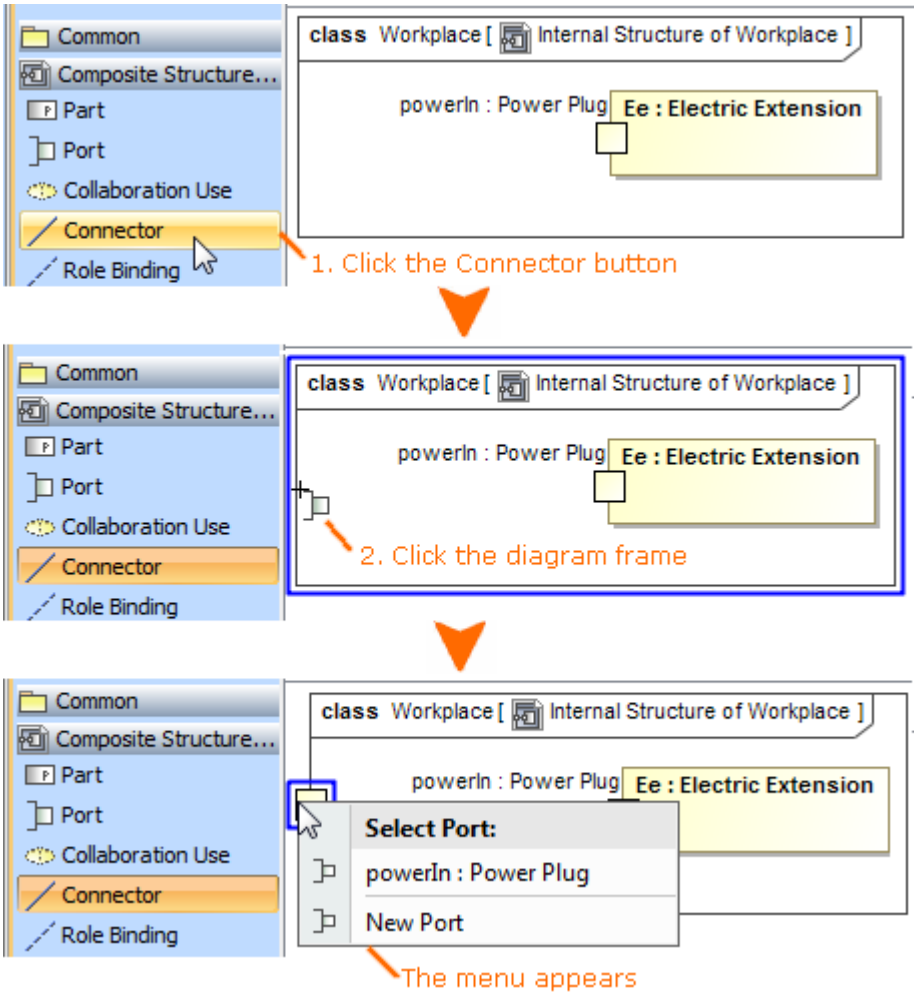
To create a Port on a composite structure diagram frame

1. Select an element and from the smart manipulator toolbar, choose the **Connector** button.
2. Click the composite structure diagram frame. The port is created and the Select Port list is displayed.
3. Select one of the following:
   - A hidden port or a nested port of the composite structure diagram.
   - **New Port** - click to create a port on a diagram frame, with the same name, type and multiplicity, as it is set on the source part or port.
   - **Nested Port** - click to create a new nested port.

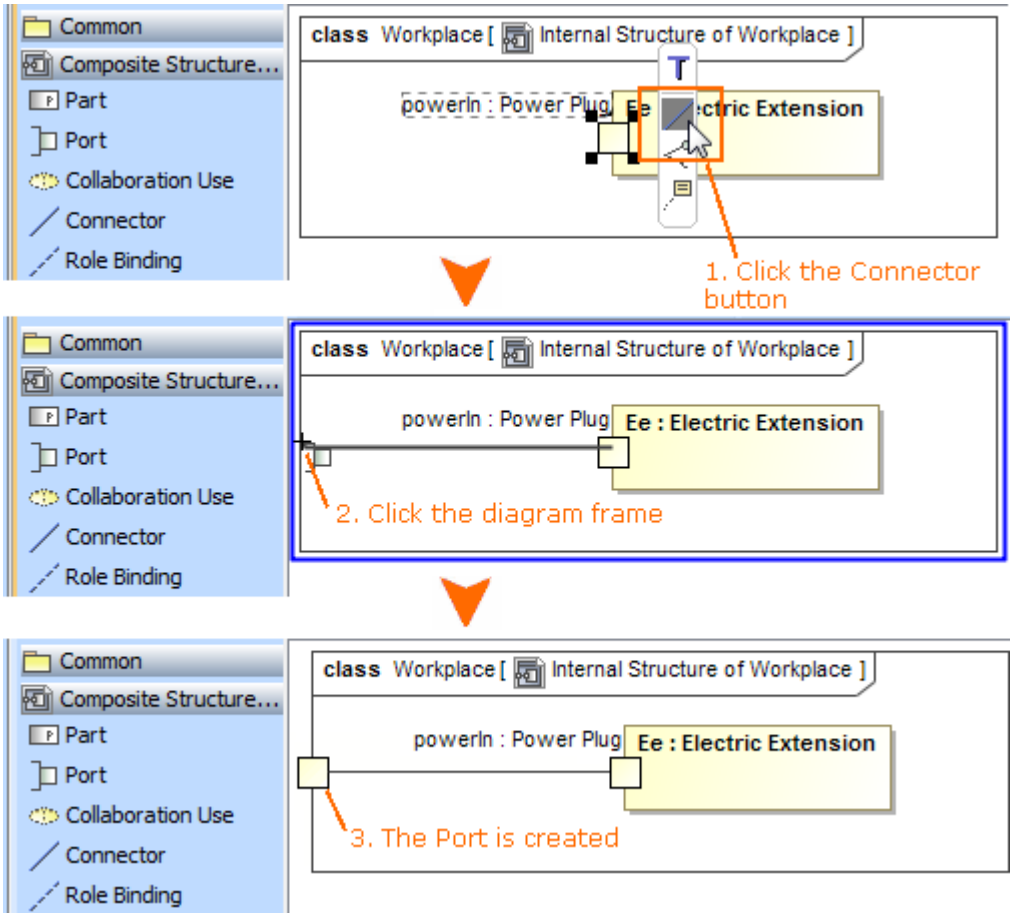
     The **Nested Port** is available only when creating a connector from the nested port to the diagram frame.

- You can also quickly create a port on the composite structure diagram frame when drawing a connector straight from a composite structure diagram frame.
- If a connector is created from a port with the defined type, then only compatible hidden ports are listed in the **Select Port** list.
- If a connector is created from a port that has interfaces, then ports that have compatible interfaces are listed in the **Select Port** list.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Displaying Port in compartment

To display Ports in the element compartment

1. On the diagram, select a symbol and click the Compartments button.
2. In the list, select **Ports**. The ports' compartment is displayed on a symbol.

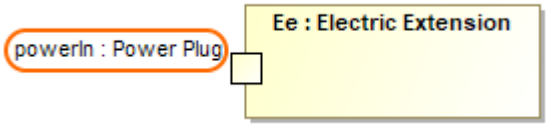
> In the **Compartment Edit** dialog, you can select which ports to display in the ports' compartment.

Changing position of Port label

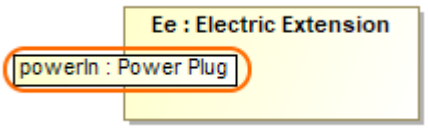
To change position of Port label

1. Right-click the Port shape and select the **Symbol Properties** command.
2. In the Symbol Properties dialog(see page 975), select the **All** property display mode.
3. Find the **Position of Labels** property and select on of the following value:
   - **Outside** to show the label outside the Port shape.

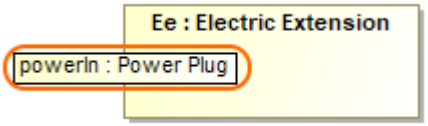
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



- **Inside** to show labels inside the Port shape.

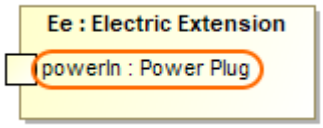


- **Name and Type Labels Inside** to show only the name and type inside the Port shape.



- **Inside Enclosing Shape** to show the label outside the Port shape but inside enclosing shape on which boundary the port is created.

> Nested ports do not support **Inside Enclosing Shape** value.



Specifying he Provided/Required Interfaces for a Port

To specify the Provided/Required Interfaces for a Port even if the Port type is not specified

---

1. In the Port Specification window(see page 913), **Provided/Required Interfaces** property group, click the **Add** button.
2. Select to either provided or required interface.
3. In the case, when the port type is not specified, the **Select Port Type** dialog is displayed. Select one of the following options:
   - Set provided interface as a port type (available only when creating a provided interface). The provided interface will be specified as the port type.
   - Create "dummy" port type automatically. Create a “dummy” port type and relations between the type and interface.
   - Select or create a port type manually. The **Select Port Type** dialog will open, to allow you to select or create a Port.
4. In the element Selection dialog, select or create an interface and click **OK**.

> - Draw a realize relationship from a port to an interface to create a provided interface.
> - Draw a usage relationship from a port to an interface to create a required interface.

Converting a Port to an Association

To convert a Port to an Association, do one of the following

- From the port's shortcut menu, select **Refactor** > **Convert To** > **Association**.
- In the port's compartment, select a port and drag it out of the shape.

> ⚠ You can convert a port to an association, only if the port type is specified and a port is not a member of any association end.

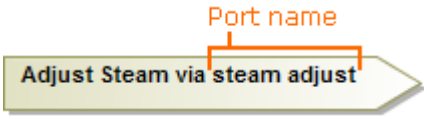
Displaying a Port name in a signature on a shape

To display a Port name in a signature on a shape

- In the **Symbol Properties** dialog(see page 975), click to select the **Show Port** property.

> ⓘ If you cannot find the **Show Port** property in the **Symbol Properties** dialog(see page 975), change the properties display mode to *Expert* or *All*.

You can display the Port name on the following shapes:
• Call Operation Action, Call Behavior Action, Send Signal Action, Send Object Action, Broadcast Signal Action, and Start Object Behavior Action.
For example: via <Port name>.



• Transition, Protocol Transition, State, and Accept Event Action.
For example: « from» <Port name>.

Hiding the multiplicity

To hide the multiplicity next to the Port's type

1. Right-click the Port shape and select Symbol Properties.
2. In the **Symbol Properties** dialog, set the **Show Multiplicity** property value to *false*.
3. Click **Close**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related pages

- Displaying parts and ports(see page 491)
- Model Elements(see page 628)
- Specification Window(see page 913)

Profile

A Profile is a kind of a package that extends a reference metamodel. The primary extension construct is a stereotype. Stereotypes are defined as a part of profiles.

A profile introduces several constraints or restrictions to ordinary metamodeling. Constraints and restrictions are realized using metaclasses defined in the package. It is a restricted form of a metamodel that always must be related to a reference metamodel, such as UML, as it is described below. It cannot be used without its reference metamodel, and it defines a limited capability to extend metaclasses of the reference metamodel. The extensions are defined as stereotypes that apply to the existing metaclasses.

Each profile contains a set of stereotypes. Profiles are defined as separate modules. Profiles are loaded on demand, that is when you start or open your project only profiles used in that project are loaded.

Profiles are defined using the UML extensibility mechanisms that allow modelers to customize UML for specific domains, for example, for software development processes. The mechanism of the profile is similar to the functionality of modules.

The program comes with some predefined profiles: UML Standard Profile, DDL, EDOC, and other. All profiles are stored in *<modeling tool installation directory>\profiles*.

Profile Properties

The profile is defined as a package(see page 800), that is, it has package properties.

You can specify profile properties in the profile's Specification window(see page 913). You can also find descriptions of each property(see page 921) in this window. Descriptions are provided in the description area below the property list.

Creating Profiles

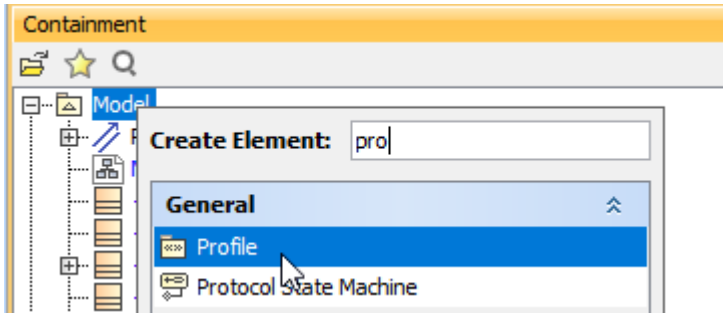
You can create a profile in one of the following ways:

- Using the package / model / profile shortcut menu(see page 816)
- Using the profile diagram palette(see page 817)
- Using the package diagram palette(see page 817)
- Using the class diagram palette(see page 817)

To create a profile using the package / model / profile shortcut menu

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

1. In the Containment tree, select a package, a model, or another profile, wherein you want to create the new profile.
2. From the shortcut menu(see page 816) of the selected package, model, or profile, select **Create Element** > **Profile**.



To create a profile using the profile diagram palette

---

1. Either create a new Profile diagram(see page 324) or open an existing one.
2. On the diagram palette, click the **Profile** button.
3. Click a free space on the diagram pane to see the created Profile.

To create a profile using the package diagram palette

---

1. Either create a new Package diagram(see page 321) or open an existing one.
2. On the diagram palette, click the **Package** arrow to see more buttons.
3. Click the **Profile** button.
4. Click a free space of the diagram pane to see the created Profile.

To create a profile using the class diagram palette

---

1. Either create a new Class diagram(see page 297) or open an existing one.
2. On the Class diagram palette, do one of the following:
   - Expand the Profile Diagram button group and click the **Profile** button.
   - Expand the Package Diagram button group, click the **Package** arrow to see more buttons, and then click the **Profile** button.
3. Click a free space of the diagram pane to see the created Profile.

Related pages

- Model Elements[154]
- Stereotype(see page 856)

154 https://docs.nomagic.com/display/MD190SP4/Model+elements

- Package(see page 800)
- Working with Profiles(see page 2036)

Profile Application

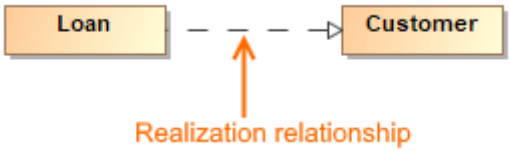
A profile application is used to show which profiles have been applied to a package.

Specify the Profile Application element, in the Specification(see page 913) dialog.

Realization

The Realization is a specialized abstraction(see page 1211) relationship between two sets of Classes, one represents a specification (the supplier), and the other represents an implementation of the latter (the client). The Realization can be used to model stepwise refinement, optimizations, transformations, templates, model synthesis, framework composition, and so forth.

The Realization relationship is drawn as a dashed line with a solid triangular arrowhead (a "dashed generalization symbol"). The client (the one at the tail of the arrow) supports at least all of the operations defined in the supplier (the one at the arrowhead), but not necessarily the data structure of the supplier (Attributes(see page 672) and Associations[155]).



More Information
For more information about working with symbols, see Diagramming(see page 482).

The Realization paths can be grouped in a tree. This feature makes the appearance of the diagram more structural and understandable.

155 https://docs.nomagic.com/display/CBMP190SP4/Association

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Realization Types

In our modeling tool, you will find three kinds of Realization relationships:

- **Interface Realization.** A dashed line with a solid triangular arrowhead. An Interface Realization is a specialized Realization relationship between a Classifier and an Interface. This relationship signifies that the realizing classifier conforms to the contract specified by the Interface.
- **Realization.** A solid line that represents a relationship between a classifier(see page 507) and an interface(see page 741).
- **Substitution.** A dashed line with an arrowhead and *«substitute»* stereotype. A Substitution is a relationship between two classifiers. It signifies that the Substituting classifier complies with the contract specified by the contract classifier. This implies that instances of the Substituting classifier are runtime substitutable where instances of the contract classifier are expected.

To specify the selected realization path in the Specification window

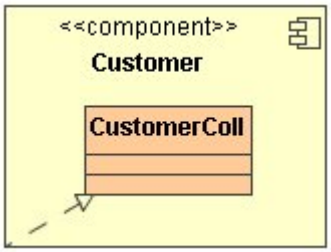
- Double-click the path.
- Select **Specification** from the path's shortcut menu.
- Select the path and press **Enter**.

Creating the realizing classifiers

- The realizing classifiers are a set of Realizations owned by the Component. The Realizations reference the Classifiers of which the Component(see page 685) is an abstraction (i.e., that realize its behavior).

To create a Realization relationship between a component and a classifier:

1. Drag the classifier shape to the Component(see page 685) shape.
2. Select a classifier or Component and select **Related Elements** from its shortcut menu, then select the **Display Paths** command.
3. The Realization relationship will be displayed on the diagram pane.



Related pages

- Activity diagram(see page 311)
- Class diagram(see page 297)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Action(see page 629)
- Package(see page 800)
- Associations(see page 651)
- Model Elements[156]

Reception

Signal Receptions can be specified for classes(see page 666) or interfaces(see page 741).

You can specify Reception properties in the Reception Specification window(see page 913). You can find the description of each property in the same window. Descriptions are presented in the description area of the Specification window.

To create a new reception

Do one of the following:

- In the Class's Specification window, click the **Signal Receptions** property group and then click the **Create** button. Select the signal and click **OK**. The Signal Reception's Specification window opens.
- In the Containment tree, right-click a Class and from its shortcut menu(see page 820), select **Create Element** and then **Signal Reception**. Select the signal and click **OK**. Type the name for the new signal reception.
- On the diagram pane, click the Class, and then click the *Create Element* smart manipulator. At the bottom of the menu, click the arrow to expand the menu and then click the **Signal Reception** command. Select the signal and click OK.
- On the diagram pane, click the Class, then press the **Ctrl+Alt+R** shortcut keys. Then, select the signal and click **OK**.
- On the diagram pane, click the Class shape, and then click the **Create Signal Reception** smart manipulator.

> **Signal Reception**
> Note that the signal reception compartment is suppressed by default.

- Drag the signal from the Containment tree or the diagram to the Class shape on the diagram pane.

To set the signal for a signal reception

Do one of the following:

- Create a signal reception. The Select Signal dialog opens. Select an existing or create a new signal for the signal reception.

156 https://docs.nomagic.com/display/MD190SP4/Model+elements

- In the Signal Reception's Specification window, select a signal in the **Signal** drop-down menu. You can also click the "..." button. In the Select Element dialog, select a signal or click the **Create** button to create a new one.

To display signal reception on shape and to customize a displayed list

1. On the diagram pane, click on a symbol, then click the *Compartments* button.
2. On the menu, click to select the **Signal Receptions** check box. The signal receptions compartment is displayed on the shape.
3. On the diagram pane, click a shape, then click the *Compartments* button. On the menu, click the **Edit Compartments** command.
4. In the Compartment Edit dialog, click the **Signal Receptions** tab.
5. Move the signal receptions from the *Hidden* to the *Selected* list or vice versa.

To change the order of the signal receptions

1. On the diagram pane, right-click the shape. From its shortcut menu, select the **Symbol Properties** command.
2. In the Symbol Properties dialog, under the **Signal Receptions** group, click to change the **Signal Receptions Sort Mode** property.

Related pages

- Class diagram(see page 297)
- Action(see page 629)
- Package(see page 800)
- Associations(see page 651)
- Model Elements(see page 628)

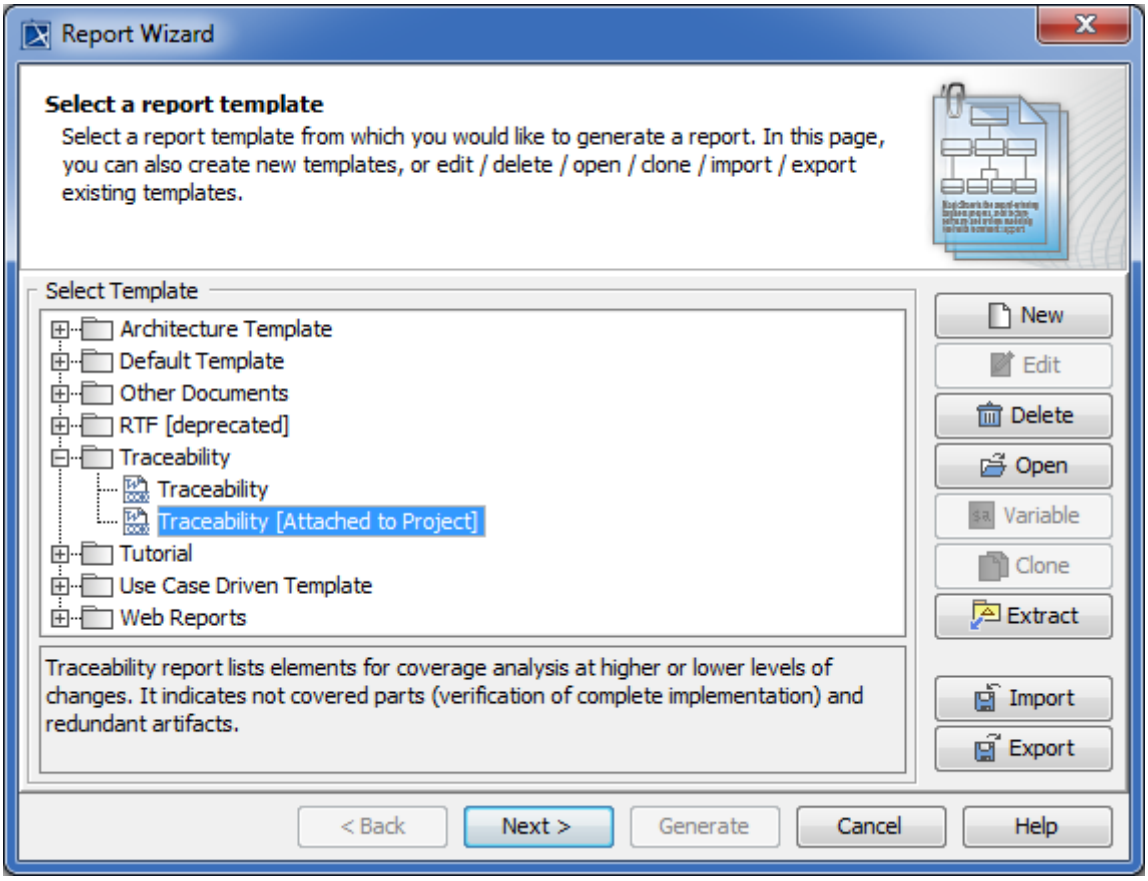
Report Template

This is a custom element used to store a report template file (*.mrzip*) attached to the project.

In contrast to the report templates stored in the local report templates folder (*<modeling tool* configuration files directory (see page 73) *>\data\reports*), a report template attached to the project as this specific type of model element is not lost when you open the project on another computer. It is safely transferred within the project (either local or server).

To attach a report template to the project

1. From the main menu, select **Tools** > **Report Wizard**. The **Report Wizard** dialog(see page 1547) opens.
2. Select the report template you want to attach to the project and click the **Attach** button. The selected report template is duplicated, and the duplicate is attached to the open project. It is visible in the Containment tree(see page 32) directly under the root package. As the following figure shows, in the **Report Wizard** dialog it as marked with the *"[Attached to Project]"* tag.



Attached to a project report template in the Report Wizard dialog.

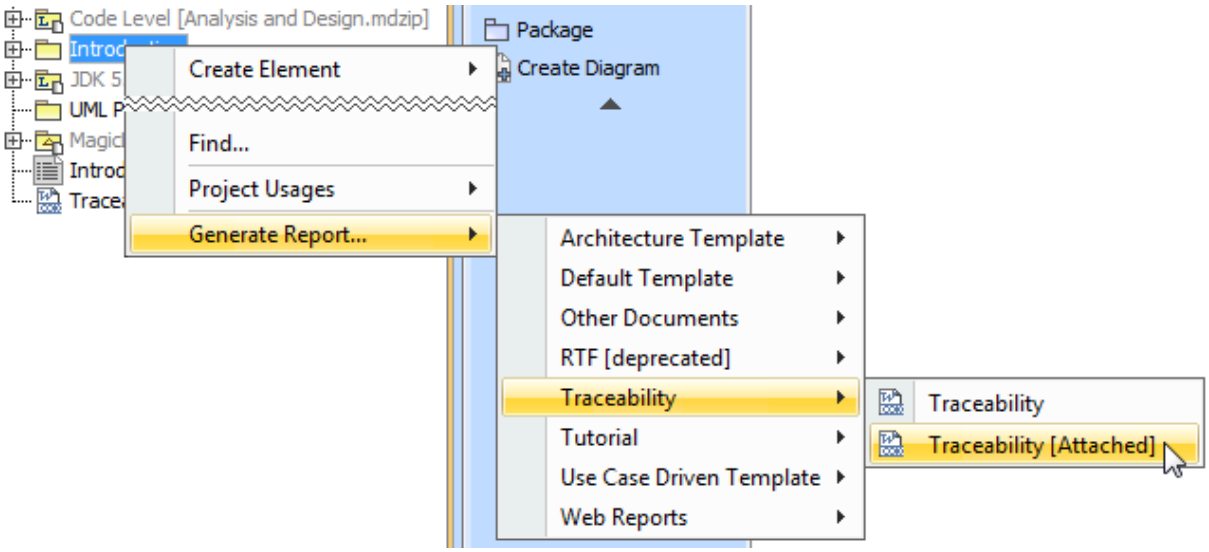
You can generate a report from the attached report template in one of the following ways:

- Right-click the attached report template in the Containment tree. From the shortcut menu select **Generate Report**. The **Report Wizard** dialog opens with the attached report template already selected. Proceed to the subsequent steps of the wizard.
- Right-click the namespace (for example, a Package) in the Containment tree. From the shortcut menu select **Generate Report**. Then select the appropriate report templates category and click the attached report template.

> The *"[Attached]"* tag helps to identify attached to project report templates on this menu.

Proceed to the subsequent steps of the wizard.

- From the main menu, select **Tools** > **Report Wizard**. In the **Report Wizard** dialog, select the report template and proceed to the subsequent steps of the wizard.



Selecting attached to project report template for report generation.

If you no longer need the attached report template, you can simply delete it from the model. Just select it in the Containment tree and press **Delete**.

If you need to modify the attached report template, do the following:

- Extract it to the local report templates folder.
- Open the extracted template for edit and make changes in it.
- After you finish editing, reattach the updated report template to your project.

To extract a report template to the local report templates folder

---

- From the main menu, select **Tools** > **Report Wizard.** The **Report Wizard** dialog opens.
- Select the report template you want to extract to the local report templates folder and click the **Extract** button.
- If a message asking to confirm the update of already existing report template appears, click **Yes.**

The report template appears in the local report templates folder and can be opened for edit. It also remains attached to the open project.

Related pages

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Report Wizard(see page 1547)
- Containment Tree(see page 693)
- Model Elements(see page 628)

Slot

A Slot specifies that an entity modeled by an Instance(see page 735) specification has a value or values for a specific structural feature. Values of each Attribute(see page 672) have specific type and are allocated in the slots of the instance or associated with that instance.

You can modify the current element and add various specifications to it in the element dialog box: click on the property group Tags, select the tag with the assigned value and click Edit Value to open the Slot Specification window(see page 913).

Creating Slots on Link creation automatically

When creating Link(see page 750) between two Instance(see page 735)s, slots are suggested to create for Instance automatically.

1. On diagram draw two Class(see page 666) shapes - A and B Classes.
2. Draw Association(see page 651) with roles between them.
3. Draw two Instance shapes. Define Class A as classifier of the first Instance and Class B as classifier of the second Instance.
4. Draw Link between Instances. The Select Association dialog will open.
5. Association which is drawn between Classes will be automatically selected in the Select Association dialog. Click **OK**.
6. The Create Slots dialog will open. Select Association ends. Ends of Association which is drawn between class A and Class B will be selected automatically in the Create Slots dialog. Click **OK**. Slots for Instance will be created.

Smart Package

This feature is available in Standard, Professional, Architect, and Enterprise editions.

Smart package is a special collection of model elements.

There are two ways an element can be included in the smart package:

- *Manually* - the user selects to include the particular element.

  You can create a smart package named *My Bookmarks* and add frequently used elements to it.

- *Automatically* - the element meets the set of criteria defined by the user.

  You can create a smart package named *Requirements v5* with the criteria “all elements of type *Requirement* under the package *Business requirements*, having tag *version=5*”.

The membership in the smart package is not the UML ownership; one element can belong to several smart packages.

Smart packages aggregate relevant elements, so that you can:

- Easily browse, navigate, list, and discover these elements in the Containment tree.
- Narrow the scope in both the **Find** dialog and the element Selection dialog.
- Define dynamic row and column scopes in dependency matrices.

Concepts

To better understand this material, please read through the following concepts.

Static contents

A collection of manually included elements.

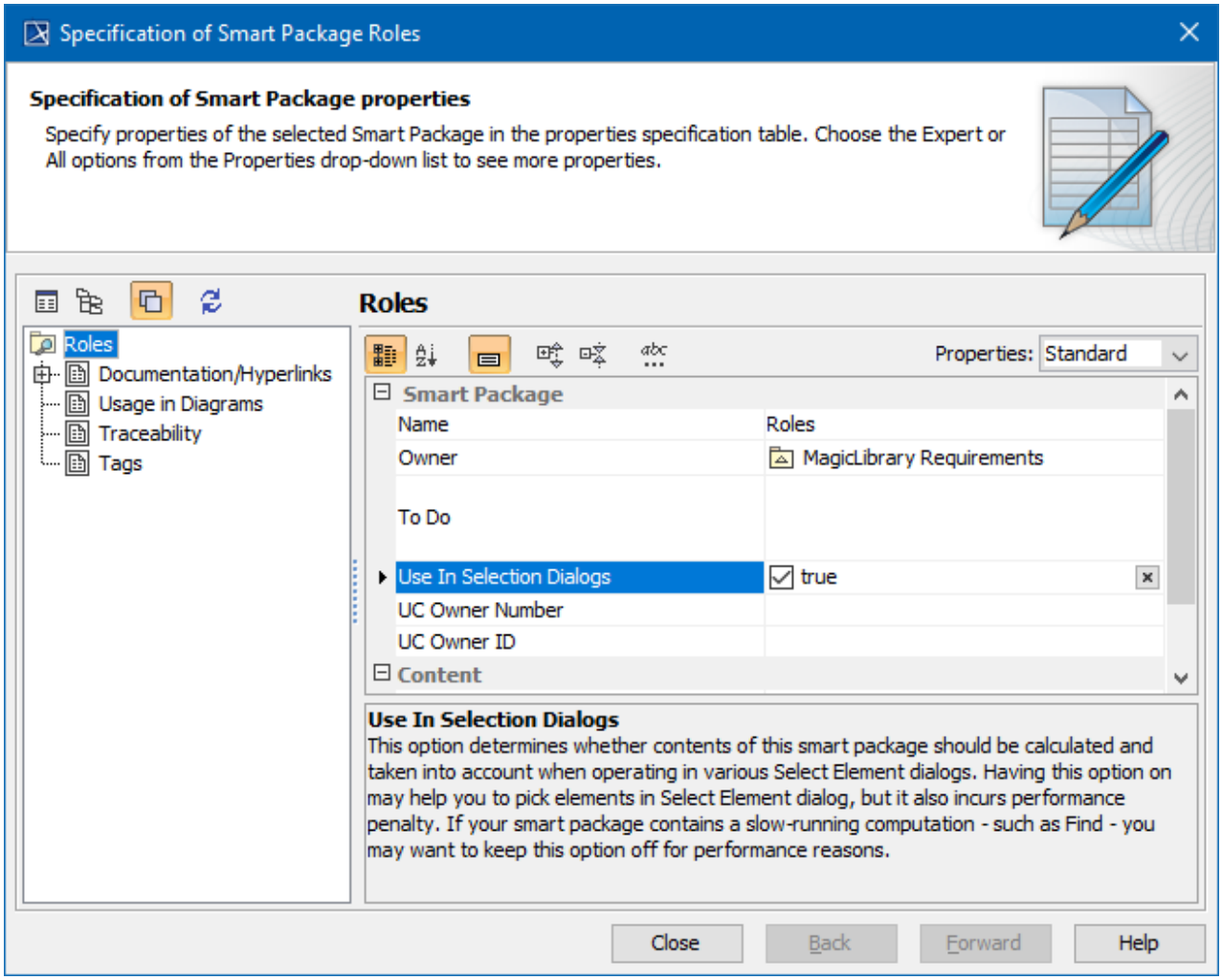
Dynamic contents

A collection of elements automatically calculated according to the set of criteria specified by the user.

Performance contents

In the **Smart Package** Specification window(see page 913), you can take advantage of a new property, **Use In Selection Dialogs,** which allows you to work better with your smart packages. Previously, smart packages slowed down the opening of **Selection** dialog boxes. **Use In Selection Dialogs** allows users to exclude the smart package from the calculation of elements to display in **Selection** dialog boxes (e.g. Select Elements(see page 1031)).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Related pages

- Creating smart packages(see page 826)
- Managing contents of smart package(see page 828)
- Smart package properties(see page 831)
- Using smart packages in your model(see page 832)
- Smart package-related performance issues(see page 839)

- Model Elements(see page 628)
- Stereotype(see page 856)
- Package(see page 800)
- Working with Profiles(see page 2036)

Creating smart packages

A smart package can be created in one of the following ways:

- **From the shortcut menu of an element**. The newly created smart package is empty.
- **From the search results** (by saving either the search options or the search results as a smart package). The newly created smart package has either dynamic or static contents.

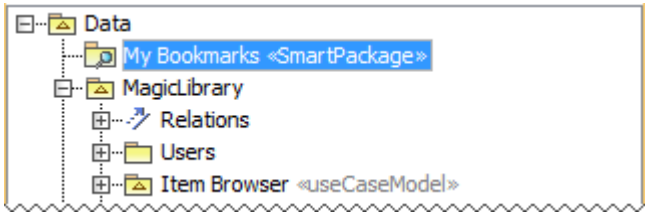
To create a smart package from the shortcut menu of an element

1. In the Containment tree, right-click the element. The shortcut menu opens.

   The element should be a namespace, such as a package, a class, or a requirement.

2. Select **Create Element** > **Smart Package**.
3. Type a name for the new smart package.
4. Press Enter to finish.

The new smart package is created, and you can proceed to managing its contents(see page 828) and reviewing its specific properties(see page 831).



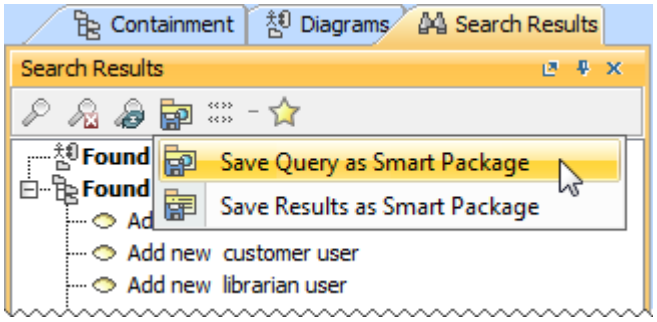
To create a smart package from the search results

1. Open the **Find** dialog, define search options, and perform the search. The search results are displayed in the **Search Results** tab on the Model Browser.

   For more information about defining search options and analyzing the search results, refer to the Find dialog(see page 217) and Search Results tab(see page 47).

2. On the toolbar of the **Search Results** tab, click and select one of the following commands:

   - **Save Query as Smart Package** to create a new smart package with dynamic contents. The search options will be saved as the **Query** property value.
   - **Save Results as Smart Package** to create a new smart package with static contents. The search results will be saved as values of the **Additional Elements** property.



   The element Selection dialog opens.

3. In the dialog, select an owner of the new smart package and click **OK**.
4. In the Containment tree, type a name for the new smart package.
5. Press Enter to finish.

The new smart package is created, and you can proceed to managing its contents(see page 828) and reviewing its specific properties(see page 831).

Related Pages

- Model Elements(see page 628)
- Stereotype(see page 856)
- Package(see page 800)
- Working with Profiles(see page 2036)

Managing contents of smart package

The following sections outline how to manage the smart package's contents once it is created.

- Including elements into a smart package(see page 828)
- Removing elements from a smart package(see page 829)
- Freezing the contents of a smart package(see page 830)
- Making a snapshot of the smart package(see page 831)

Including elements into a smart package

An element can be included in the smart package in one of two ways:

- *Manually* - the user selects to include the particular element. Manually included elements constitute the static contents.
- *Automatically* - the element meets the set of criteria defined by the user. Automatically included elements constitute the dynamic contents.

The membership in the smart package is not the UML ownership association. The owner of the element will not change after the element is included in the smart package.

To include an element into a smart package manually

Do either:

- In the Containment tree, select an element and drag it to the smart package.
  See the element included into the contents of the smart package. The element becomes a value of the **Additional Elements** property as well.

  You can include more than one element at once.
  - To include *adjacent* elements, select a single element, and then hold down the Shift key while you click any other elements you want to add.
  - To include *nonadjacent* elements, select a single element, and then hold down the Ctrl key while you click any other elements you want to add.

- Open the Specification window of the smart package and click the cell of the **Additional Elements** property value. Then click ... and in the open dialog, select one or more elements. Click **OK** when you are done. See the element(s) included in the contents of the smart package.

To include elements into a smart package automatically

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

1. Open the Specification window of the smart package.
2. Under the **Contents** category, click the cell of the **Query** property value.
3. Click . The **Query** dialog opens.
4. In the dialog, define a set of criteria (query) for gathering the elements into the contents of the smart package.

   For instructions about defining criteria in the **Query** dialog, refer to Specifying Criteria for Querying Model.

   You can select to show implied relationships(see page 335) (logical relationships calculated from inherited or inner elements of the model element) to provide additional information.

5. Close the Specification window. See the contents of the smart package updated with elements that meet the defined set of criteria.

   If the contents of the smart package are not updated, select **View** > **Refresh** from the main menu.

   - If the smart package contains manually included elements (values of the **Additional Elements** property), they are joined with the automatically included elements, skipping duplicates.
   - If the smart package has excluded elements (values of the **Excluded Elements** property), they do not appear in the dynamic contents of the smart package.

Removing elements from a smart package

To remove an element from a smart package

Do either:

- In the Containment tree, under the smart package, right-click the element. From the shortcut menu, select **Delete from Contents** or press **Delete**.

  You can exclude more than one element at once.
  - To exclude *adjacent* elements, select a single element, and hold down the Shift key while you click other elements you want to exclude.
  - To exclude *nonadjacent* elements, select a single element, and hold down the Ctrl key while you click other elements that you want to exclude.

- Open the Specification window of the smart package and click the cell of the **Excluded Elements** property value. Then click and in the open dialog, select one or more elements. Click **OK** when you are done.
- Open the Specification window of the smart package and click the cell of the **Additional Elements** property value. Then click and in the open dialog, unselect one or more elements. Click **OK** when you are done.

  This is only valid for manually added elements.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

See the element removed from the contents of the smart package, though it is not removed from the model. You can still see the element displayed under the owner of the element in the Containment tree.

Freezing the contents of a smart package

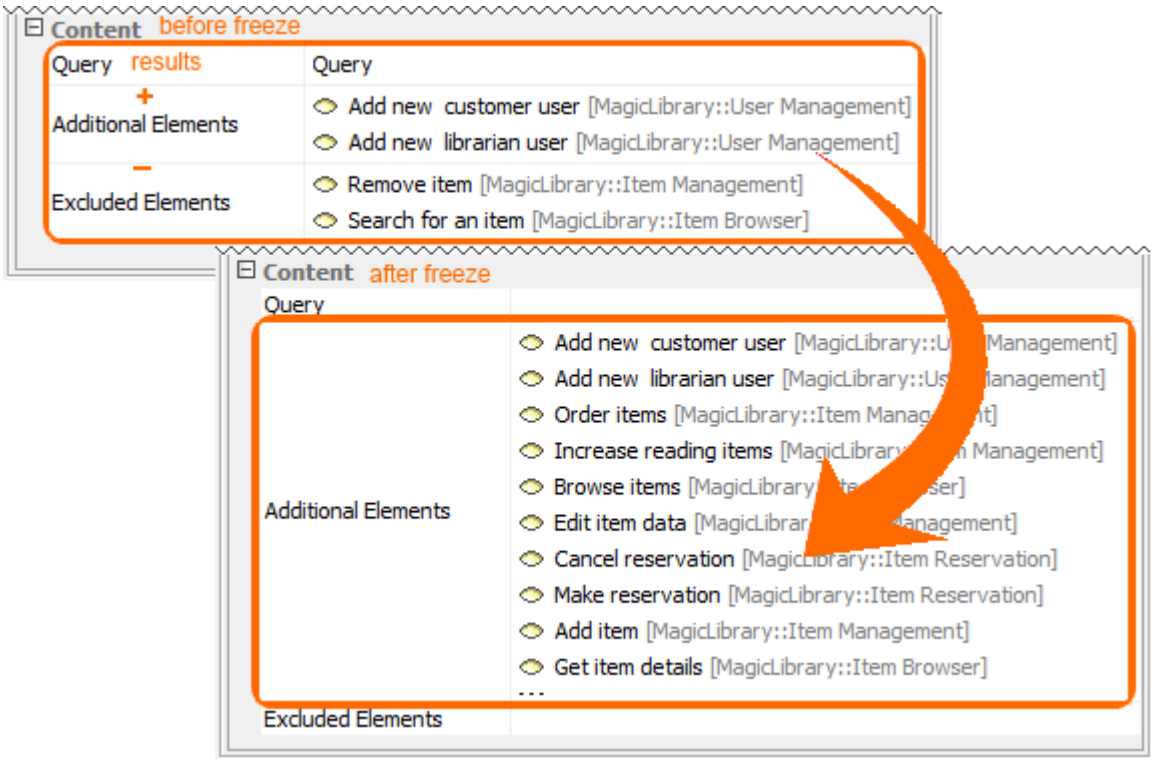
If you no longer need the contents of a smart package to be dynamic, you can simply freeze it. The dynamic contents will be converted to the static list of elements and joined with the list of manually added elements (values of the **Additional Elements** property), skipping both duplicates and elements defined as excluded (values of the **Excluded Elements** property).

To freeze the contents of a smart package

- In the Containment tree, right-click the smart package with dynamic contents and from the shortcut menu select **Freeze Contents**.

  See that there are no changes visible in the Containment tree. Open the Specification window of the smart package to see that the values of both the **Query** property and the **Excluded Elements** property have passed into empty, and the value of the **Additional Elements** property stores the new list of elements.

  > The contents of the smart package will no longer be automatically updated after relevant changes in the model.



Making a snapshot of the smart package

You can create a copy of the smart package with the static contents, that is, a snapshot.

To make a snapshot of a smart package

1. In the Containment tree, right-click the smart package with dynamic contents and from the shortcut menu select **Snapshot Contents**. The element Selection dialog opens.
2. In the dialog, select an owner of the new smart package and click **OK**.

   See the new smart package created under the selected owner in the Containment tree with the name *Snapshot of <smart package name>*, for example, *Snapshot of My Bookmarks*. The contents of the snapshot equal the contents of the smart package, but the list of elements is static.

Related Pages

- Model Elements(see page 628)
- Stereotype(see page 856)
- Package(see page 800)
- Working with Profiles(see page 2036)

Smart package properties

Every smart package has several properties that allow you to specify the contents of the smart package. These properties are described in the following table.

Property	Description
Query	Stores a set of criteria for automatic inclusion of elements into the contents of the smart package. For more information, see Including elements into a smart package automatically(see page 828).
Additional Elements	Stores the list of the elements manually included into the contents of the smart package. For more information, see Including elements into a smart package manually(see page 828).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Property	Description
Excluded Elements	Stores the list of the elements excluded from the contents of the smart package. For more information, see Removing elements from a smart package.(see page 829)

Related Pages

- Model Elements(see page 628)
- Stereotype(see page 856)
- Package(see page 800)
- Working with Profiles(see page 2036)

Using smart packages in your model

To better understand the usability of the smart packages, please see the following case studies.

Case study #1: Gathering use cases

The case study uses the sample project *use case diagram.mdzip*, which can be found in *<MagicDraw installation folder>\samples\diagrams*.

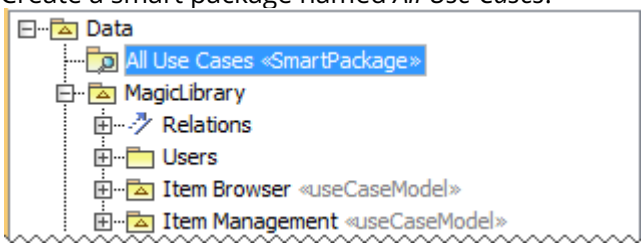
For example, we have a lot of use cases owned by different system boundaries in the project.

We need all these use cases to be in a single package, except for several particular ones. Also, we need all newly created use cases to be automatically included into this package.

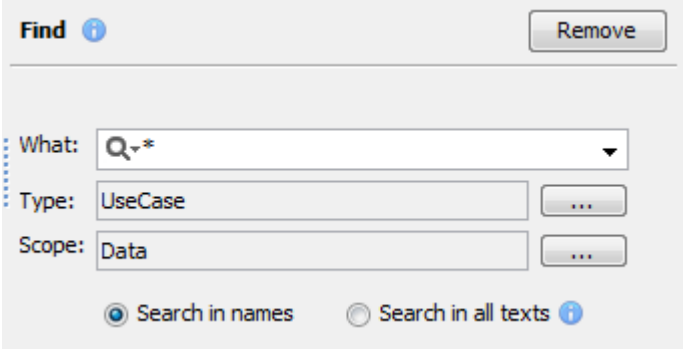
In this case, the smart packages feature is very useful. We will create a smart package with dynamic contents to gather all use cases in the model, then demonstrate how it manages further changes in the model. Finally, we will create a snapshot of the smart package to have a static list of use cases as a milestone of the model development.

To accomplish this, do the following:

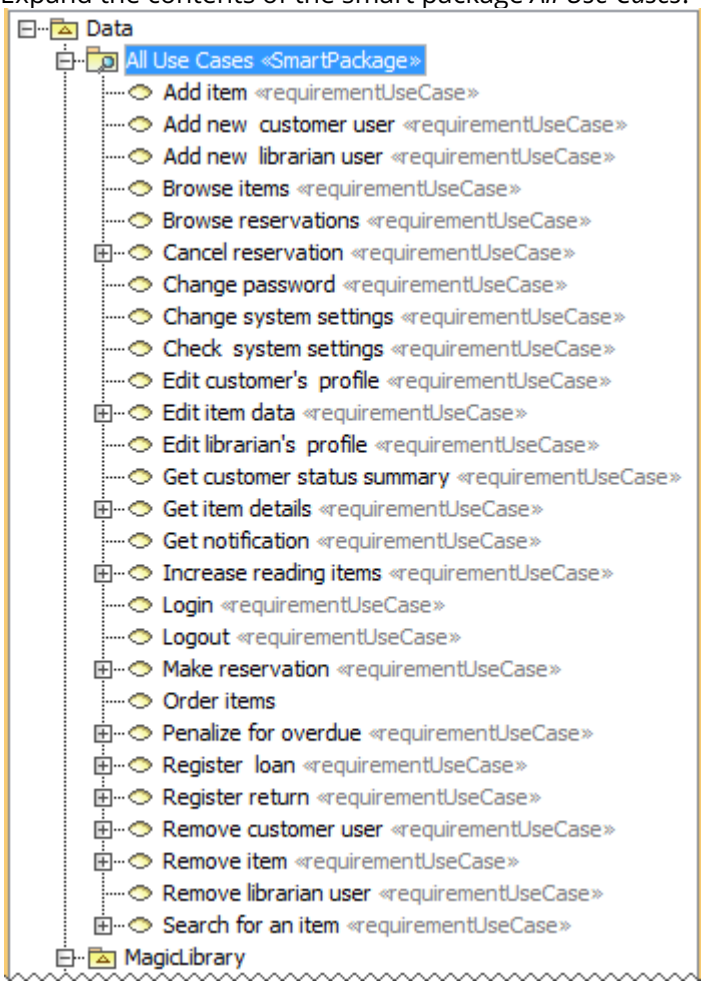
1. Create a smart package named *All Use Cases*.



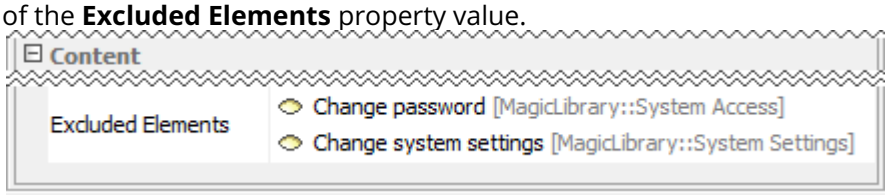
2. Define the criteria for gathering the contents of the smart package. Specify the search options to find all use case type elements from the root package *Data*.



3. Expand the contents of the smart package *All Use Cases*.

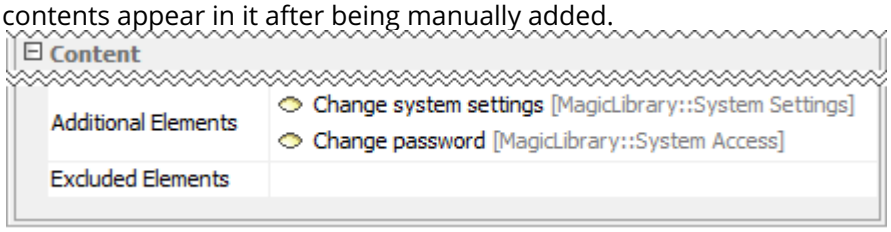


4. Exclude use cases *Change password* and *Change system settings* from the contents. See the contents of the smart package shortened.
5. Open the Specification window of the smart package and see the excluded use cases in the cell of the **Excluded Elements** property value.

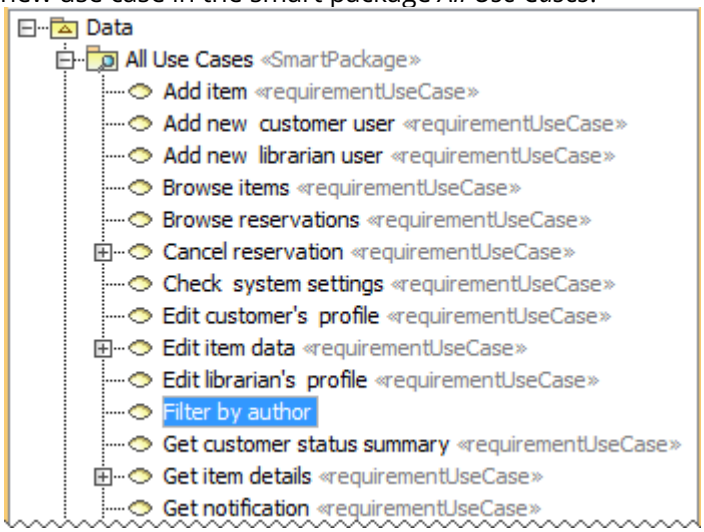


6. Drag use cases *Change password* and *Change system settings* to the smart package. See them in the contents of the smart package again.

7. Open the Specification window of the smart package again and see these manually included use cases in the cell of the **Additional Elements** property value. Elements excluded from the contents appear in it after being manually added.



8. Create a new use case named *Filter by author* under the system boundary *Item Browser*. See the new use case in the smart package *All Use Cases*.



9. Create a snapshot of the smart package *All Use Cases* to have a static list of use cases as a milestone of the model development.

10. Expand the contents of the snapshot and see that it equals the contents of the smart package *All Use Cases*.

Case study #2: Performing the requirements coverage analysis

> ⚠ The case study uses the sample project *hybrid sport utility vehicle.mdzip*, which can be found in *<MagicDraw installation folder>\samples\SysML*, if the SysML plugin is installed.

Here is another case where the smart package feature is very helpful: we need to have all unsatisfied requirements in a separate package. Also, we need requirements to automatically disappear from this package after becoming satisfied.

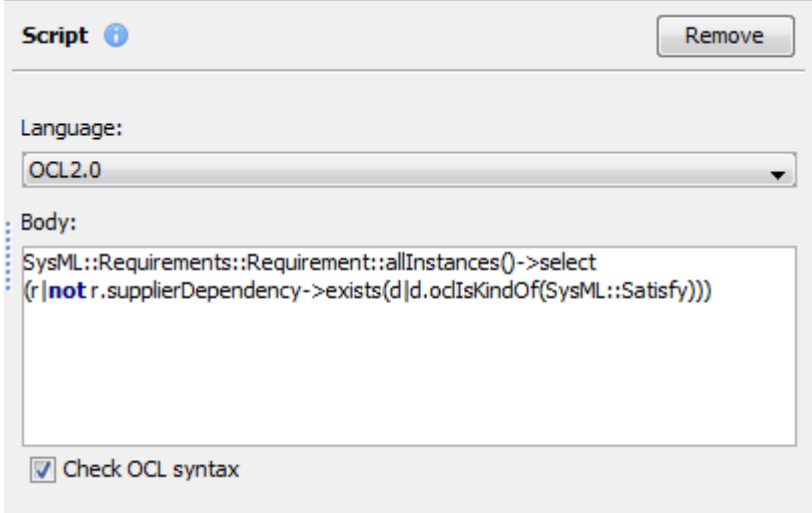
We will create a smart package with dynamic contents to gather all the unsatisfied requirements in the model and a dependency matrix for performing the requirements coverage analysis. Then we will demonstrate how both the smart package and the dependency matrix reflect the transition of a requirement to satisfy.

Let's do the following:

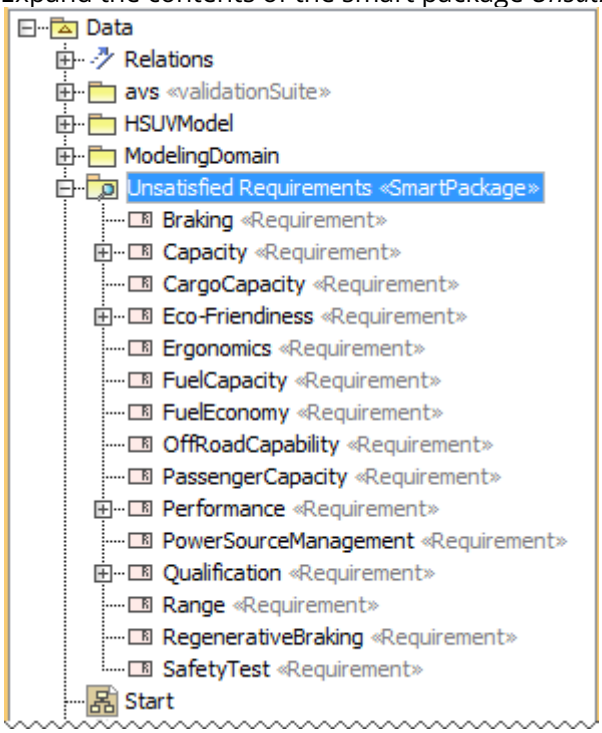
1. Create a smart package named *Unsatisfied Requirements*.

2. To gather the contents of the smart package, add a new script operation (in the **Expert** mode of the **Query** dialog) and define the following OCL 2.0 expression as the criteria:

```
SysML::Requirements::Requirement::allInstances()->select(r|not
r.supplierDependency->exists(d|d.oclIsKindOf(SysML::Satisfy)))
```



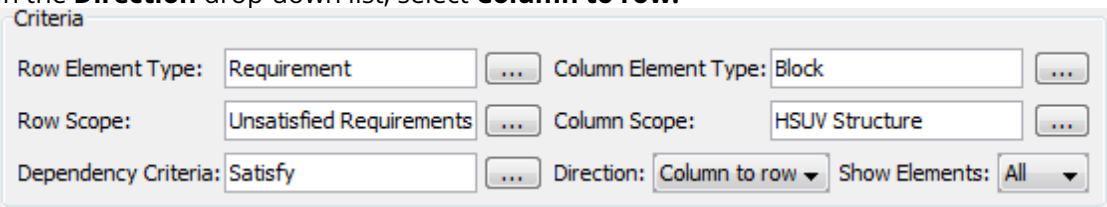
3. Expand the contents of the smart package *Unsatisfied Requirements*.



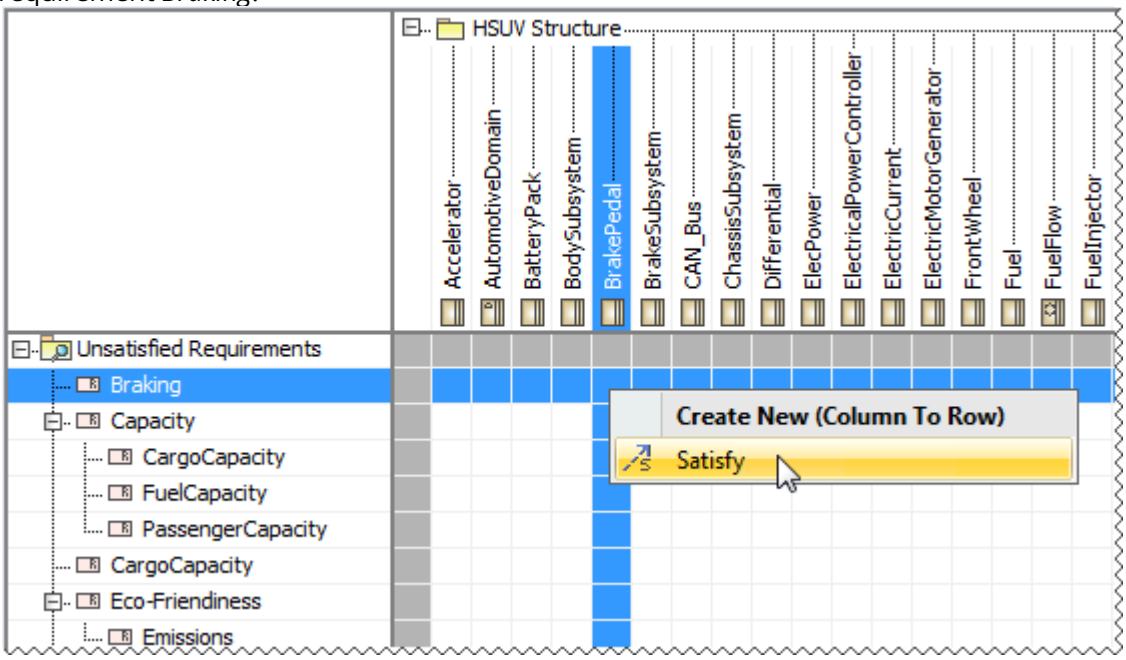
4. Create a dependency matrix and define the following criteria:
   - Specify Requirement as the row element type
   - Specify Block as the column element type
   - Specify the smart package *Unsatisfied Requirements* as the row scope
   - Specify the package *HSUV Structure* as the column scope
   - Specify the Satisfy relationship as dependency criteria

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- In the **Direction** drop-down list, select **Column to row.**



5. On the dependency matrix, create a Satisfy relationship between the block *BrakePedal* and the requirement *Braking*.



The requirement becomes satisfied and thus disappears from the contents of the smart package *Unsatisfied Requirements* and from the dependency matrix as well.

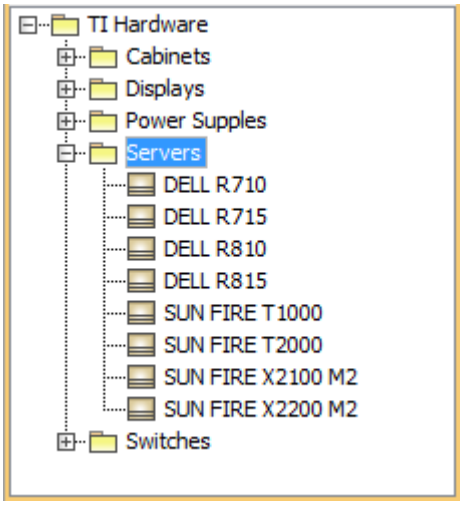
Case Study #3: Configuration management of the complex system - creating dynamic configuration catalogs

The efficient configuration management process is a challenge in the evolution of any industrial scale product family. Smart packages are a real-life out of the box solution supporting the configuration management approach.

Now study the case that illustrates the efficient management of the complex system configurations with the help of smart packages. We have a library (static package) of system components, which we need to see in several different views of the model, that is, catalogs, according to their characteristics. Using the catalogs will not extend the scope of the model, since they do not require duplication of the

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

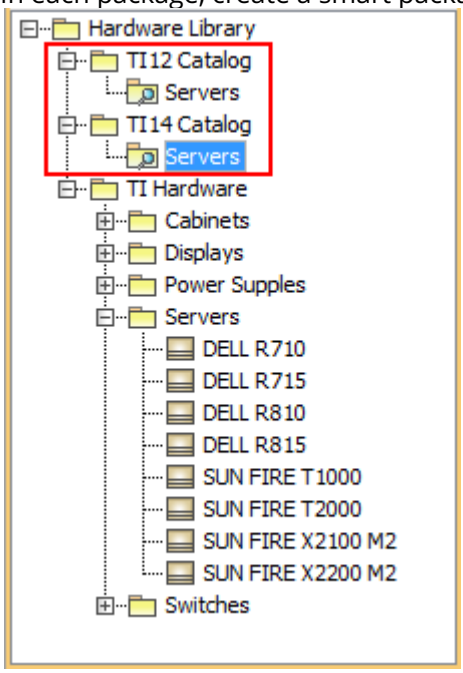
elements.



We will create two dynamic system configuration catalogs, that is, smart packages with dynamic contents, to gather the servers from the library *TI Hardware* according to the configuration version defined in a tag value of their specification.

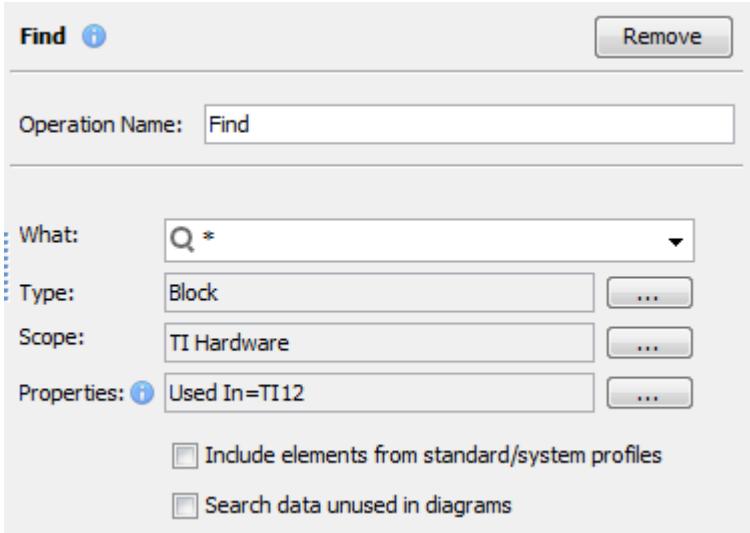
Do the following:

1. Create two packages: *TI12 Catalog* and *TI14 Catalog*.
2. In each package, create a smart package named *Servers*.

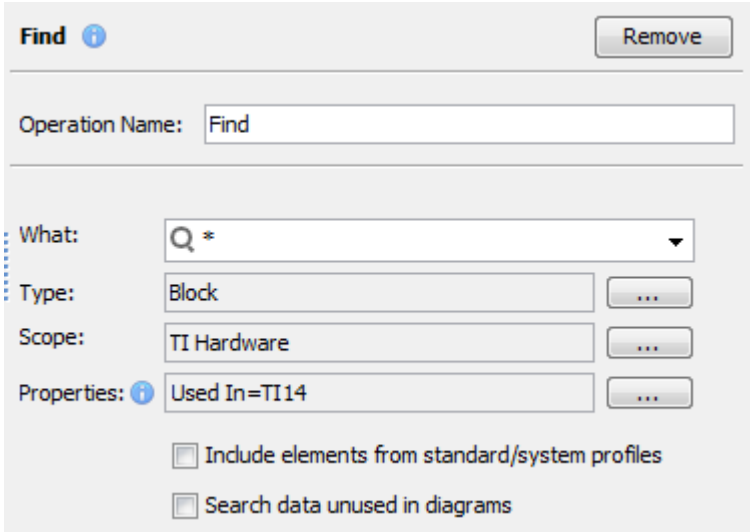


3. Define criteria for gathering the contents of the smart package *Servers* in *TI12 Catalog*. Specify search options to find in the package *TI Hardware*, all block type elements with tag value Used

In=TI12.

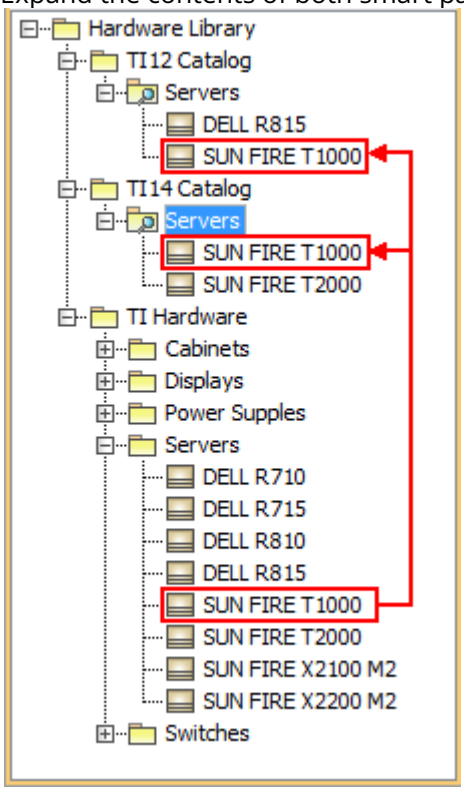


4. Define criteria for gathering the contents of the smart package Servers in TI14 Catalog. Specify search options to find in the package TI Hardware, all block type elements with tag value *Used In=TI14*.



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

5. Expand the contents of both smart packages.



> ⚠ The block *SUN FIRE T1000* appears in both *TI12 Catalog* and *TI14 Catalog*, since it has both tag values *Used In=TI12* and *Used In=TI14*.

6. Also, you can add a block to a smart package manually. Just drag the block to the smart package.

Related Pages

- Model Elements(see page 628)
- Stereotype(see page 856)
- Package(see page 800)
- Working with Profiles(see page 2036)

Smart package-related performance issues

When working with large projects, smart packages can sometimes cause performance issues. They can influence the time of a search, commit, and derived properties related operations. However, there is a way to identify which smart packages or properties cause the problem by getting notifications about their evaluation time.

To get notifications about the evaluation time of smart package content and derived properties

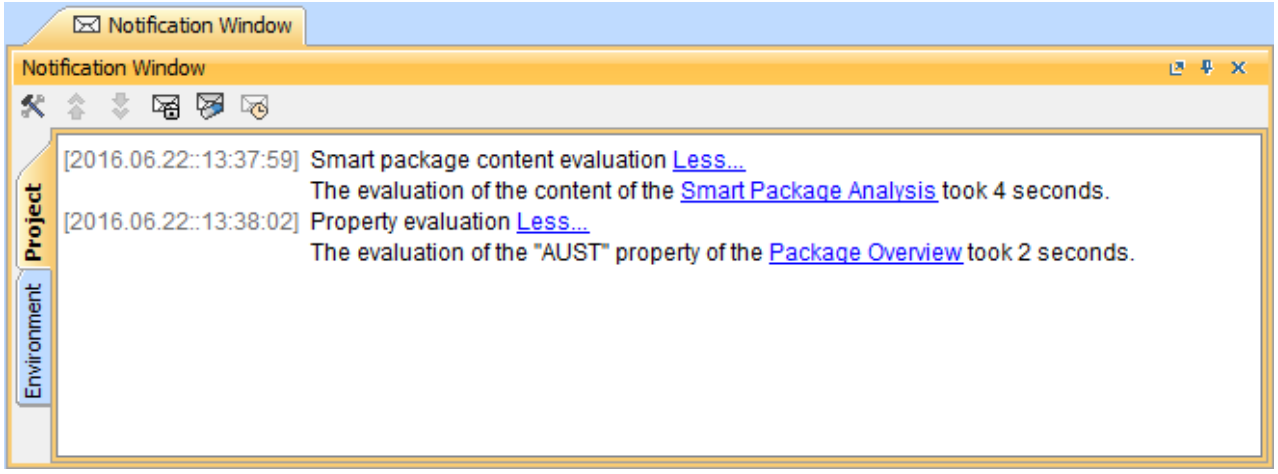
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

1. Go to *<modeling tool installation directory>/bin* and open the properties file of your modeling tool.
2. Enter the *md.expression.evaluation.duration.notification.threshold* Java argument and define the threshold time in seconds for getting notifications, e.g.:
   `-Dmd.expression.evaluation.duration.notification.threshold=30`

> **Note**
> The default value of the *md.expression.evaluation.duration.notification.threshold* argument is 60 seconds. If you add the argument to the properties file of your modeling tool but do not define its value, the default value is used.

3. Save and close the file
4. Restart your modeling tool if it's open.

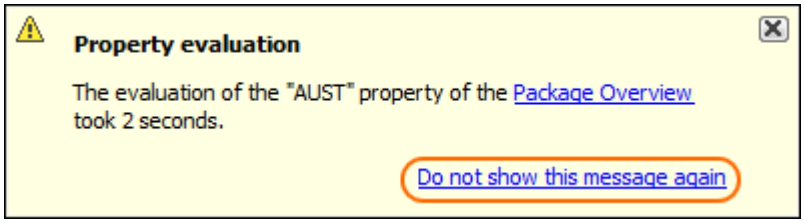
Your modeling tool is now configured to show notifications about the evaluation time of smart package content and derived properties. The notifications are shown in the **Notification Window** area at the bottom of a modeling tool window.



Smart package content and property evaluation notifications - the time threshold applied is 1 second.

To disable notifications

- Click **Do not show this message again** when a new notification appears on the right side of a modeling tool window.



Disabling notifications from a modeling tool.

This action disables notifications about the evaluation time of both smart package content and derived properties.

Tip
You can increase the performance of your modeling tool by following these recommendations:

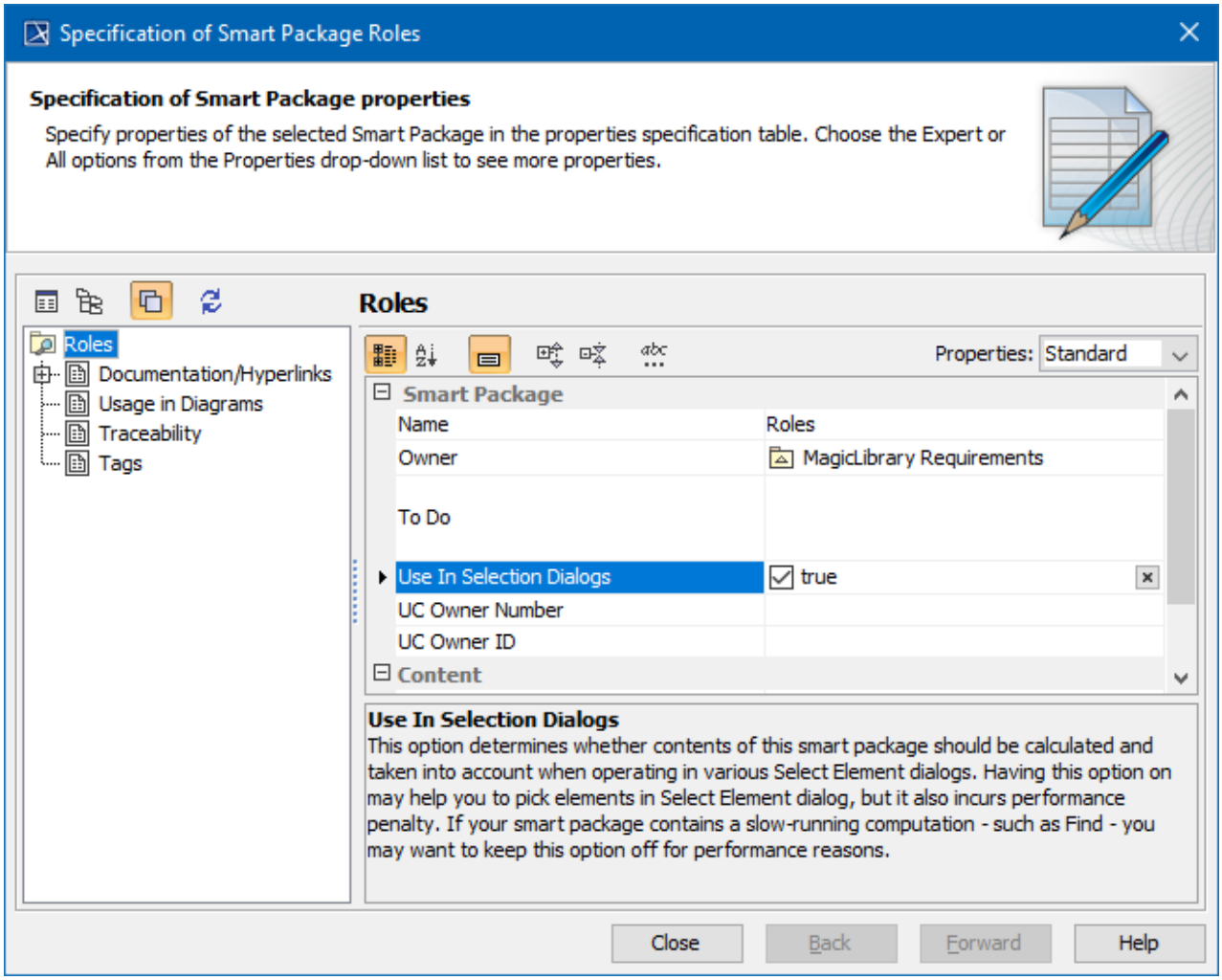
- Avoid using smart packages with Find queries, especially if they are used to find common elements (e.g., Class, Package, Block, etc.).
- Avoid using smart packages with Find queries which have wide scope.
- Avoid nesting smart packages, i.e., using one smart package as the part of another smart package's content.

Tip
You can prevent a number of performance issues stemming from the use of smart packages by using tables and matrices with the specified expression(see page 839) instead.

Performance contents

In the **Smart Package** Specification window(see page 913), you can take advantage of a new property, **Use In Selection Dialogs,** which allows you to work better with your smart packages. Previously, smart packages slowed down the opening of **Selection** dialog boxes. **Use In Selection Dialogs** allows users to exclude the smart package from the calculation of elements to display in **Selection** dialog boxes (e.g. Select Elements(see page 1031)).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Related pages

- Model Elements(see page 628)
- Stereotype(see page 856)
- Package(see page 800)
- Working with Profiles(see page 2036)

State Machine

A State Machine is a type of Behavior(see page 662) that describes how a system or system component (represented by a block) reacts to events or changes in its environment. State machines are typically the classifier behavior of a block; they describe the whole life behavior of the block.

State machines consist of State(see page 848), Transitions and Pseudo States(see page 843). A state represents a significant and persistent condition of a block. A block will remain in a given state until triggered to transition to another state. When the triggering event occurs, the state machine will transition to another state within the region based on the triggers and guard condition specified for the transition. State machines can invoke other behaviors, including activities, interaction and other state machines. These behaviors may be invoked from within the state as entry, do, or exit behaviors or during the transition as an effect behavior. Pseudo states describe the initialization and termination of

region and may be used to control the path taken by a given transition. A state machine is created when creating a State Machine diagram(see page 308).

Specify the State Machine element in the State Machine Specification window(see page 913).

Pseudo States

The Pseudo state is typically used to connect multiple transitions into more complex state transition's paths. For example, by combining a transition entering a fork pseudo state with a set of transitions exiting the fork pseudo state, we get a compound transition that leads to a set of orthogonal target states.

You can specify pseudo state properties(see page 921) in the pseudo state Specification window(see page 913). In the same window, you can find the description of each property. Descriptions are presented in the description area of the Specification window.

Related Pages

- Model Elements(see page 628)
- Specification Window(see page 913)

Related pages

Initial state

Every object belongs to a particular state as soon as it is created. So, it is useful to explicitly show that particular state. A solid filled circle represents the initial state of an object. There can only be one initial state for an object. The initial state denotes the starting place for a transition, the target of which is a composite state.

Related Pages

- Model Elements(see page 628)
- Specification Window(see page 913)

Related pages

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Final state

The final state symbol (a circle surrounding a smaller solid circle) is used to represent the object destruction. The final state is optional in the diagram because there is a system that runs without interruption after the start of the activities. Also, there can be several final states in the same state diagram, denoting that the life of the
object can finish depending on several conditions.

Related Pages

- Model Elements(see page 628)
- Specification Window(see page 913)

Related pages

Terminate

Entering a terminate pseudo state implies that the execution of the state machine by means of its context object is terminated. The state machine does not exit any states nor does it perform any exit actions other than those associated with the transition leading to the terminate pseudo state.

Related Pages

- Model Elements(see page 628)
- Specification Window(see page 913)

Related pages

Entry Point

An entry point connection point reference as the target of a transition implies that the target of the transition is the entry point pseudo state as defined in the submachine of the submachine state. As a result, the regions of the submachine state machine are entered at the corresponding entry point pseudo states.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related Pages

- Model Elements(see page 628)
- Specification Window(see page 913)

Related pages

Exit Point

An exit point connection point reference as the source of a transition implies that the source of the transition is the exit point pseudo state as defined in the submachine of the submachine state that has the exit point connection point defined. When a region of the submachine state machine has reached the corresponding exit points, the submachine state exits at this exit point.

Related Pages

- Model Elements(see page 628)
- Specification Window(see page 913)

Related pages

Deep History

The Deep History represents the most recent active configuration of the composite state that directly contains this pseudo state; e.g. the state configuration that was active when the composite state was last exited. A composite state can have at most one deep history vertex.

Related Pages

- Model Elements(see page 628)
- Specification Window(see page 913)

Related pages

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Shallow History

The Shallow History represents the most recent active substate of its containing state (but not the substates of that substate). A composite state can have at most one shallow history vertex. A transition coming to the shallow history vertex is equivalent to a transition coming to the most recent active substate of a state.

Related Pages

- Model Elements(see page 628)
- Specification Window(see page 913)

Related pages

Junction

The junction vertices are semantic-free vertices that are used to chain multiple transitions together. They are used to construct the compound transition paths between states. For example, a junction can be used to combine multiple incoming transitions into a single outgoing transition representing a shared transition path (this is known as merge). Conversely, it can be used to split an incoming transition into multiple outgoing transition segments with different guard conditions.

Related Pages

- Model Elements(see page 628)
- Specification Window(see page 913)

Related pages

Choice

The choice vertices, when reached, result in the dynamic evaluation of the guards or the triggers of its outgoing transitions. This realizes a dynamic conditional branch. It allows splitting of transitions into multiple outgoing paths such that the decision on which path to take may be a function of the results of prior actions performed in the same run-to-completion step.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related Pages

- Model Elements(see page 628)
- Specification Window(see page 913)

Related pages

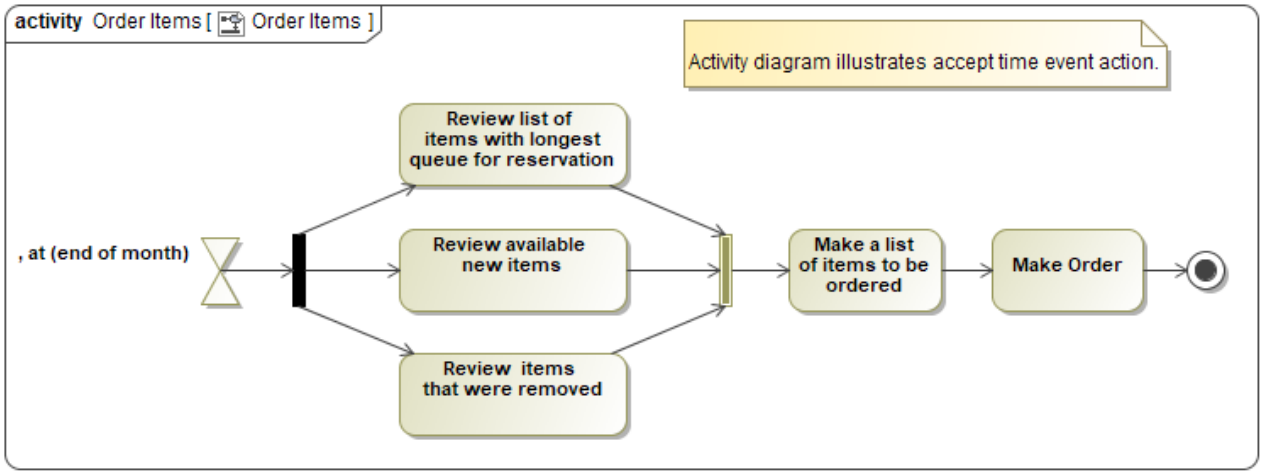
Fork and Join

The fork vertices are used to split an incoming transition into two or more transitions terminating on the orthogonal target vertices (i.e., vertices in different regions of composite state). The segments going out of a fork vertex must not have guards or triggers.

The join vertices are used to merge several transitions emanating from the source vertices in different orthogonal regions. The transitions entering a join vertex cannot have guards or triggers.

1. A Fork Node can have exactly one incoming Activity Edge (or Transition), though it may have multiple outgoing Activity Edges (or Transitions).
2. A Join Node can have exactly one outgoing Activity Edge (or Transition) but may have multiple incoming Activity Edges (or Transitions).

The Fork and Join are represented by the same symbol. Although, you can set the **Use different Fork/ Join and Decision/Merge notations** project option(see page 2221) to draw different notations for the **Fork** and **Join**.



An example shows the different Fork and Join notations

Related pages

- Activity diagram(see page 311)
- Interaction Overview diagram(see page 332)

- State Machine diagram(see page 308)
- Protocol State Machine diagram(see page 310)
- UML elements(see page 628)
- Specification Window(see page 913)

State

A State is a condition during the lifetime of an object or an interaction during which the object meets certain conditions, performs an action, or waits for an event. The State is defined by the concepts of duration and stability. An object can not be in an unknown or undefined State. A State may have two compartments to provide more information about that State:

- The first compartment is the name compartment, it contains the State name, for example: running, going up.
- The second compartment is the activity compartment, it contains the events and actions of the State.

You can specify State properties in the State Specification window. In the same window, you can find the description of each property. Descriptions are presented in the description area of the Specification window

Simple State

A simple State is a State without any regions, internal Vertices or Transitions.

Composite State

You can create a composite State by converting a simple State(see page 0).

A composite State either contains one region or is decomposed into two or more orthogonal regions. Each region has a set of mutually exclusive disjoint subvertices and a set of transitions. A given State may only be decomposed in one of these two ways.

Any State enclosed within a region of the composite State is called a subState of that composite State. It is called a direct subState when it is not contained by any other State; otherwise it is referred to as an indirect subState.

Each region of the composite State may have an initial pseudoState and a final State. A transition to the enclosing State represents a transition to the initial pseudoState in each region.

You can specify composite State properties in the State Specification window. In the same window, you can find the description of each property. Descriptions are presented in the description area of the Specification window.

Orthogonal State

An orthogonal State is a composite State with at least 2 regions.

Submachine State

A submachine State specifies the insertion of the specification of a submachine State Machine(see page 842). The State machine that contains the submachine State is called the containing State machine. The

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

same State machine can be a submachine more than once in the context of a single containing State machine.

The submachine State is semantically equivalent to a composite State. The regions of the submachine State machine are the regions of the composite State. The entry, exit, behavior actions, and internal transitions, are defined as part of the State. The submachine State is a decomposition mechanism that allows factoring of the common behaviors and their reuse.

You can specify submachine properties in the State Specification window. In the same window, you can find the description of each property. Descriptions are presented in the description area of the Specification window.

On this page

- Simple State(see page 848)
- Composite State(see page 848)
- Orthogonal State(see page 848)
- Submachine State(see page 848)

Related Pages

- Model Elements(see page 628)
- Stereotype(see page 856)
- State Machine diagram(see page 308)

Related References

Assigning behavior to state

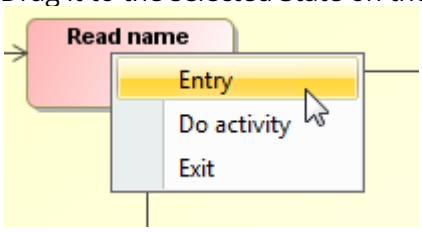
You can define a behavior to be executed correspondingly to the listed events while being in the State whenever the State is entered and exited.

To assign a behavior to a State by dragging

1. In the Model Browser, select a behavior type element (Activity, Function Behavior, Interaction, Opaque Behavior, State Machine, or Protocol State Machine) or a Signal element.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

2. Drag it to the selected State on the diagram. The list of activities appears.



- Click a required activity.
  The behavior is assigned to the selected State's activity.

> - If you drag a behavior type element (Activity, Function Behavior, Interaction, Opaque Behavior, State Machine, or Protocol State Machine), the Activity diagram under the Activity element with a Call Behavior Action element are created automatically.
> - If you drag a Signal element, the Activity diagram under the Activity element with a Send Signal Action element are created automatically.

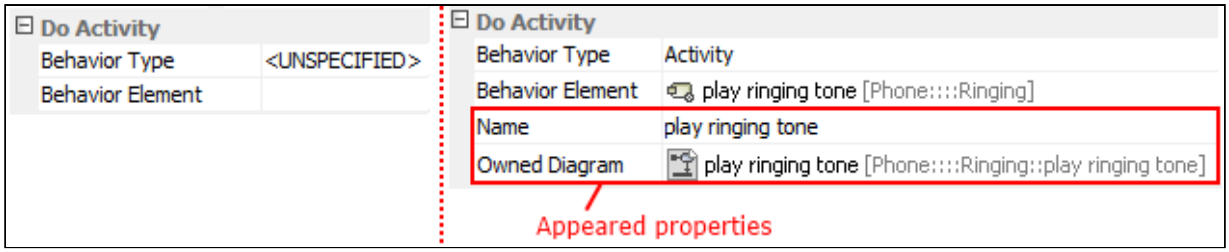
To assign a behavior to a State by using the transition's Specification window

---

1. Open the State's Specification window.
2. In the **Entry, Do Activity**, or **Exit** category, click the **Behavior Type** property value cell. The list of available behavior types opens. Select the behavior type from the list.

> If you do not see the **Behavior Type** property, click the **+** button near the **Entry**, **Do Activity**, or **Exit** category to expand its content.

3. According to the selected behavior type, the **Behavior Element** value is defined automatically. Additional properties **Name** and **Owned Diagram** appears. Specify these properties if it is needed.
4. Click the **Close** button after you have defined desired properties.



If a behavior of an Activity, Interaction, Protocol State Machine, or State Machine type is assigned to an Entry, Exit, or Do activity, a diagram for that behavior is created automatically.

Related Pages

- Model Elements(see page 628)
- Stereotype(see page 856)
- State Machine diagram(see page 308)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related pages

Managing regions

To create a region

Do one of the following:

- In the Containment tree, right-click the State, select **Create Element**, and then click **Region**.
- On the diagram, right-click the State, then select the **Add New Region**.
- On the diagram, select the State symbol and click the **Create Region** button ⊕.
- In the State Specification window, click the **Inner Elements** property group, then click **Create** and select **Region**. Specify the region and click **Close** or **Back**.

To remove a region

Do one of the following:

- On the diagram, right-click the State and select **Remove Region**.
- In the State Specification window, click the **Inner Elements** property group. Then click the region and click the **Delete** button.

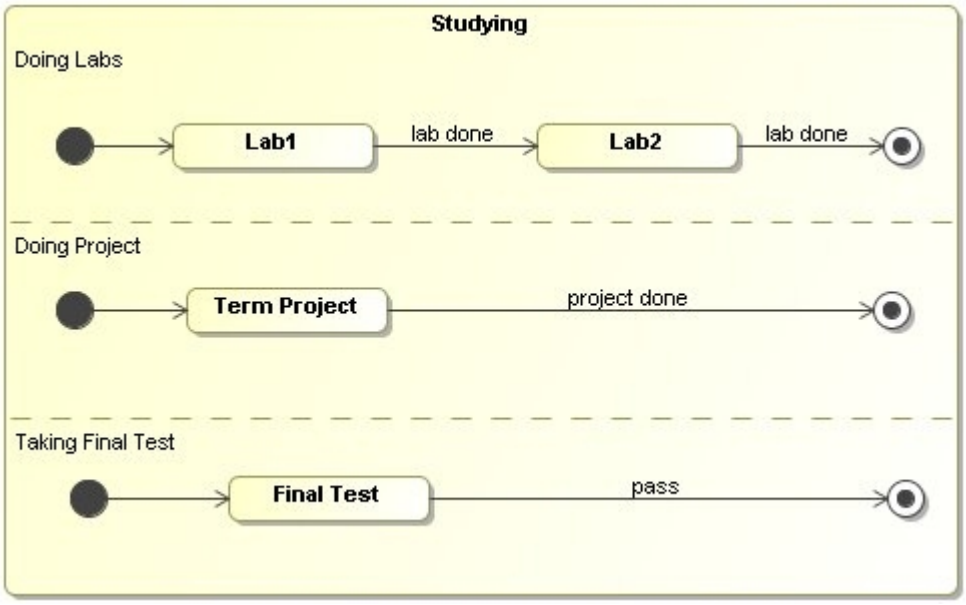
To insert a State to the region quickly

- In the Containment tree or on the diagram, select the State, and drag it to the State's region.

To display or hide the region name

1. On the diagram, right-click the State and select **Symbol Properties**.
2. In the **Symbol Properties** dialog, in the right-top corner, click to expand the menu, and select the **Expert** mode.
3. Select or clear the **Show Region Name** check box.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Related Pages

- Model Elements(see page 628)
- Stereotype(see page 856)
- State Machine diagram(see page 308)

Related pages

Changing state to composite/submachine/orthogonal state

You can change your current State to a simple State, composite State, orthogonal State, and submachine State.

To change a State to a composite State

- Drag a vertex on a simple State symbol. A region is created and the simple State is converted to a composite State.

To change a State to a orthogonal State

- Add two regions to the State. Read more about creating regions(see page 0).

To change a State to a submachine State

1. On the diagram, right-click the State, then on the shortcut menu, click **Submachine**.
2. In the dialog, click to select a submachine that you wish to apply on the State or click the **Create** button, to create the submachine.

To change a State to a submachine State, the State must contain no regions.

Related Pages

- Model Elements(see page 628)
- Stereotype(see page 856)
- State Machine diagram(see page 308)

Related pages

Connection Point Reference

The Connection Point Reference represents an entry to or an exit from the submachine State. It can be used as the source or target of a transition.

The following procedures show you how to work with Connection Point Reference:

- Drawing the Connection Point Reference(see page 853)
- Checking the assigned entry/exit point to Connection Point Reference(see page 853)
- Assigning the entry/exit points to the Connection Point Reference(see page 854)
- Displaying the Connection Point References(see page 854)

Drawing the Connection Point Reference

To draw the Connection Point Reference on the submachine State

1. On the State diagram palette, click the **Connection Point Reference** button.
2. Click the shape of the submachine State. The **Select Entry/Exit Point** dialog opens.
3. Select *entry point* to define the entry into the submachine State, or *exit point* to define the exit from the submachine State. The Connection Point Reference is drawn on the submachine State with a defined entry or exit point.

Checking the assigned entry/exit point to Connection Point Reference

To see the assigned entry/exit point

- Open the **Connection Point Reference** dialog. The **Entry** or **Exit** properties display the defined entries.

Assigning the entry/exit points to the Connection Point Reference

To assign the entry/exit points to the Connection Point Reference

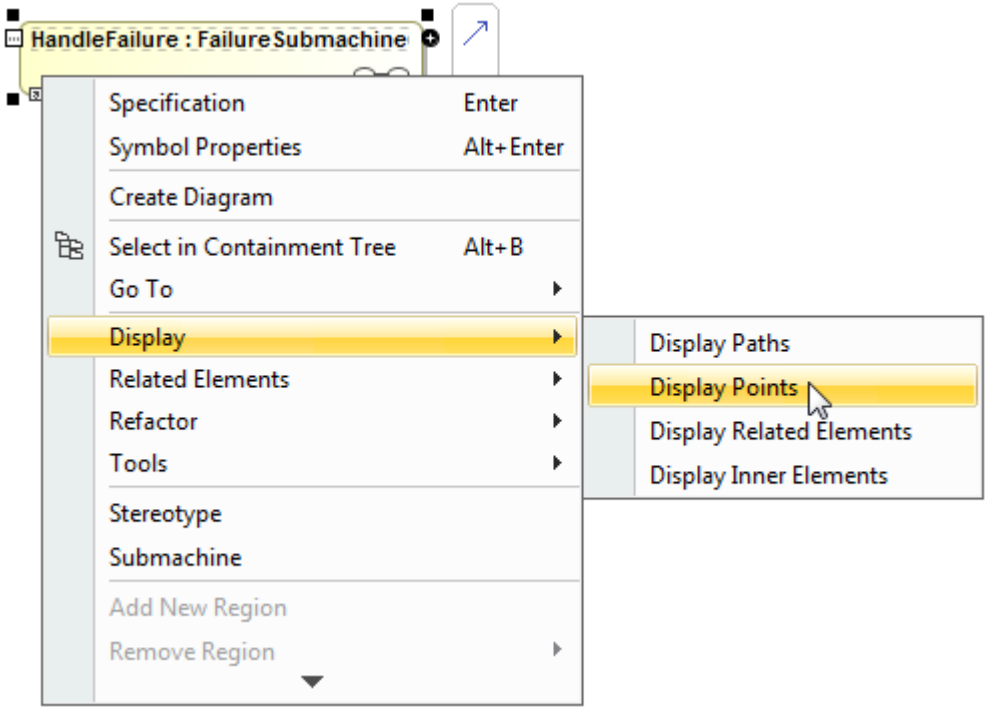
- Right-click the Connection Point Reference shape, and, on the shortcut menu ,click **Select Entry/ Exit Point**.

> In the **Select Entry/Exit Point** dialog, only the entry and exit points that are created at the same State Machine as the submachine State are listed. If there are no entry/exit points at the same State machine, the **Select Entry/Exit Point** dialog is not opened when drawing the Connection Point Reference.

Displaying the Connection Point References

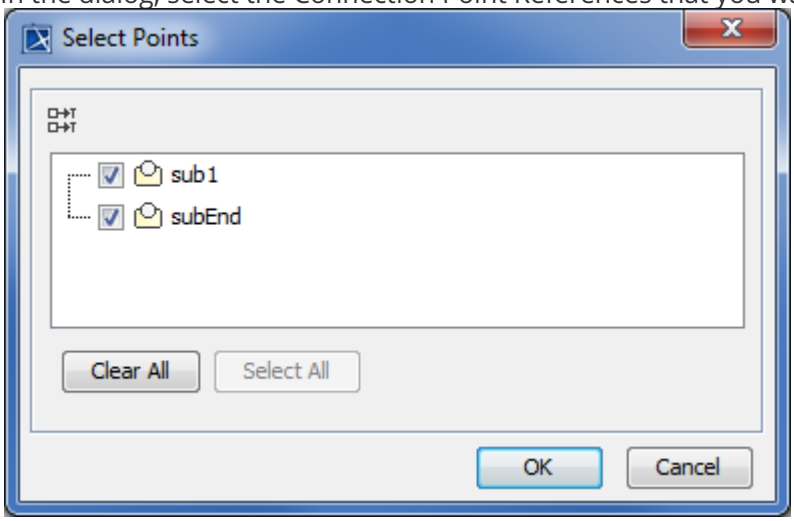
To display the Connection Point References on the submachine State shape

1. Select the submachine State shape of the Connection Point References you want to display.
2. Open the **Select Points** dialog by performing one of the following steps:
   - From the shortcut menu, select **Display** > **Display Points**.
   - On the diagram toolbar, click and select **Display Points**.

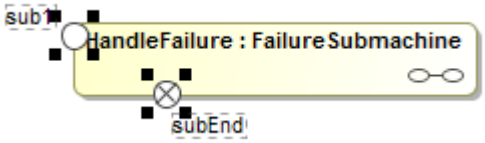


Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

3. In the dialog, select the Connection Point References that you want to display, and click **OK**.



Selected Connection Point References are displayed on the submachine State shape.



Related Pages

- Model Elements(see page 628)
- Stereotype(see page 856)
- State Machine diagram(see page 308)

Related pages

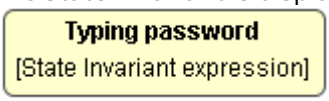
State Invariant

Defining State Invariant

To define a State condition

- Double-click the State to open the **State** Specification window and in the **State Invariant** property value cell, type a condition and press Enter.
- Near the **State Invariant** property value cell, click the **...** button. The **Edit State Invariant** dialog opens. Type the condition and close the dialog.

The State Invariant is displayed in square brackets on the State shape:



Related Pages

- Model Elements(see page 628)
- Stereotype(see page 856)
- State Machine diagram(see page 308)

Related pages

Stereotype

A stereotype defines how an existing metaclass may be extended. It enables the use of a platform, a domain specific terminology, or a notation in place of, or in addition with, the ones used for the extended metaclass. Similar to a class, a stereotype may have properties which may be referred to as tag definitions. When a stereotype is applied to a model element, the values of the properties may be referred to as tagged values. A stereotype can extend any model element from the reference metamodel (any UML model element). For example in UML, States, Transitions, Activities, Use cases, Components, Attributes, Dependencies, etc. can all be extended with the stereotypes. The stereotypes are created as separate model elements and can be drawn in almost all diagrams. You can specify stereotype properties(see page 921) in the stereotype Specification window(see page 913). In the same window, you can find the description of each property. Descriptions are presented in the description area of the Specification window(see page 913).

Stereotype notation in diagrams use the guillemets instead of symbols «» (see the following figure). However, when editing elements in a diagram, you can still enter the stereotype names between the «» symbols.

The following procedures describe how to work with stereotypes:

- Creating a stereotype(see page 857)
- Creating stereotype with an image(see page 857)
- Applying a stereotype(see page 857)
- Changing the stereotype's metaclass(see page 858)
- Displaying the stereotype icon instead of the shape on the diagram pane(see page 858)
- View assigned image properties(see page 858)
- Changing the order of the stereotypes(see page 858)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Saving the stereotype information in XMI(see page 859)
- Changing the stereotype display mode(see page 859)

Creating a stereotype

To create a stereotype, do one of the following

- Open or create the Profile Diagram(see page 324). On the diagram pallet(see page 24), click the Stereotype button and place it on the diagram pane.
- In the Containment tree, select a Profile, and from the shortcut menu, select **Create Element** > **Stereotype**.

Creating stereotype with an image

To create a stereotype with an image

---

1. In the stereotype's Specification window(see page 913), click the **Icon** specification cell to edit and select an image you want to place(see page 982) for the stereotype by selecting an image form:
   - the image library, click .
   - the file system, click .
   - using URL, click .
2. Select an image and click the **Open** button.
3. Close the Specification window.

Applying a stereotype

To apply a stereotype, do one of the following:

- In the corresponding element's Specification window(see page 913), click the **Applied Stereotype** property specification cell, then click , and select stereotypes from the open list. After you have selected, click **Apply**.
- In the element's shortcut menu, select **Stereotype** and select a stereotype that you wish to apply. Click **Apply** when you are done.
- On the diagram pane, select an element and in that element's name area, type the stereotype name between brackets « ». The element's name comes after the stereotype.

  > If you want to name the element Books and assign it the «table» stereotype, in the element's name area type the following: «table» Books. The name completion for the stereotypes works in the name editing mode, press Ctrl+Spacebar or Ctrl+Backspace to get a list of available stereotypes.

You can change the order of the stereotypes applied to the element. The symbol style of the first stereotype in the list will be applied to the element.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Changing the stereotype's metaclass

To change the stereotype's metaclass

1. Do one of the following:
   - On a diagram pane, select the stereotype and from its smart manipulator toolbar, choose the Metaclass button.
   - In the stereotype's Specification window(see page 913), click the **Metaclass** specification value cell, and then click .
2. In the **Select Metaclass** list, select the metaclass you want to apply.
3. Click **Apply** when you are done.

Displaying the stereotype icon instead of the shape on the diagram pane

To display the stereotype icon instead of the shape on the diagram pane

1. Select the shape on the diagram and click the Compartments smart manipulator(see page 570).
2. Click to expand the menu and select **Suppress All**.

View assigned image properties

To view assigned image properties

- In the stereotype Specification window(see page 913), right-click the **Icon** property specification cell and select **Open Specification**. The Image specification window opens wherein you can see and specify image properties.

Changing the order of the stereotypes

If a symbol has several stereotypes applied, you can change the order of stereotypes for visual purposes.

To change the order of the stereotypes

1. Right-click the symbol to open the shortcut menu, click **Stereotype**. The **Select Stereotype** dialog opens.
2. Click the **Order** button. The **Order Stereotypes** dialog opens.
3. Select a stereotype and click **Up** or **Down** buttons to change the order of the stereotypes.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Saving the stereotype information in XMI

You can choose the location of the applied stereotypes' information in the XMI file. Information can be saved at the end of the file or inside the element's information. By default, stereotype information is stored at the end of the XMI file.

Storing Information
To store information inside an element

- From the main menu, select **Options** > **Environment**.
- The **Environment Options** dialog opens.
- In the **General** options group > **Save/Load**, set the **Save Stereotype Information within Element** to *true*.

Changing the stereotype display mode

You can change the stereotype name and its icon visibility on an element's shape.

To change the stereotype display mode on the element shape

1. Click the shape, and then, click buttons to suppress all the compartments(see page 570).
2. Click the shape and then click the Compartments button(see page 570). From the menu, click the **Stereotypes** and then select a desired display mode that are described in the following table:

Show Stereotypes Property Value	Shape	Icon of the stereotype on the shape	Name of the stereotype	Image of the stereotype instead of the element shape
Text and icon		displayed	displayed	-
Text		not displayed	displayed	-
Icon		displayed	not displayed	-
Shape Image and Text		-	displayed	displayed*

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Show Stereotypes Property Value	Shape	Icon of the stereotype on the shape	Name of the stereotype	Image of the stereotype instead of the element shape
Shape Image		-	not displayed	displayed*
Do Not Display		not displayed	not displayed	-

* To display the image of a stereotype instead of the element shape, all element compartments should be suppressed.

Related pages

- Profile diagram(see page 324)
- Model Elements(see page 628)
- UML Profiling and DSL Guide(see page 2034)

Substitution

A substitution is a relationship between two Classifiers(see page 461). In these cases, the substituting Classifier complies with the contract specified by the contract classifier. Instances of the substituting Classifier are runtime substitutable where instances of the contract classifier are expected. Specify the Substitution element in the Substitution Specification window(see page 913).

You can see which elements the Substitution is drawn between in the Substitution dialog, Source and Target boxes.

You can format the Substitution symbol properties in the Symbol Properties dialog(see page 975).

Subsystem

A subsystem is treated as an abstract single unit. It groups model elements by representing the behavioral unit in a physical system.

To draw a subsystem

- On the Use Case diagram palette, click the **Subsystem** button, and then click the diagram pane.

Related Pages

- Model Elements(see page 628)
- Stereotype(see page 856)
- Use Case diagram(see page 300)

Swimlanes

Actions and subactivities can be organized into Swimlanes in the Activity diagrams. Swimlanes are used to organize responsibility for actions and subactivities according to class. They often correspond to the organizational units in a business model.

Swimlanes limit and provide a view of the behaviors invoked in the activities. They consist of one or more partitions and can be vertical or horizontal.

An Activity diagram can be visually divided into Swimlanes, each separated from the neighboring Swimlanes by vertical or horizontal solid lines on both sides. Each Swimlane represents a responsibility for part of the overall activity, and may eventually be implemented by one or more objects. The relative ordering of the Swimlanes has no semantic significance, but can indicate some affinity. Each action is assigned to one Swimlane. Transitions can cross lanes. The routing of a transition path is non-essential.

You can customize Swimlane representation by changing its symbol property values. Press Alt + Enter to open the appropriate dialog. Each property is described in the description area on this dialog.

To create a Swimlane

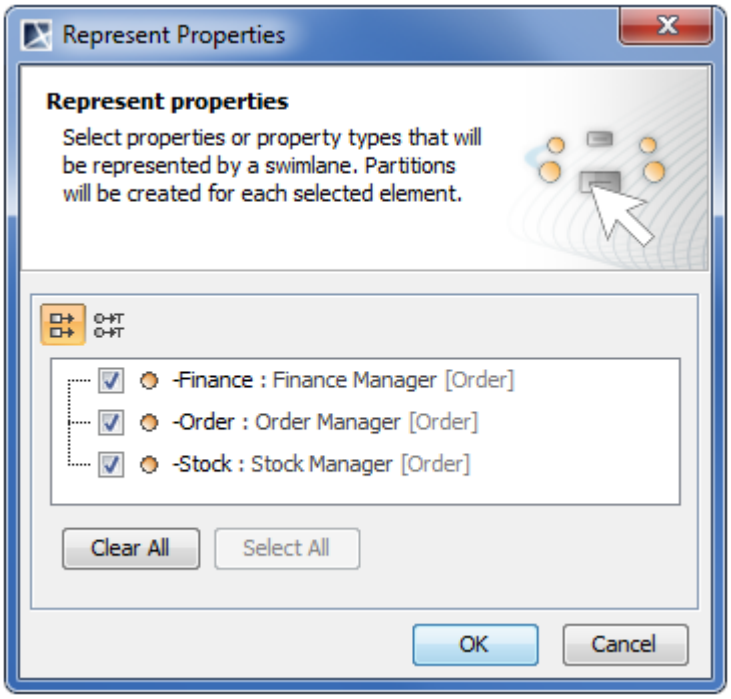
---

1. On the Activity diagram palette, click the **Vertical Swimlanes** button and then click an empty space on the diagram pane.
2. If the **Represent Properties** dialog opens, select the properties you want to display as Swimlanes in the diagram (see the following figure) and click **OK**. Otherwise, a few empty Swimlanes are created.

   The **Represent Properties** dialog opens if:
   a. The Activity diagram is owned by a classifier, such as a Class, Component, Use Case.
   b. The classifier or its subject(see page 878) has properties.

   Partitions are created for each property you select to represent as a Swimlane in the Activity diagram.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



> You can also create a Swimlane by dragging one or more properties, Actors, Classes, or Instance Specifications from the Containment tree to the Activity diagram pane. Partitions for each element are created as well.

To set representative elements, do one of the following

- Open the Specification window of the Swimlane and click the cell of the **Represents** property value. Then click ... and select the element. Click **OK** when you are done.
- From the Containment tree, drag the representative element to the partition on the diagram.

To add an additional partition, do one of the following

- On the diagram, right-click the Swimlane, select **Insert Swimlane** and then **Insert Vertical Swimlane** or **Insert Horizontal Swimlane**.
- From the Containment tree, drag one or more property, actor, class or instance specification elements to the Swimlane on the diagram.

To draw multidimensional Swimlanes

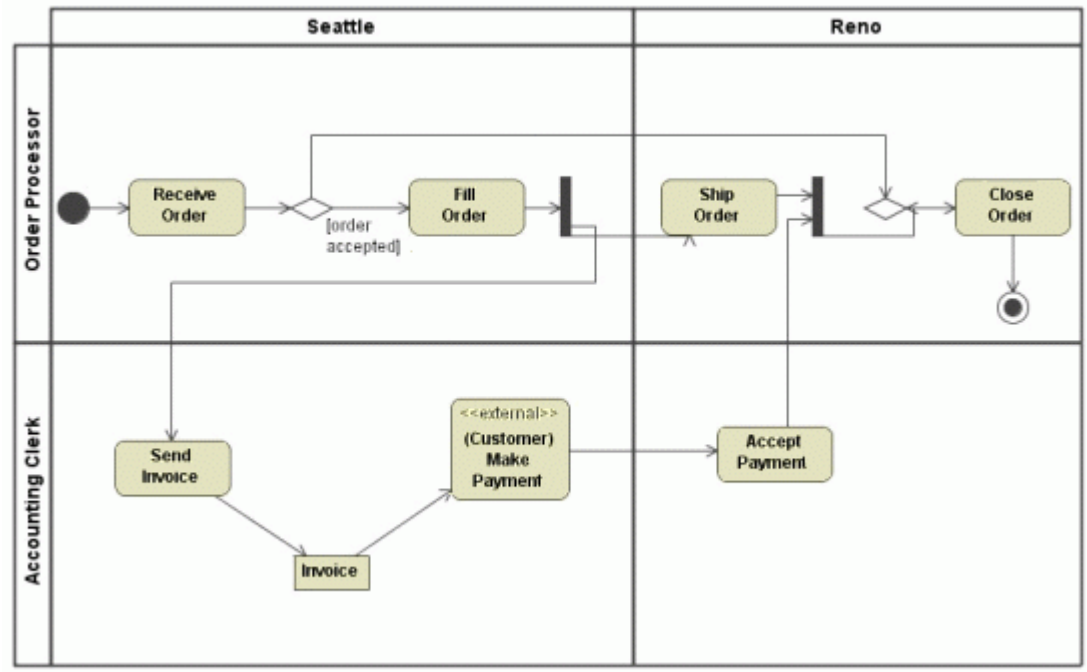
1. Draw a vertical Swimlane.
2. From the Swimlane shortcut menu, select **Insert Horizontal Swimlane**.
3. Insert as many horizontal and vertical Swimlanes as you need.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To add model elements to a Swimlane

- If a Swimlane is already drawn in the Activity diagram, drawing an action (or any other element) highlights the Swimlane in blue. This means the action shape depends on the Swimlane symbol.

  ⚠ If the model elements depend on a Swimlane symbol, they are deleted if the Swimlane symbol is deleted.



To rearrange Swimlanes, do one of the following

- Drag any of the Swimlanes in the direction where you want it to be.
- Click the heading of any Swimlane and use the displayed control arrow to move the Swimlane left, right, up or down.

To delete a single Swimlane entirely, do one of the following

- Click the header of the Swimlane, then press Delete.
- Click the header of the Swimlane, then press Ctrl + D.
- Right-click the header or any empty space on the Swimlane, then select **Delete Swimlane**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To delete all Swimlanes and their contents

- Click any empty space on the Swimlanes, then press Ctrl + D.

To delete Swimlanes without deleting contained elements, do one of the following

- Click any empty space on the Swimlanes, then press Delete.
- Right-click the headers or anywhere on the Swimlanes, then select **Delete All Swimlanes, Keep Contents**.

Rrelated pages

- Formatting symbols(see page 975)
- Model Elements(see page 628)
- Activity diagram(see page 311)

Tag

Just like a class, a stereotype (see page 856)can have properties, which can be referred to as tag definitions. When a stereotype (see page 856)is applied to a model element, the values of the properties may be referred to as tagged values.

The tag definitions are used to define new meta attributes of the extended metaclass, they are used as regular class attributes.

An actual instance of the tag definition is a tagged value (tag).

A tag holds extra information like:

- additional information that does not come with UML, for example Precondition for Use Cases.
- management data about the state and progress of the project such as author, status, and tested.
- language specific data for code generation tools.

A tagged value consists of two parts: name and value (example: Author = Joe).

To create a new tag definition

1. Create a new stereotype.
2. Open the stereotype Specification window(see page 913).
3. In the **Tag Definitions** property group, click the **Create** button to add a new tag definition for stereotype.
4. Select the type of this property.

   ⚠ Type of the property can be a standard UML data type or another user defined Stereotype. Regular classes should not be used as types of tag definition.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related References

Related Pages

- Model Elements(see page 628)
- Stereotype(see page 856)
- State Machine diagram(see page 308)

Editing tagged value

To create a new tagged value, do one of the following

- In the element Specification window(see page 913), **Tags** property group, select an available tag definition and click the **Create Value** button.
- In the Containment tree, select an element and drag it to the Specification window tag value area.

To create default tag values

1. In the stereotype's Specification window, expand the **Tag Definitions** property group and select the tag definition.
2. Assign a value for the **Default Value** property.
3. Create an element and assign a stereotype for this element. The element will have tags with assigned default values.

Setting a default tag value to tag with empty value

MagicDraw allows for setting default tag values to a tag with an empty value. This functionality is needed when a stereotype is already assigned to an element and a new mandatory tag definition with a default value is created for the stereotype. After creating such a tag definition, the model elements that have the modified stereotype applied will have the newly created tags unset.

To set default values instead of empty values

1. In the main menu, select **Tools** > **Set Empty Tags to Defaults**. The **Select Package** dialog opens.
2. Select the scope of elements to which you want to set the default tag values.

   If you want to assign default values to all project tags, in the *All* data tree select the root package *Data* and click the **Add All** button. The root package *Data* is added to the **Selected** objects list. Click **OK**.

3. In the opened dialog, click **Yes**. Now, the element with stereotype tags has been assigned default values.

The following conditions are required to set the default tag values

- The element should have an assigned stereotype with a specified default value property.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- The tag definition, to which you want to assign a default value, should contain no value.

The default tag values are not set when the stereotype property (tag definition) multiplicity is equal to 0, 0..1, 0..*, *.

Related Pages

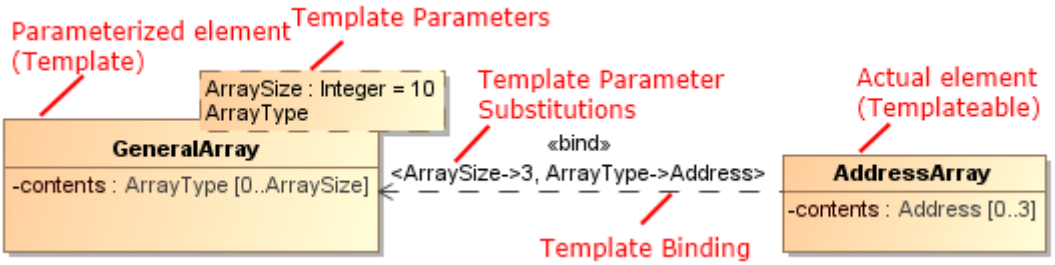
- Model Elements(see page 628)
- Stereotype(see page 856)
- State Machine diagram(see page 308)
- Tag(see page 864)

Template

Use a Template to define concrete types for elements without concrete types. You can create a template and then use it as many times as you need. Moreover, when you have elements of the same structure (or with the same parameters) but with different values, the template you use can easily define parameters for each element. You can apply template parametrization to classifiers, packages, and operations. Most often. template parametrization is used for classes and interfaces. Templates are often used in code engineering (Java, C#, and other programming languages).

The template consists of:

- A parameterized element (an element with defined template parameters). The parameterized element can also be called a template element.
- An actual element (an element for which a template is applied, an element with actual values). The actual element can also be called a templateable element.
- A template binding relationship (a relationship from the actual element to the parameterized element).
- A template parameter substitution (used to define actual parameters of the actual element that substitutes formal template parameters). Template parameter substitutions are created on the template binding relationship.



Example of Template

In this example, you can see the *GeneralArray* template class, which has the *ArraySize* and *ArrayType* template parameters, represented by the dashed rectangle in the upper-right corner of the class. These parameters represent the size and type of the general array.The attribute *GeneralArray::contents :*

ArrayType [0..ArraySize] represents the *ArrayType* and the *ArraySize* template parameters. The *ArraySize* template parameter is of the Integer type, and its default value is 10. The *ArrayType* template parameter is of the Class type (it is not represented in the shape), and its default value is not specified. The default value of the template parameter is used if no actual value is supplied for the parameter in a binding. Typically, the parameter types are classifiers, but they can also be integers or other types.

The *GeneralArray* class and its template parameters create a template of the general array of undefined types. You can use this template in other concrete arrays as many times as you need.

How to use the template and how to create an actual element

To create an array of addresses, you would create the *AddressArray* class. Alone (without a template), the *AddressArray* class would have undefined types. A template binding is created from the *AddressArray* class to the *GeneralArray* class. Now the template binding connection points to the *GeneralArray* template class.

The template binding specifies template parameter substitutions, the actual values of parameters for the address array. In the example, we created the following actual values of the AdressArray class: the array size having the value 3 and the array of the *Address* type.

The output of these definitions results in the *AddressArray* class having the attribute *AddressArray::contents : Address [0..3]*.

Attributes for the actual element classes should be created manually. In the example, the *AddressArray::contents : Address [0..3]* attribute is not created automatically - you must create this attribute manually. Use these optional solutions to create these attributes in your model or not.

Working With Templates
Step by step instructions for modelling a template in the MagicDraw are described in Working With Templates(see page 867).

You can format template parameter symbol properties in the **Symbol Properties** dialog(see page 975).

You can specify template parameter properties in the Template Parameter Specification window(see page 913) and template binding properties in the Template Binding Specification window. In the specification window, you can find the description of each property. Descriptions are presented in the description area of the Specification window.

Related pages

- Model elements[157]
- Auxiliary Diagram Symbols(see page 900)
- Diagramming(see page 482)

Working with Template

This section demonstrates how to create a template and how to use it.

157 https://docs.nomagic.com/display/MD190SP4/Model+elements

Follow these instructions from the first step consistently. If you do not, and if you do not define the template parameter, you will not be able to draw a template binding relationship. You can omit optional procedures only.

> **Precondition**
> Let's assume you already have an element to which you want to apply a template. This element is an actual element. The actual element can be a classifier, package, or operation.

To create and apply a template to an actual element, see the following procedures:

- To create a template(see page 0)
- To apply a template to an actual element(see page 0)

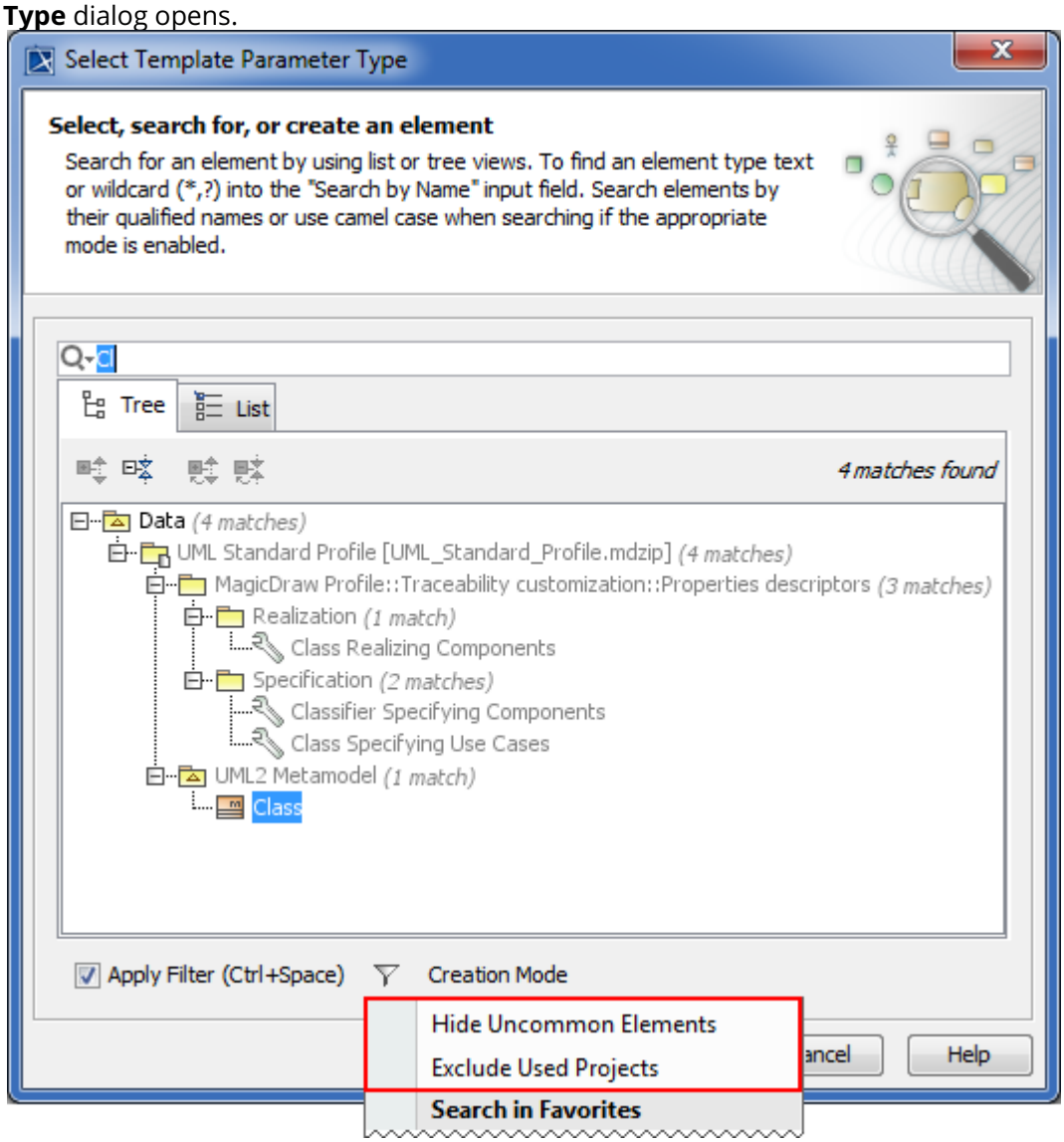
In the following procedures, we will show you how to create a template with parameters and how to define these parameters.

To create a template

1. Create or select a template element for which you will create template parameters.
2. Open the element's Specification window(see page 913).
3. Change the property display mode to **All**, if it is not changed already. The **Template Parameters** property group appears in the property group list.
4. Click the **Template Parameters** property group.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

5. To add a new template parameter, click the **Create** button. The **Select Template Parameter Type** dialog opens.

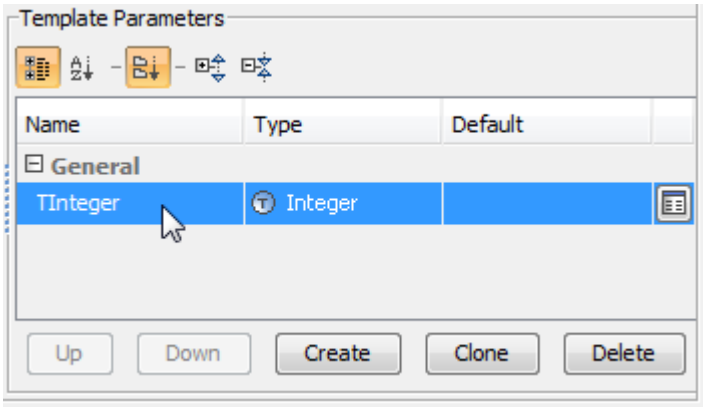


6. In the Model tree, select an existing or create a new type of the template parameter.

> **Template Parameter**
> If you do not want to specify the concrete template parameter type, you can select the *Class* type (the *Class* from the UML metamodel). When the *Class* type is specified, type is not displayed on the diagram pane. Only the name of the template parameter is represented.
> If you need to select one of the UML metamodel elements, such as *Class*, be sure to clear the **Hide Uncommon Elements** and **Exclude Used Projects** commands (as in the preceding figure). For more information, read Searching for elements in the Element Selection dialog(see page 1033).

7. Click **OK**. The template parameter is created. The name of the template parameter is defined automatically by compounding the *T* letter and the name of the template parameter type - *T<type name>*. For example, *TInteger*. You can change the name of the template parameter if you need

to.



8. (Optional) Assign the default value to the created template parameter.

> **More Information**
> For more information about symbol representation properties, see Formatting Symbols(see page 975).

> **Value Specification**
> To assign a particular value of the value specification, you must specify a Value Specification first and then type its particular value in the Actual value cell. For more information about Value Specification, read Value Specification(see page 899).

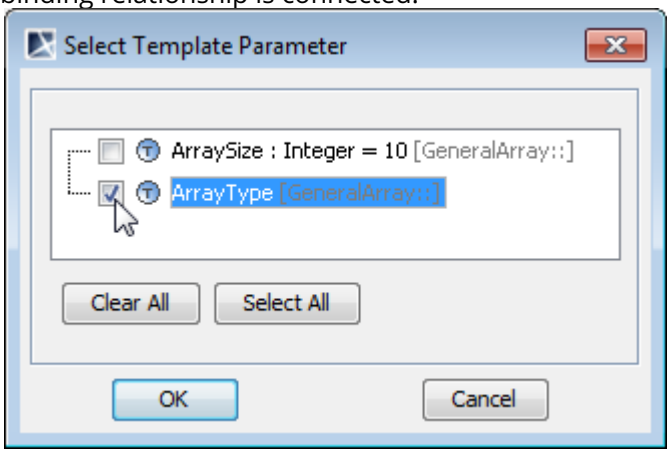
After the template is created, you can apply it to the actual element. The following procedures describe how to apply the template to the actual element.

To apply a template to an actual element

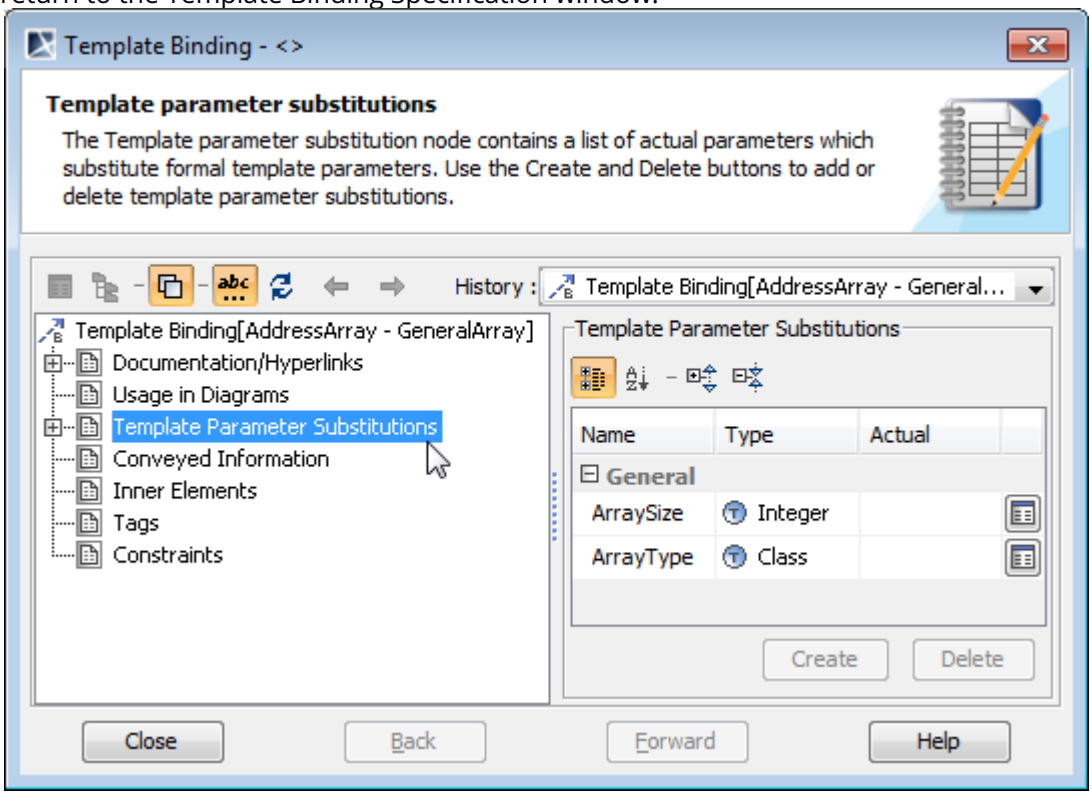
---

1. Bind the actual element to the template element. In the diagram palette, click the Template Binding button or click the **B** shortcut key and draw a template binding relationship from the actual element to the template element.
2. Define a template binding substitution. Open the Template Binding Specification window and click the **Template Binding Substitutions** property group.
3. To add a new template binding substitution, click the **Create** button. The Select Template Parameter dialog opens, listing the template parameters of the element to which the template

binding relationship is connected.



4. Select one or more template parameters to add as template parameter substitutions. Click **OK** to return to the Template Binding Specification window.



5. (Optional) Assign the actual value for each template binding substitution.

More Information
For more information about specifying property values, see Editing property values(see page 921).

Value Specification
To assign a particular value of the value specification, you must specify a Value Specification first and then type its particular value in the Actual value cell. For more information about Value Specification, read Value Specification(see page 899).

Related pages

- Model elements[158]
- Auxiliary Diagram Symbols(see page 900)
- Diagramming(see page 482)

Transition

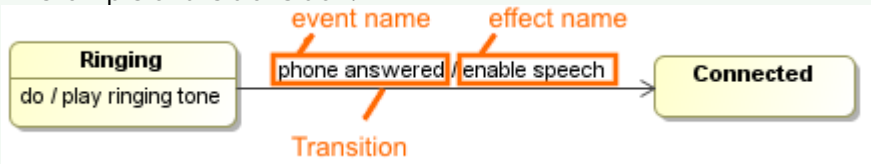
A transition is a directed relationship between a source vertex and a target vertex. It can be a part of a compound transition, which takes the state machine from one state configuration to another, representing the complete response of the state machine to an occurrence of an event of a particular type. The transition relationship is created between two states. The transition specifies event occurrences and guard conditions.

When these events occur and conditions are satisfied with the object in the source, state will perform the specified effect and will enter the target state. So, in the transition, the main subjects are as follows:

- Event trigger and its parameter
- Guard condition
- Effect

On a transition path the event trigger, guard condition, and effect has the following syntax:
< event name> (< parameter >) [<guard condition >] / <effect name>

An example of the transition.



In the preceding example, the transition relationship is created from the Ringing state to the Connected state. The transition has the specified event - phone answered, and the specified effect - enable speech. When the phone answered event occurs, the Ringing state performs the speech effect, and then the Connected state is
entered.

The event on the transition is specified as event of the concrete type. The event trigger is the event whose reception in the source state makes the transition eligible to fire. That is, the trigger is specified by the event. Event (see page 708)types are as follows:

- Any Receive Event
- Call Event
- Change Event
- Signal Event
- Time Event

158 https://docs.nomagic.com/display/MD190SP4/Model+elements

- You can assign an event type(see page 873) in the Transition Specification window, or there is a quick way to assign an event type straight on the transition from the diagram pane.
- The effect on the transition is specified as one of the behaviors types(see page 874).
- You can format the transition symbol properties in the Symbol Properties dialog(see page 975).

You can specify transition properties(see page 921) in the Transition Specification window(see page 913). In the same window, you can find the description of each property. Descriptions are presented in the description area of the Specification window(see page 913).

Related pages

Related Pages

- Model Elements(see page 628)
- Event(see page 708)
- State Machine diagram(see page 308)

Assigning Event Type

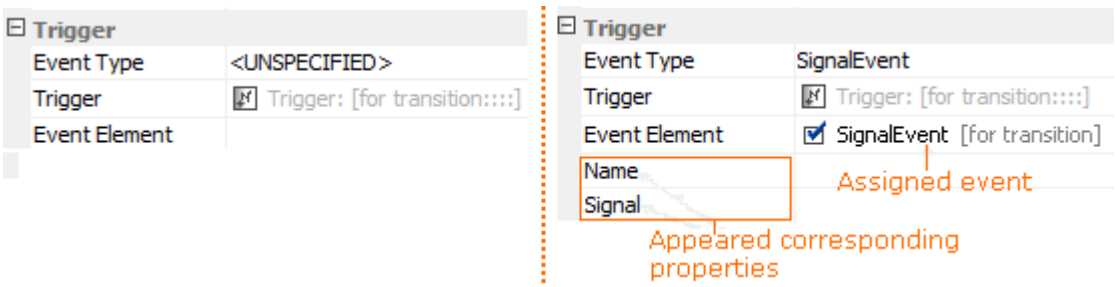
You need to assign an event to the transition in order to specify an occurrence of an event which can trigger a state transition.

You can assign an event type to a transition in the following ways:

- Using the transition's Specification window(see page 913).
- By typing an appropriate command straight on the selected transition path.

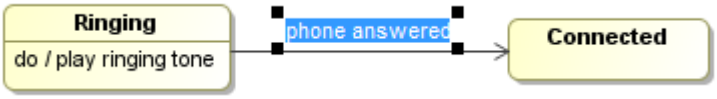
To assign an event type in the Transition Specification window

1. Open the transition's Specification window.
2. In the **Trigger** category, select the **Event Type** property value cell. The list of available event types opens.
3. Select the event type from the list. According to the selected event type, Event Element, and Trigger property values are defined and additional corresponding properties appears in the Trigger category (see the following figure).
4. Click the **Close** button after you have defined desired properties.



To assign an event type straight on a transition path

1. Select a transition path on the diagram pane.
2. Type the command of the appropriate event type. The event type is assigned to the transition. In the following figure, see an example of the signal event type assignment.



See the list with the event type and its command syntax in the following table.

Name	Command syntax
Any Receive Event	*all*
Call Event	*<operation name* ()*>* **NOTE**: Before assigning operation straight on the transition path the Call Event together with this operation should be already created in the same State Machine as the transition.
Change Event	*when* (*<expression>*)
Signal Event	*<signal name>*
Time Event	• *after* (*<time>*) - for an event occurring at a relative time • *at* (*<time>*) - for an event occurring at an absolute time

The event type assigning by typing the command straight on the diagram pane is also available for a transition to self in the State Machine diagram, as well as for a protocol transition and protocol transition to self in the Protocol State Machine diagram.

Related pages

Related Pages

- Model Elements(see page 628)
- Transition(see page 872)
- State Machine diagram(see page 308)

Assigning Behavior Type

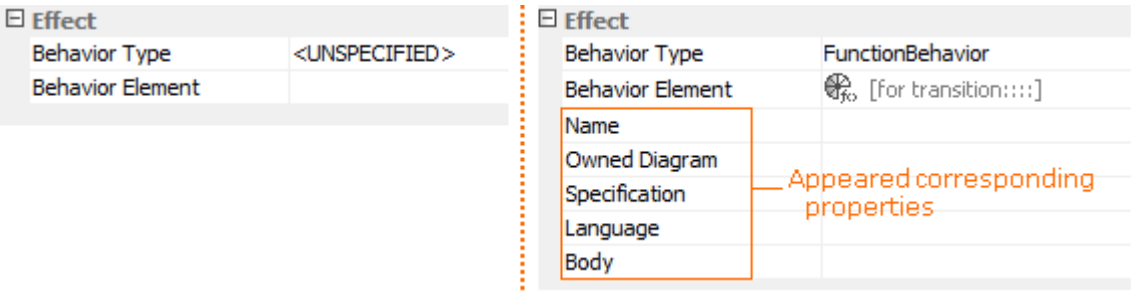
A behavior type is a behavior (see page 662)that is performed when the transition fires.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To assign a behavior type, do one of the following

- In the Containment tree, select a behavior type element and drag it to the selected transition on the diagram.
- In the transition's Specification window(see page 913) > **Effect** category, click the **Behavior Type** property value cell and select the behavior from the list.

> According to the selected behavior, additional corresponding properties will appear in the **Effect** category. Specify desired properties.



> - To quickly trace the effect of the transition, from the transition shortcut menu, select **Go To** and then the effect element.
> - If a behavior (see page 662)of an activity, interaction, protocol state machine, or state machine type is assigned to a transition, a diagram for that behavior is created automatically.

Related pages

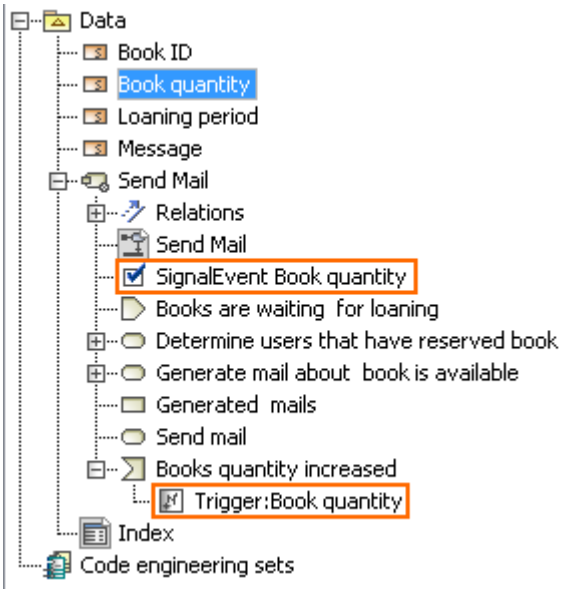
Related Pages

- Model Elements(see page 628)
- Transition(see page 872)
- State Machine diagram(see page 308)

Trigger

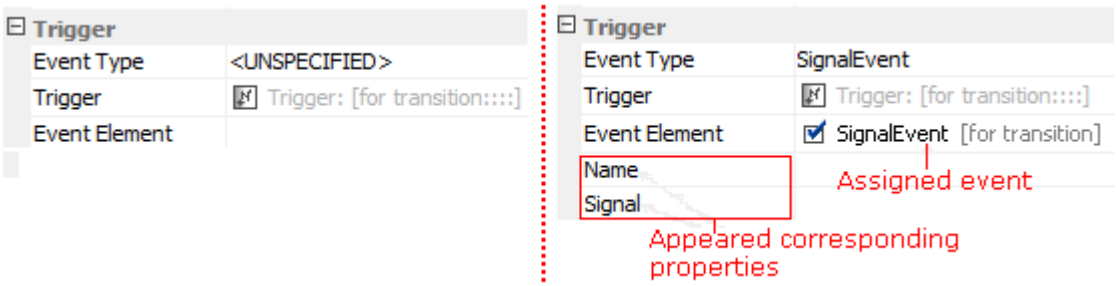
A Trigger used in a State Machine(see page 842) is an extension of the basic definition found within the behavioral and structural portions of the specification. A Trigger defines the types of events that can initiate a transition between states. An event is anything that can happen in a system, such as a signal sent by a Behavior(see page 662), a call to a specific operation, reaching a point in time, or a change in values within the system.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



To create the Trigger

- For a State(see page 875) element, in the State dialog box, click Add. The Trigger dialog box opens.
- For a Transition relationship, in the Transition dialog box, select the Trigger box, and click Add (note that the Trigger property is hidden by default). The Trigger dialog box opens.



Specify the Trigger element in the Trigger Specification window(see page 913).

In the Trigger dialog box, near the Event box, click the **...** button and select existing or create new Event(see page 708).

Use Case

Use Cases are a means of specifying the required usages of a system. Typically, they are used to capture the requirements of a system, that is, what a system is supposed to do.

In the following figure you can see a fragment of the Library System. The Use Case *Register Return* is associated to the Actor *Librarian*, which means that this Actor initiates that Use Case.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Behavior of a Use Case can be specified by an Activity, State Machine, or Sequence. It can also be described by a textual description - a Use Case scenario that can be depicted in an Activity diagram.

A Use Case can be specified by changing its property values in the Use Case Specification window(see page 861). Each property is described in the description area on this window.

Use Case relationships

A Use Case can be associated with an Actor to specify that the Actor initiates the associated Use Case.

A Use Case can be related to other Use Cases by

- Generalization
- Include
- Extend

Use Case shape with rake icon

The rake icon is displayed on the shape, if the Use Case is realized by other behavior diagrams, such as Use Case, Activity, State Machine, or Sequence. To hide the rake icon you need to set the **Show Rake Icon** property value to *false* in the **Symbol Property** dialog.

Use Case Actors

An Actor can be related to the Use Case by Association.

You can see the Actors that are related to the selected Use Case in Specification window(see page 913) of that Use Case, the Actors property group. This group displays a list of Actors that are directly (through the Association relationship) or indirectly (through the Extend, Include, or Generalization relationships) connected to the selected Use Case.

Related pages

- Model Elements(see page 628)
- Transition(see page 872)
- State Machine diagram(see page 308)
- Activity diagram(see page 311)
- Sequence diagram(see page 304)
- Use Case diagram(see page 300)

Extension Point

An extension point in the Use Case represents the location at which the extension can be inserted. In the Use Case diagram, the extension point is used together with the extend relationship.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To add an extension point to the Use Case

1. Draw an extend relationship from the extending Use Case to the extended Use Case.
2. Subsequent actions depend on whether the extended Use Case has at least one extension point already:
    - If there are no extension points, a dialog appears asking for confirmation to add a new extension point. Click **Yes** and type the name of the extension point.
    - If there is at least one extension point, the **Extension Points** dialog opens, displaying the list of extension points of that Use Case. Click **Create**, type the name of the new extension point, and click **Close**.

You can also add extension points via the Specification window(see page 913) of the Use Case. To do this, click **Extension Point** > ⊞, and then type the name of the new extension point.

Related pages

- Extension Point(see page 877)
- Specifying a Use Case subject(see page 878)
- Use Case description profile(see page 880)
- Use Case Scenario sketch(see page 882)

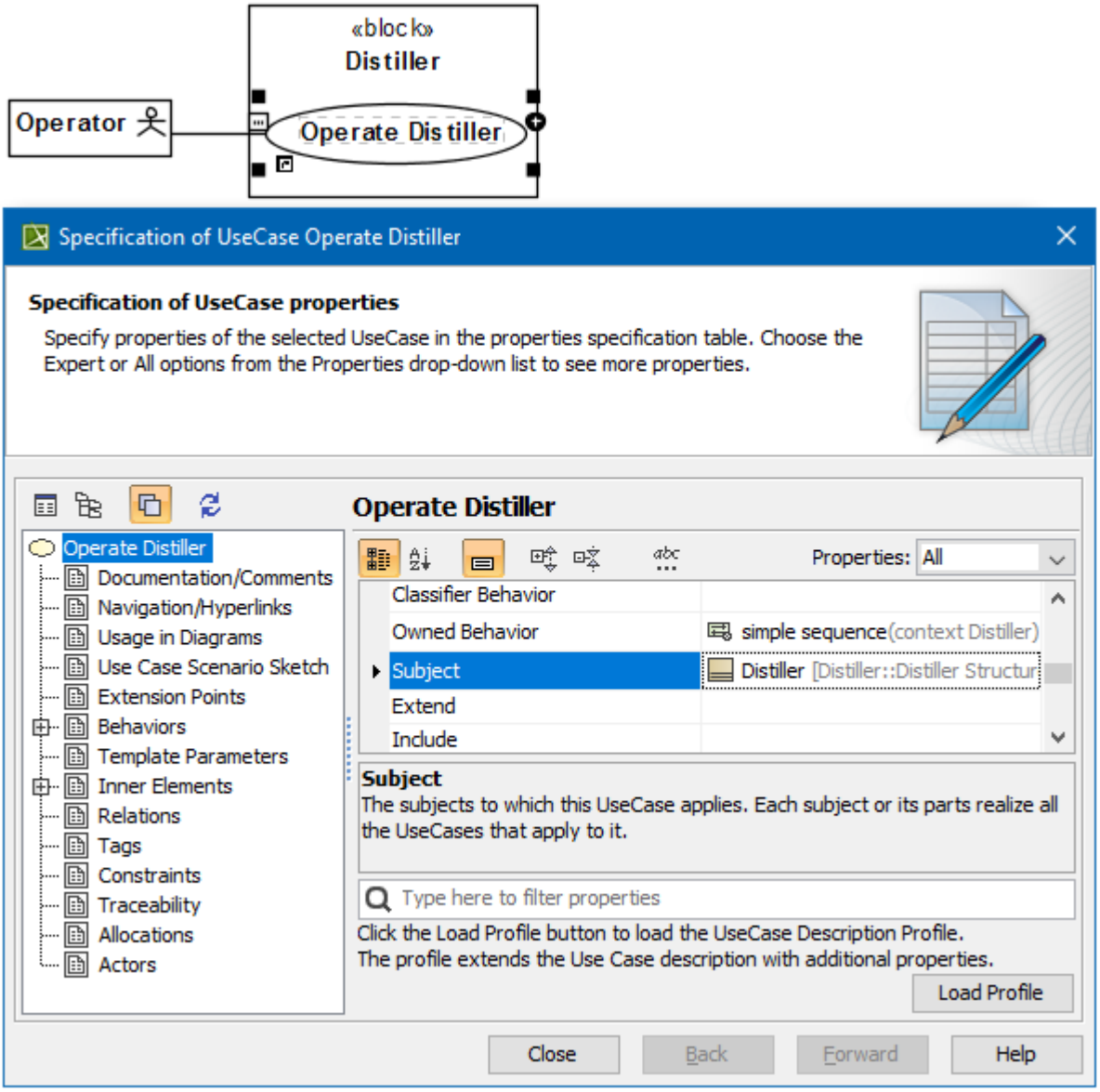
Specifying a Use Case subject

The subject of a Use Case represents a system under consideration to which that Use Case applies. The subject can be any element that may have a behavior, such as a Component or Class (SysML Block).

To specify the Use Case subject

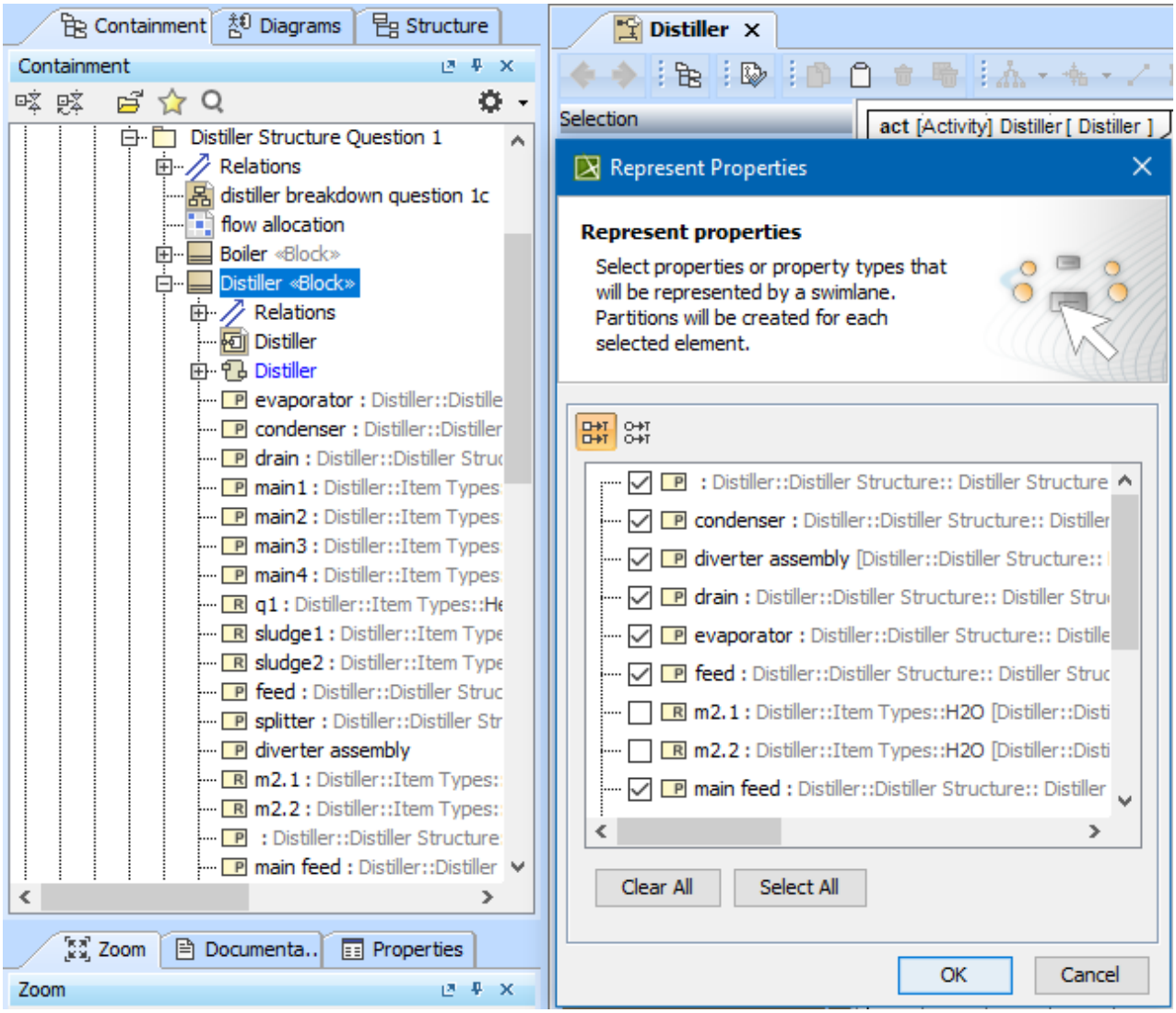
1. Select the Use Case or its shape.
2. Drag it onto a Class, SysML Block, or Component shape. The Use Case becomes nested in that shape, and the appropriate model element is set as the Use Case subject.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Having a Use Case with the subject specified jumpstarts the creation of Swimlanes in the Activity diagram owned by that Use Case. In this case, you can select the subject properties you wish to be represented by the Swimlanes on that diagram.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Related pages

- Extension Point(see page 877)
- Specifying a Use Case subject(see page 878)
- Use Case description profile(see page 880)
- Use Case Scenario sketch(see page 882)

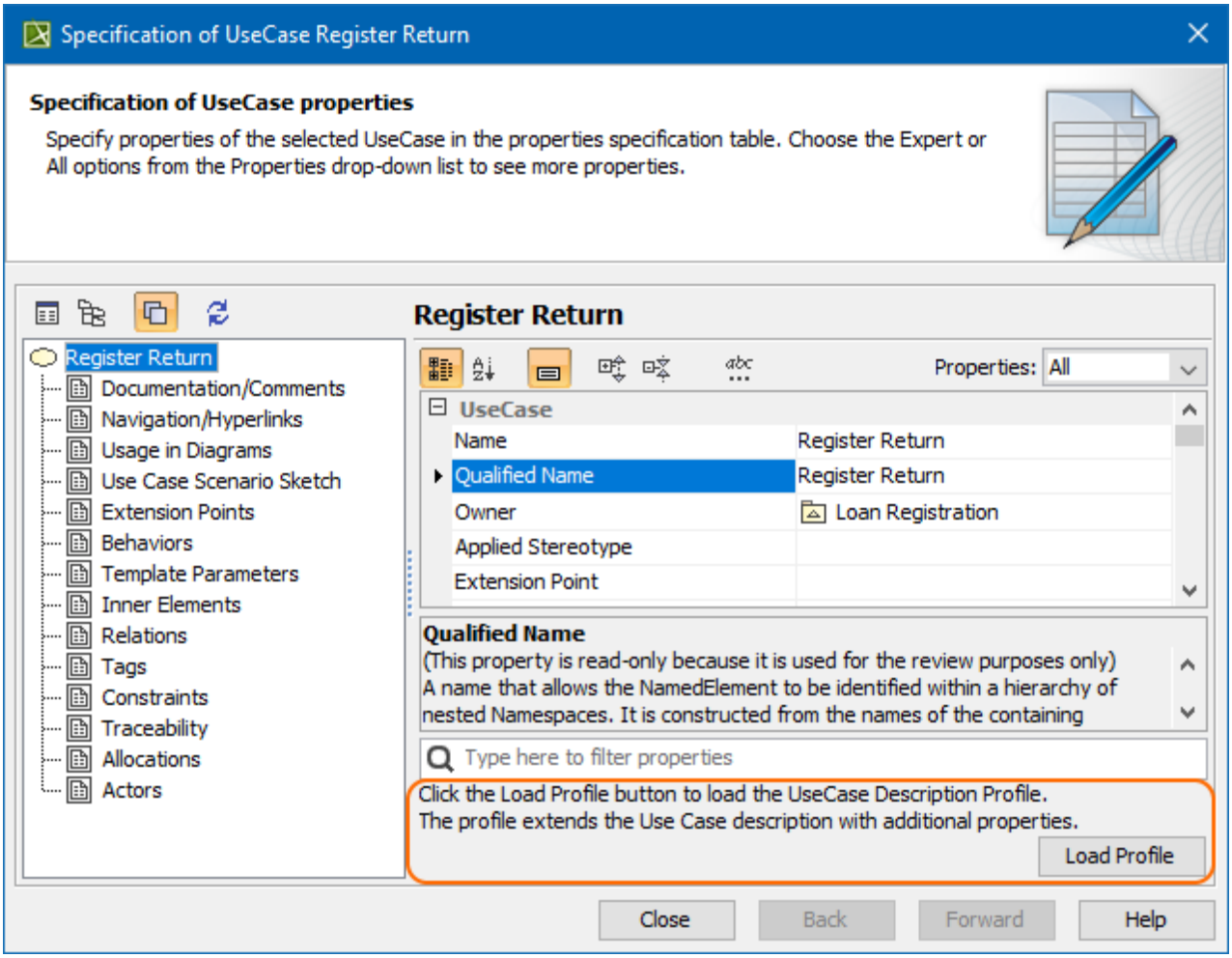
Use Case description profile

You can extend the Use Case description with additional Use Case properties, such as Use Case ID, Author, Date, Use Case Complexity, Pre Condition, and others.

To extend the Use Case with additional properties, you must load the Use Case Description Profile manually; in a new project, it is not loaded by default.

To load the Use Case Description Profile

1. Open the **Use Case** Specification window(see page 913).
2. Click the **Load Profile** button. The Use Case Description Profile is loaded and additional properties are added to the Use Case.

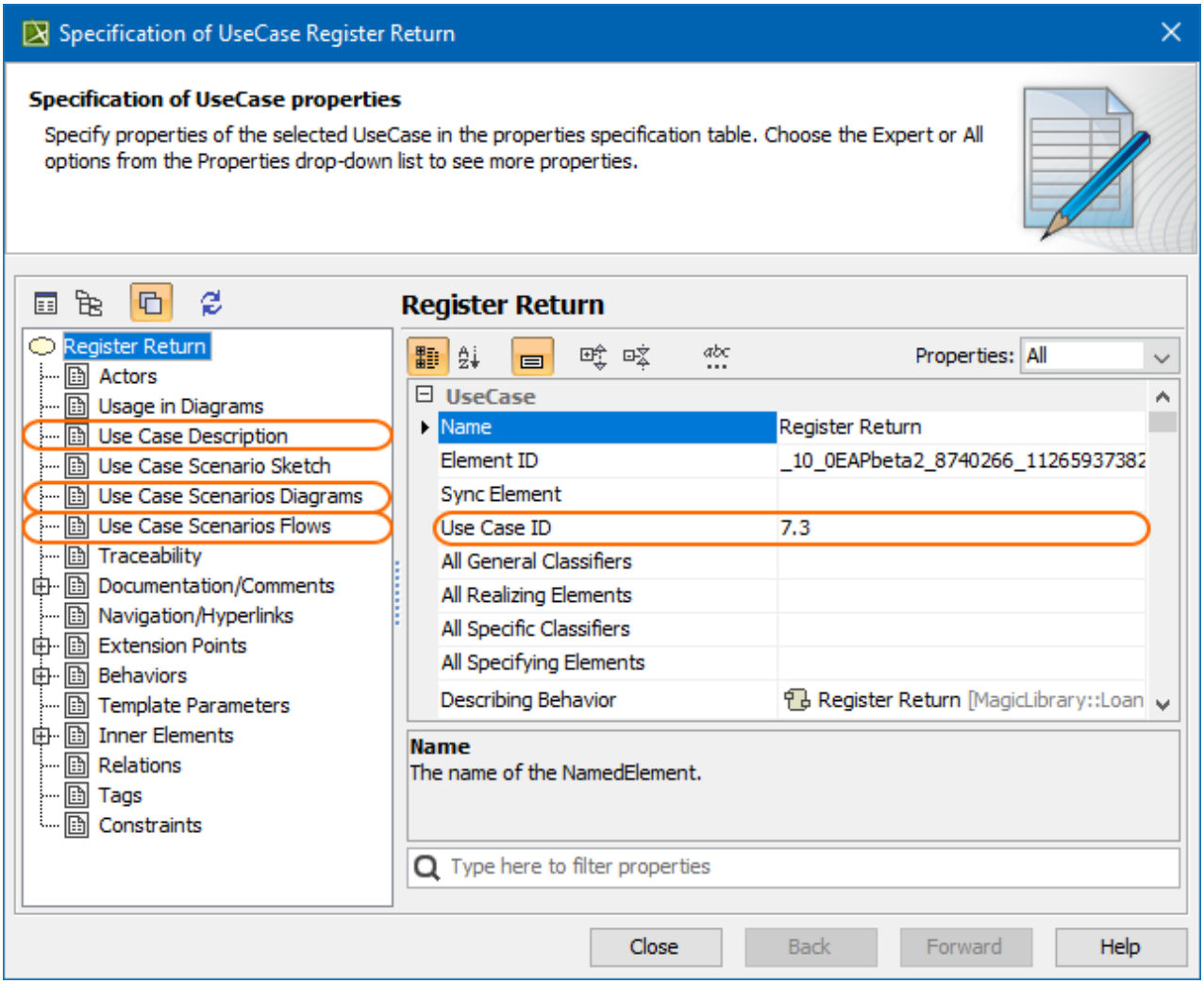


After the Use Case Description Profile is loaded, the following additional properties appear in the Use Case Specification window(see page 913):

- The **Use Case ID** property in the UseCase general property group. This property is designed to number use cases to correspond to particular sequences.
- The **Use Case Description** property group. In this property group, you can specify properties describing the particular Use Case, such as an Author, Date, Goal, and other.
- The **Use Case Scenarios Diagrams** property group allows specifying diagrams for basic, alternative, and exceptional Use Case scenarios.
- The **Use Case Scenarios Flows** property group allows specifying flows for basic, alternative, and exceptional Use Case scenarios.

The following figure illustrates an example of the **Use Case** Specification window(see page 913) after the Use Case Description Profile is loaded.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



You can specify the Use Case extension properties in the Use Case Specification window(see page 913). Descriptions of each property appear in the description area of the Specification window.

Related pages

- Extension Point(see page 877)
- Specifying a Use Case subject(see page 878)
- Use Case description profile(see page 880)
- Use Case Scenario sketch(see page 882)

Use Case Scenario sketch

On this page

- Basic flow(see page 884)
- Alternative flow(see page 885)
- Exceptional flow(see page 886)

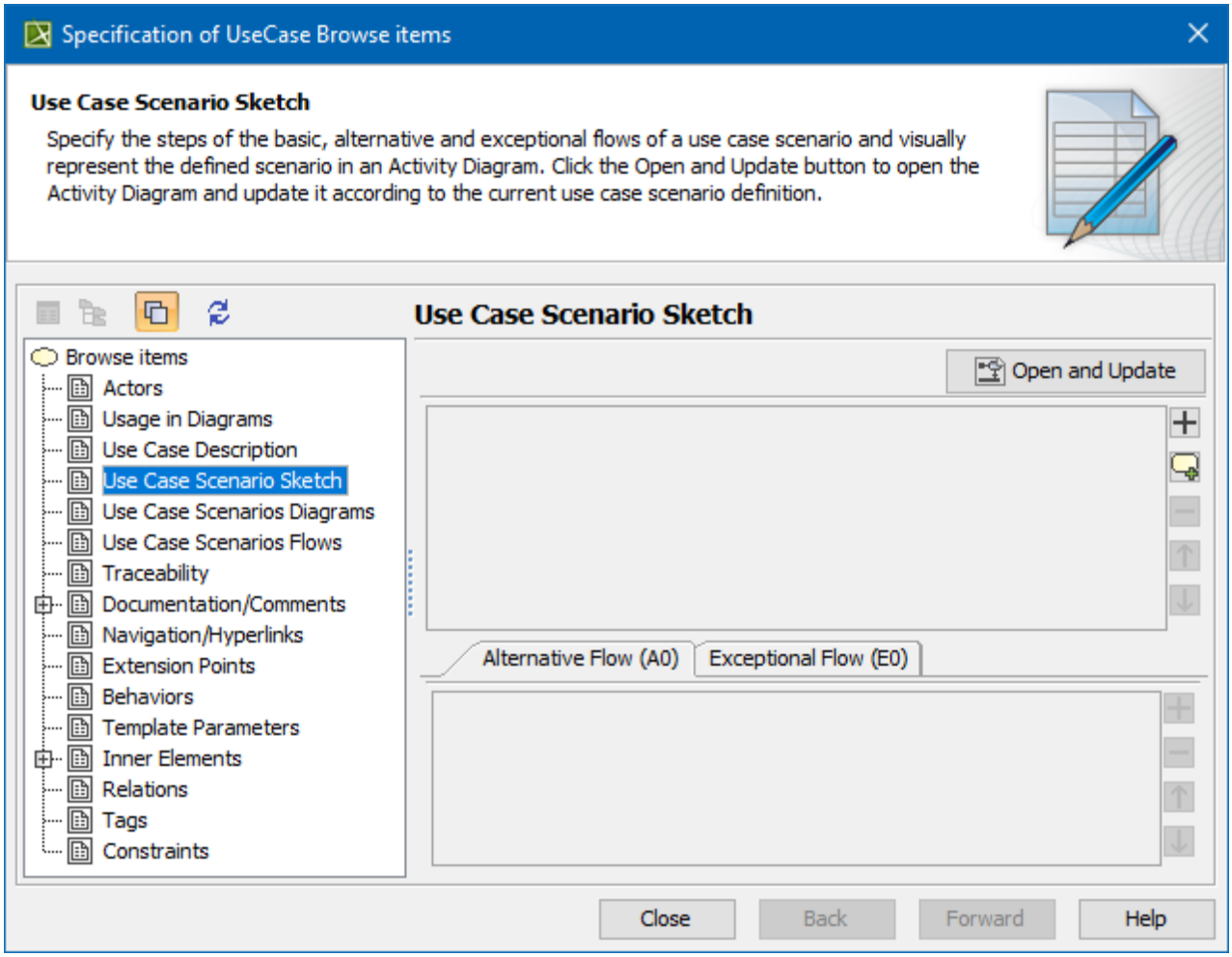
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

This functionality is available in the Standard, Professional, Architect, and Enterprise editions.

Use Case Scenario Sketch is designed for creating the very first Use Case Scenario. After the scenario is created, you may edit it directly in the Activity diagram.

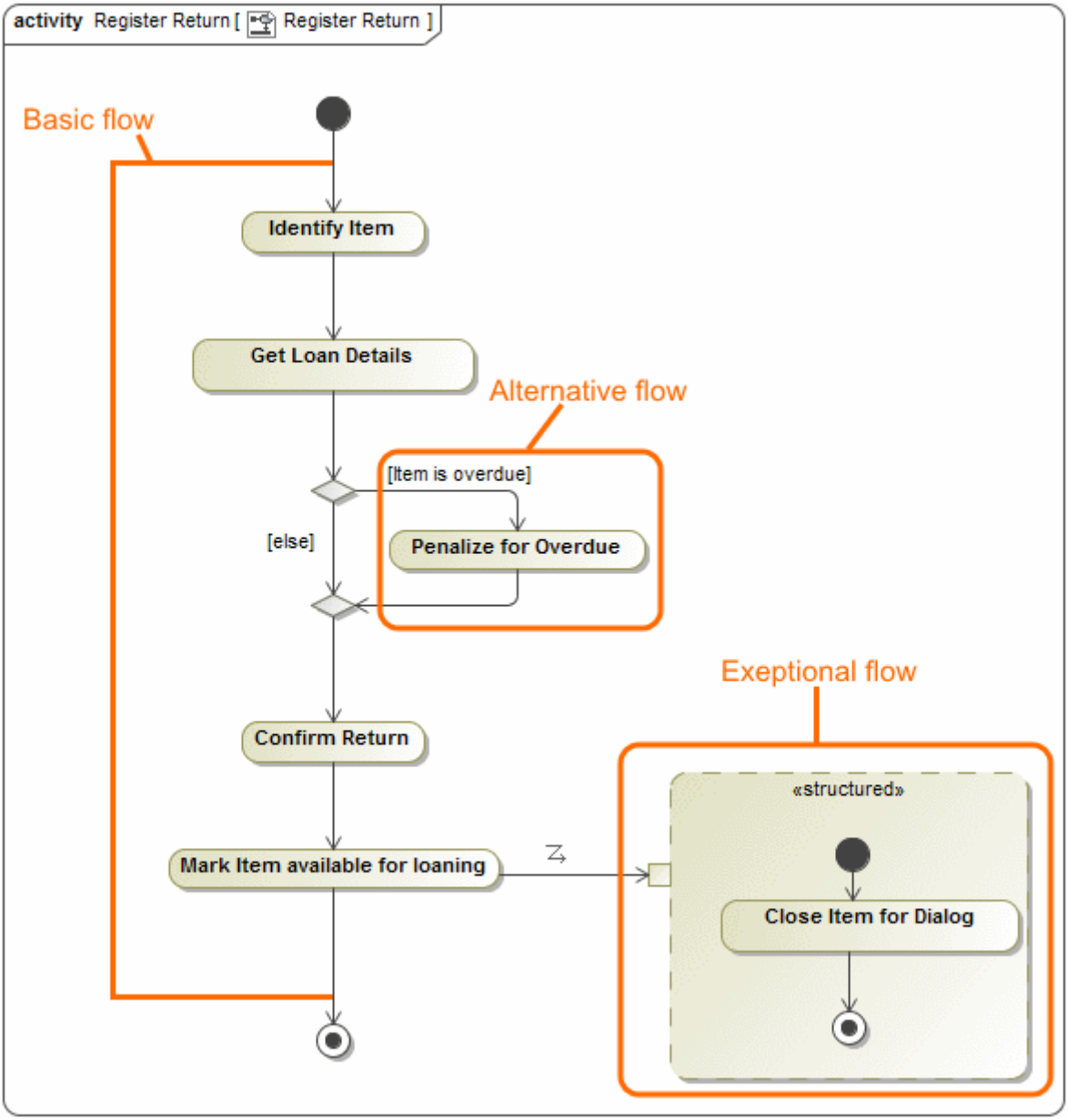
With the help of the Use Case scenario editor, you can:

- Create, review, and edit steps of the Use Case scenario by using the convenient textual values editor.
- Automatically create an Activity diagram representing the textual Use Case scenario flow. This enables you to have the textual numbered action flow and its graphical representation.
- Create the Use Case scenario from the Activity diagram for the particular Use Case and automatically represent this action flow as textual information in the Use Case scenario.



The Use Case Scenario Sketch creating in the Specification window.

The following figure depicts the same scenario represented in the Activity diagram.



A Use Case scenario can have defined basic(see page 884), alternative(see page 885), and exceptional(see page 886) flows.

Basic flow

A basic flow represents the sequence of basic steps or actions of the Use Case scenario. You can also add steps from included or extended use cases to the flow. Each basic flow step can have one or more alternative and exceptional paths. The two figures above depict the basic flow, consisting of four steps. Letters and numbers in the brackets next to the basic flow step indicate how many alternative conditions and exceptional types that particular step has. The letter A indicates alternatives flows; the letter E indicates exceptional flows.

In an Activity diagram, the basic flow is the main path down from the initial node to the final node.

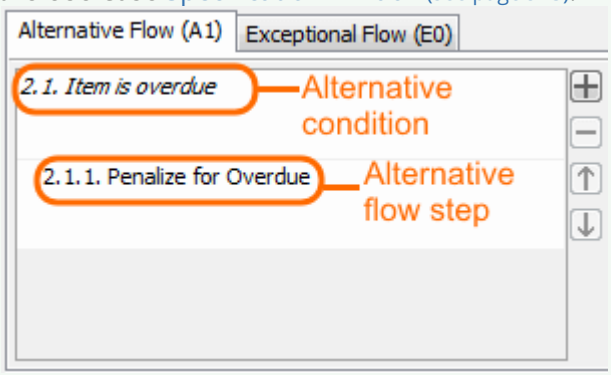
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Alternative flow

An alternative flow is an alternative path of the basic flow. You can define an alternative flow from a particular step of the basic flow. The alternative flow is an alternative solution, performed after the defined condition is satisfied. The alternative flow contains steps that are executed if one or more conditions occur.

Thus, in the Use Case scenario, the alternative flow is specified by these two parameters: the alternative condition and the alternative flow steps.

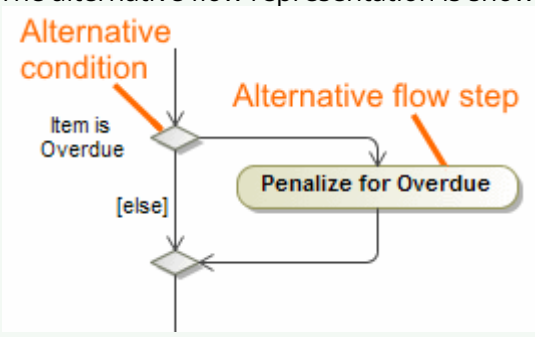
The following figure shows the content of the **Alternative Flow** tab of the Use Case Scenario in the Use Case Specification window(see page 913).



Here, the alternative flow is created for the second step of the basic flow — 2. *Get Loan Details*. The alternative flow has the condition named *2.1 Item is overdue*. The condition contains one alternative flow step named 2.1.1 *Penalize for overdue*.

In the Activity diagram, the alternative flow is created between the decision and merge nodes. The alternative condition is represented as a decision node; the name of the alternative condition in the Activity diagram is the name of the decision node. In the Activity diagram, the alternative flow steps are represented as Call Behavior actions.

The alternative flow representation is shown in the activity diagram in the following figure.



In the Activity diagram, you can read the alternative flow as follows: if the item is overdue, then penalize for overdue; if not, continue to the basic flow.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Exceptional flow

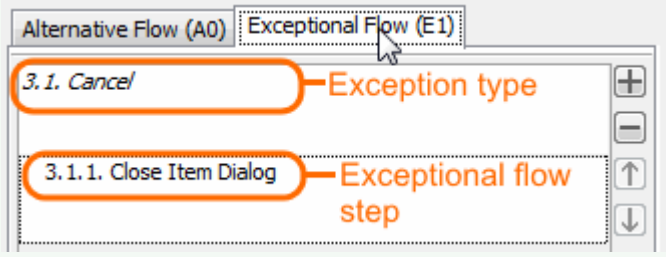
An exceptional flow is an exceptional path from the particular basic flow step, or it can be a quick solution for exit. The exceptional flow contains steps that are executed if something goes wrong, such as an input from the actor that the system cannot handle. An example of this would be if the user clicks the *Cancel* button in the opened dialog.

In the Use Case scenario, the exceptional flow is specified by two parameters: the exception type and the exceptional flow steps.

The type of the exception object indicates the nature of the exception. Any class could be assigned as a type of the exception.

The exceptional flow steps are performed as actions when the execution occurs. The exceptional flow steps are specified for the concrete type. Normally, there is one exceptional flow (in other words, one exception type) assigned for one basic flow step. There can be any number of exceptional flow steps specified for one exception type.

The following figure shows the content of the **Exceptional Flow** tab of the Use Case Scenario in the Use Case Specification window.



Here, the exception flow is created for the third step of the basic flow, 3. *Confirm Return*. The exceptional flow has the type named *3.1 Cancel*; that is, the *Cancel* class is assigned as the exception type.The type contains one exceptional flow step named 3.1.1 *Close Item Dialog.*

In the Activity diagram, the exceptional flow is connected using the Exception Handler relationship. The Exception Handler has the type specified, the same as the exception type, in our sample *Cancel* class. Exceptional flow steps in an activity diagram are represented as Call Behavior actions.

You can create a Use Case scenario in the Use Case Specification window using the **Use Case Scenario Sketch** property group, where you can specify the basic, alternative, and exceptional flows, as well as open the Activity diagram.

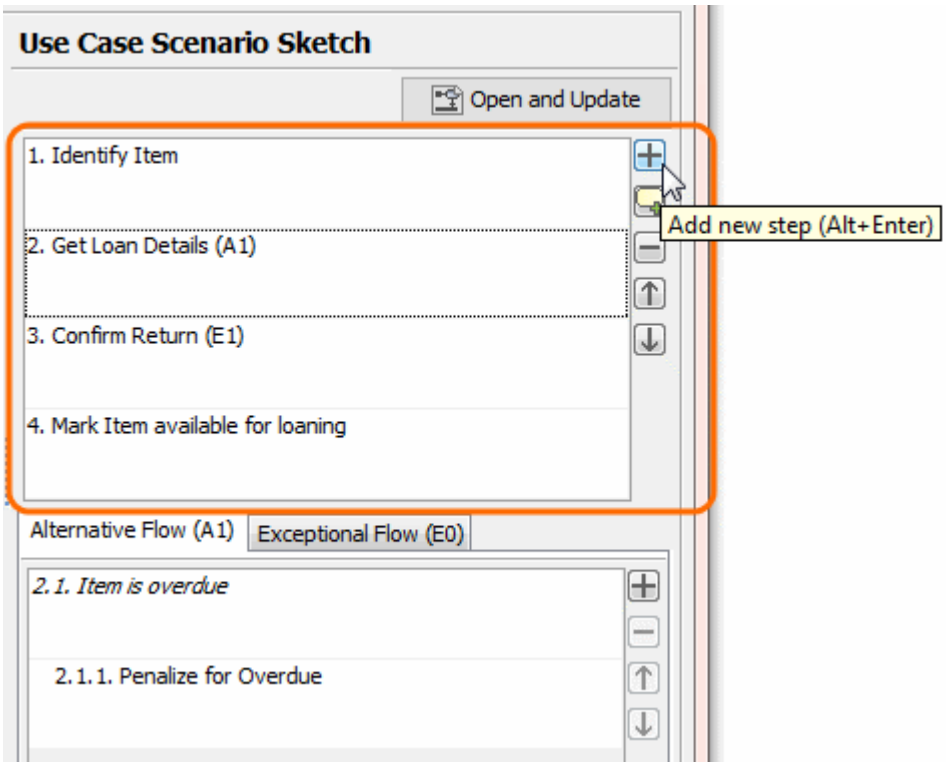
Related pages

- Creating a basic flow(see page 886)
- Adding a basic flow step from other use cases(see page 887)
- Creating an alternative flow(see page 889)
- Creating an exceptional flow(see page 891)
- Representing Use Case scenarios in Activity diagrams(see page 892)
- A Use Case scenario and an Activity diagram mapping schema(see page 893)
- The Use Case scenario created with version 17.0.1 or earlier(see page 898)

Creating a basic flow

To create a basic flow

1. Open the Use Case Specification window and click the **Use Case Scenario Sketch** property group.
2. In the **Basic Flow** area, click the Add new step button or press Alt+ENTER and type a name of the basic flow step. Repeat this action for each basic flow step you need to create.



Steps of the basic flow are numbered automatically. Use the Up and Down buttons to change the order of steps. In the activity diagram, steps are represented according to their order in the Use Case scenario.

Related pages

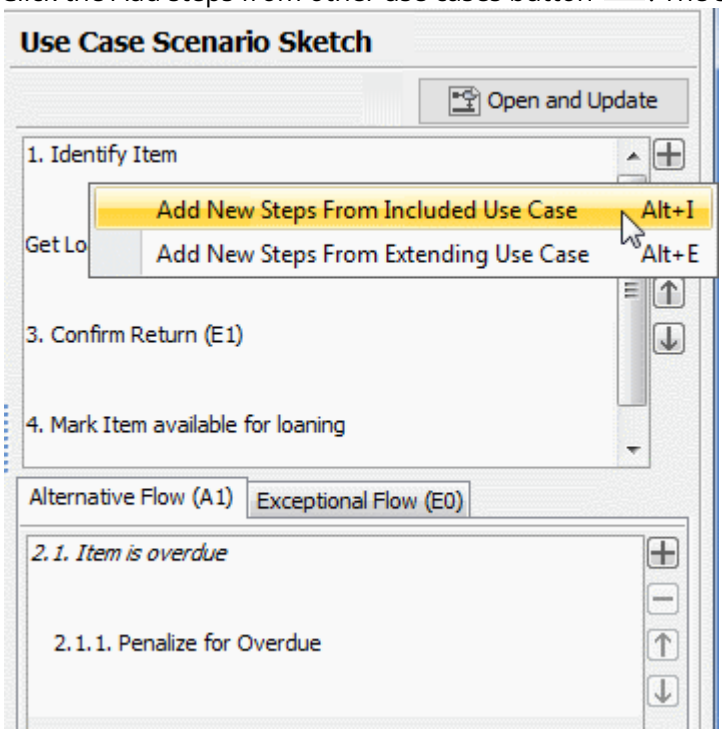
- Model Elements(see page 628)
- Transition(see page 872)
- State Machine diagram(see page 308)
- Activity diagram(see page 311)
- Sequence diagram(see page 304)
- Use Case diagram(see page 300)

Adding a basic flow step from other use cases

Make sure the selected Use Case has included or extending use cases.

To add a basic flow step from other use cases

1. Open the Use Case Specification window and click the **Use Case Scenario Sketch** property group.
2. Select the basic flow step where you want to add steps from another Use Case.
3. Click the Add steps from other use cases button . The shortcut menu opens.



4. Do one of the following:
   - Click **Add New Steps From Included Use Case** or press ALT+I to add steps from the included Use Case.
   - Click **Add New Steps From Extending Use Case** or press ALT+E to add steps from the extending Use Case.

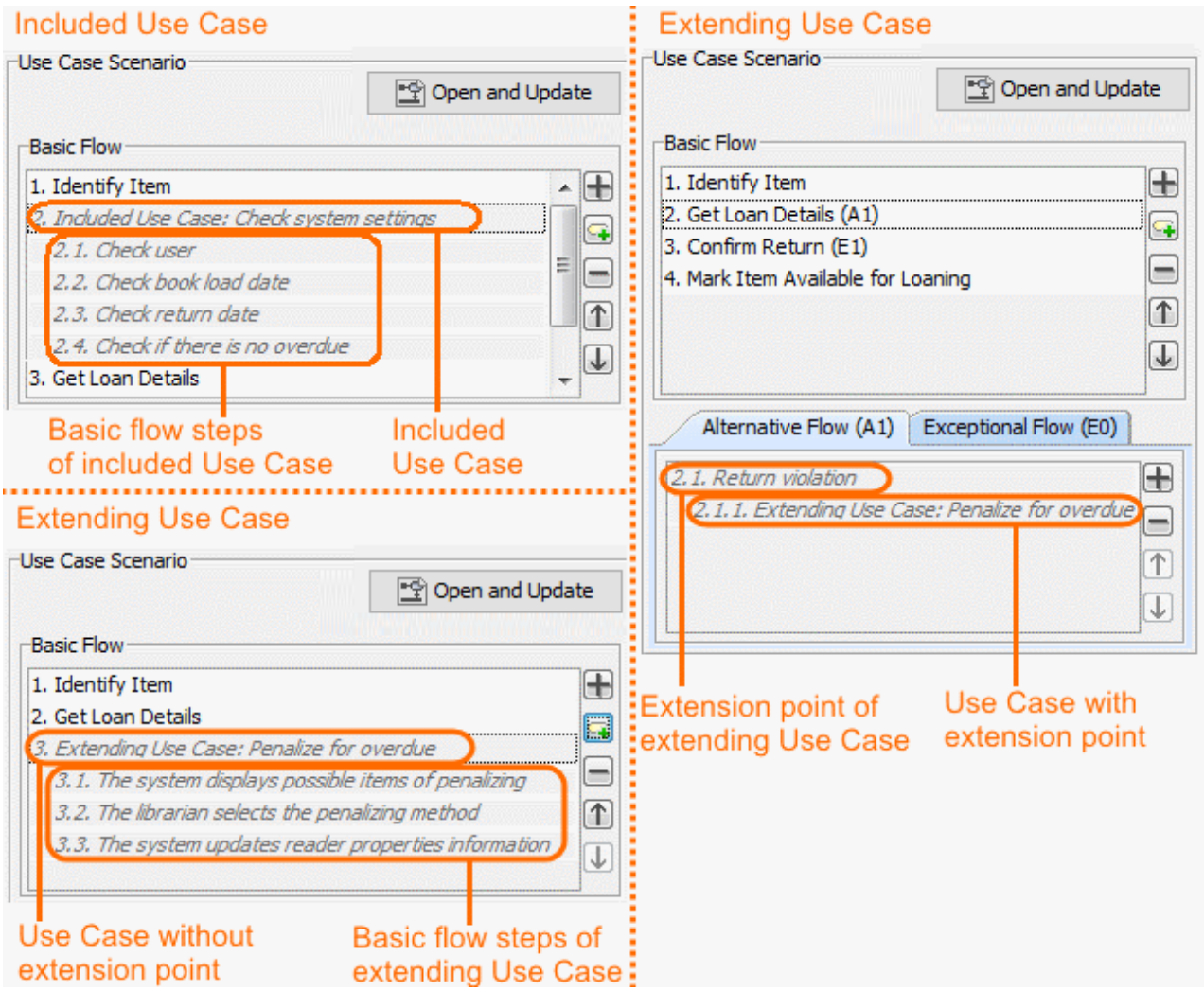
   Depending on your choice, the **Select Included Use Case** or **Select Extending Use Case** dialog opens.

5. In the opened dialog, select the use case(s) from which you want to add steps to the selected step of the basic flow. Click **OK** after you have selected a Use Case.

- Steps from included use cases are added to the basic flow as substeps of the newly created basic flow step.
- Steps from extending use cases with no extension points use cases are added to the basic flow as substeps of the selected basic flow step.
- Steps from extending use cases with extension points are added as steps of an alternative flow to the selected basic flow step. The extension point is considered the alternative condition.

See the examples in the following figures.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Steps added from included or extending use cases are read-only. You are not allowed to modify them in the current scenario.
You can modify added steps only in use cases from which these steps are added.

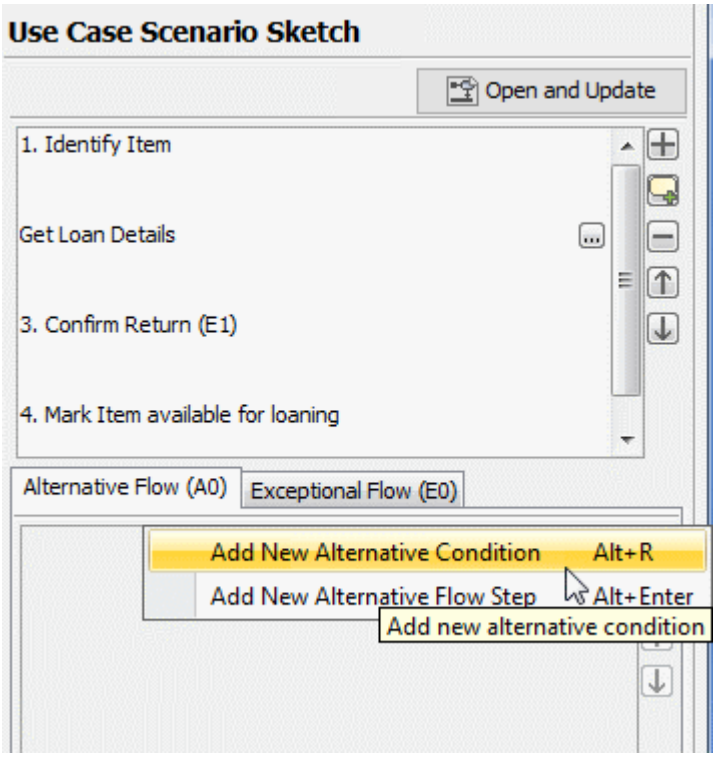
Related pages

- Model Elements(see page 628)
- Transition(see page 872)
- State Machine diagram(see page 308)
- Activity diagram(see page 311)
- Sequence diagram(see page 304)
- Use Case diagram(see page 300)

Creating an alternative flow

To create an alternative flow

1. Open the Use Case Specification window and click the **Use Case Scenario Sketch** property group.
2. Select the basic flow step you want to create an alternative flow for.
3. In the **Alternative Flow** tab, click the Add new button . The shortcut menu opens.



4. Do one of the following:
   - Select the **Add New Alternative Condition** command or press ALT+R to add a new alternative condition. A new alternative condition is created together with the alternative flow step. Type the name of the alternative condition. The alternative condition for the basic flow is created.
   - Select the **Add New Alternative Flow Step** command. Press ALT+ENTER to add a new alternative flow step. If there was an alternative condition created in the alternative flow already, a new alternative flow step is created for the selected alternative condition. If in the alternative flow there was no alternative condition, a new alternative flow step is created together with a new alternative condition. Now you can type the name of the new alternative flow step.

Conditions and steps of the alternative flow are numbered automatically. Use the Up and Down buttons to change the order of conditions in the alternative flow or the order of steps in a particular condition. In the Activity diagram, steps are represented according to their order in the Use Case scenario.

Related pages

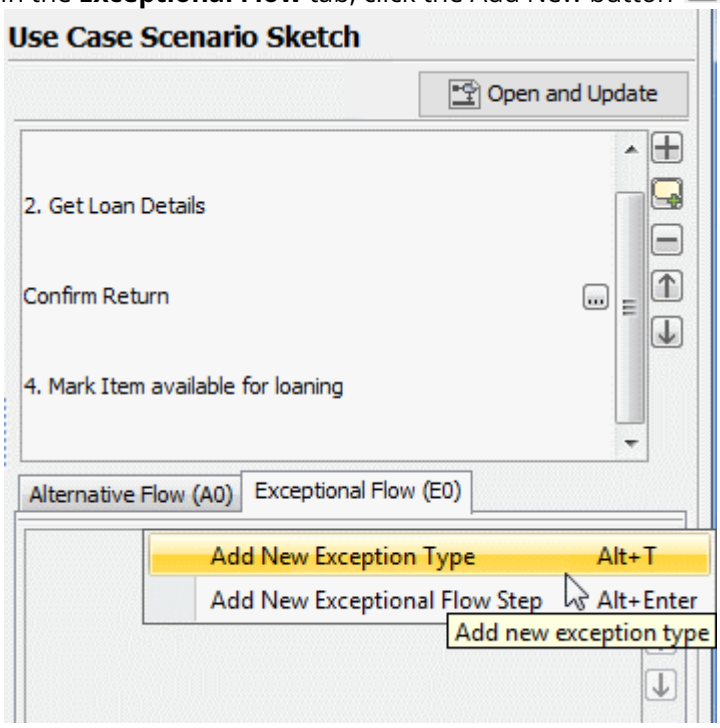
- Model Elements(see page 628)
- Transition(see page 872)
- State Machine diagram(see page 308)
- Activity diagram(see page 311)
- Sequence diagram(see page 304)
- Use Case diagram(see page 300)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Creating an exceptional flow

To create an exceptional flow

1. Open the Use Case Specification window and click the **Use Case Scenario Sketch** property group.
2. In the use case scenario, select the basic flow step you want to create an exceptional flow for.
3. In the **Exceptional Flow** tab, click the Add New button . The shortcut menu opens.



4. Do one of the following:
   - Select the **Add New Exception Type** command or press Alt+T. The drop down list opens. Type the name of a new exception type. The exception type is created (a new class is created). The created exception type contains an unnamed exceptional flow step.
   - Select the **Add New Exceptional Flow Step** command or press Alt+Enter. If there was an exceptional type created in the exceptional flow already, a new exceptional flow step is created for the selected exception type. If there was no exception type in the alternative flow, a new exceptional flow step is created together with a new exception type. Now you can type the name of the new exceptional flow step.

Types and steps of the exceptional flow are numbered automatically. Use the Up and Down buttons to change the order of types in the exceptional flow or the order of steps in the particular type. In the Activity diagram, steps are represented according to their order in the Use Case scenario.

Related pages

- Model Elements(see page 628)
- Transition(see page 872)
- State Machine diagram(see page 308)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Activity diagram(see page 311)
- Sequence diagram(see page 304)
- Use Case diagram(see page 300)

Representing Use Case scenarios in Activity diagrams

After you have created a Use Case scenario, you can represent this scenario in an Activity diagram. The following procedure describes how to represent the Use Case scenario in the Activity diagram and how to turn the automatic layout on or off in the Activity diagram when representing the Use Case scenario.

To represent a Use Case scenario in an Activity diagram

1. Open the Use Case Specification window and click the **Use Case Scenario Sketch** property group.
2. Click the **Open Activity Diagram** button

. The Activity diagram opens with the Use Case scenario represented on it.

You need to reopen the Activity diagram each time after you make modifications to the Use Case scenario in the Specification window(see page 913). Modifications to the Activity diagram where the Use Case scenario is represented automatically appear in the textual Use Case scenario.

> For Use Cases having read-only accessibility, the use Case Scenario cannot be represented on the Activity diagram.

In the Activity diagram, all symbols are laid out automatically every time the diagram is opened.

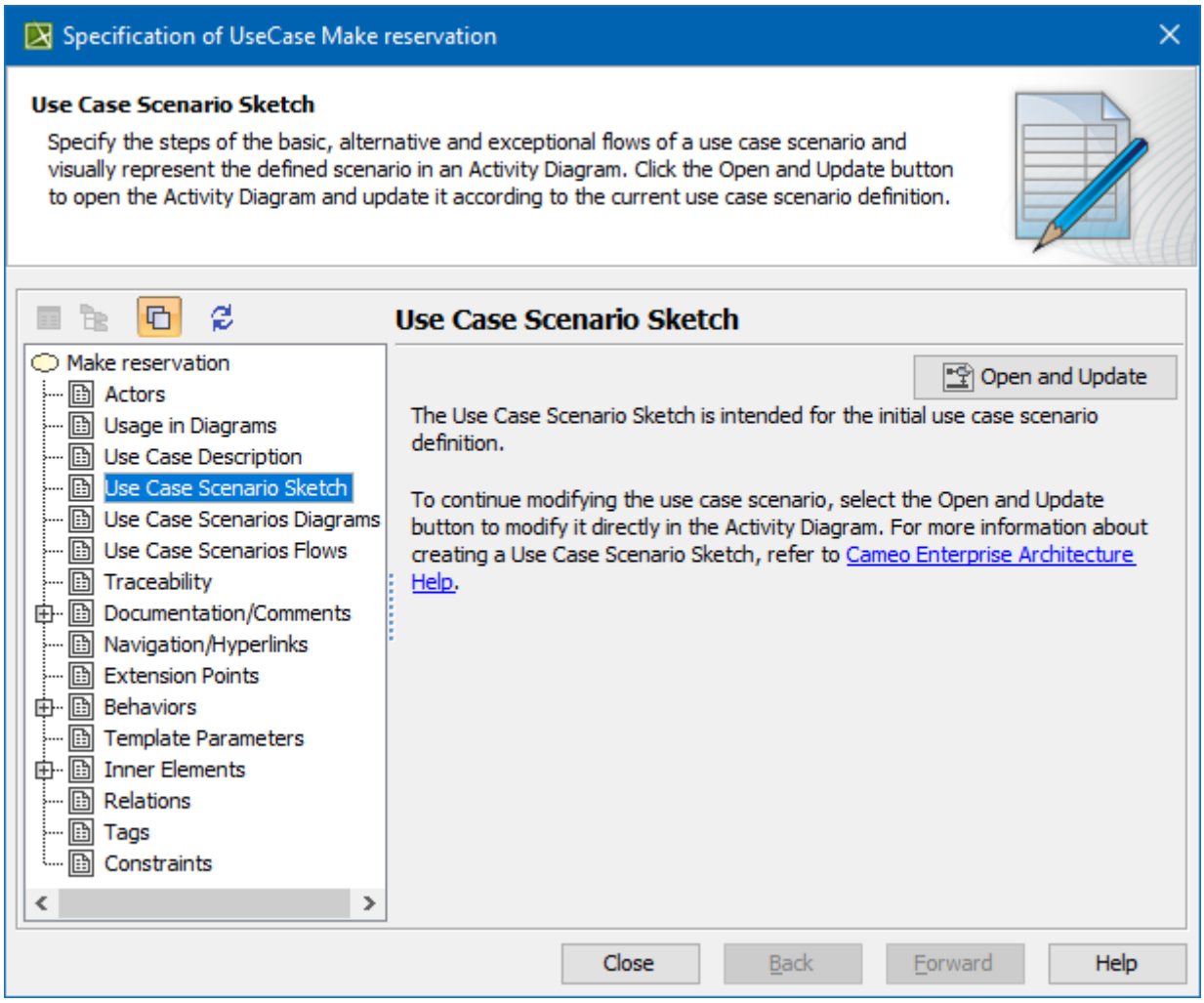
> To leave symbols in the same place while opening the Activity diagram after modifying the Use Case scenario in the Specification window(see page 913), change the **Layout use case scenario activity diagram** option value to *false*. This option appears in the **Project Options** dialog, in the **General project options** group.

Your modeling tool might not be able to interpret the Use Case scenario in some cases. These are the most common reasons why:

There is no merge node or call behavior action after the element.
There is no initial node in Activity.
There is no final node in Activity.
There are no outgoing control flows from the element.
There is no merge node after the element.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

There are no outgoing flows from the element.
There is no alternative flow step after the element.
Cyclic scenario.



Use Case Scenario cannot be interpreted

Related pages

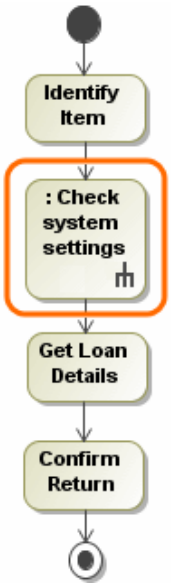
- Model Elements(see page 628)
- Transition(see page 872)
- State Machine diagram(see page 308)
- Activity diagram(see page 311)
- Sequence diagram(see page 304)
- Use Case diagram(see page 300)

A Use Case scenario and an Activity diagram mapping schema

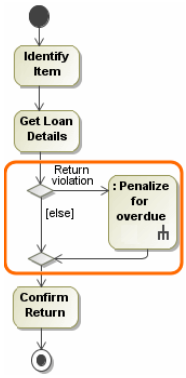
To understand the Use Case scenario representation in the Activity diagram, see the following table.

Use Case Scenario	Mapping in Activity diagram	Example
One (the first) Basic flow(see page 882) step	1. The Activity with the Activity diagram is created under the owning Use Case just after the first basic flow step is created. 2. The name of the created Activity and Activity diagram is the same as the owning Use Case name. 3. In the Activity diagram, the Call Behavior Action is created for the basic flow step. 4. The Initial Node is created before the Call Behavior Action. 5. The Final Node is created after the Call Behavior Action. 6. The Control Flow relationships are created from the Initial Node to the Call Behavior Action and from the Call Behavior Action to the Final Node.	
Basic flow(see page 882) steps	1. The Call Behavior Action is created for each basic flow step. 2. The Initial Node is created before the first Call Behavior Action. 3. The Final Node is created after the last Call Behavior Action. 4. The Control Flow relationships are created from the Initial Node to the first Call Behavior Action, between each Call Behavior Action, and from the last Behavior Action to the Final Node.	

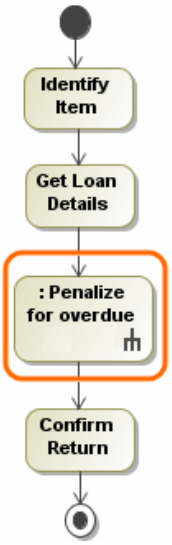
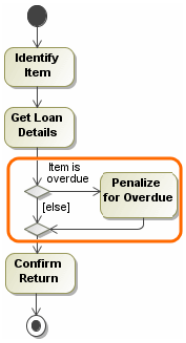
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Use Case Scenario	Mapping in Activity diagram	Example
Included use case(see page 887)	1. The Call Behavior Action is created for the basic flow step of the included Use Case. 2. The Call Behavior Action is inserted to the activity basic flow according to the order it was inserted in the basic flow. 3. The Call Behavior Action is connected with the Control Flow relationships. The Call Behavior Action is not named. 4. The Call Behavior Action has the Activity behavior defined. This Activity name corresponds to the name of the included Use Case. 5. The activity (the Behavior or the Call Behavior Action) is owned by the included Use Case. 6. If the included Use Case has its own Use Case scenario, this scenario is represented in the activity; the Activity diagram is created inside the Activity and flows are represented. To represent the included Use Case from the Activity diagram to the Use Case scenario, you must follow all the rules. Additionally, you must connect the including Use Case with the included Use Case with the included relationship in your project.	

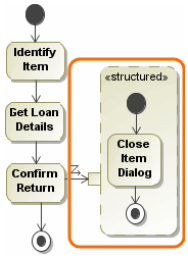
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Use Case Scenario	Mapping in Activity diagram	Example
Extending use case(see page 887) (with the extension point)	1. The extending Use Case with the extension point is added as an alternative flow; the Decision and Merge nodes are created. 2. The Decision node name corresponds the name of the alternative condition. 3. The Call Behavior Action is created for the alternative flow step of the extending Use Case. The Call Behavior Action is not named. 4. The Call Behavior Action has the Behavior defined: the Activity whose name corresponds to the name of the extending Use Case. 5. The Activity (the Behavior or the Call Behavior Action) is owned by the extending Use Case. 6. If the extending Use Case has its own Use Case scenario, this scenario is represented in the Activity diagram; the Activity diagram is created inside the Activity under the extending Use Case. 7. The *[else]* Guard property is defined for the Control Flow created from the Decision node to the Merge node. To represent the extending Use Case from the Activity diagram to the Use Case scenario, you must follow all the rules described above. Additionally, you must connect the extended Use Case with the extending Use Case with the extend relationship in your project.	

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Use Case Scenario	Mapping in Activity diagram	Example
Extending use case (see page 887)(without any extension point)	1. The Call Behavior Action is created for the basic flow step of the Use Case scenario. 2. The Call Behavior Action is connected with the Control Flow relationships according to the extending Use Case order in the basic flow. 3. The Call Behavior Action is not named. 4. The Call Behavior Action has the Behavior defined - the Activity whose name corresponds to the name of the extending Use Case. 5. The Activity (the Behavior or the Call Behavior Action) is owned by the extending Use Case. 6. If the extending Use Case has its own Use Case scenario, this scenario is represented in the Activity; the Activity diagram is created inside the Activity under the extending Use Case. To represent the extending Use Case from the Activity diagram to the Use Case scenario, you must follow all the rules described above. Additionally, you must connect the extended Use Case with the extending Use Case with the extend relationship in your project.	
Alternative flow(see page 882): alternative condition, alternative flow step	1. The alternative flow is interrupted in the basic flow by using the Decision and Merge nodes. 2. The Decision and Merge nodes are created after the Call Behavior Action. The Call Behavior Action in the Use Case scenario represents the basic flow step of this alternative flow. In other words, in an Activity diagram, the elements of the alternative flow are created after the basic flow step this alternative flow belongs to. 3. The Decision node name corresponds to the name of the alternative condition. 4. The Call Behavior Action is created for each alternative flow step. 5. All these elements are connected with the Control Flow relationships. 6. The *[else]* Guard property is defined for the Control Flow created from the Decision node to the Merge node.	

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Use Case Scenario	Mapping in Activity diagram	Example
Exceptional flow(see page 882): exception type, exceptional flow step	1. The Structured Activity Node with the Input Pin is created. 2. The Call Behavior Action for which the exceptional flow was created is linked with the Input Pin using the Exception Handler relationship. 3. The Class element is created under the Activity. The Class corresponds to the exception type in the Use Case scenario, that is, the Class name corresponds to the exception type name in the Use Case scenario. 4. The Class is assigned to the Input Pin as a type property. 5. In the Structured Activity Node the Call Behavior Action is created for each Exceptional Flow step. 6. The Call Behavior Actions inside the Structured Activity Node are connected with the Control Flow relationships. 7. In the Structured Activity Node the Initial Node is created before the first Call Behavior Action, and the Final Node is created after the last Call Behavior Action. 8. In the Structured Activity Node, the Initial Node is connected with the first Call Behavior Action by using the Control Flow relationship and the last Call Behavior Action is connected with the Final Node by using the Control Flow relationship.	

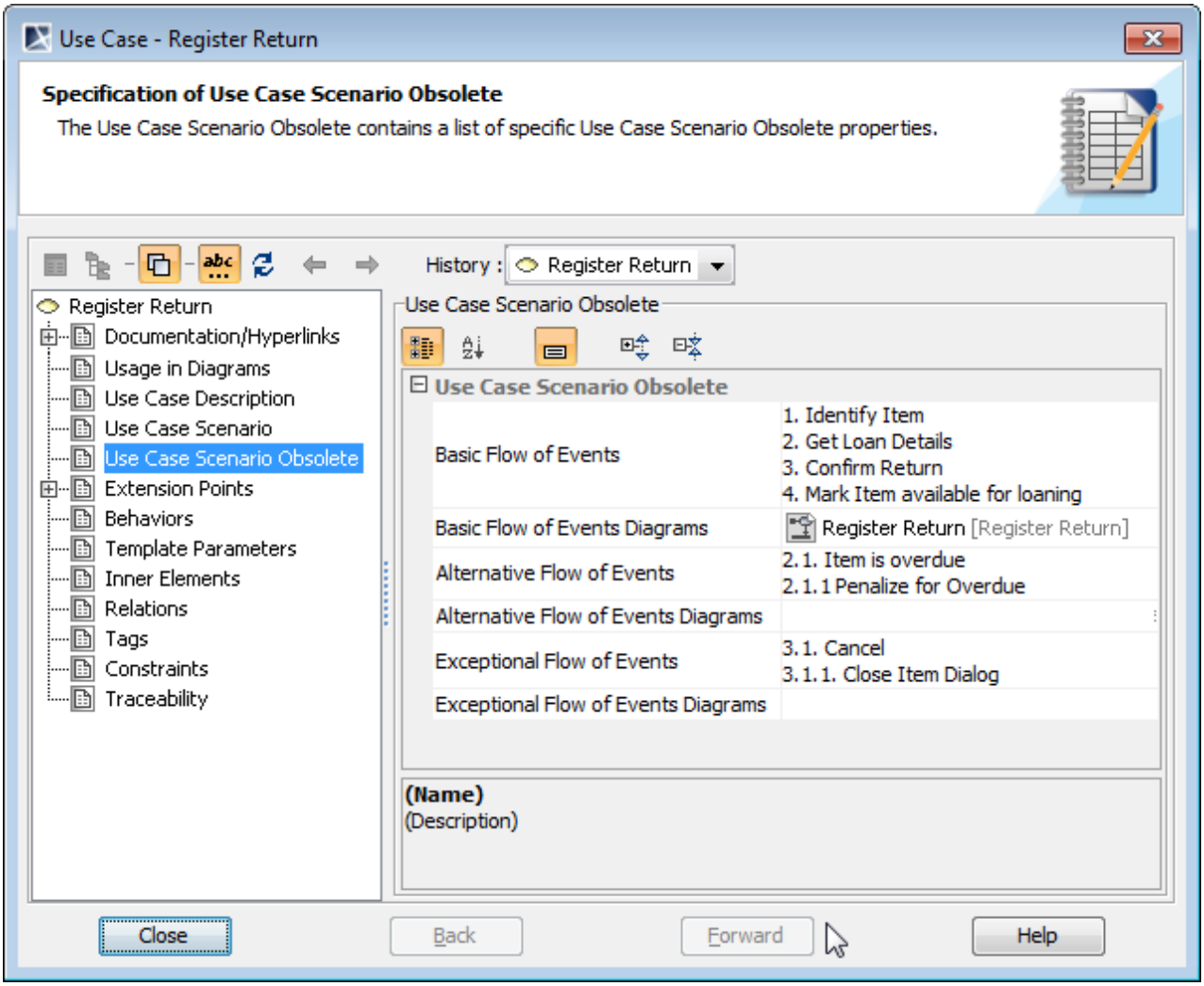
Related pages

- Model Elements(see page 628)
- Transition(see page 872)
- State Machine diagram(see page 308)
- Activity diagram(see page 311)
- Sequence diagram(see page 304)
- Use Case diagram(see page 300)

The Use Case scenario created with version 17.0.1 or earlier

Beginning with version 17.0.2 of MagicDraw or other modeling tools developed by No Magic Inc., Use Case scenarios of projects created with earlier versions are stored in the **Use Case Scenario Obsolete** property group in the Use Case Specification window(see page 913).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



- To see the **Use Case Scenario Obsolete** property group, change the property display mode to **Expert**.
- The **Use Case Scenario Obsolete** property group is displayed only if the Use Case Description Profile(see page 880) is loaded.

Related pages

- Model Elements(see page 628)
- Transition(see page 872)
- State Machine diagram(see page 308)
- Activity diagram(see page 311)
- Sequence diagram(see page 304)
- Use Case diagram(see page 300)

Value Specification

You can create a standalone value specification in a model from the appropriate element's shortcut menu.

You can use any of the following ways to create a value specification:

- Via the element's shortcut menu.
- Via the element's Specification window.
- By defining a default value to an element.

You can also change an assigned value specification after it has been created.

To create a value specification from the element's shortcut menu

1. In the Containment tree, select an appropriate element.
2. From the selected element's shortcut menu, select **Create Element** > **Value Specification** and choose a suitable value specification.

To assign a value specification in the element's Specification window(see page 913)

1. In the Specification window of the selected element, choose an appropriate property value cell and click the **Show Shortcut Menu** button (the black arrow).
2. In the shortcut menu, click **Value Specification** and select a value specification.

To create a value specification automatically

1. Assign a default value to a property for which you want to create a value specification.
2. The value specification of the corresponding type will be assigned automatically according to the assigned default value in your model.

To change an assigned value specification

1. In the Specification window of the selected element, choose an appropriate property value cell and click the **Show Shortcut Menu** button (the black arrow).
2. From the shortcut menu, select **Value Specification** > **Delete** <value specification>.
3. Assign a new value specification.

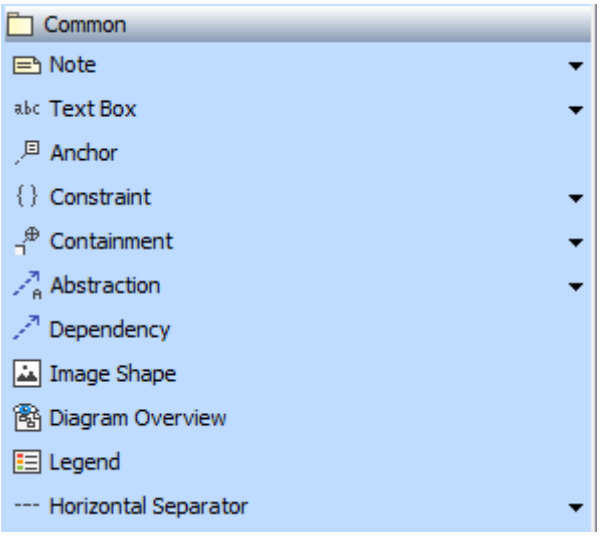
Related Pages

- Model Elements(see page 628)
- Specification window(see page 913)
- Transition(see page 872)
- State Machine diagram(see page 308)
- Activity diagram(see page 311)
- Sequence diagram(see page 304)
- Use Case diagram(see page 300)

Auxiliary diagram symbols

This section lists auxiliary diagram symbols, including simple text box, image shape, whole diagram overview, and others. These symbols can be used in all diagrams.

You can create auxiliary diagram symbols from the **Common** category in the diagram palette.



Representation of the Common category in the Diagram pallet

Refer to the following symbol descriptions:

Related pages

- UML elements(see page 628)
- Diagramming(see page 482)

Image Shape

The image shape provides a simple and quick way to insert a image into a diagram. Images may include logos, graphs, tables, or others. The default shape size after the insertion is the actual image size.

You can format image shape representation properties in the **Symbol Properties** dialog(see page 975).

To insert an image on the diagram pane

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

1. On the diagram palette, in the **Common** category, click the **Image Shape** button or press the **I** key.
2. Click on the diagram pane. The **Open** dialog opens.
3. Browse for an image in the *.gif, *.jpg, *.jpeg, *.svg, or *.png format and click **Open**.

Related pages

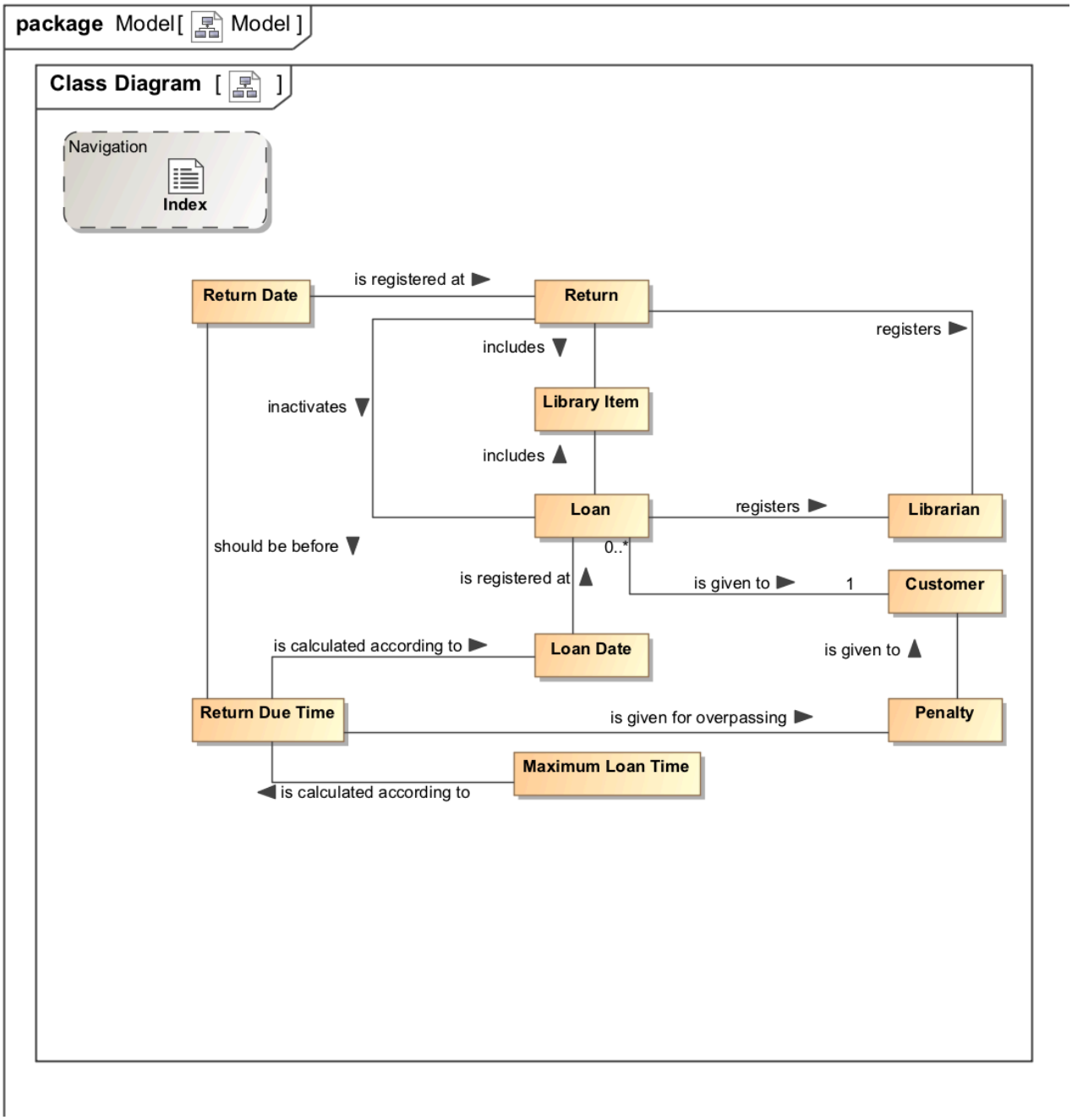
- UML elements(see page 628)
- Auxiliary Diagram Symbols(see page 900)
- Diagramming(see page 482)

Diagram Overview

> **Functionality Availability**
> The diagram overview functionality is available in Standard, Professional, Architect, and Enterprise editions.

A diagram overview shape provides the ability to overview other diagrams(see page 526) on a diagram pane. When using the diagram overview shape, you can overview dependency matrices(see page 334), tables(see page 405), and relation maps(see page 364) on the diagram pane.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Example of using the Diagram Overview functionality

You can format the diagram overview representation properties in the **Symbol Properties** dialog(see page 975).

To open the particular diagram's Specification window(see page 913), double-click the diagram overview shape. The description of each diagram property appears in the same window. Descriptions are found in the description area at the bottom of the Specification window.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To create a diagram overview shape from the diagram palette

---

1. On the diagram palette, under the **Common** category, click the **Diagram Overview** button. The **Select Diagram** dialog opens.
2. Select the diagram for which you want to create a diagram overview shape and click **OK**.
3. Click the diagram pane outside the dialog. The diagram overview shape is created and shows the full content of the corresponding diagram.

Related pages

- UML elements(see page 628)
- Auxiliary Diagram Symbols(see page 900)
- Diagramming(see page 482)

Sample model

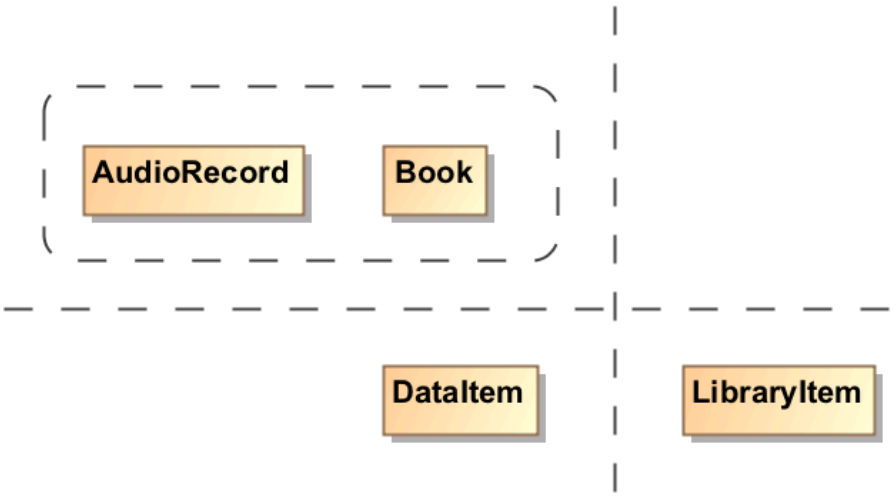
A diagram overview sample project in *<modeling tool installation directory>\samples\product features\diagram overview.mdzip*.

Separator

You can use either a horizontal or vertical separator to rule off different parts of a diagram. The **Rotate** button allows changing the separator from the horizontal to vertical position, and vice versa.

See some examples of the types of separators in the following figure.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Using the vertical, horizontal, and rectangular separators

You can format the separator symbol properties in the **Symbol Properties** dialog(see page 975).

You can display text next to the separator as plain text or HTML text. You can edit the HTML text using the HTML editor(see page 1055).

To change separator properties

1. From the separator's shortcut menu, select **Symbol Properties** or select the separator and press Alt+Enter.
2. The **Symbol Properties** dialog opens. In that dialog, set particular property values, such as **Text Position**, **Line Style**, **Line Width** and other, as you need.
3. Click **Close**, when you are done.

To change the separator's position

1. Select the separator on the diagram pane.
2. Click the *Rotate* button at the corner of the selection.

To select the type of separator from the diagram palette

1. Find the Separator under the **Common** pallet.
2. Click the arrow on the right side of the button to reveal a drop-down menu with the following options: **Vertical Separator**, **Horizontal Separator**, and **Rectangular Shape**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

3. Select the separator you would like to use for your model.

To set the text position on a separator

1. From the separator's shortcut menu, select **Symbol Properties**.
2. In the **Symbol Properties** dialog, change the **Text Horizontal Position** property to one of the following: **Left** (Default), **Center**, or **Right**.

To set a separator line style

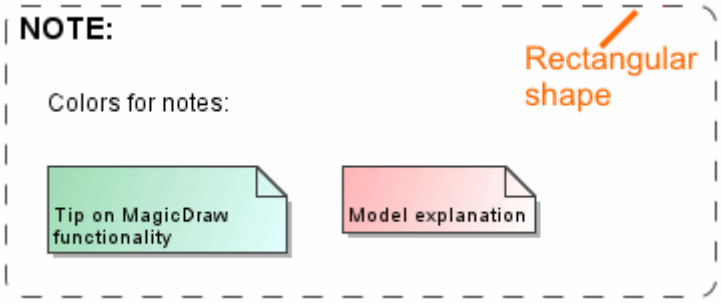
1. From the separator's shortcut menu, select **Line style**.
2. Select the style you need: **Dashed** (Default), **Solid**, or **Dotted**.

Related pages

- Model element(see page 628)
- Auxiliary Diagram Symbols(see page 900)
- Diagramming(see page 482)

Rectangular shape

You can use a rectangular shape to delineate different parts of a diagram, as seen in the following figure.



Example of a rectangular shape

You can format the rectangular shape representation properties in the **Symbol Properties** dialog(see page 975).

A text next to the separator can be displayed as plain text or HTML text. You can edit the HTML text using the HTML editor(see page 1055).

To change rectangular shape properties

---

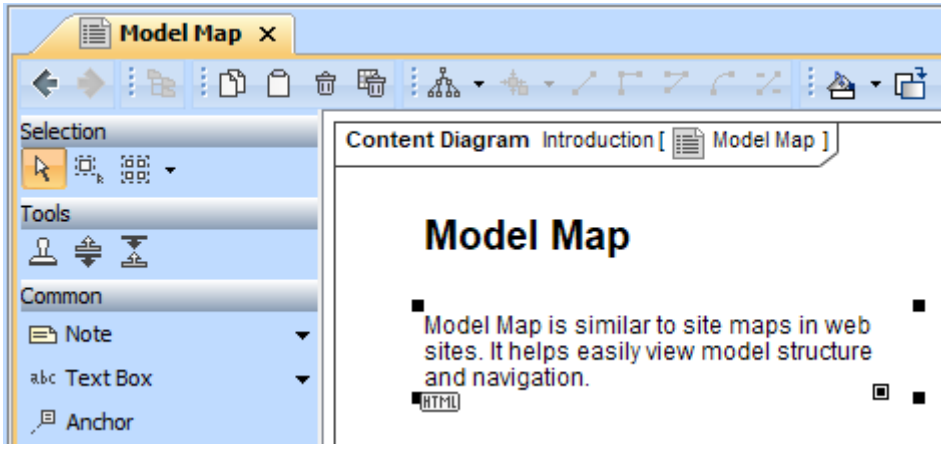
1. From the rectangular shape's shortcut menu, select **Symbol Properties** or select the rectangular shape and press Alt+Enter.
2. The **Symbol Properties** dialog opens. In that dialog, set particular property values, such as **Fill Color**, **Line Style**, **Line Width** and other, as you need.
3. Click **Close**, when you are done.

Related pages

- UML elements(see page 628)
- Auxiliary Diagram Symbols(see page 900)
- Diagramming(see page 482)

Text Box

A text box is a box where you can type any text. Text in the text box can be displayed as plain text or HTML text.

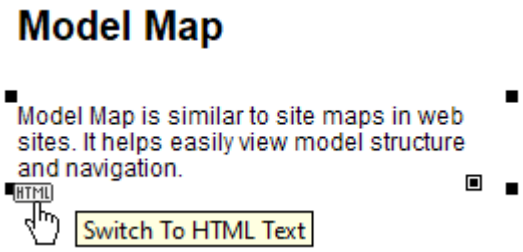


Example of a text box

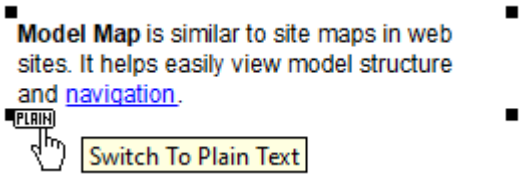
You can format the text box symbol properties in the **Symbol Properties** dialog(see page 975).

To switch the text of the text box to HTML or Plain text do one of the following

- On the diagram pane, select the text box. Click the button that appears on the lower left corner of the shape:
  - The **HTML** button converts the text to HTML text.



  - The **Plain** button converts the text to plain text.

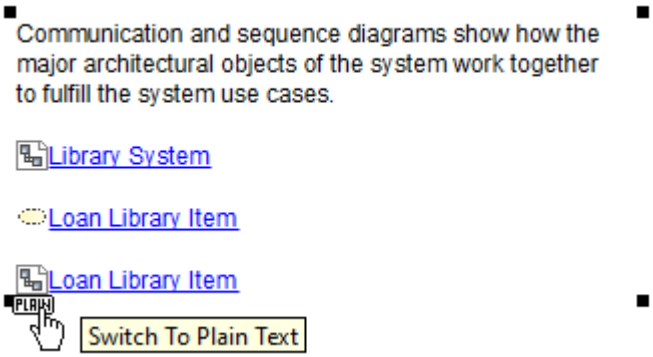


- Open the selected text's shortcut menu and select or click to clear the HTML Text command.

You can edit the HTML text using the HTML editor(see page 1055).

To add a hyperlink to an element on a text box

1. Select one or more elements in the Model Browser(see page 29), or Specification window(see page 913).
2. Drag the selected elements on the text box on the diagram pane. The hyperlinks to specifications of dragged elements are created on the text box. Click the link on the text box to open the Specification window of the selected element.



Text Format
After the hyperlink to the element on the text box is added, the text format in the text box changes to HTML.

Related pages

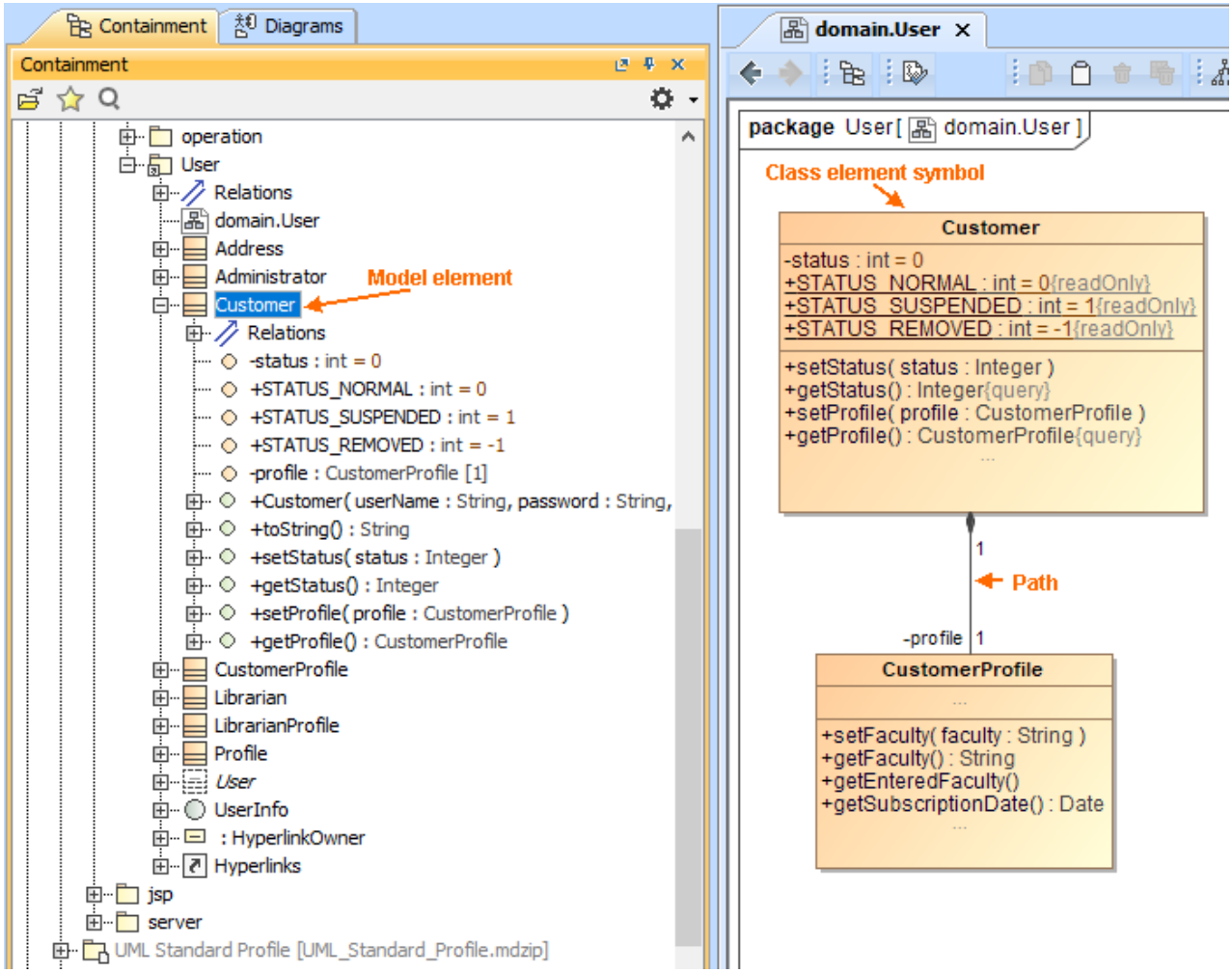
- Model elements[159]
- Auxiliary Diagram Symbols(see page 900)
- Diagramming(see page 482)

Working with model elements

The chapter outlines various methods of working with your model's elements. You will find detailed instructions on creating and defining model elements, default property values, extracting, defining hyperlinks, etc. You will also learn how the modeling tool shortcut menus, toolbar, and browser enable you to easily edit model elements.

In the model, a model element is stored in the Model Browser and its symbol or path (for relationships) is represented on the diagram pane, as shown below.

159 https://docs.nomagic.com/display/MD190SP4/Model+elements



Model elements and symbols.

You can represent several symbols or paths of the same element in the diagram.

To create elements in the Containment tree

1. In the Containment Tree, right-click the element owner where you want to create a new model element.
2. From the shortcut menu, select **Create Element**.
3. A list with available elements appears. Select the element you need or start typing an element name in the **Search** box to find it faster.
4. Type the name of the element.

You can also create an element directly on the diagram pane. In this case, a symbol of the element and the element in the diagram owner is created.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To create elements on the diagram pane

1. Open the diagram or create a new diagram where you want to create a new model element.
2. From the diagram palette, select an element and drag it on the diagram pane. The element symbol is created.
3. Click the symbol and type the element name.

Containment Tree
Please note, the new element is created in the Model Browser as well.

Related pages

- UML elements(see page 628)
- Creating diagrams(see page 483)
- Understanding the user interface(see page 24)

Creating elements from other resources

It is possible to copy a listed text from other resources (Word, Excel, HTML, etc.) and create as many elements as you need by pasting it. Each item of this list becomes a new element in the model. The basic numbering and bullets are not reproduced in the model.

You can do this either:

- Creating elements in the Containment tree(see page 911)
- Creating elements on the diagram pane(see page 912)

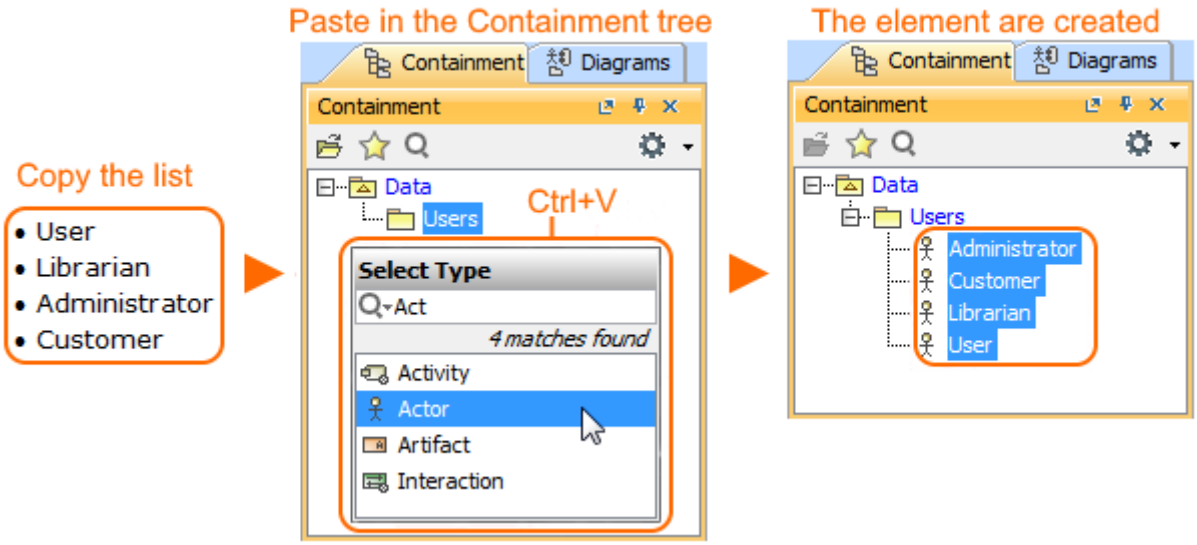
Avoid names duplication, because elements with the same name cannot be created for the same owner.

Creating elements in the Containment tree

To create the elements from the clipboard in the Containment tree

1. Copy a list.
2. Select the possible owner in the Containment tree(see page 32).
3. From the shortcut menu select **Paste** or press Ctrl+V.
4. From the **Select Type** dialog choose the element type.
   The named elements are created in the model. As example, see the figure below where the Actors are created in the Containment tree by pasting the list.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Steps of creating elements from clipboard in Containment tree

Creating elements on the diagram pane

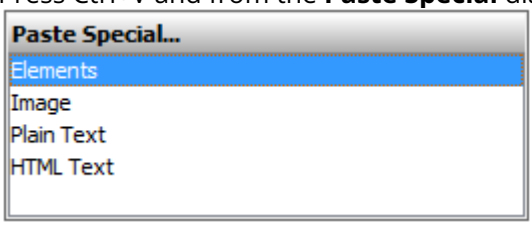
There are two ways to create the elements from the clipboard on the diagram pane:

- Using **Select Type** dialog.(see page 912)
- Using diagram palette.(see page 913)

To create the elements from the clipboard on the diagram pane using **Select Type** dialog

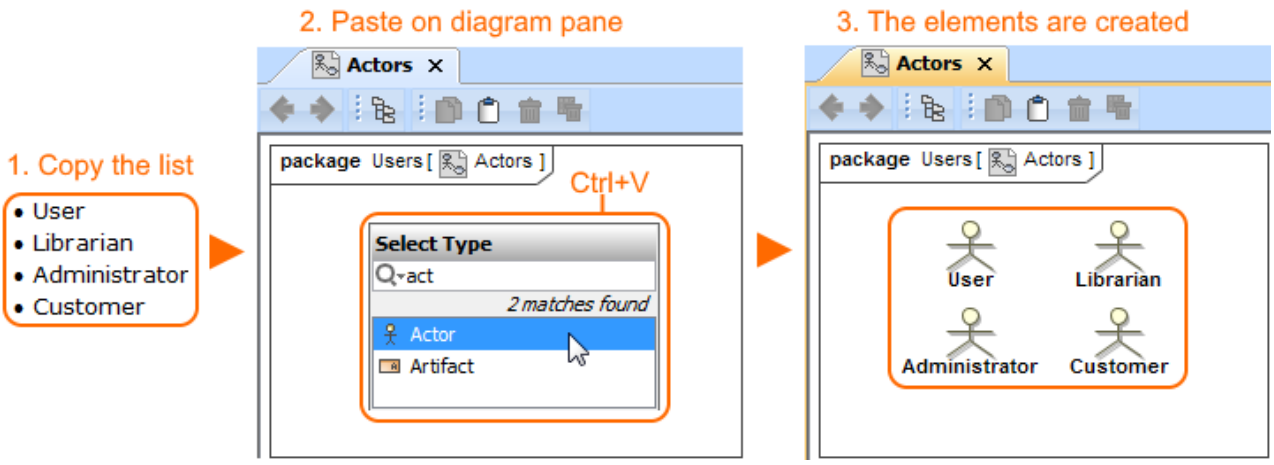
---

1. Copy a list.
2. Click a free space on the diagram pane.
3. Press Ctrl+V and from the **Paste Special** dialog select **Elements**.



4. From the **Select Type** dialog choose the element type.
   The named elements are displayed on the diagram pane and are created in the model.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



To create the elements from the clipboard on the diagram pane using diagram palette

1. Copy a list.
2. From the diagram palette(see page 24) select the element type.
3. Press Ctrl+V.
   The named elements are displayed in the diagram pane and are created in the model.

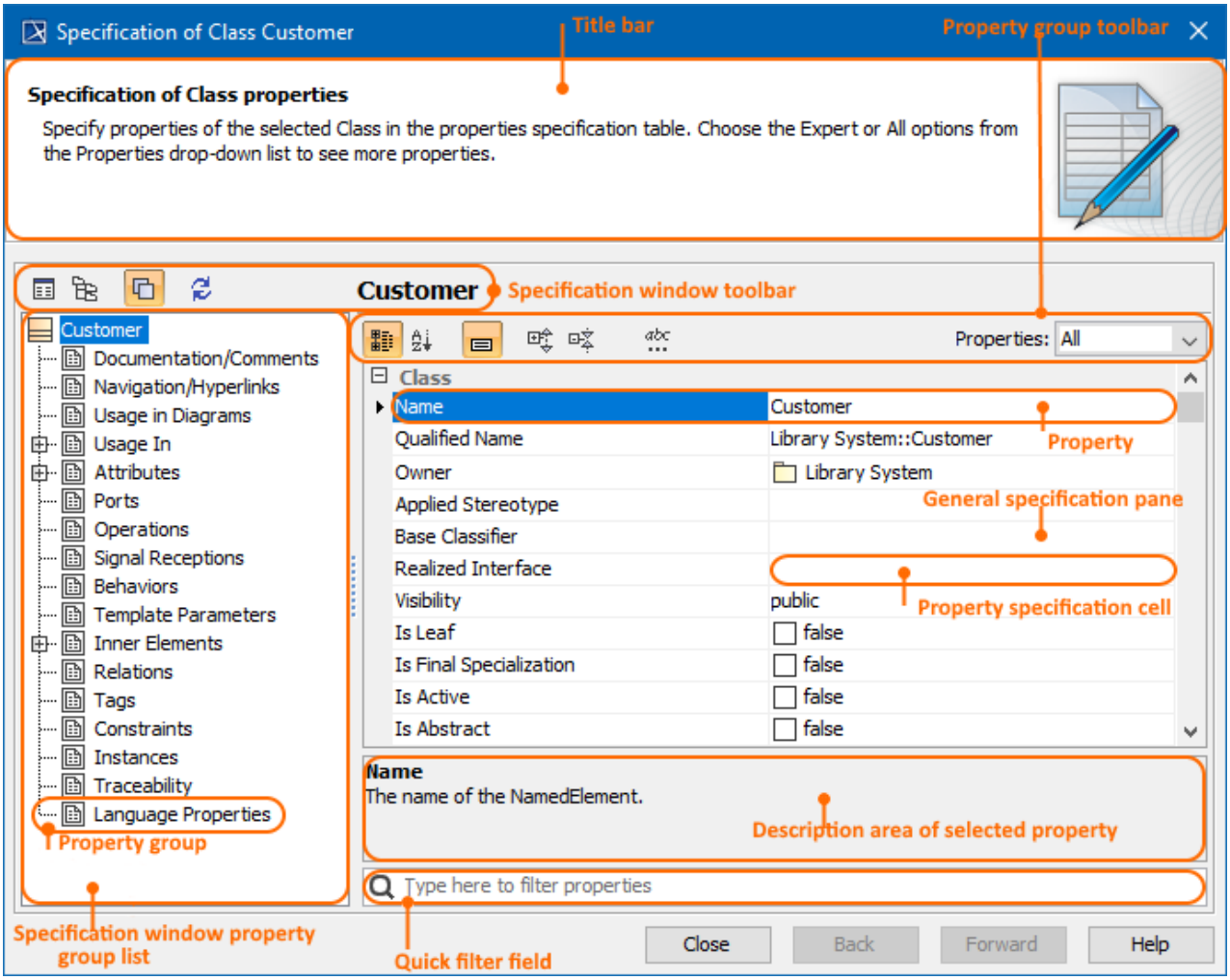
Specification window

You can define all model elements in the Specification window.

MagicDraw shortcut menus, toolbars, and browser help ease the task of editing model elements.

> ⚠ You can also edit model elements and symbol properties directly in the **Properties** tab(see page 55), which is located at the bottom of the Model Browser(see page 29).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The Specification window is used to define UML model elements such as class, package, activity. The Specification window is a non-modal window in which you can edit model element property values(see page 921) and work with a model simultaneously.

To open the Specification window, do one of the following

- From the selected symbol shortcut menu, select **Specification**.
- Double-click a symbol on the Diagram pane or in the Model Browser.
- Select a symbol on the Diagram pane and press ENTER.
- The element Specification window opens when you add a model element to an owning model element in its Specification window. The second Specification window opens on top of the first. Use the **Back to** or **Forward to** arrow buttons for switching between windows.

Related pages

- Functions of Specification window(see page 915)
- Specification window toolbar(see page 916)
- Property group toolbar(see page 917)
-

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Quick filter(see page 918)

Functions of Specification window

Using Specification window you can perform various actions that are necessary when working with model elements. Every model element has its own specification.

The following table describes only common functions of the Specification window.

Function	Description
Add/modify model element properties and inner elements	Choose the desired property group and specify its properties in the general specification pane.
Open referenced element specifications	From the shortcut menu, select **Open Specification** or click . When another element specification is open, you can work with both specifications in the same window.
Navigate between open specifications	To navigate between open specifications, use the **Back** and **Forward** buttons.
Manage relationships	• View all relationships in which the element participates. Modify the name of the relationship. • View/change the direction of the relationship. • Modify the target element. • Create new outgoing or incoming relationships.
Add/edit element documentation	Add element description, describe use case scenarios, or add other relevant information that can be used for generating reports or technical documentation. Documentation also can be written in HTML using the HTML editor(see page 1055).
Manage element hyperlinks	Open, edit, add, or remove hyperlinks (see page 1008)from the selected model element to a file, address, other element / symbol, or requirements.
Manage element tags and tagged values	Create tag definitions, view, create, modify, assign(see page 865) default tagged values.
Manage element constraints	Define new, edit, or apply constraints(see page 690) to an element.
Track element symbol usage in diagrams	Find out in which diagrams the symbol is used. Open these diagrams directly from the specification window.
Select model element in the Containment tree	From the shortcut menu, choose **Select in Containment Tree** or click .

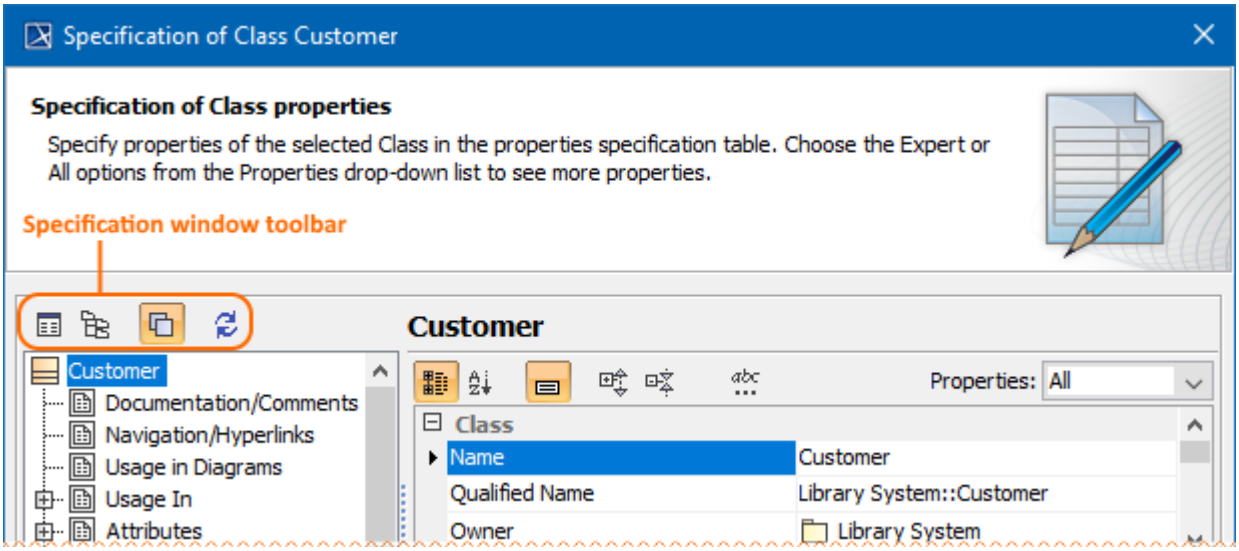
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Function	Description
View element traceability	Monitor element changes and the impact of those changes using traceability(see page 1201).

> Select a specification property and from the shortcut menu, select the action you want to perform.

Related pages

Specification window toolbar



Icon	Name	Description
	Open Specification	Opens specification of the selected referenced element in the same window.
	Select in Containment Tree	Selects the chosen element in the Containment tree of the Model Browser.

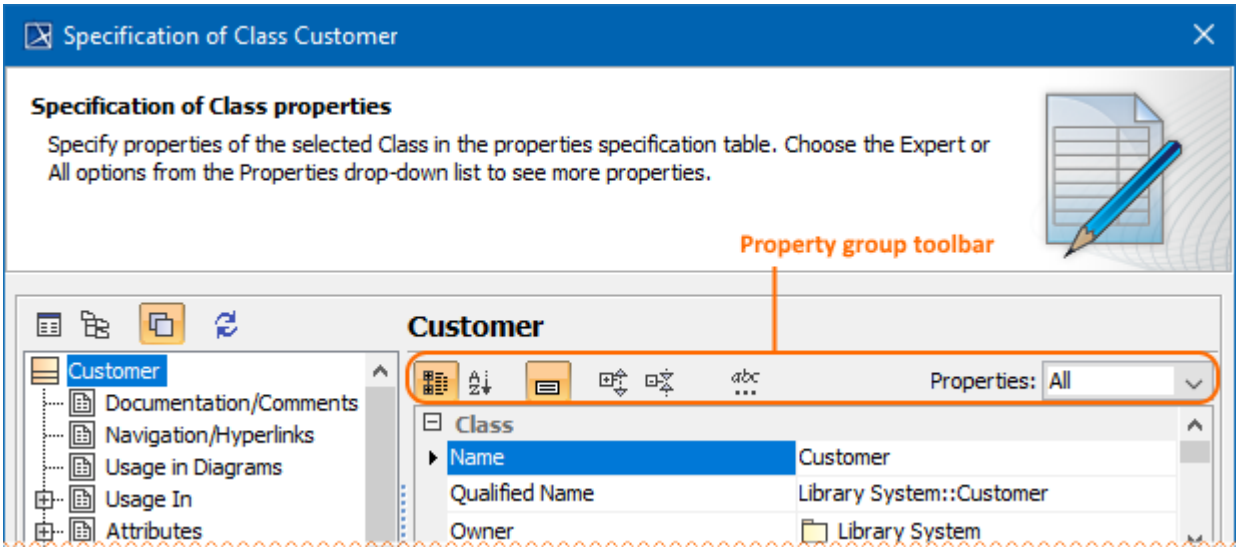
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Icon	Name	Description
	Specification Opening Mode	If selected, the referenced Specification windows are opened in the same window. To navigate between specifications, use **Back** and **Forward** buttons. You can cancel the selection, if you need the Specification window to be open for each element in a separate window, for example, to compare specifications of several elements at the same time.
	Refresh	Refresh data of the Specification window. Refresh the Specification window of an element to see if any changes that are not directly related to that element were made. Also you may need to refresh data if the Specification window is open for a long time.

Related pages

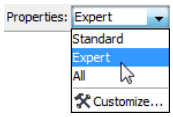
- Functions of Specification window(see page 915)
- Specification window toolbar(see page 916)
- Property group toolbar(see page 917)
- Quick filter(see page 918)

Property group toolbar



Icon	Name	Function
	Categorized view	Groups properties by the categories.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Icon	Name	Function
	Alphabetical view	Lists properties alphabetically.
	Show Description	Shows or hides the description area of the selected property.
	Expand All	Expand all collapsed categories.
	Collapse All	Collapse all expanded categories.
	Strip Multiline Text	If the text in the property cell (e.g., ToDo) covers more than five rows, it is finished with **...** showing only those five rows. The complete text is available in the tooltip or in the cell edit mode.
	Properties	Change the properties display mode to **Standard**, **Expert**, or **All**. Select **Customize**, to customize displayed properties(see page 1001).
	Show Private Inherited	Show the inherited properties.
	Hide Inherited Relations	Hide the relationships that are inherited.

The toolbar configuration can vary according to the selected property group.

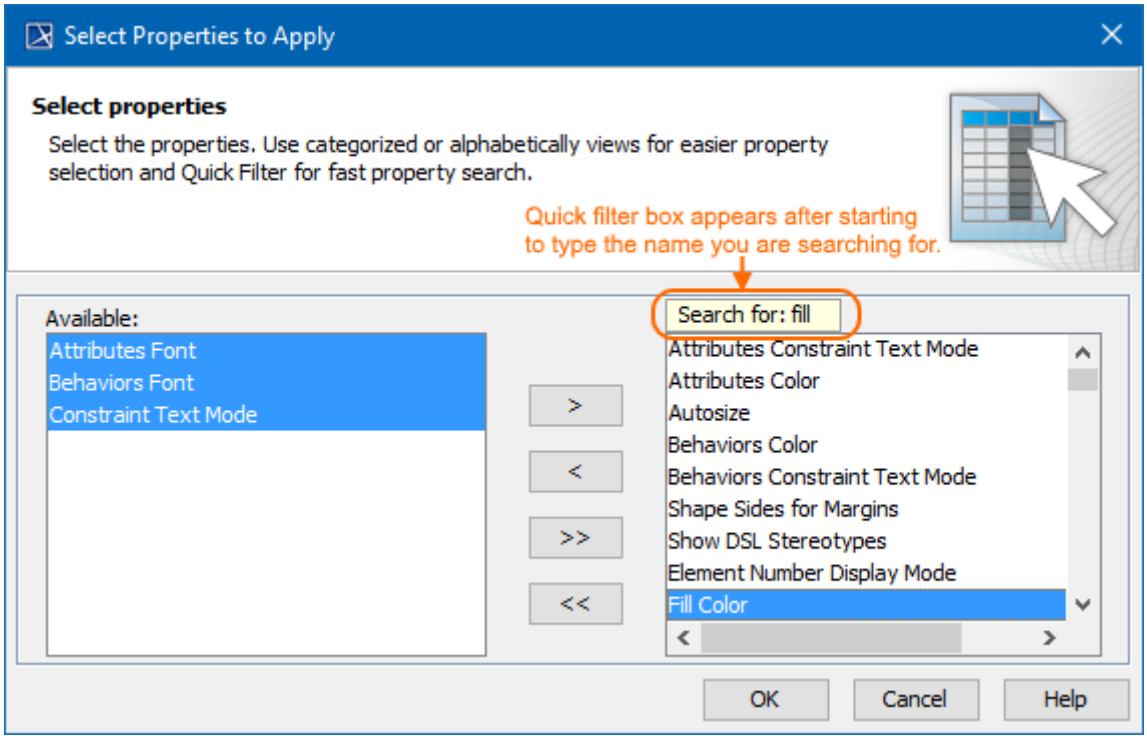
Related pages

- Functions of Specification window(see page 915)
- Specification window toolbar(see page 916)
- Property group toolbar(see page 917)
- Quick filter(see page 918)

Quick filter

To find a property, just start to type the desired property name in the open dialog.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



A Quick filter box appears after starting to type the name you are searching for.

> ⚠ The quick filter box is displayed only when ten or more properties are displayed in the property list.

The quick filter allows you to quickly find the required property in the property list. This is especially handy with a long properties list. Properties can be filtered by the keyword entered in this field.

You can also use the quick filter in Environment Options(see page 65), Project Options(see page 214), and Symbol Properties(see page 975) dialogs in the Specification window as well.

Related pages

- Functions of Specification window(see page 915)
- Specification window toolbar(see page 916)
- Property group toolbar(see page 917)
- Quick filter(see page 918)

Default property values

On this page

- Sharing the default property values(see page 921)

The modeling tool allows you to define the initial (default) properties for elements.

The Default element properties can be defined for:

- The whole project
- The specific diagram

To set the default properties for the whole project

1. From the **Options** menu, select **Project**. The **Project Options** dialog opens.
2. Expand **Default model properties**. Select the desired elements and change their property values.

After creating a new element it will have new property values. Values for previously created elements will not be changed.

To reset element properties to the default value

- In he **Project Options** dialog, **Default model properties** group, click the **Reset to Defaults** button.

Exception: interface attribute default visibility will always be #public, no matter what your settings are.

To set default properties for the specific diagram

1. From the **Diagrams** menu, select **Customize**. The Customize Diagrams(see page 2045) dialog opens.
2. Select the diagram type you want to customize.
3. Click the **Edit** button. The **Customize Diagram Wizard** opens.
4. In the fourth step**Specify toolbar buttons**, click the **Add** button and select **New Button**. The **Edit Button** dialog opens.
5. In the **General** > **Model Element Type** drop-down list, select the element, which default properties you want to customize.
6. In the **Edit Button** dialog, open the **Element Properties** tab.
7. Select the **Specify own values** radio button. Now change the default property values of the chosen element.

Create a new element from the customized diagram toolbar and the element default properties will be the same as you had defined.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Sharing the default property values

You can share the default properties with other users for their new project. After changing the property values in the **Project Options** dialog, create a project template.

To create a template

1. From the File menu select **Export** > **Template**
2. Enter the template name and description.
3. Click **OK**. A template will be created in *<moleding tool installation directory>\ templates*.

To import a template

1. From the **File** menu, select **New Project**. The **New Project** dialog opens.
2. From the **Other** domain, select the **Project from Template** icon.
3. In the **Select template** tree, choose your template. The project options are imported to the project together with the template.

Related pages

- Working with model elements(see page 909)

Editing property values

You can edit property values in:

- Specification window(see page 913)
- Tables(see page 405)
- **Element** tab on the **Properties** tab(see page 55) in the Model Browser

> Property values in gray are non-editable. They are derived from other elements used in your model.

Learn about editing values of different property types in:

- Editing textual properties(see page 922)
- Editing logical values(see page 923)
- Selecting values(see page 924)
- Cases of advanced editing(see page 929)

Related pages

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Editing textual properties

Textual type property values are usually simply typed in a property value cell. In some specific cases you can also use advanced editing features, i.e., long textual values editor and multiline textual values editor (both available in HTML).
Examples of textual type properties: *Name*, *Body*.

To edit a textual property value

1. Click an appropriate property value cell.
2. Type a value.

Learn about advanced cases of editing textual values in:

- Editing long textual values(see page 922)
- Editing multiline textual values(see page 922)

Related pages

Editing long textual values

In case of editing a value that consists of several words, the value cell may seem too short. In order to avoid this inconvenience you can edit the property value in a dialog specially designated for typing long values.

To edit a textual value consisting of several words

1. Click an appropriate property value cell.
2. Click the Edit button (see the highlighted button in the following figure).
3. In the opened dialog, type a value and click **OK**.

There are some textual properties, whose values can keep text formatting settings. You can use the HTML editor(see page 1055) in such cases. To use the HTML editor(see page 1055), select the **HTML** check box in the dialog opened for editing this special kind of textual property value.

Editing multiline textual values

Some properties can have more than one textual value, for example, the **Body** property of an opaque expression, the **Pre Condition** and **Post Condition** properties of a use case. You can use a special editor to edit multiline textual values.

Using the editor for multiline textual values, you can perform the following actions:

- Create a new value.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Remove a selected value.
- Edit each value in the HTML editor separately.
- Reorder values.
- Strip long values (available only in the Specification window).

> To strip long values, make sure that the Strip Multiline Text mode is turned on in the Specification window(see page 913).

The following table describes functions of buttons used in the editor.

Button	Shortcut Key	Description
		Select the HTML check box to edit a selected row in the HTML mode. The HTML editor toolbar will appear. For more information about the HTML toolbar, refer to "HTML editor toolbar" on page 428.
	ALT+ENTER	Click to add an empty row after the selected one to type a new value.
	ALT+BACKSPACE	Click to remove a selected row.
	ALT+SHIFT+UP ARROW	Click to moves up a selected row.
	ALT+SHIFT+DOWN ARROW	Click to moves down a selected row.

> Mac OS users should use MAC shortcut key as an alternative for the ALT key.

Editing logical values

Logical (boolean) type property value can be either *true* or *false*. Examples of logical type properties: *Is Abstract*, *Is Read Only*.

To edit a logical property value, do one of the following

- Select the check box to set the value to *true*.
- Clear the check box selection to set value to *false*.

Related pages

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Selecting values

Selectable value type property values are selected from lists. There can be two types of lists:

- **Non-editable** - for properties with value ranges restricted by UML (for example, Visibility, Message Sort, Event Type).
- **Editable** - for properties with value ranges not restricted by UML and can be defined by the user (for example, Type Modifier, Multiplicity).

Learn about selecting property values from lists in:

- Selecting single property value from non-editable list(see page 924)
- Selecting single property value from editable list(see page 925)
- Selecting multiple property values(see page 927)

Related pages

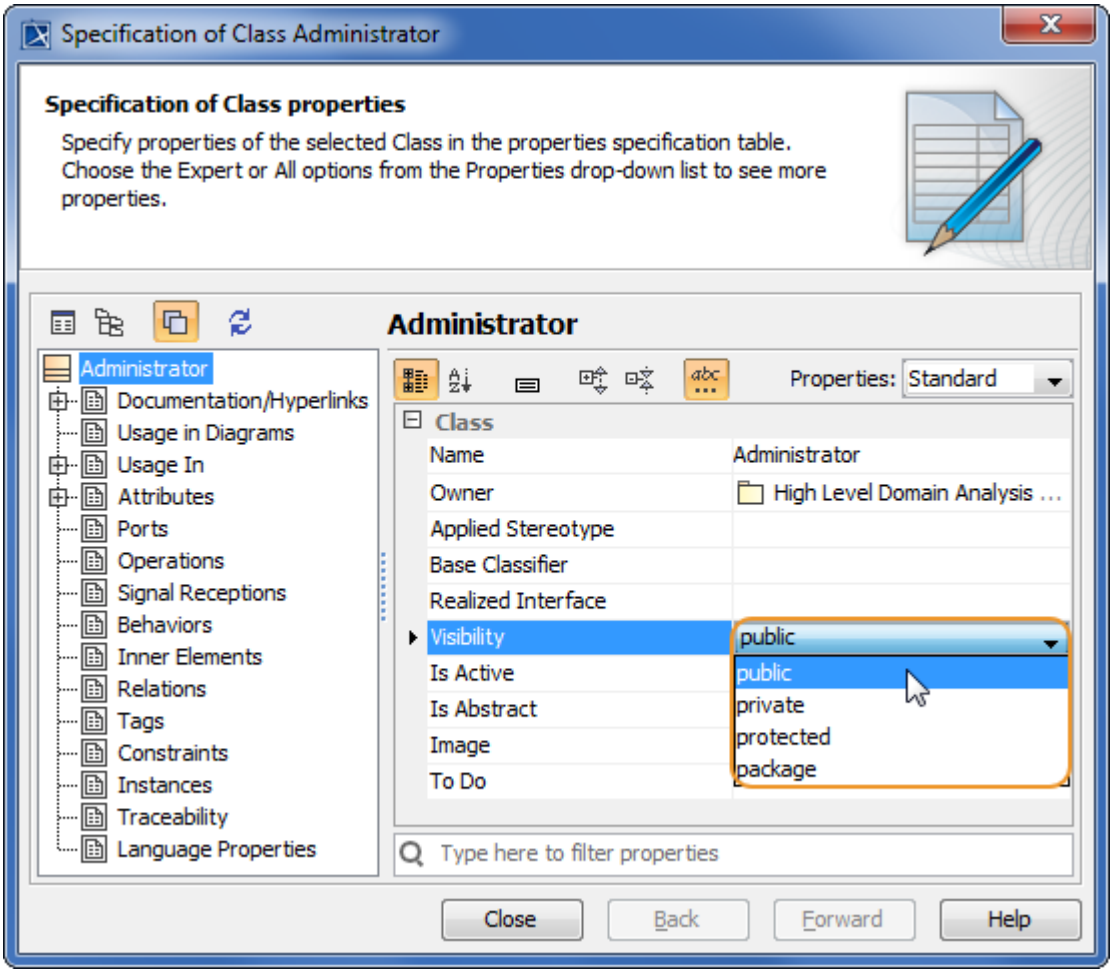
Selecting single property value from non-editable list

In this case you can choose one of the predefined values.

To select a property value from the non-editable predefined list

1. Click an appropriate property value cell.
2. In the drop-down list, select the value.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



To remove a property value selected from the non-editable predefined list

1. Click an appropriate property value cell.
2. In the drop-down list, select <*UNSPECIFIED*>.

> ⓘ The property value can be removed only for those properties, that can have an empty value.

Selecting single property value from editable list

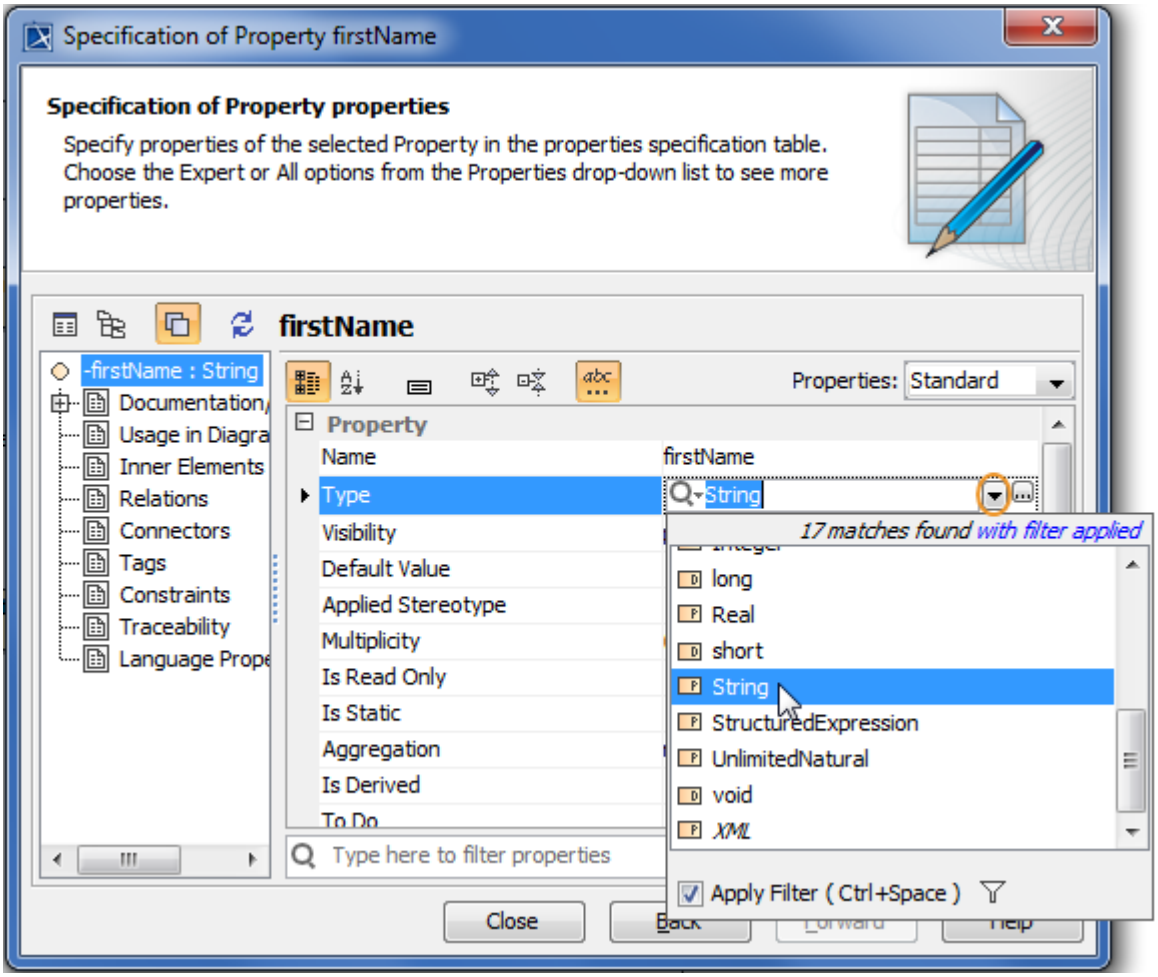
This is the case of assigning a value to a property in one of the following ways:

- Selecting a value either from a predefined value (auto-completion) list or from the whole model i n the **Select Element** dialog.
- Creating an element and selecting it as property value.

To select a property value from a predefined (auto-completion) list

1. Click an appropriate property value cell.
2. Click the arrow (see the highlighted button in the following figure).

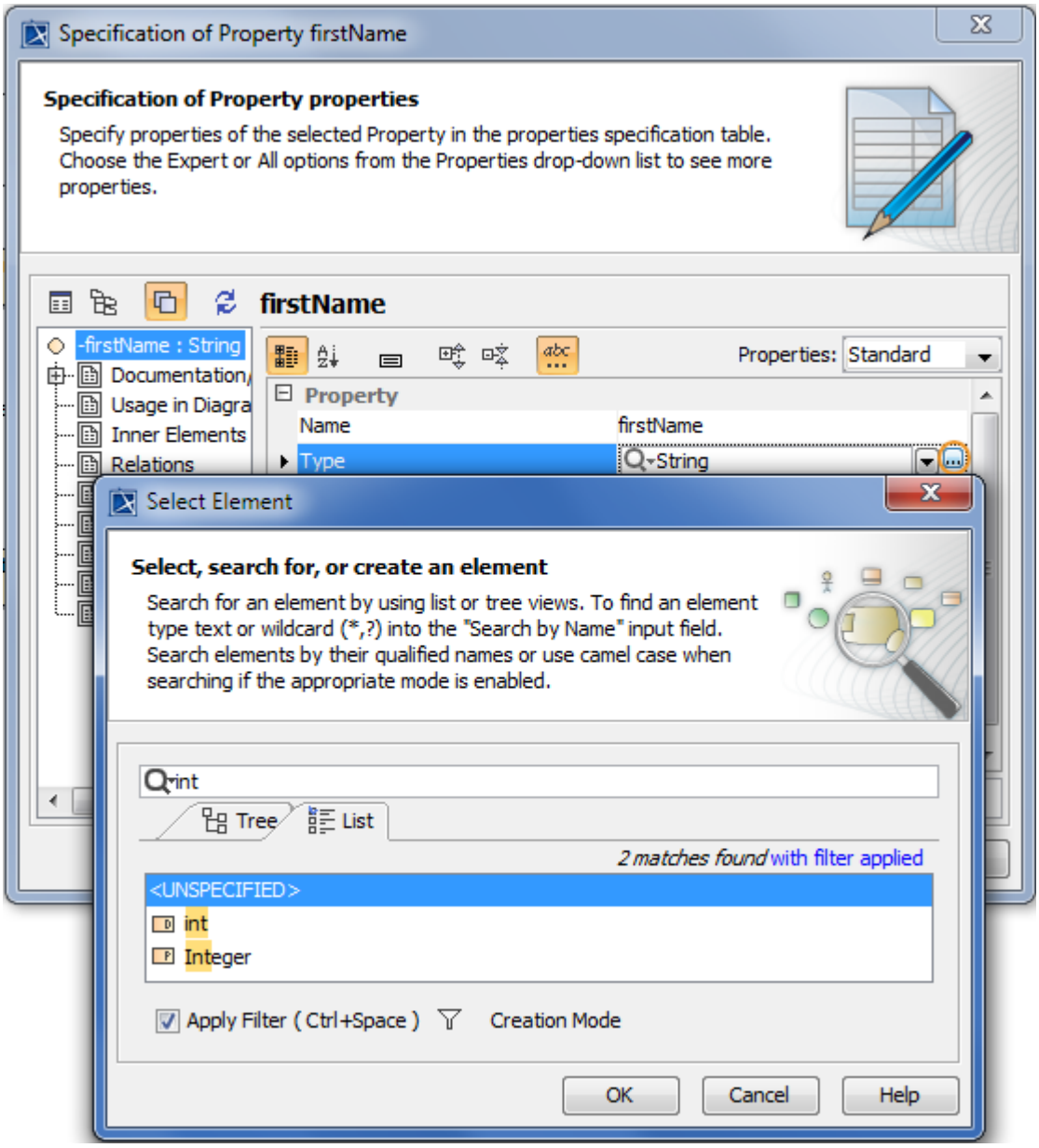
3. In the drop-down list, select a value.



To assign a property value in the Select Element dialog

---

1. Click an appropriate property value cell.
2. Click the Edit button (see the highlighted button in the following figure). The Select Element(see page 1031) dialog opens.
3. Select an existing or create(see page 1035) an element.
4. Click **OK** when you are done.



To remove a value selected for a single value type property

1. Click an appropriate property value cell.
2. In the drop-down list, select *<UNSPECIFIED>*.

Selecting multiple property values

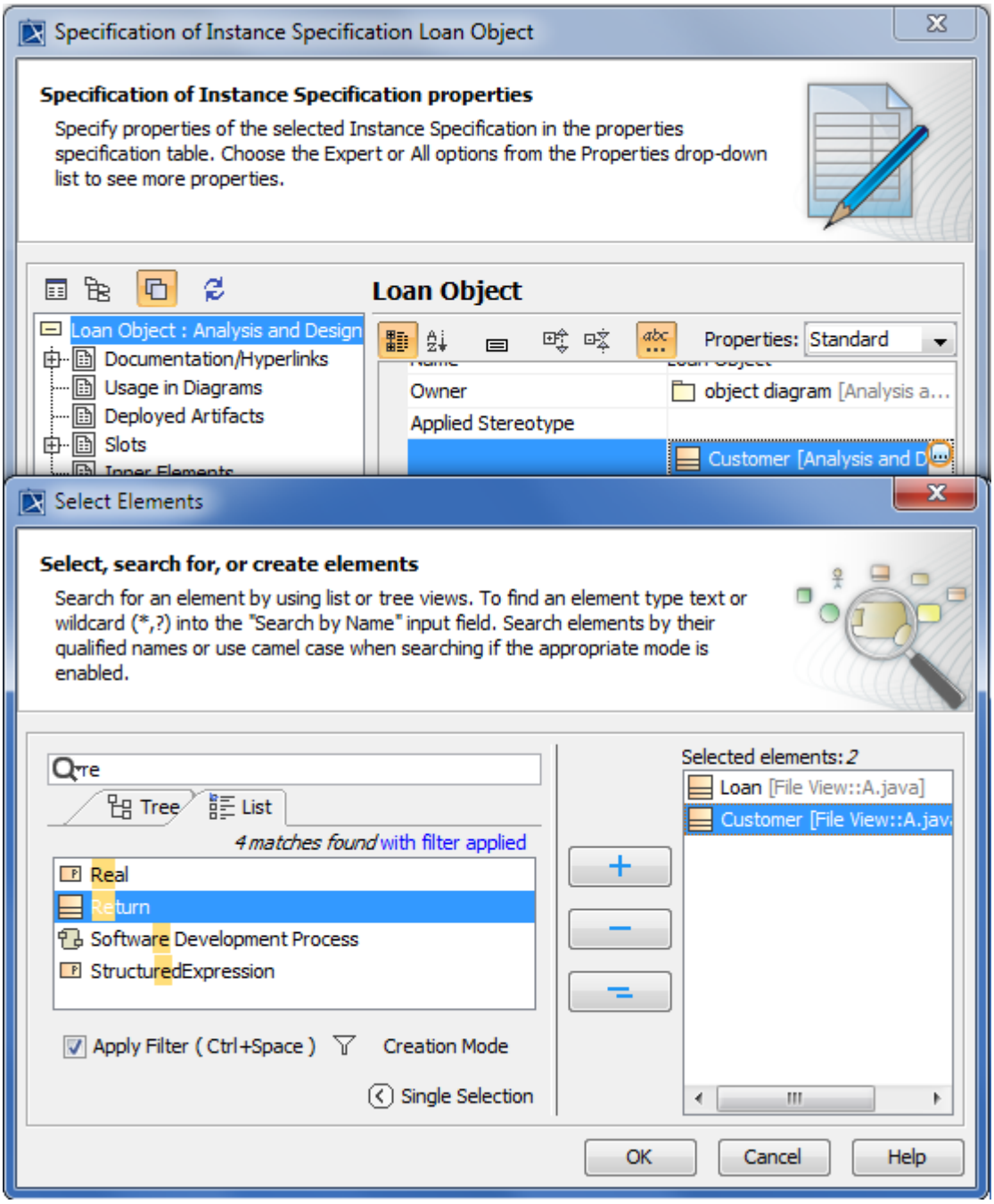
In this case you can assign more than one value to a property in the **Select Element** dialog. Examples of properties that may have multiple values: *Classifier*, *Method*.

To assign multiple property values

1. Click an appropriate property value cell.

2. Click the Edit button (see the highlighted button in the following figure). The Select Element(see page 1031) dialog opens.
3. Select existing and / or create elements.
4. Click **OK** when you are done.

> Make sure the Multiple Selection mode is turned on.



Cases of advanced editing

You may have noticed that editing some property values is a sophisticated process. These are the cases of editing such property values:

- Creating inner element as property value(see page 929)
- Assigning value specification as property value(see page 930)
- Editing property values in property groups(see page 931)

Related pages

Creating inner element as property value

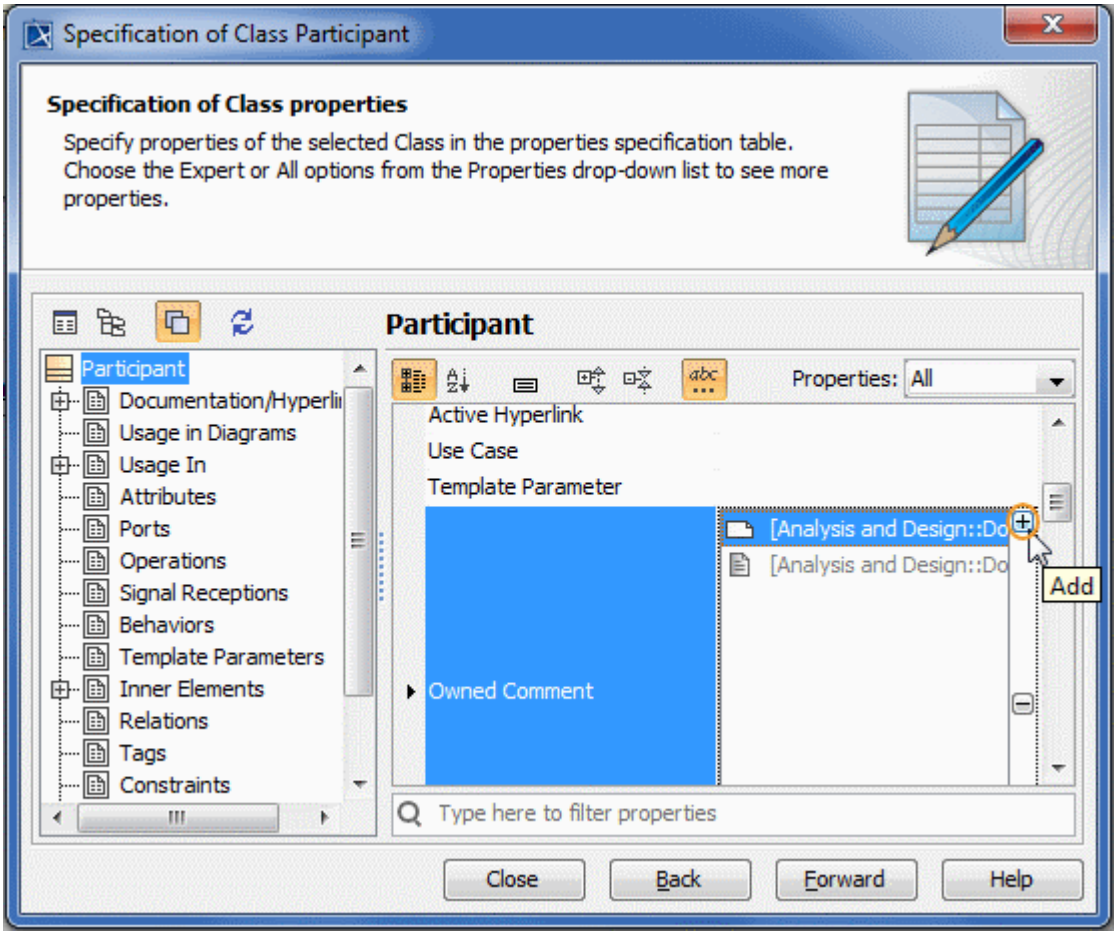
There are some properties whose values can be their inner elements. Specifying such property value requires to create a new element. The element, which has the property with the inner element assigned as its value, becomes the owner of this inner element.
Examples of properties whose values are their inner elements: *Owned Comment*, *Owned Attribute*.

To create an inner element as a property value

1. Click an appropriate property value cell.
2. Click the Add button (see the highlighted button in the following figure).
3. Select an element you want to create. The Specification window of the created element opens.
4. Specify properties of the new inner element.
5. When you are done, do one of the following:

- Close the Specification window.
- Click **Back** to return to the Specification window of the owning element.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



To remove an inner element from a property value list

1. Click an appropriate property value cell.
2. Select the value you want to remove.
3. Click the Remove button.

> ⓘ The element is removed from both the property value list and the model.

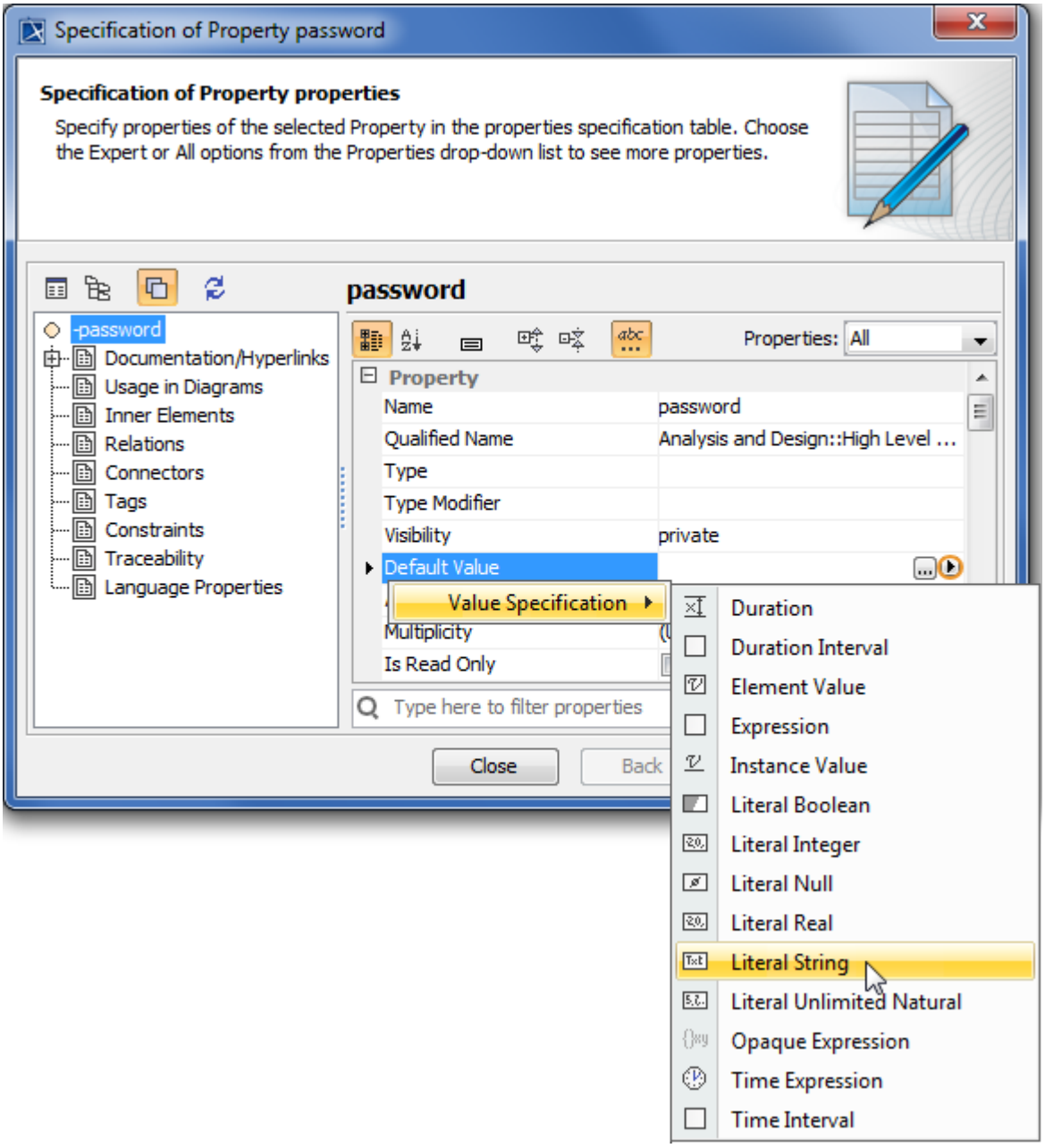
Assigning value specification as property value

There are some properties whose values can be value specifications(see page 899).

Examples of properties whose values are value specifications: *Default Value*, *Value*.

To assign a value specification

1. Click an appropriate property value cell.
2. Click the Show Shortcut Menu button (see the highlighted button in the following figure).
3. From the shortcut menu, select Value Specification(see page 899).
4. Select a value specification from the list.



To change an assigned value specification

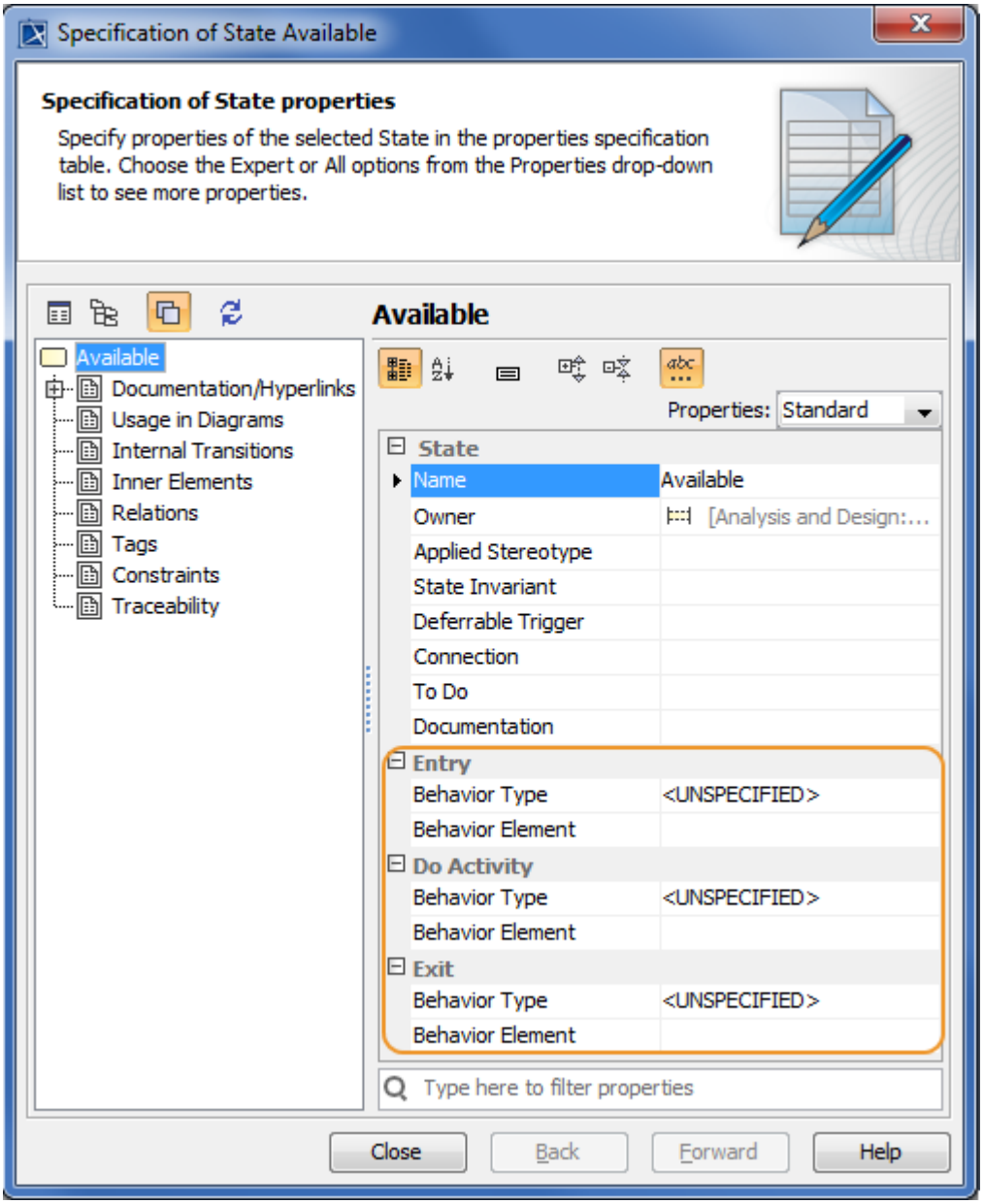
1. Click an appropriate property value cell.
2. Click the Show Shortcut Menu button (see the highlighted button in the preceding figure).
3. From the shortcut menu, select Value Specification(see page 899) > **Delete** <value specification>.
4. Assign a new value specification.

Editing property values in property groups

Some elements can be related to each other as the owner and the owned one, for example, a transition(see page 872) can be an owner of a trigger, and the trigger can be owned by the transition.

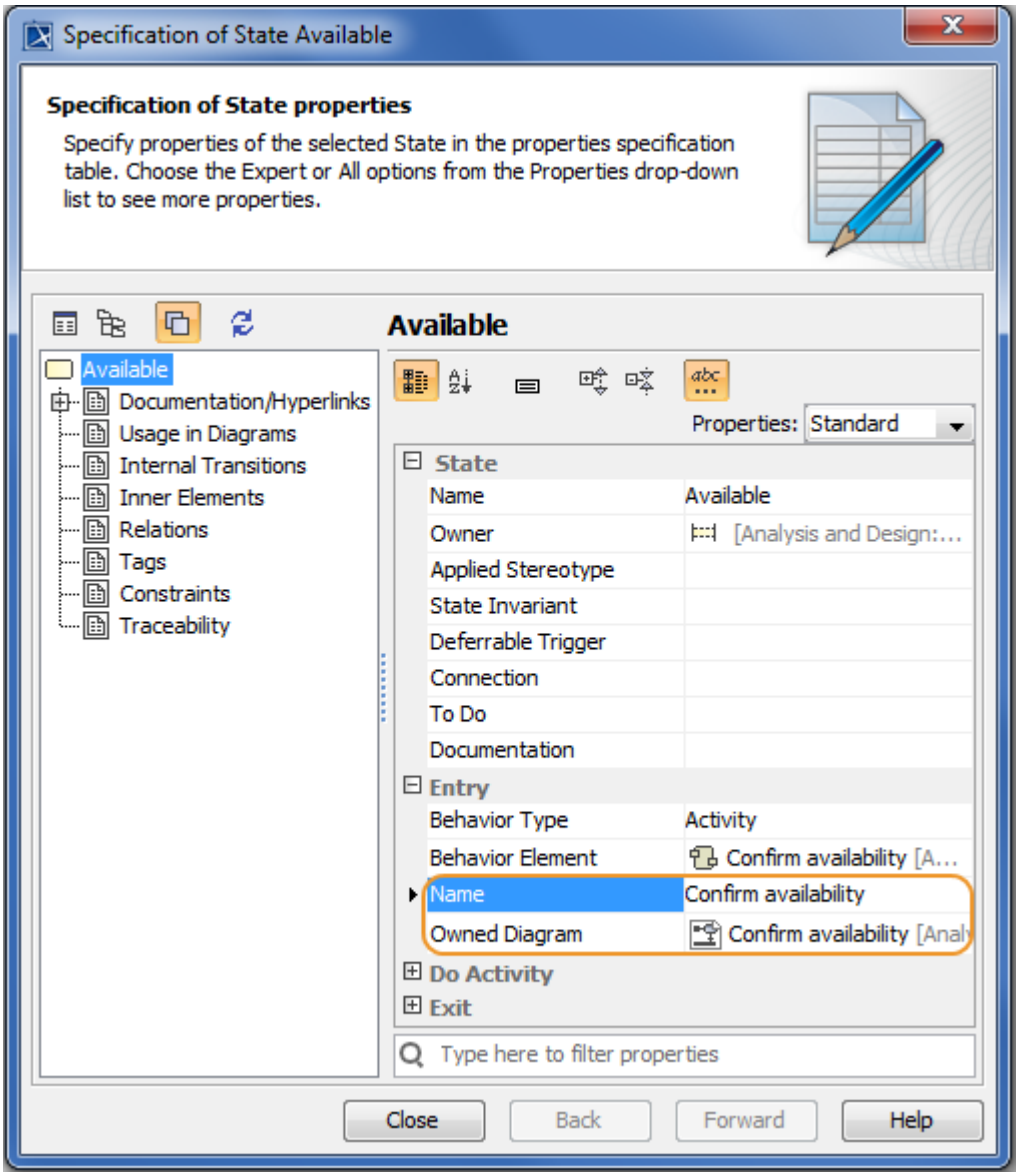
MagicDraw allows the handy editing of owned element property values via owner's properties. Owned element properties are available as appropriately named property groups, for example, the *Entry*, *Do Activity*, and *Exit* property groups in the state's Specification window (see the following figure).

> ⓘ When editing owned element properties, keep in mind that you create a new element in your model.



A property set in the property group may vary before and after owned element specification. For example, let's say, we have an activity assigned as a behavior type in the **Entry** property group. As a result the additional properties appear in the **Entry** property group (**Name** and **Owned Diagram**) after creating a new activity (see the following figure).

Keep in mind that the owner's specification contains only ownership-relevant properties of an owned element.
To view and edit all other properties of the owned element, open its Specification window.



To create an owned element as a property value

1. In the property group, wherein you want to create the owned element, select either a type or an element property specification cell.
2. From the list, select an element you need to create.
3. Specify additional properties corresponding to the created element.

To remove an owned element, do one of the following

- Either

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

1. Select the **<owned element type>** property specification cell.
2. In the drop-down list, select *<UNSPECIFIED>*.

- Or

1. Select the <owned element> property cell and click **...** button.
2. From the list, select **Delete <element>**.

Sorting elements

On this page

- Sorting elements manually(see page 934)
- Sorting elements alphabetically(see page 935)

Sorting elements manually

When creating a model, most of the elements are sorted alphabetically in the model browser. However, if you want to customize the structure of your model, you can sort the inner elements of any model element manually. For example, you can change the order of the Packages owned by the parent *Model* element.

> **Sorting elements in server projects**
> Model elements can only be reordered if they or elements with the same owner are not locked by another user.

To sort model elements manually

1. In the model browser, double click the model element that owns the elements you want to sort. Its Specification window opens.
2. In the property group list on the left side of the window, select the **Inner Elements** node.
3. Make sure the sorting view is selected in the Specification window toolbar, as displayed in the figure below.
4. On the right side of the window, select the element that you want to move up or down in the model browser.

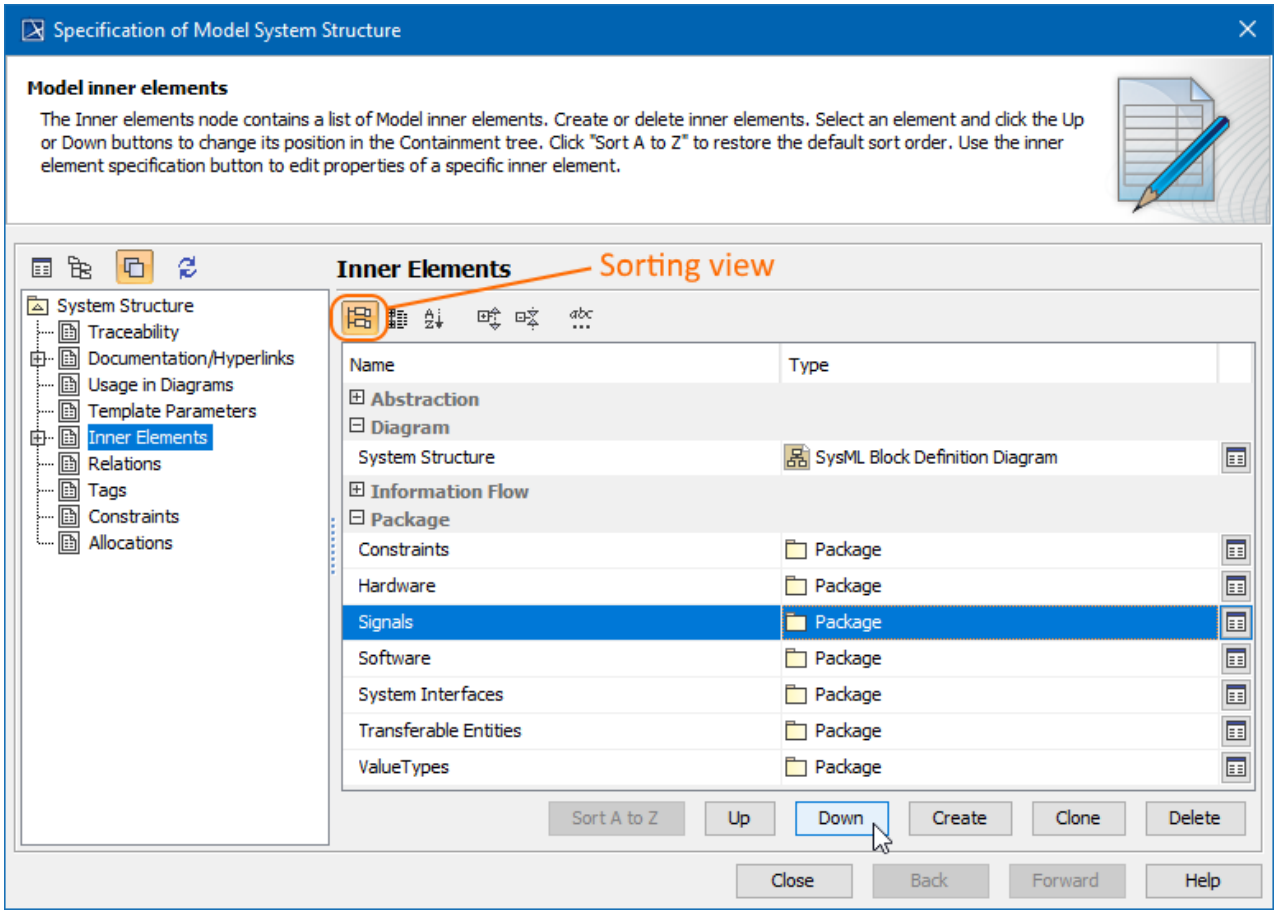
   > **Sorting multiple elements**
   > If you want to reorder a group of elements, select multiple elements by clicking the first or last element of the group. Then press Shift+▼ to select elements below it or Shift+▲ to select elements above it.

5. Do one of the following:
   - Click the **Up** button on the bottom of the window, to move the element up.
   - Click the **Down** button on the bottom of the window, to move the element down.
6. Repeat step 5 until the element is in the desired position.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

7. Close the Specification window.

Once you sort elements in the Specification window of the owning element, the elements are instantly reordered in the model browser as well. The newly created elements of the sorted element type are automatically placed at the bottom.



Sorting inner Packages of the *System Structure* Package.

Sorting elements alphabetically

If you have sorted the elements of your model manually, as described above, but want to restore alphabetical element sorting, you can do it in the Specification window of the owning element.

> **Sorting elements in server projects**
> Model elements can only be reordered if they or elements with the same owner are not locked by another user.

To sort model elements alphabetically

1. In the model browser, double click the model element owning the elements you want to sort. Its Specification window opens.
2. In the property group list on the left side of the window, select the **Inner Elements** node.
3. Make sure the sorting view is selected in the Specification window toolbar, as displayed in the figure above.
4. On the right side of the window, select any element in the group of elements that you want to sort.
5. Click the **Sort A to Z** button.

   > **Tip**
   > If the **Sort A to Z** button is disabled (grayed out), the elements of the selected element group are already sorted alphabetically.

6. Close the Specification window.

After completing the steps above, all the elements in the selected element group are sorted alphabetically.

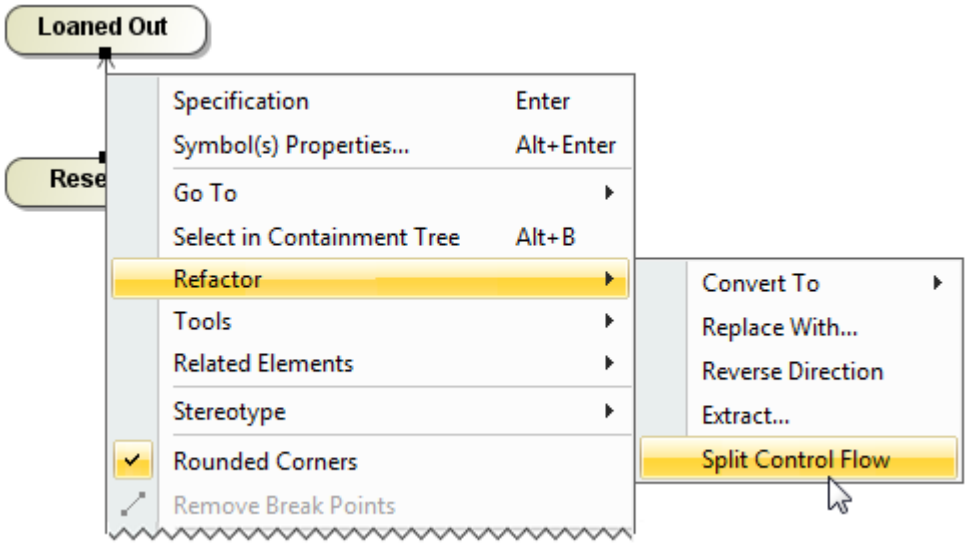
Refactoring

Refactoring allows you to modify model elements including all related data. Within refactoring, you can convert element types, replace elements, decompose diagrams, and perform other actions.

All the refactoring commands have been moved under the Refactor command in the shortcut menu of the particular element. Now the refactoring covers the following commands:

- Convert To.
- Replace With.
- Reverse Direction.
- Reverse Flow Direction.
- Extract.
- Split Control Flow for the selected Control Flow edge.
- Split Object Flow for the selected Object Flow edge.
- Join Connector for the selected split Control Flow or Object Flow edge.
- Swap.
- Redefinition.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



To perform refactoring

1. Right-click an element in the Model Browser or on the diagram pane.
2. From the shortcut menu, point to Refactor and select one of available commands for the selected element.

> ⓘ Some of the Refactor sub-commands are available only in element shortcut menu that is opened from the diagram pane, not available in the Model Browser.

Related pages

Redefinition

Any inherited part can now be quickly redefined directly in the composite structure diagrams. A new part redefining the original one is created and graphically replaced in the diagram with all ports and connectors remaining in their places.

To redefine an inherited part

1. Right-click a part and choose **Refactor** > **Redefine To**. The **Select Type** dialog opens.
2. Do one of the following:
   - Type a name, if you want to create a new type.
   - Select one of the existing types.

Converting Elements

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Original element

The original element is the element that will be converted. In other words, it is the conversion source.

Conversion target

The conversion target is the element type, to which the original element is going to be converted.

Converted element

The converted element is the element that type has been changed during the conversion.

Related elements

The related elements are elements having relations to the original element. Related elements can be the following:

- If the original element is a relationship, elements connected by this relationship are considered as related element to relationship.
- If the original element is not a relationship, relationships going from or to the original element are the related elements (such as an association, dependency, generalization, and others).
- Other related elements associated to the original element. They are properties of the original element, such as inner elements, types, and others.

Compatible properties

The compatible properties are original element specification that are compatible with the conversion target. The term “properties” includes related elements, inner elements, and all other elements listed in the element Specification window.

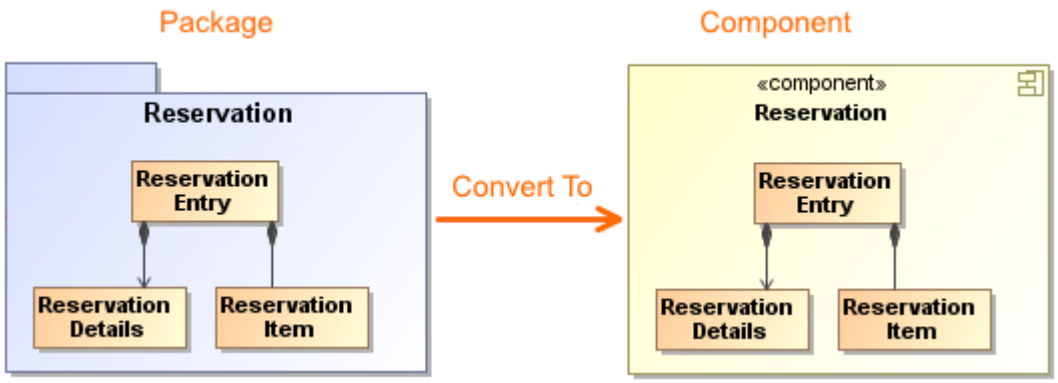
Compatible means that the converted element can have or own the particular property or properties of the original element.

Incompatible properties

Incompatible properties are original element specification that are incompatible with the conversion target. Incompatible means that the converted element can NOT have or can NOT own the particular property(-ies) of the original element.

Element conversion allows converting one element type to another. To be more specific, the element conversion functionality allows the UML model element conversion from one metaclass to another. For example, a class element can be converted to a use case, component, interface, or other. All compatible properties and all compatible related elements are moved from the source element to the converted element during the conversion.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



- Relationships conversion is available.
- Multiple elements can be selected on the diagram or in the Model Browser to convert them to other types.

To convert elements

1. From the selected element's shortcut menu, select **Refactor** > **Convert To**. The list with available conversion target types opens.
2. Do one of the following:
   - Select the element type from the commonly used element type list. The element is converted.
   - Select **More Elements** to see the whole list of available conversion targets. In the opened **More Elements** dialog, select the element type you need and click **OK**. The element is converted and selected element type is included as commonly used element type.

As of the MagicDraw version 17.0.2, the following commands have been moved under the **Refactor** command in the shortcut menu of the particular element:

- **Convert To** > **Output Pin** for the selected Object Node or Input Pin.
- **Convert To** > **Input Pin** for the selected Object Node or Output Pin.
- **Convert To** > **Object Node** for the selected Input Pin or Output Pin.
- **Convert To** > **Attribute(s)** for the selected Association that has defined roles, that is, roles have names.
- **Convert To** > **Association(s)** for the selected Attribute. **NOTE**: The Association(s) command is available only if the attribute(s) type is specified.

- If the **Refactor** command is unavailable, make sure you have permissions to edit the selected element or related element(s).
- Use **Undo** to restore the original element and its data.

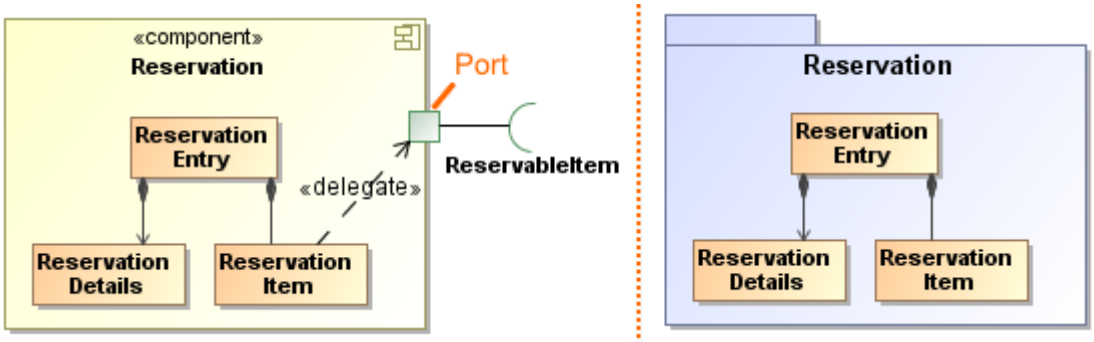
Related pages

Incompatible properties and related elements

After the conversion, a converted element owns the same compatible specification properties and is connected with the same related elements as the original element was connected. The incompatible properties of the original element are lost, that is, are deleted from the project.

Incompatible means that the converted element can NOT own particular properties or can NOT be associated with particular related elements of the original element.

For example, let's convert the component to package. See an example in the following figure.



As you see, the component has the assigned port. According to UML guidelines, a package cannot own ports. So, in this case, ports are incompatible with the packages. After the conversion, the package, that is, a converted element, has no assigned port. The port is deleted from the project.

To prevent from losing model data, you always get a message with a warning that incompatible properties will be lost after the element conversion.

Changes after conversion

Converting a property to a port
If the property visibility is *private*, *protected,* or *package* , then after the conversion visibility of the created port will be set to *public*.

On this page

- Changes after conversion(see page 940)
  - Converting a property to a port(see page 940)

Replacing elements

This functionality is available in Standard, Professional, Architect, and Enterprise editions.

Original element

The original element is the element that is going to be replaced. In other words, the replacement source.

Replacement target

The replacement target is the element with which the original element is going to be replaced.

Replaced element

The replaced element is the element to which the original element has been changed during the replacement.

Element properties

The element properties are so called the element specification properties (specified in the element Specification window) and the element symbol properties (defined in the element Symbol Properties dialog).

Inner elements

Inner elements are the elements stored inside the element (that is, owned by that element), for example, the class as inner package element.

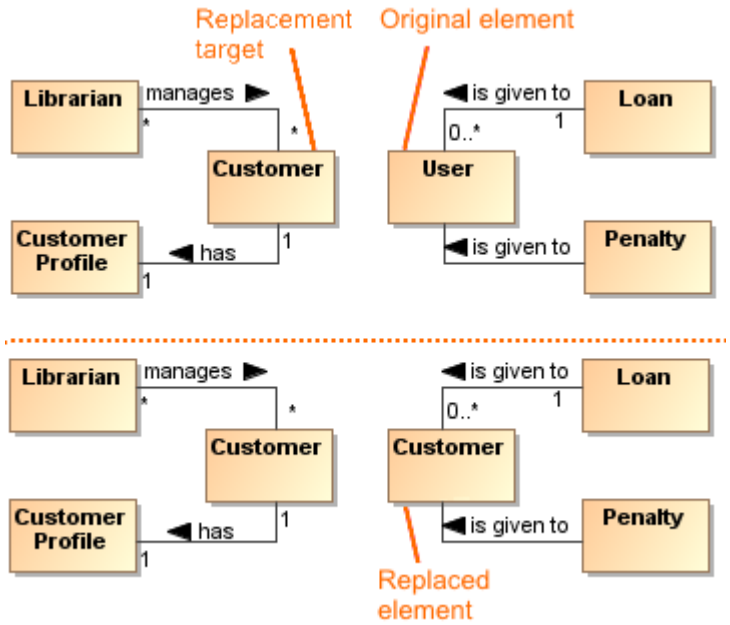
You can replace one model element with another of the same type (metaclass) element. The model element replacement is useful when during the modeling process you notice that one model element needs to be replaced with another. All relations and references of the original element are transferred to the newly replaced element.

After the replacement, an original element will be replaced with a replacement target:

- The replacement target will be displayed in all diagrams instead of the original element.
- All references to the original element will be replaced by references to the replacement target.
- The replacement target after the replacement will have all the relationships of the replacement target and the original element.
- The original element will be deleted from the project.
- The original element properties and inner elements will be deleted from the project.

See the replacement example in the following figure.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The preceding figure illustrates the replacement of the *User* class instead of the *Customer* class. After the replacement the *User* class has been removed and the *Customer* class has been created instead of it.

To replace one element with another

1. Select a symbol on the diagram pane or an element in the Model Browser and open its shortcut menu.
2. Click **Refactor** > **Replace With**. The warning message appears:
   - Click **Yes** to continue the element replacement. The properties and inner elements of the original element will be lost.
   - Click **No** to cancel the element replacement. The selected element will not be replaced.
3. If you decided to continue the replacement, the element selection dialog opens.
4. In the element selection dialog, select the element with which you want to replace the original element.
5. Click **OK**. The original element is replaced with the target element.

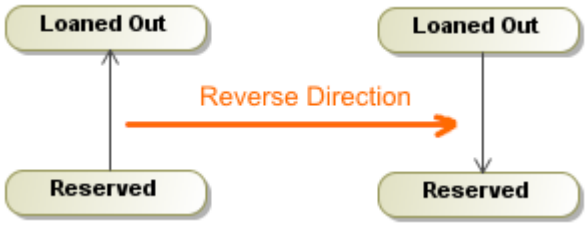
Related pages

- Refactoring(see page 936)
- Working with model elements(see page 909)

Reversing relationships direction

This functionality is available in Standard, Professional, Architect, and Enterprise editions.

The direction of the particular relationship can be reversed. You can reverse the direction of the transition, control flow, object flow, and other relationships.



The relation can be reversed if:

- The selected relationship and elements connected by this relationship are editable.
- The source and target elements are compatible, that is, they can be counterchanged.

To reverse a relationship direction

1. Select the relationship in the Model Browser or the relationship's path on the diagram pane.
2. From the element's shortcut menu, select **Refactor** > **Reverse Direction**. The direction of the relationship changes.

The direction of the information flow(see page 732) will not be changed on the direction reverse action. Additionally, in some cases when the information flow(see page 732) is not valid anymore, the information flow(see page 732) will be removed from the project after the direction reverse action.

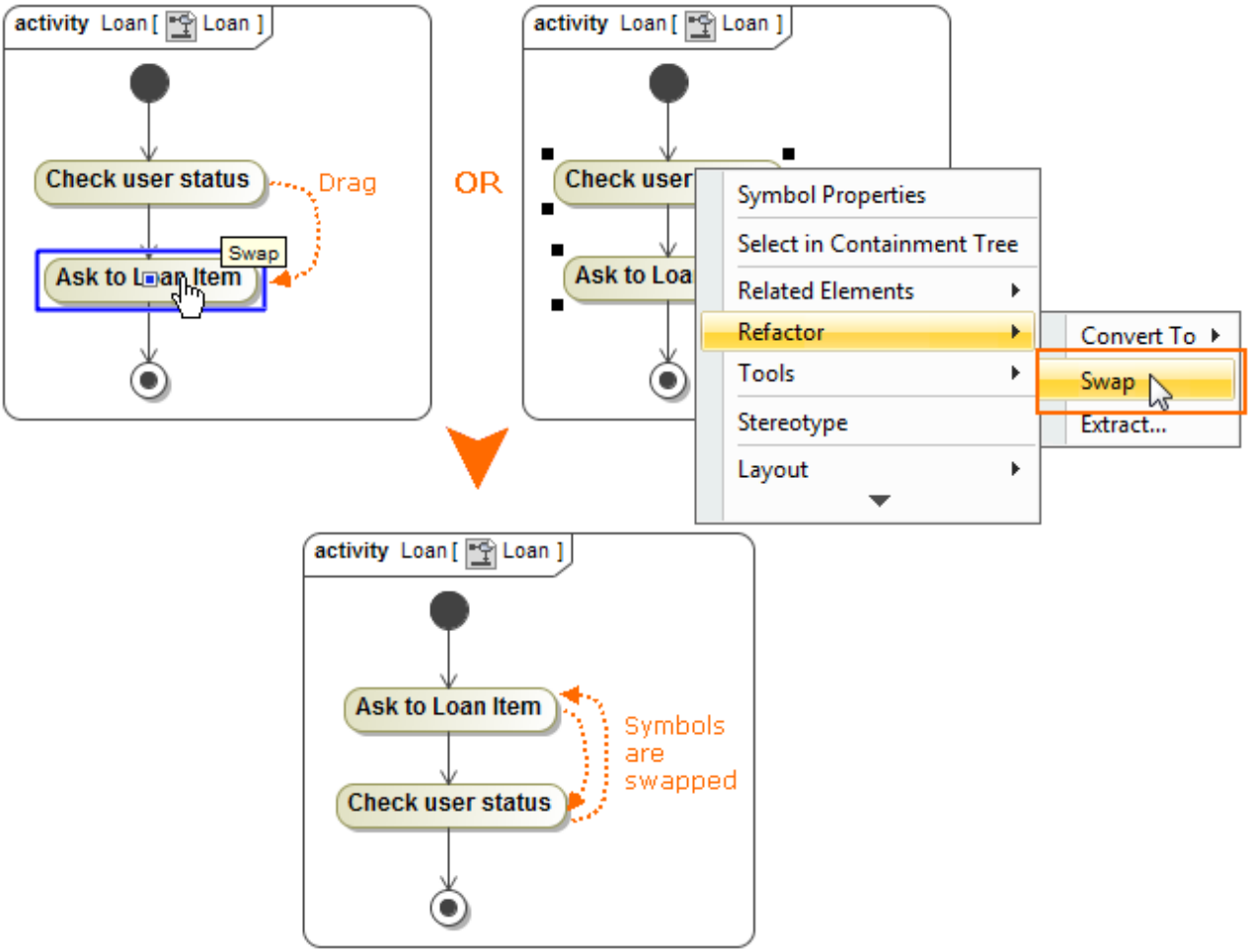
Related pages

- Refactoring(see page 936)
- Working with model elements(see page 909)

Swapping

Swapping allows counter-change symbols with each other on the diagram.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



You can swap the symbols in the following diagrams:

- In the Activity diagram, you can swap actions, merge or decision nodes, join or fork nodes with each other, if the symbols are connected through control flow.
- In the State diagram, you can swap states or pseudo states (initial state, deep history, join, fork, junction, choice, entry point, etc.) with each other, if symbols are connected through transition.

To swap elements, do one of the following

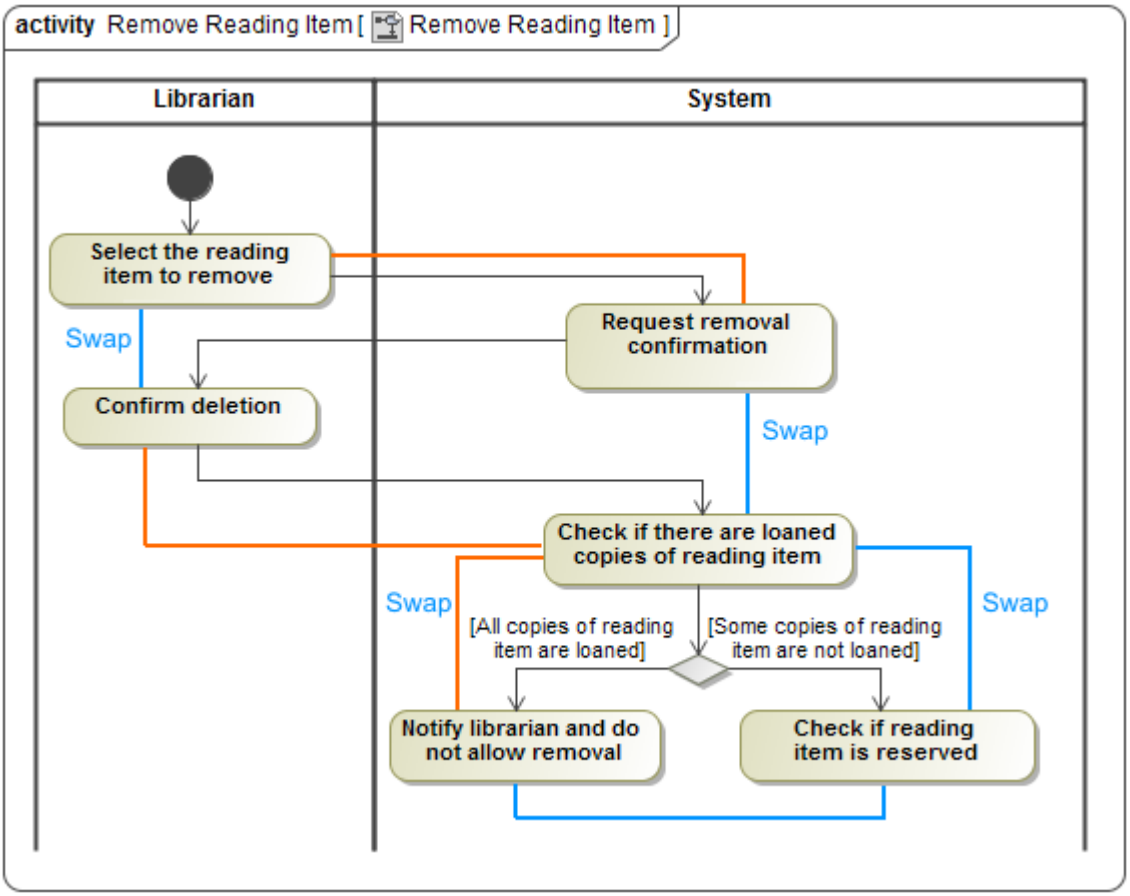
- Drag one shape onto another.

  > ⚠ Swap does not work when dragged shape can be added into the target shape (or in other words, can be added as a child).

- On the diagram, select the two shapes that you want to swap, right-click, and from the shortcut menu, select **Refactor** > **Swap**.

> ⚠
> - Swapping does not work when symbols have different parents.
> - If you cannot swap within a teamwork project, make sure you have the right to edit the model of this project and then try to lock for edit this diagram.

In the following example, you can see the cases when you can swap the elements and the cases when you cannot swap the elements.



Rule	Example
You **can** swap the elements that are in the same branch	You **can** swap the following elements: • “Select the reading item to remove” with “Confirm deletion” • “Request removal confirmation” with “Check if there are loaned copies of reading item” • “Check if there are loaned copies of reading item” with “Notify librarian and do not allow removal” • “Check if there are loaned copies of reading item” with “Check if reading item is reserved”

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

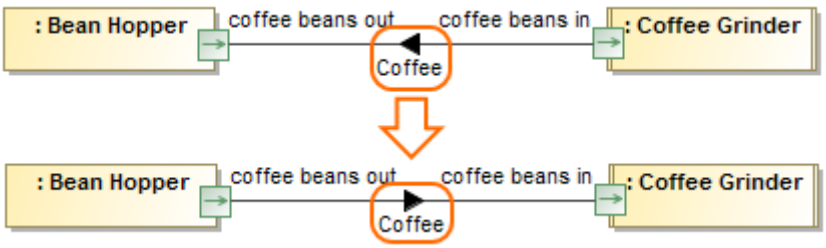
Rule	Example
You **cannot** swap the elements that have different parents	You **cannot** swap the following elements: • "Select the reading item to remove" with Request removal confirmation, because one element is in the "Librarian" partition and the other is in the "System" partition. • "Confirm deletion" with "Check if there are loaned copies of reading item" because one element is in the "Librarian" partition and the other is in the "System" partition.
You **cannot** swap the elements that are in different branches	You **cannot** swap the following elements: • "Notify librarian and do not allow removal" with "Check if reading item is reserved".

Related pages

- Refactoring(see page 936)
- Working with model elements(see page 909)

Reversing Information Flow direction

You can reverse the direction of the Information Flow(see page 732) on Connectors(see page 686), as shown in the following example. After reversing the direction, the flow directions are updated in all the related diagrams.



The direction of *Coffee* Information Flow is reversed. The illustration displays concepts from SysML Plugin.

To reverse Information Flow direction

1. Select the Connector that has the Information Flow set in the Containment tree or on the diagram pane.
2. From its shortcut menu, select **Refactor** > **Reverse Flow Direction**.
   The direction of the Information Flow is changed and updated in all the related diagrams.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related pages

- Refactoring(see page 936)
- Working with model elements(see page 909)
- Connector(see page 686)
- Information Flow(see page 732)

Extracting

Extracting feature is available in Architect and Enterprise editions.

The extraction functionality allows for moving a selected part of a diagram to a newly created diagram. A referencing element will be created in a place of the extracted element(s) in the source diagram.

In order to make the diagrams more readable and usable, you can extract a specific part of the diagram to another diagram. By using the extraction, you can reuse that part of the diagram to simplify a complicated system.

Definitions

Source diagram

A source diagram is the diagram from which the elements are extracted.

Target diagram

A target diagram is the newly created diagram resulting from the extraction. The selected elements are moved from the source diagram to the target diagram.

Extracting

Extracting means moving selected elements from one location to another. In other words, the selected elements are deleted from the source diagram and moved to the newly created target diagram.

Referencing element

A referencing element is created as the result of the extraction. The referencing element is created in the source diagram and represents the elements moved to the target diagram during the extraction.

A referencing element contains the reference to the target diagram. Therefore, double-clicking the referencing element on the diagram pane or in the Model Browser will open the target diagram.

Intersected relationship

An intersected relationship is the relationship selected in the source diagram. The intersection appears when the selection intersects the relationship - one end element of the relationship is selected, but not both.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

You can use the extraction functionality in the following diagrams:

- Extracting in Sequence Diagram
- Extracting in Activity Diagram
- Extracting in State Machine diagram
- Extracting in Composite Structure diagram

To extract a selected part

1. On the diagram, select a part that you want to extract.
2. From the selected part's shortcut menu, select **Refactor** > **Extract**. An extract wizard specific to the particular type of diagram opens.

Related pages

Related pages

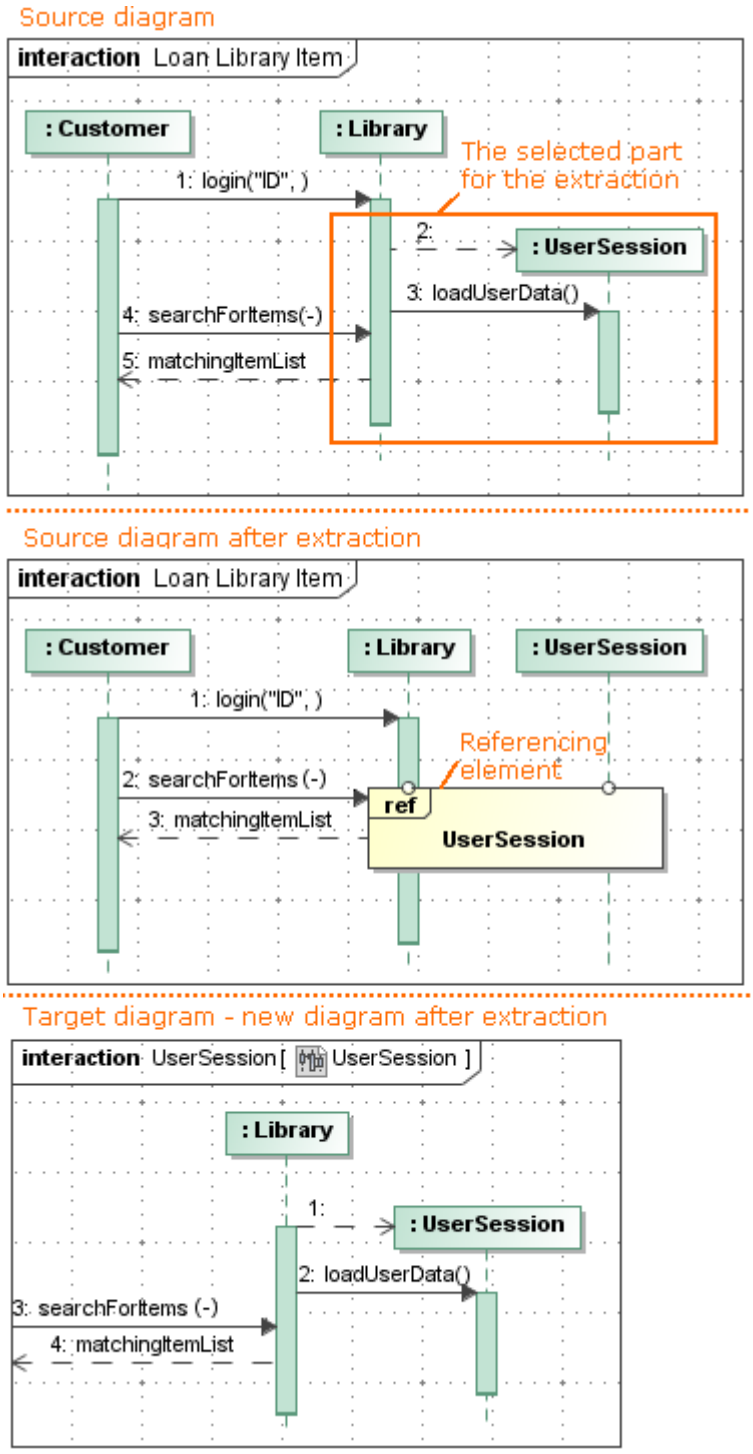
- Working with model elements(see page 909)

Extracting in Sequence diagram

The extraction functionality allows extracting a selected part of a sequence diagram to a newly created diagram. After extraction, an interaction use is created in the source diagram in place of the extracted part(s).

That is, the interaction use is a referencing element that references the newly created sequence diagram.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



As you can see in the example, during the extraction, the selected part of the sequence diagram is moved to the newly created *UserSession* sequence diagram. The interaction use element that references the *UserSession* interaction is created in the source diagram.

Related pages

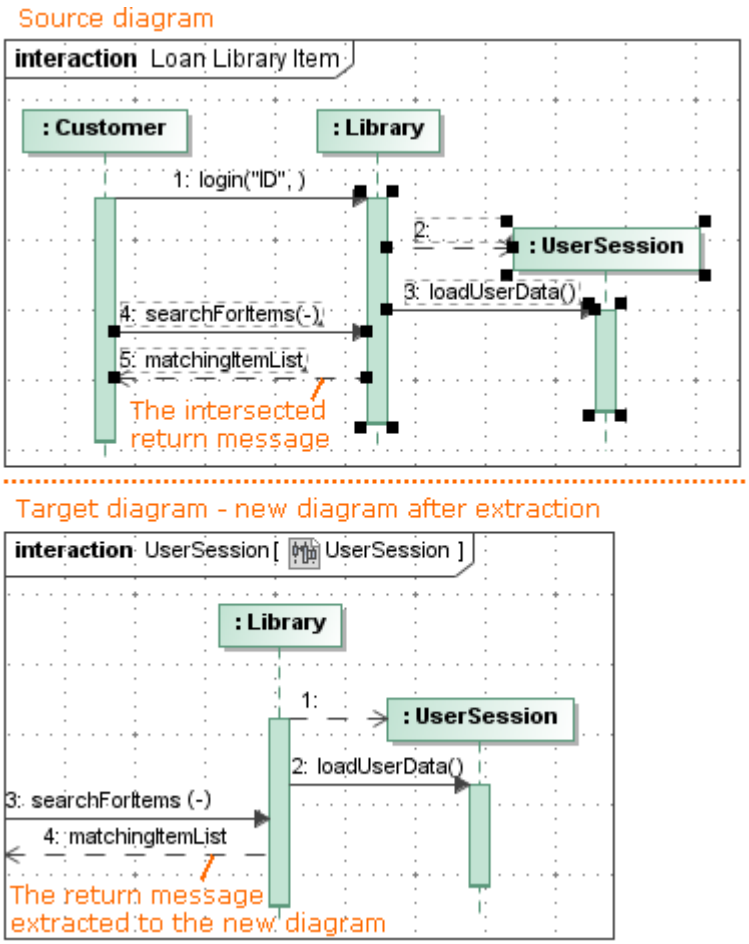
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related pages

- Extracting(see page 947)
- Working with model elements(see page 909)

Rules of extracting in Sequence diagram

Outgoing intersected messages are not selected for the extraction. The exception to this rule is the reply message. You can see an example in the following figure.



Related pages

- Extracting(see page 947)
- Extracting in Sequence diagram(see page 948)
- Extract Sequence Wizard(see page 951)
- Working with model elements(see page 909)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Extract Sequence Wizard

The **Extract Sequence Wizard** is used for extracting a selected part of the sequence diagram to a separate diagram. Using this wizard, you can:

- Create and specify the new element wherein the extracted part will be moved.
- Select the intersected incoming messages for which the formal gates will be created in the target diagram.

To start the **Extract Sequence Wizard**

---

1. On a sequence diagram, select a part that you want to extract.
2. From the selected part's shortcut menu, select **Refactor** > **Extract**. The **Extract Sequence Wizard** opens.

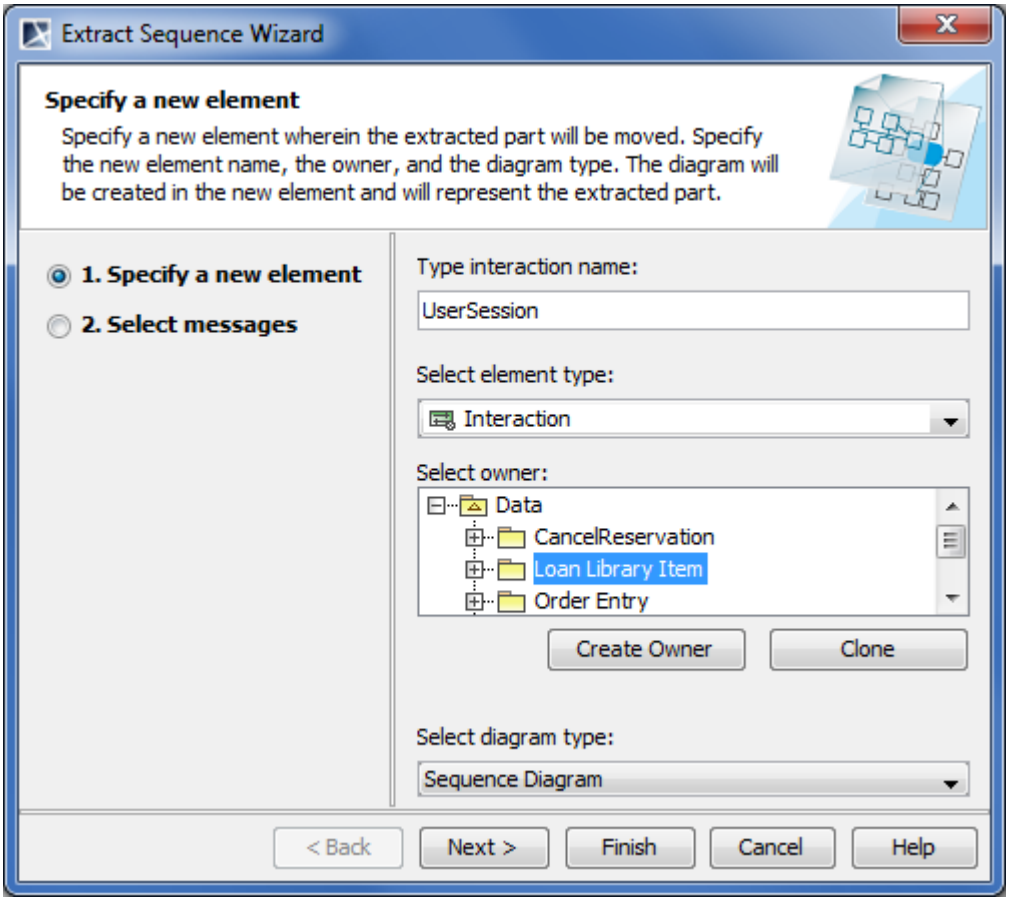
The **Extract Sequence Wizard** consists of two steps:

1. Specification of a new element
2. Selection of messages

Step #1. Specification of a new element

In this step, you can specify a diagram into which the extracted part will be moved and an owner wherein this new diagram will be created. You can also define the type of the newly created element that will own the diagram. The interaction element and the sequence diagram type are selected by default. The name of the sequence diagram will be the same as the specified name of the interaction element.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Element name	Description
Type interaction name	Type the name of the new interaction, wherein the extracted part will be stored and the new sequence diagram will be created. The newly created sequence diagram name will be the same as the interaction name.
Select element type	Open the list and select the element type in which the extracted part and new diagram will be created. The default element type is interaction. By using the customization plugin, you can add custom elements to the element type list.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Element name	Description
Select owner	In the model tree, select a package, profile, or model as an owner of the newly created element. You can also: • Create a new owner. Click the **Create Owner** button, select an element type, and specify its properties. • Clone an existing package, profile, or model. To clone, in the model tree, select an owner, click the **Clone** button, and specify properties of the cloned element.
Select diagram type	Open the list and select the diagram type for the extracted part. The default diagram type is a sequence diagram. You can also select to create a new time diagram. By using the customized plugins, you can add custom diagrams to the diagram type list.

After you have specified the new element name, type, owner, and the type of the new diagram, do one of the following:

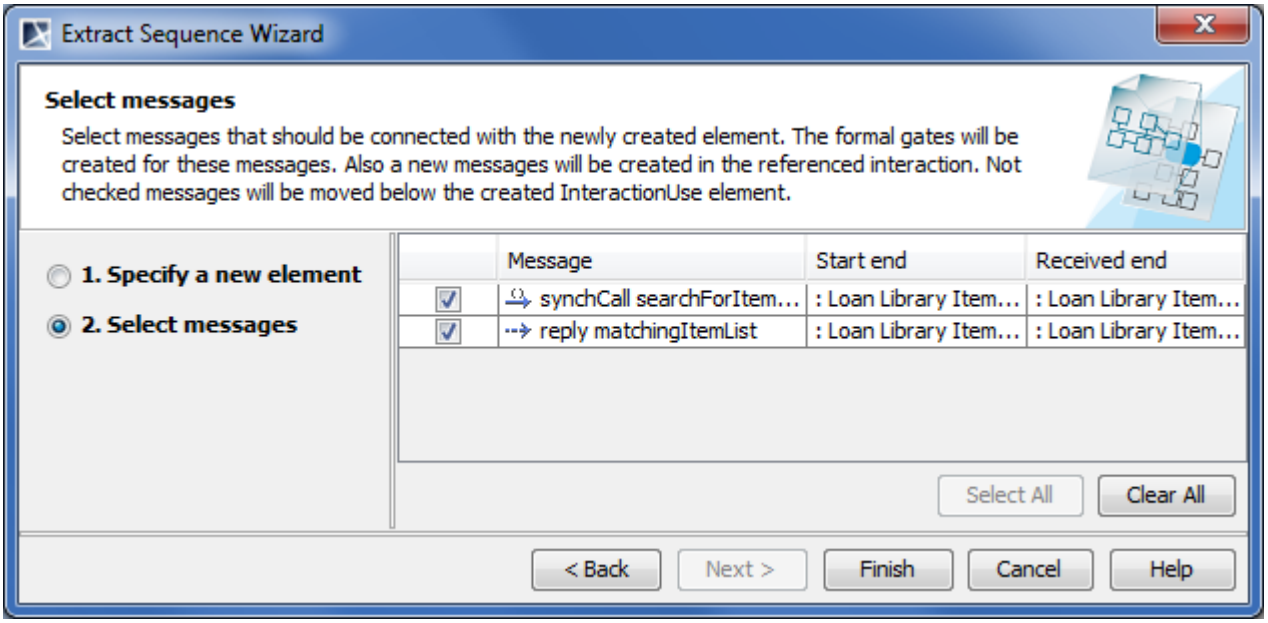
- Click **Next**, if you want to select intersected messages that will be created in the new target diagram.
- Click **Finish**. The selected part of a sequence diagram is extracted to a new diagram. The second wizard step is skipped, and the messages that by default are selected in this step, will be created.

Step #2. Selection of messages

The incoming intersected messages selected in the source diagram are listed in this step. Select the messages that will be extracted to the new target diagram. The formal gates will be created for the selected messages.

- The **Select messages** list is empty unless there are intersected incoming messages that are selected in the source sequence diagram.
- If there are intersected outgoing replay messages when they are listed in the **Extract Sequence Wizard**, the second step. For more information, see Rules of extracting in Sequence diagram(see page 950).
- For more information about intersected messages, see concepts in Extracting(see page 947).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Click **Back** to return to the first step.After you have selected the messages, do one of the following:

- Click **Finish**. The selected part of the sequence diagram is extracted to a new diagram.

On this page

- Step #1. Specification of a new element(see page 951)
- Step #2. Selection of messages(see page 953)

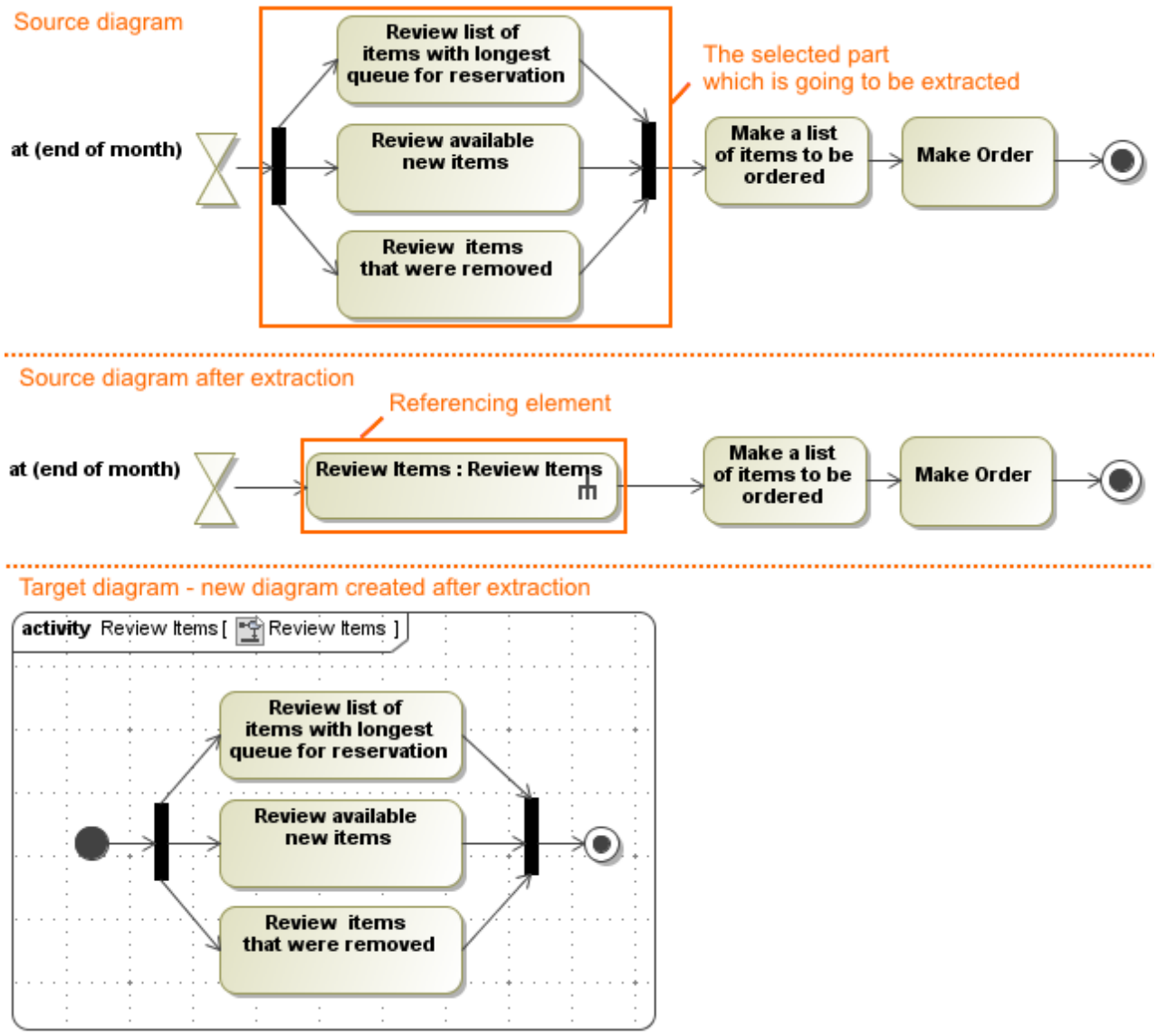
Related pages

- Extracting(see page 947)
- Extracting in Sequence diagram(see page 948)
- Rules of extracting in Sequence diagram(see page 950)
- Working with model elements
  (see page 909)

Extracting in Activity diagram

The extraction functionality allows for extracting a selected part of an activity diagram to a newly created diagram. After extraction, in the source diagram, a call behavior action is created instead of the extracted part. That is, the call behavior action is a referencing element that references the newly created activity diagram.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



As you can see in the example, during extraction, the selected part of the activity diagram is moved to the newly created *Review Items* activity diagram. In the source diagram, a *Review Items* call behavior action is created in place of the selected elements.

Related pages

Related pages

- Extracting(see page 947)
- Working with model elements(see page 909)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Extraction rules in Activity diagram

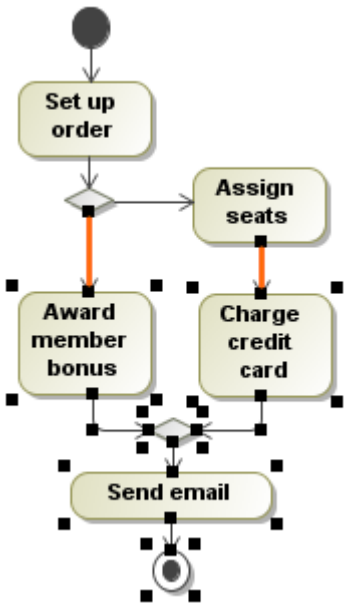
Extracting rules are the rules that should be followed in order to proceed with the correct extraction. For example, there can be invalid selections in a source diagram. Or there are unsupported elements that cannot be extracted. Without knowing the rules you will not be able to proceed with extraction (you will have to change your selected elements and/or re-select elements according to the rules) or you may lose some data. Here are the extraction rules:

- Rule 1. Only one incoming and/or outgoing control flow is allowed for a valid diagram extraction(see page 956)
- Rule 2. The selected elements should exist in the same owner, action, or activity partition(see page 956)
- Rule 3. The intersected exception handler is not extracted to a new diagram(see page 957)

Rule 1. Only one incoming and/or outgoing control flow is allowed for a valid diagram extraction

If the selected part of an activity diagram has two or more intersected incoming or outgoing control flows, the extraction is not valid.

Let's analyze an example of the extraction when two incoming control flows are selected in the activity diagram as depicted in the following figure.



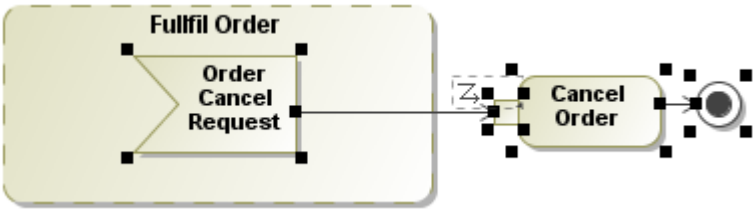
After you start the extraction, you will be warned that only one incoming control flow and only one outgoing control flow is allowed. Click the **OK** button to close the message and then change the selected elements in the activity diagram.

Rule 2. The selected elements should exist in the same owner, action, or activity partition

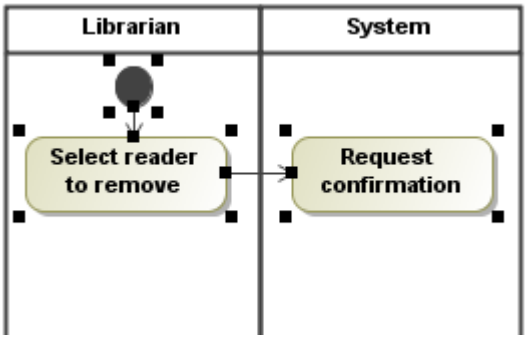
The extraction will be invalid if:

- The selected elements do not belong to the same owner. For example, if one of the selected actions is located in a structured activity node and the other does not belong to the

same structured activity node.



- Selected elements are in different activity partitions.

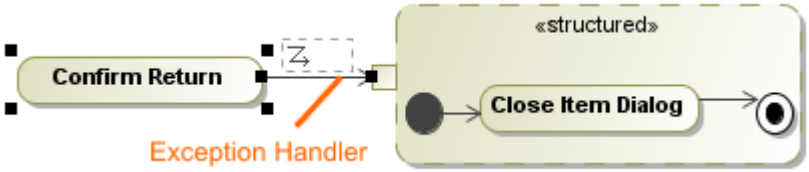


If an attempt is made to extract using an invalid selection, a warning will be displayed. Make a valid selection in order to proceed with the extraction.

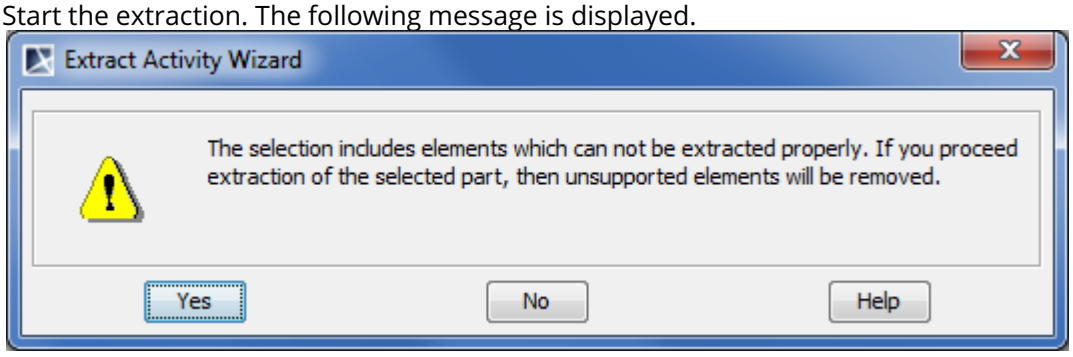
Rule 3. The intersected exception handler is not extracted to a new diagram

Let's analyze an example of the extraction when the intersected exception handler is selected in the source activity diagram.

1. In the source activity diagram, select an exception handler along with an action.

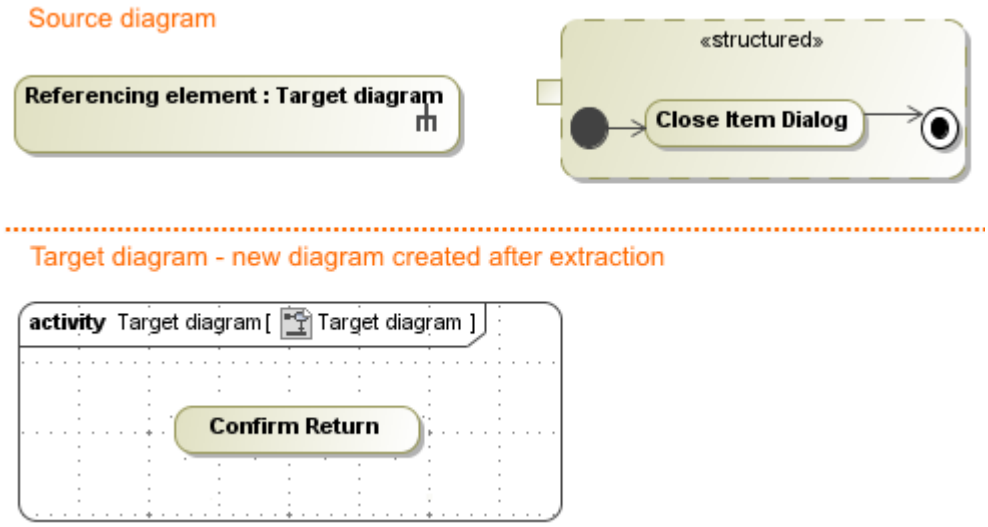


2. Start the extraction. The following message is displayed.



3. Click **Yes** to continue with the extraction. See the example of the extraction result in the following figures. After the extraction the referencing element is created in the source activity diagram in place of the selected exception handler and the *Confirm Return* action. The path of the exception handler is not represented in the new target diagram because in this case the exception handler

does not have the two symbols between which it could be added.



- Click **No** to close the message. The extraction is canceled and you can select elements for extraction again.
- Use **Undo** to restore the original element and its data.

Related pages

- Extracting(see page 947)
- Extracting in Activity diagram(see page 954)
- Extract Activity Wizard(see page 958)
- Working with model elements
  (see page 909)

Extract Activity Wizard

The **Extract Activity Wizard** is used for extracting a selected part of an activity diagram to a separate diagram.

Using this wizard, you can:

- Create and specify the target diagram that will be created as the result of the extraction. The selected part will be extracted to the newly created diagram.
- Select the parameters that will be created in a new diagram. The parameters are listed for each intersected object flow from the source diagram.
- Specify a referencing element that is created in the source activity diagram and represents the elements moved to the target diagram during the extraction.

To start the **Extract Activity Wizard**

1. In the activity diagram, select a part that you want to extract.
2. From the selected part's shortcut menu, choose **Refactor** > **Extract**. The **Extract Activity Wizard** opens.

This section describes components of the **Extract Activity Wizard**, such as text boxes and buttons.

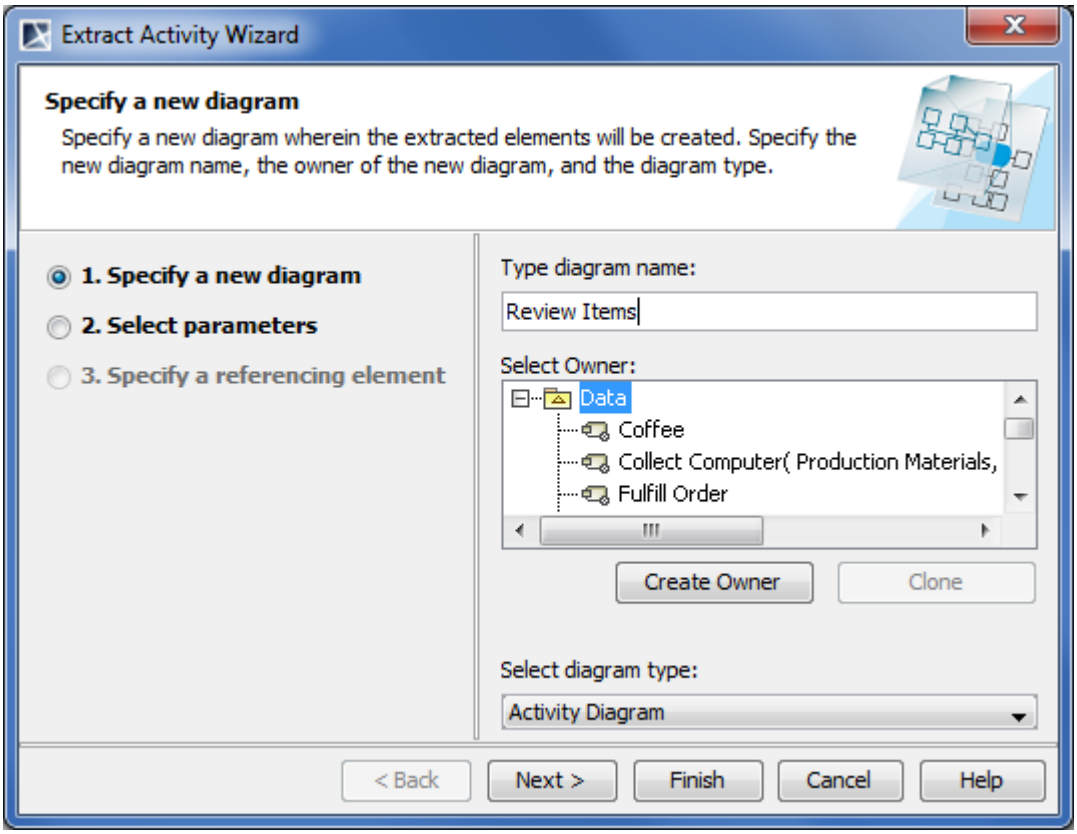
The **Extract Activity Wizard** consists of three steps:

1. Specification of a new element
2. Creation of parameters
3. Creation of a reference element

Step #1. Specification of a new diagram

In this step, you can specify a diagram into which the extracted part will be moved and a package wherein this new diagram will be created. You can also define the type of a newly created diagram here. The activity diagram type is selected by default.

The new diagram will be created along with a new activity - the diagram will be created as an inner activity element.



Element name	Description
Type diagram name	Type the name of the new diagram wherein the extracted part will be stored.

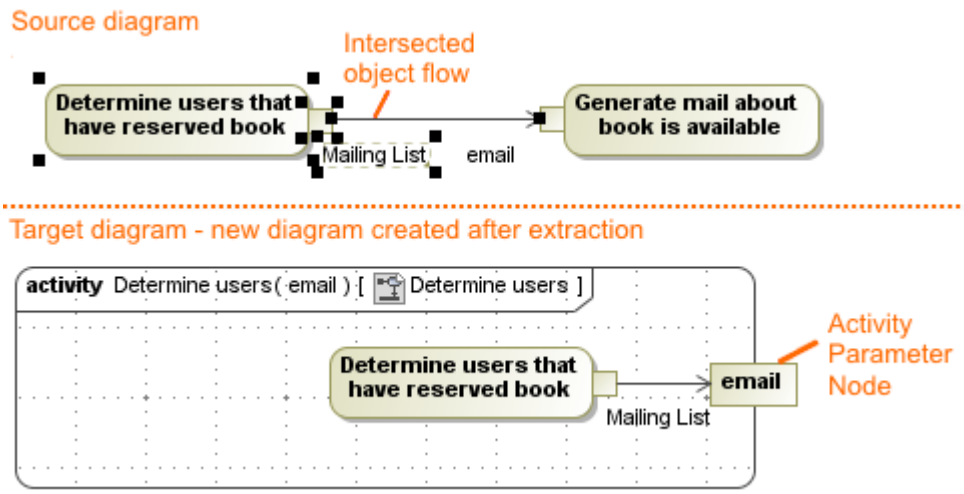
Element name	Description
Select owner	In the model tree, select a package, profile, or model as the owner for the newly created activity. You can also: • Create a new owner. Click the **Create Owner** button, select an element type, and specify its properties. • Clone an existing package, profile, or model. In the model tree, select the owner, click the **Clone** button, and specify properties of the cloned element.
Select diagram type	Open the list and select the diagram type for the extracted part. The default diagram type is activity diagram. You can also select to create an interaction overview diagram. By using the customized plugins, you can add custom diagrams to the diagram type list.

After you have specified the new diagram name, owner of the diagram, and the type of the new diagram, do one of the following:

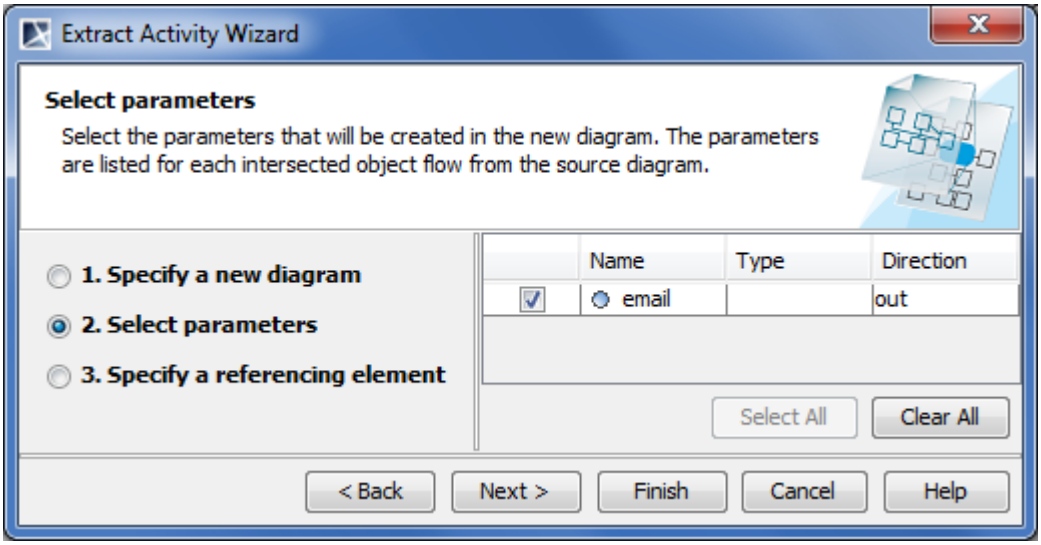
- Click **Next**, if you want to specify parameters and to define a referencing element. The next wizard step opens.
- Click **Finish**. The selected part of an activity diagram is extracted to a new diagram that is specified in this step. Instead of the extracted part, the Call Behavior Action is created in the activity diagram referencing to the newly created diagram. The second and the third wizard steps are skipped - that is the default options are taken.

Step #2. Selection of parameters

The parameter(s) that will be created for the intersected object flow is listed in this step. Along with the parameter(s) in the newly created target diagram, the activity parameter node will be created. The activity parameter node has the pin created as an inner element, as well as the assigned parameter.

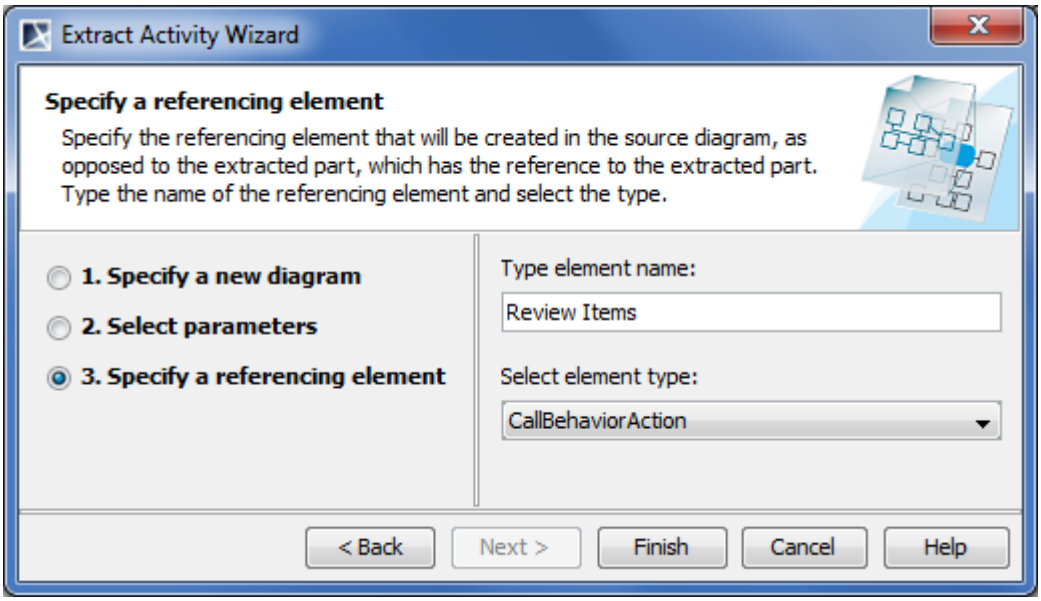


- The **Select parameters** list is empty unless there are intersected object flows selected in the source activity diagram.
- For more information about intersected object flows, see concepts in Extracting(see page 947).



Step #3. Specification of a referencing element

A referencing element is created in the source diagram instead of the extracted part. Double-click the referencing element to navigate to the extracted part.



Element name	Description
Type element name	Type the name of the referencing element that will be created in the source activity diagram in place of the selected elements.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Element name	Description
Select element type	Open the list and select the referencing element type. The default referencing element type is a call behavior action. Custom elements can be added to the referencing element type list using customized plugins.

After you have selected the parameters, do one of the following:

- Click **Back** to return to the first step.
- Click **Finish**. The selected part of an activity diagram is extracted to a new diagram.

On this page

- Step #1. Specification of a new diagram(see page 959)
- Step #2. Selection of parameters(see page 960)
- Step #3. Specification of a referencing element(see page 961)

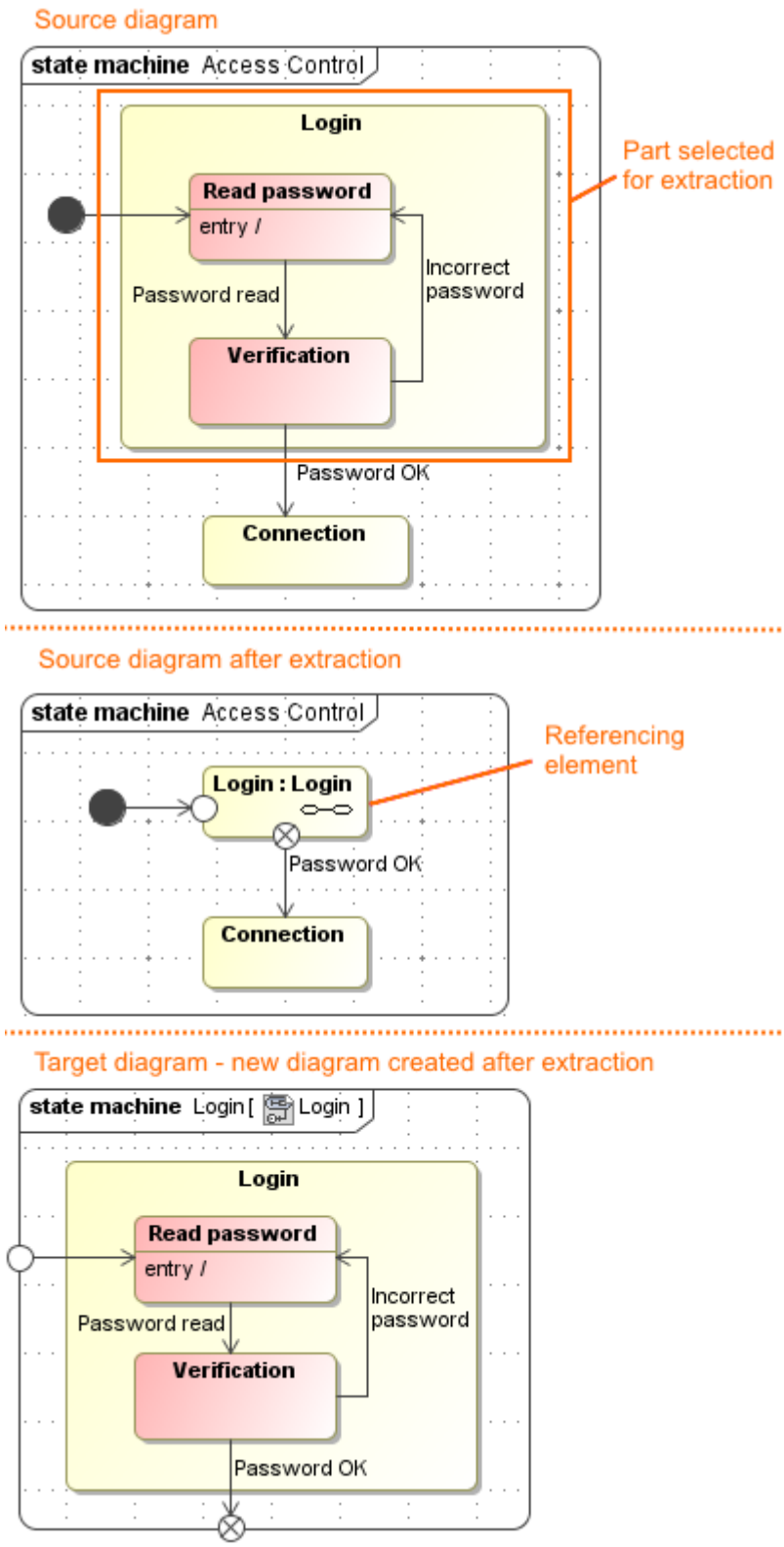
Related pages

- Extracting(see page 947)
- Extracting in Activity diagram(see page 954)
- Extraction rules in Activity diagram(see page 956)
- Working with model elements(see page 909)

Extracting in State Machine diagram

The extraction functionality allows for extracting a selected part of the state machine diagram to a newly created diagram. After extraction, a submachine state is created in the source diagram in place of the extracted part(s). The submachine state is a referencing element that references the newly created state machine diagram.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



As you can see in the example, during the extraction, the selected part of the state machine diagram is moved to the newly created Login state machine diagram. In the source diagram, a Login submachine state is created in place of extracted elements.

Related pages

Related pages

- Extracting(see page 947)
- Working with model elements(see page 909)

Extract State Machine Wizard

The **Extract State Machine Wizard** is used for extracting a selected part of a state machine diagram to a separate diagram. Using this wizard, you can:

- Create and specify the target diagram that will be created as the result of the extraction. The selected part will be extracted to the newly created diagram.
- Select the entry and exit points that will be created in a new diagram. The entry and exit points are listed for each intersected transition from the source diagram.
- Specify a referencing element that is created in the source state machine diagram and represents the elements moved to the target diagram during the extraction.

To start the **Extract State Machine Wizard**

---

1. In the state machine diagram, select a part that you want to extract.
2. From the selected part's shortcut menu, choose **Refactor** > **Extract**. The **Extract State Machine Wizard** opens.

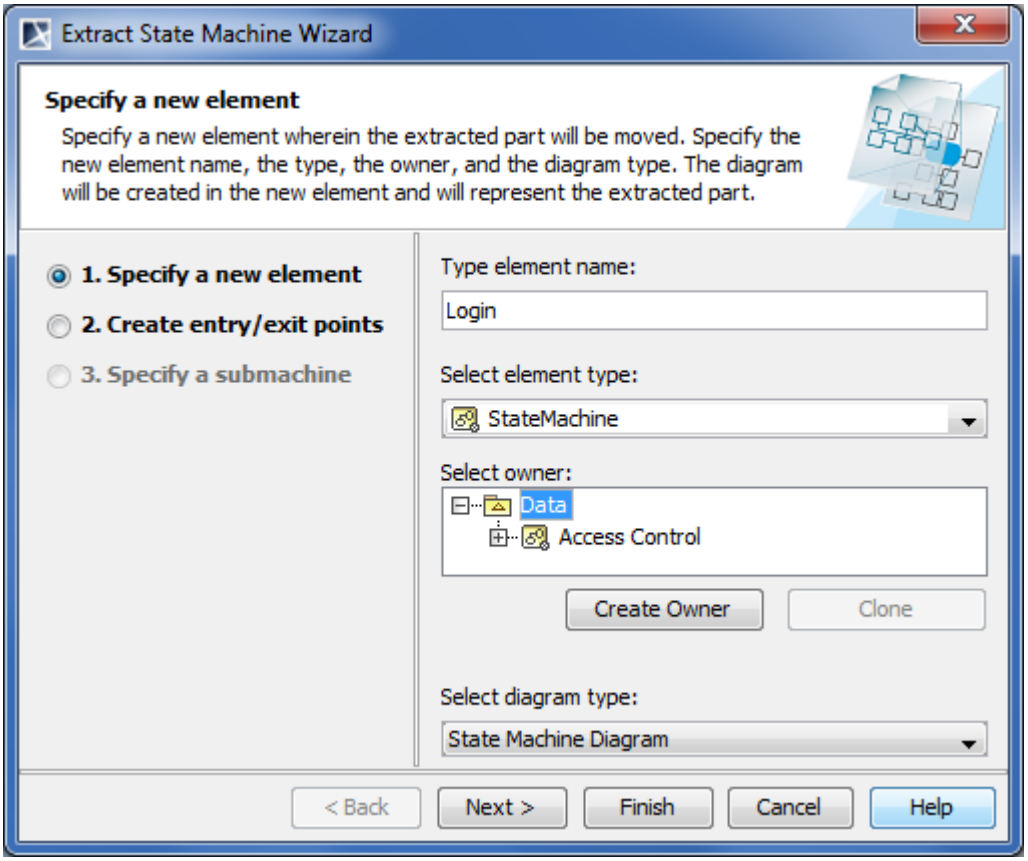
The **Extract State Machine Wizard** consists of three steps:

1. Specification of a new element
2. Creation of entry/exit points
3. Specification of a submachine

Step #1. Specification of a new element

In this step, you can specify a diagram into which the extracted part will be moved. The new diagram will be created along with a new state machine. The diagram will be created as an inner state machine element. You can also define the element type, name, and the owner. Note, that the diagram name will be the same as the element name. The state machine type is selected by default.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Element name	Description
Type element name	Type the name of the new element wherein the extracted part will be stored. The new element will contain the new diagram.
Select element type	Open the list and select the element type in which the extracted part and new diagram will be created. The default element type is a state machine. By using customized plugins, you can add custom elements to the element type list.
Select owner	In the model tree, select a package, profile, or model as an owner of the newly created state machine. You can also: • Create a new owner. Click the **Create Owner** button, select an element type, and specify its properties. • Clone an existing package, profile, or model. To clone, in the model tree, select an owner, click the **Clone** button, and specify properties of the cloned element.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

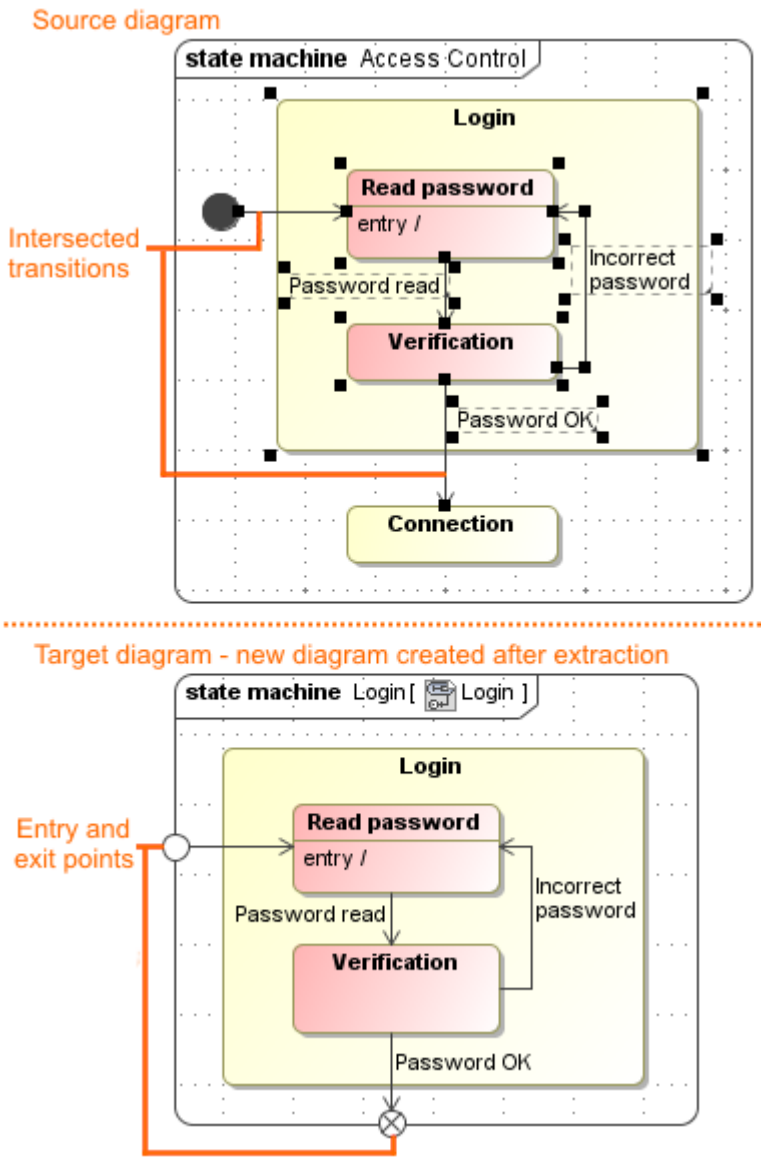
Element name	Description
Select diagram type	Open the list and select the diagram type for the extracted part. The default diagram type is a state machine diagram. By using customized plugins, you can add custom diagrams to the diagram type list.

After you have specified the new element name, an owner of the element, and the type of the new diagram, do one of the following:

- Click **Next** , if you want to specify entry/exit points and define a referencing element. The next wizard step opens.
- Click **Finish**. The selected part of a state machine diagram is extracted to a new diagram that is specified in this step. Instead of the extracted part, the submachine state is created referencing to the newly created diagram. The second and the third wizard steps are skipped - that is the default options are taken.

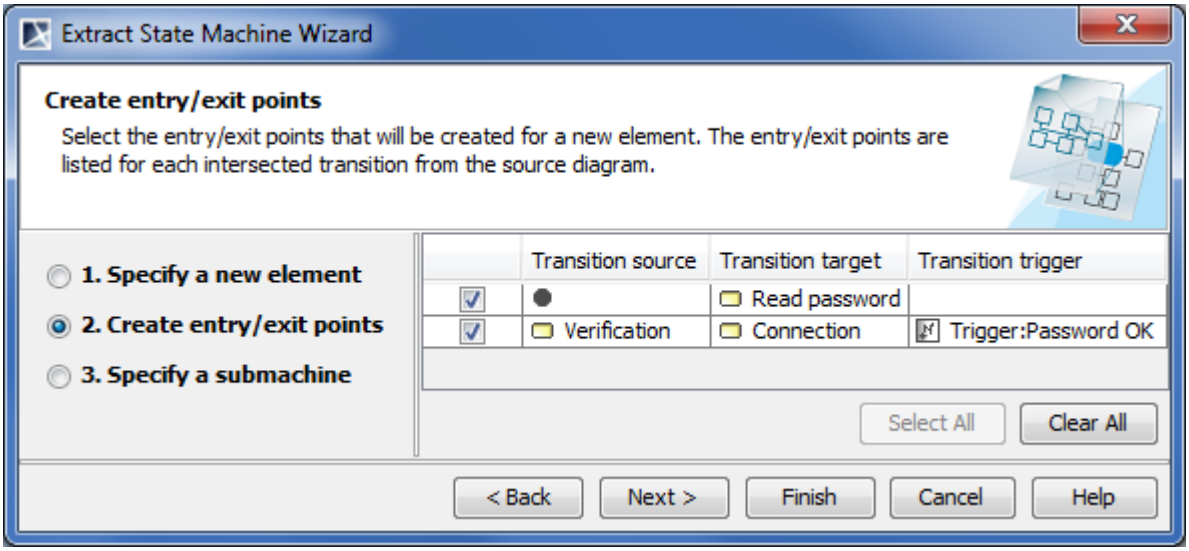
Step #2. Creation of entry/exit points

The entry and exit points that will be created for the intersected transitions are listed in this step.



As you can see in the example, in the *Access Control* source diagram, the intersected transitions are selected for the extraction. After the extraction, in the target diagram, the entry and exit points are created for each intersected transition.

> ⚠ In the **Extract State Machine Wizard**, the **Create entry/exit point** step is empty unless there are intersected transitions selected in the source state machine diagram.

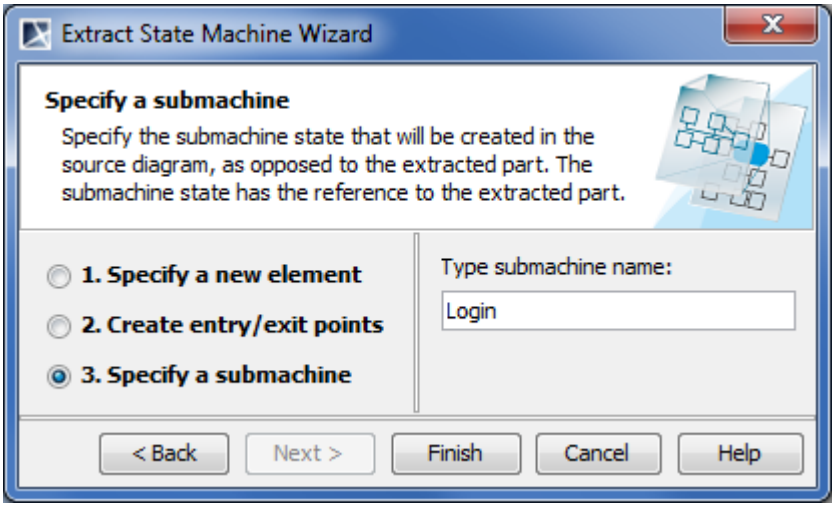


After you have selected the parameters, do one of the following:

- Click **Back** to return to the first step.
- Click **Next** to define a referencing element. The next wizard step opens.
- Click **Finish**. The selected part of the state machine diagram is extracted to a new diagram. The third wizard step is skipped - that is the default submachine name is taken.

Step #3. Specification of a submachine

A referencing element is created in the source diagram instead of the extracted part. Double-click the referencing element to navigate to the extracted part.



After you have typed the submachine name, do one of the following:

- Click **Back** to return to the second step.
- Click **Finish**. The selected part of the state machine diagram is extracted to the new state machine diagram.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

On this page

- Step #1. Specification of a new element(see page 964)
- Step #2. Creation of entry/exit points(see page 966)
- Step #3. Specification of a submachine(see page 968)

Related pages

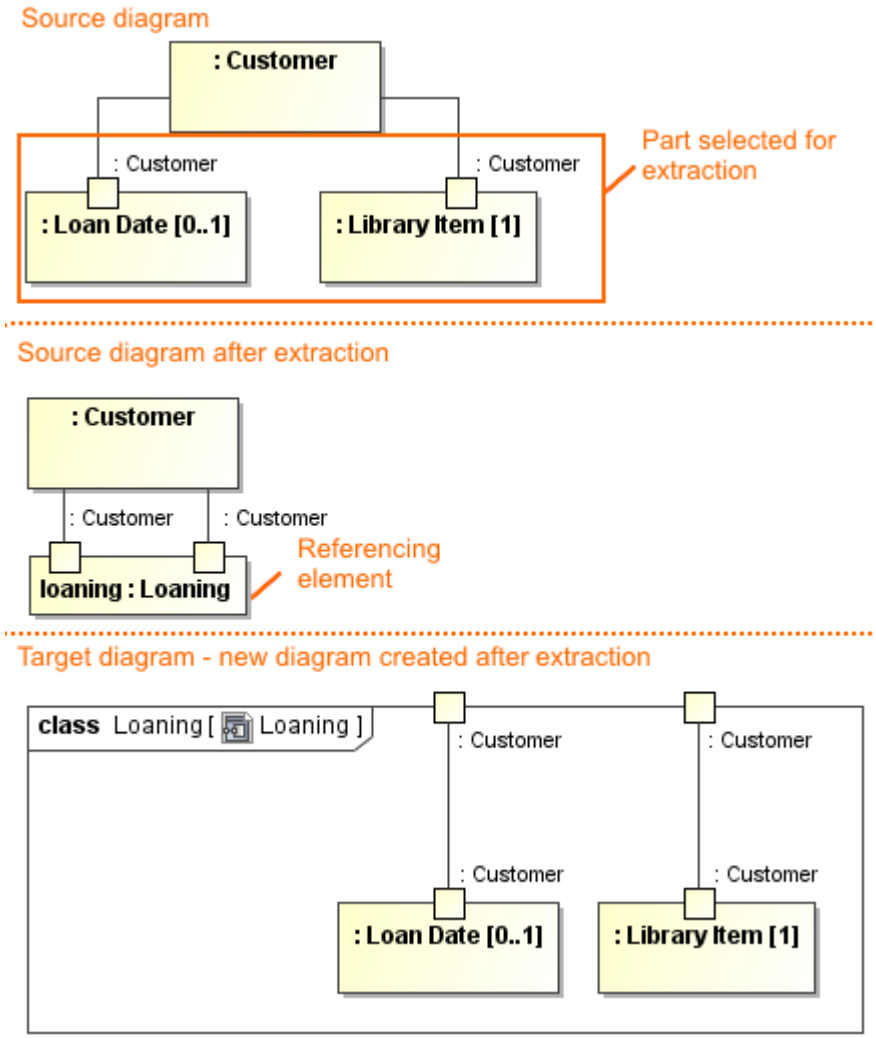
- Extracting(see page 947)
- Extracting in State Machine diagram(see page 962)
- Working with model elements(see page 909)

Extracting in Composite Structure diagram

With the SysML plugin, the extracting in composite structure diagram feature is available in Standard, Professional, Architect, and Enterprise editions.

The extraction functionality allows for extracting a selected part of a composite structure diagram to a newly created diagram. After extraction, a property is created in the source diagram in place of the extracted part(s). That is, the property is a referencing element that references the newly created activity diagram.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



As you can see in the example, during extraction, the selected part of the composite structure diagram is moved to the newly created *Loaning* composite structure diagram. In the source composite structure diagram, a *loaning* property is created in a place of the selected elements.

Related pages

Related pages

- Extracting(see page 947)
- Working with model elements(see page 909)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Extract Structure Wizard

The **Extract Structure Wizard** is used for extracting a selected part of a composite structure diagram to a separate diagram. Using this wizard, you can:

- Create and specify the target diagram that will be created as the result of the extraction. The selected part will be extracted to the newly created diagram.
- Select the ports that will be created in a new diagram. The ports are listed for each intersected connector from the source diagram.
- Specify a referencing element (in this case, a property) that is created in the source composite structure diagram and represents the elements moved to the target diagram during the extraction.

To start the **Extract Structure Wizard**

---

1. In the composite structure diagram, select a part that you want to extract.
2. From the selected part's shortcut menu, choose **Refactor** > **Extract**. The **Extract Structure Wizard** opens.

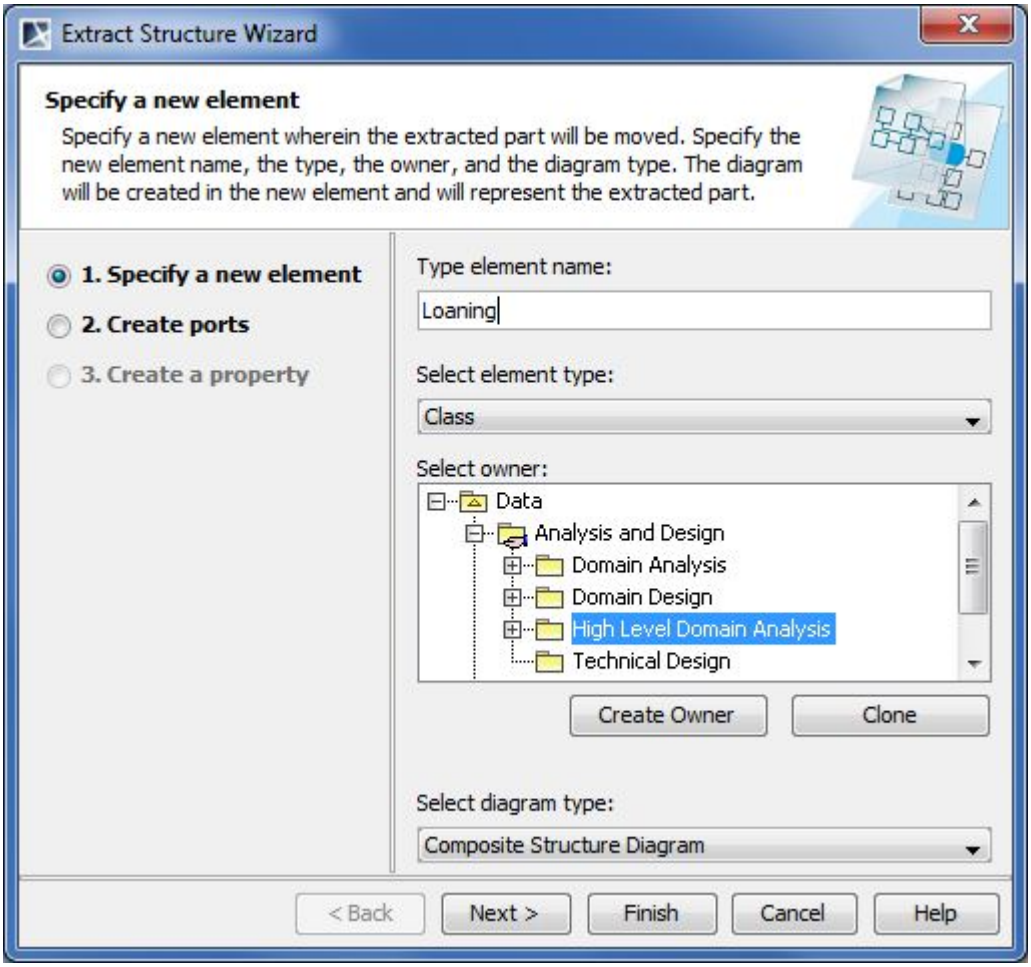
This section describes components of the **Extract Structure Wizard**, such as text boxes and buttons. The Extract Structure Wizard consists of three steps:

1. Specification of a new element.
2. Creation of ports.
3. Creation of a property.

Step #1. Specification of a new element

In this step, you can specify a diagram into which the extracted part will be moved and an owner wherein this new diagram will be created. You can also define the type of the newly created classifier element that will own the diagram. The class element and the composite structure diagram type are selected by default. The name of the composite structure diagram will be the same as the specified name of the classifier element.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Element name	Description
Type element name	Type the name of the new element, that is, classifier, wherein the extracted part will be stored.
Select element type	Open the list and select the element type. The default element type is a class. By using the customized plugins, you can add custom elements to the element type list.
Select owner	In the model tree, select a package, profile, or model as an owner of the newly created element. You may also: • Create a new owner. Click the **Create Owner** button, select an element type, and specify its properties. • Clone an existing package, profile, or model. In the model tree, select an owner, click the **Clone** button, and specify properties of the cloned element.
Select diagram type	Open the list and select the diagram type for the extracted part. The default diagram type is a composite structure diagram. By using the customized plugins, you can add custom diagrams to the diagram type list.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

After you have specified the new element name, type, owner of the diagram, and the type of the new diagram, do one of the following:

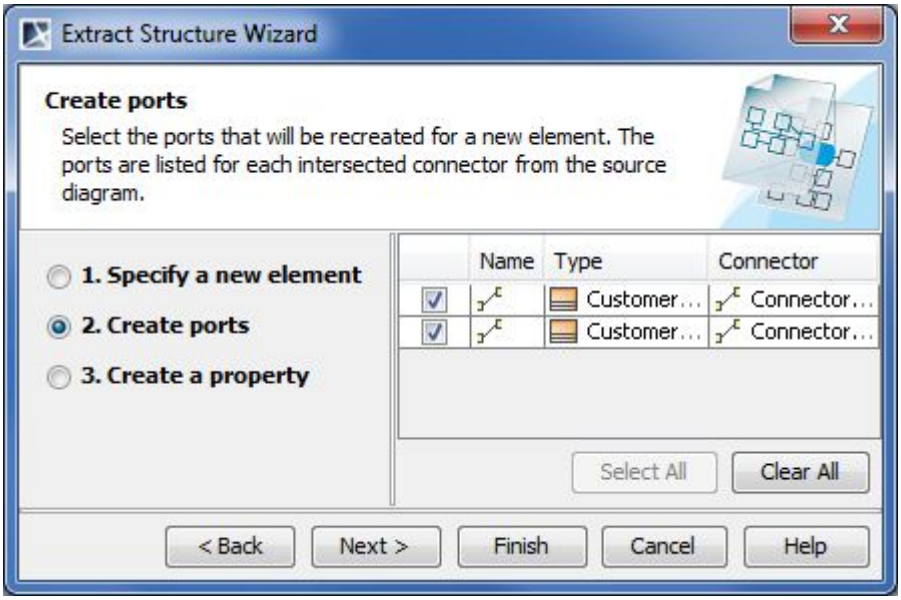
- Click **Next** , if you want to specify ports and to define a referencing element. The next wizard step opens.
- Click **Finish**. The selected part of a composite structure diagram is extracted to a new diagram that is specified in this step. Instead of the extracted part, the selected element type is created in the composite structure diagram referencing to the newly created diagram. The second and the third wizard steps are skipped, that is, the default options are taken.

Step #2. Creation of ports

In this step, there are listed the port(s) that will be created for the:

- intersected connector(s)
- incoming connector(s)
- outgoing connector(s)

> The **Create ports** list is empty unless there are intersected, incoming and/or outgoing connector(s) selected in the source composite structure diagram.



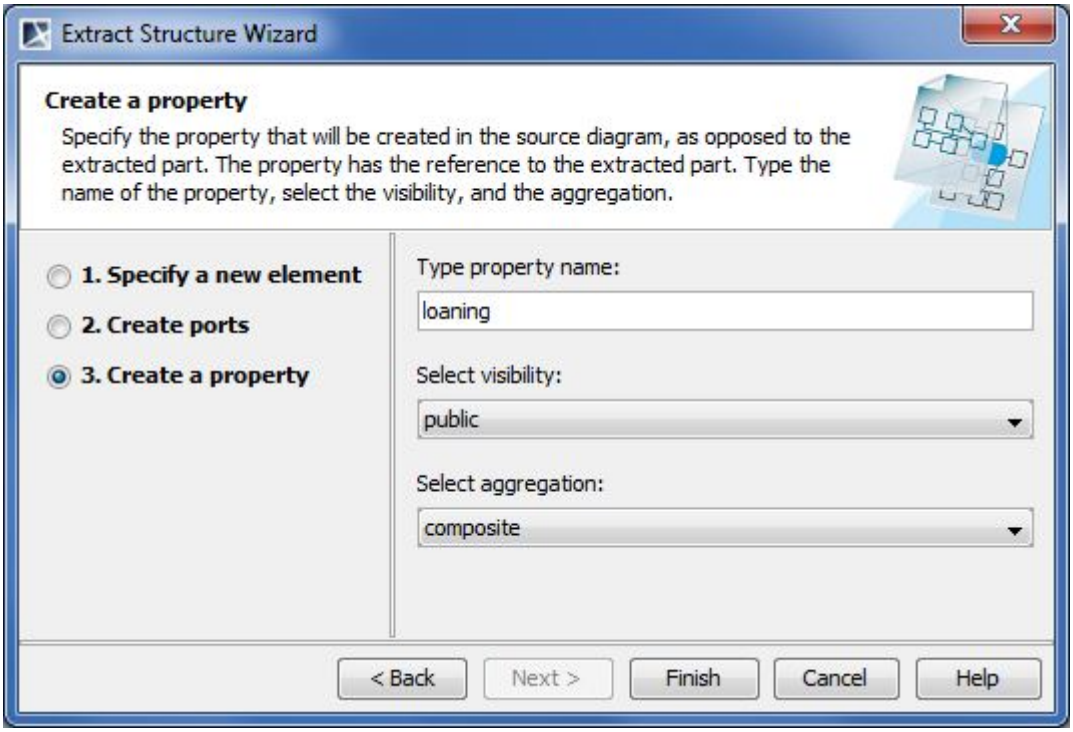
After you have selected the ports, do one of the following:

- Click **Back** to return to the first step.
- Click **Next**, if you want to define a property (that is, referencing element). The next wizard step opens.
- Click **Finish**. The selected part of the composite structure diagram is extracted to a new diagram. The third wizard step is skipped, that is, the default options are taken.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Step #3. Creation of a property

A property is created as the result of the extraction. The property is created under the classifier that was specified in the first wizard step. The symbol of the property is created in the source composite structure diagram and represents the elements moved to the target diagram during the extraction.



After you have selected the parameters, do one of the following:

- Click **Back** to return to the previous step.
- Click **Finish**. The selected part of a composite structure diagram is extracted to a new diagram.

On this page

- Step #1. Specification of a new element(see page 971)
- Step #2. Creation of ports(see page 973)
- Step #3. Creation of a property(see page 974)

Related pages

- Extracting(see page 947)
- Extracting in Composite Structure diagram(see page 969)
- Working with model elements(see page 909)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Formatting symbols

Definition	Description
Symbol	The term "symbol" means a visual representation of some model elements in the diagram. Symbols are subdivided into shapes and paths (lines in the model, for displaying various relationships).
Symbol properties	Every symbol may have its own style: color, font, size, and so forth. Symbol properties may be defined for the concrete symbol, for all symbol of one element, or according to the diagram type.

You may define the symbol properties in the following ways:

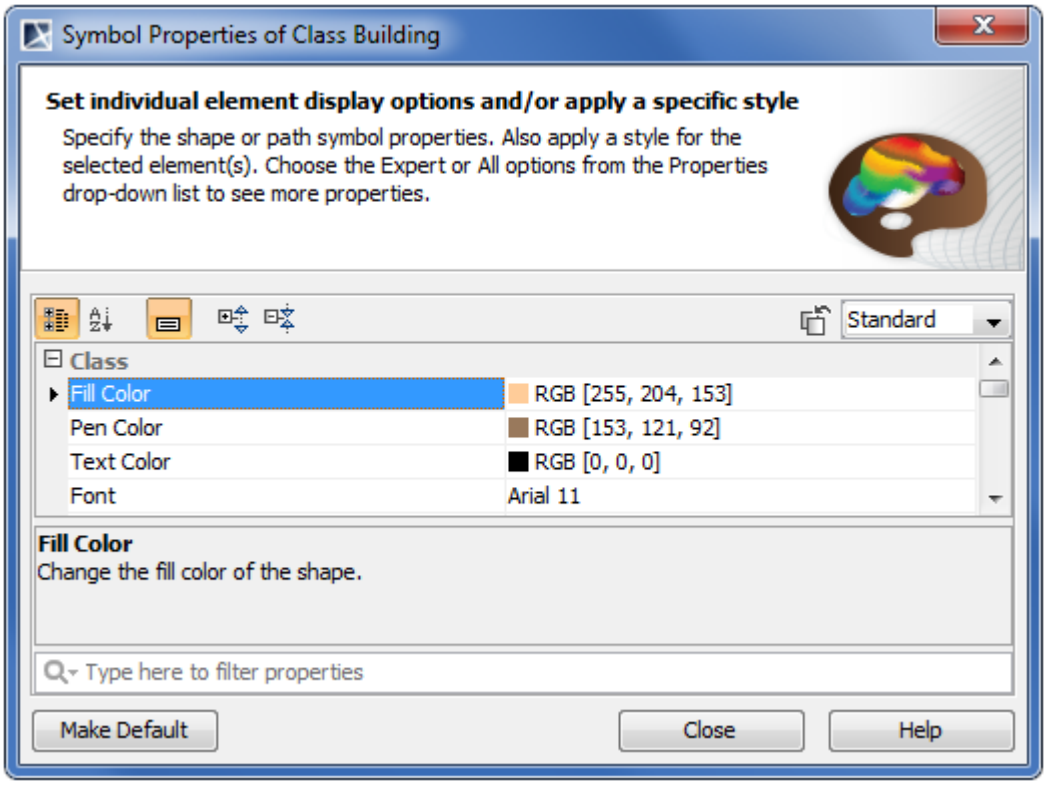
- In the **Symbol Properties** dialog(see page 975).
- In the **Symbol** tab on the Properties panel at the bottom of the Model Browser.
- In the **Project Options** dialog. Using this dialog you can change all available symbol properties, create your own style for the project, apply different symbol properties for different diagrams, define stereotype properties that may be bound to the symbol, and more.
- On the diagram toolbar. Using this toolbar you can change the color, font, path style of a symbol, and paste the symbol properties.

Related pages

Symbol Properties dialog

In the **Symbol Properties** dialog, you may define various symbol (or group of symbols) properties, such as fill color, font type, whether to wrap text. The **Symbol Properties** dialog is a non-modal window - you can edit the symbol properties and see the changes on the diagram instantly. In addition, while the dialog is active you can still work with diagrams or other dialogs.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



To open the **Symbol Properties** dialog

1. On the diagram, select a symbol or a group of symbols.
2. Do one of the following:
   - From the main menu, select **Edit** > **Symbol** > **Symbol Properties**.
   - From the selected elements' shortcut menu, choose **Symbol Properties**.
   - Press Alt+Enter.

In the following table, you will find the described common functions that **Symbol Properties** dialog provides.

Icon	Title	Type	Description
	Apply Style	Button	Click to open the menu with the available styles(see page 988).

Icon	Title	Type	Description
	Show All Properties	Button	Click to display all the properties for the selected symbol types. Note that the *Show All Properties* button is displayed only when at least two shapes of different element types are selected, for example, a class and a package.
Make Default	**Make Default**	Button	Click to change the default symbol properties to the properties that are specified in this dialog. It is highly recommended to use the **Make Default** button only if you fully understand the symbol styles and how your symbol styles can be affected.

If your project is of the earlier version you need to set up a new symbol properties manually.

Related pages

- Symbol Properties dialog(see page 975)
- Applying image(see page 977)
- Displaying icon or image(see page 982)
    - Displaying rake icon(see page 984)
- Symbol styles option group(see page 985)
- Working with Symbol styles(see page 988)
- Properties Inheritance(see page 990)
- Stereotype properties(see page 990)
    - Collecting symbol styles from modules[160]
- Copy and paste symbol style(see page 1000)
- Making your own default style for new projects(see page 1001)

Applying image

On this page

160 https://docs.nomagic.com/display/MD190SP4/Collecting+symbol+styles+from+modules

- Ways to apply image on a symbol(see page 978)
    - The Image Library tab(see page 978)
    - The Select Image dialog(see page 979)
    - The external resources(see page 980)
- Applying image to Property or its type(see page 981)

Ways to apply image on a symbol

You can apply an image on a symbol using:

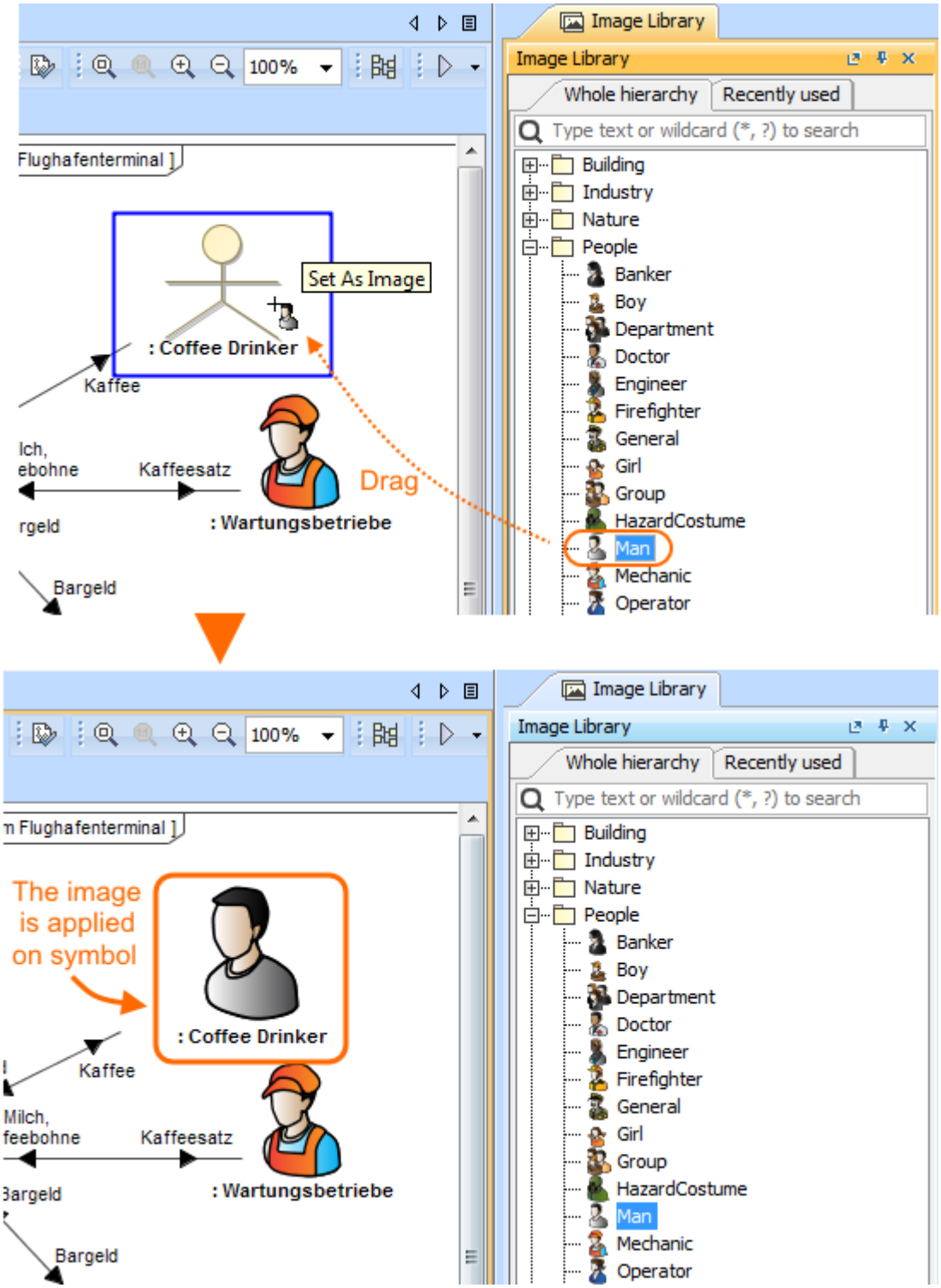
-
    - The Image Library tab(see page 978)
    - The Select Image dialog(see page 979)
    - The external resources(see page 980)

The Image Library tab

To apply an image on a symbol using the **Image Library** tab

---

1. From the **Window** menu, select **Image Library.** The Image Library tab opens on the right side of the window.
2. Select the image you want to apply from the Image Library tab(see page 57) tree.
3. Drag the image directly on the symbol.
   The image is applied on a symbol.



The Select Image dialog

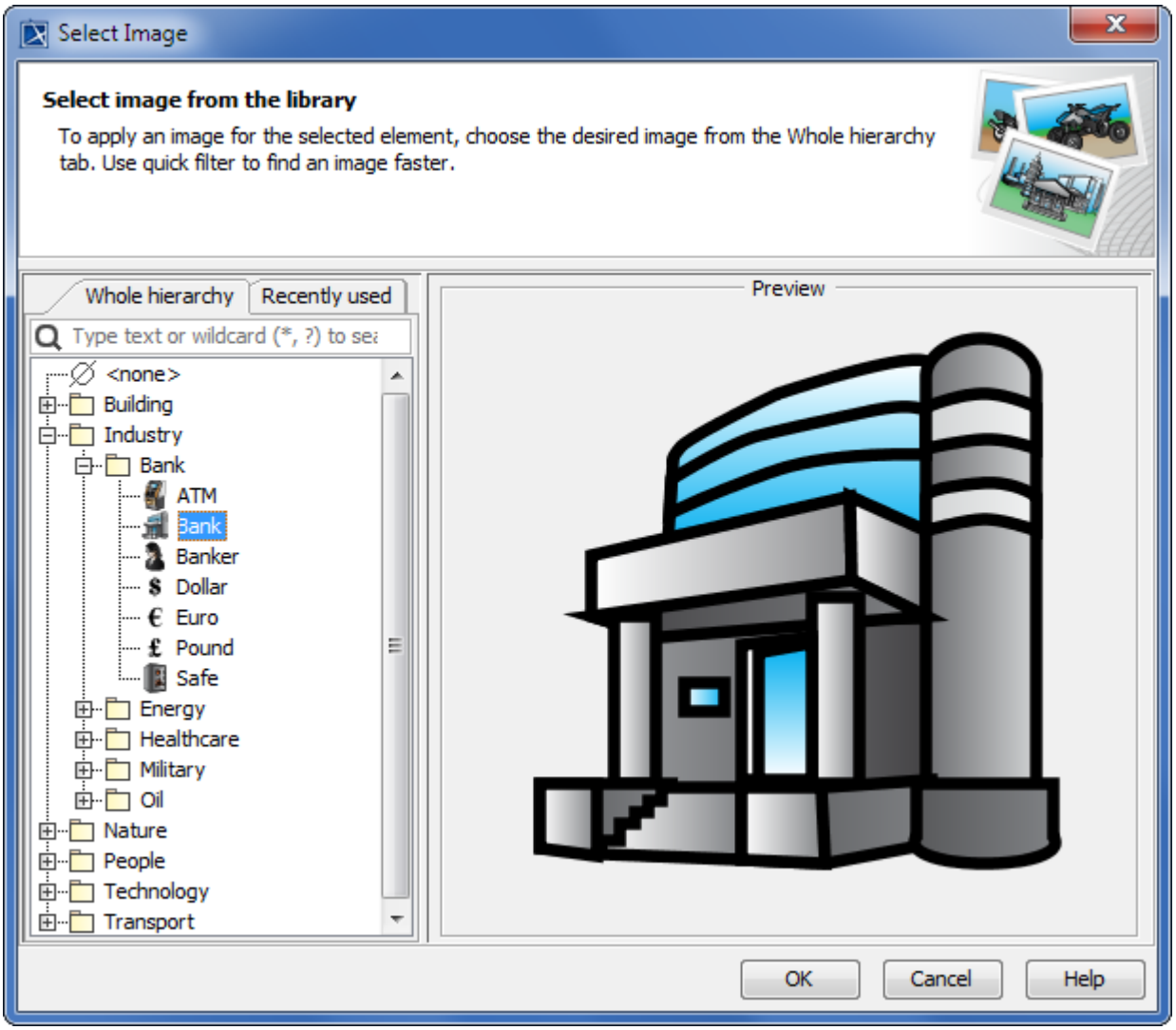
To apply an image on a symbol using **Select Image** dialog

1. Open the **Select Image** dialog by doing one of the following:

- Open the Specification window of any element, select **Image** (**Icon** for Stereotype) property cell and click .
- Select Class, Actor, Port, Block, Part, Node, Performer, Concept Role, Stereotype, or Text Box ((HTML Text) on the diagram pane and click on the smart manipulator toolbar.

> Only mentioned elements support the **Image Library** button on the smart manipulator by default. You can customize the smart manipulator of any other element.

2. Select the image you want to apply.
3. Click **OK**.



The image is applied on a symbol.

The external resources

To apply an image on a symbol from the external resources

1. Select an image from any external resource (e.g. web browser, email).

Image formats
You can apply images of the following formats: .gif, .jpg, .jpeg, .svg, .png, .wmf.

2. Drag it directly on a symbol.
   The image is applied on a symbol.

   If the the image that you applied on a symbol is large, you can resize that symbol to a preferred size automatically.
   To resize a symbol to a preferred size

   a. Select the symbol with an image applied.
   b. Click smart manipulator that appears at the right bottom side of the shape's horizontal border.

Applying image to Property or its type

When applying image to the Property (see page 804)shape (which type is specified) by dragging or selecting an image from the Image Library(see page 978) or other resources(see page 980), and the **Type Selection Mode**(see page 1036) is switched on, the image is applied to the Property type and the notification message is shown by default. For example, if you drag an image on the *workplace* Property shape that type is *Workplace* Class, the image will be applied to the Class but not to the Property.

Exceptions
- If the Property already has the image applied, the new image replaces it and is set to the Property but not to it's type.
- If the Property type already has an image applied, you cannot replace it with the new one by dragging it on Property shape.

To apply image to Property type

1. In the diagram palette(see page 24), switch on the **Type Selection Mode**(see page 1036).
2. Select an image from the Image Library(see page 978) or other resources(see page 980).
3. Drag selected image on the Property shape which:
   - Type is specified.
   - The Property is without image.
   - The Property type is without image.

   The image is applied to the Property type and the notification message is shown by default. You can specify to apply image on Property by changing the **Apply Image To** property value.

Symbol display mode
You can change the symbol display mode. How to change the stereotype name and its icon visibility on an element's shape >>(see page 859)

To specify the **Apply Image To** property value

1. On the main menu, select the **Options** > **Project**.
2. In the project Options dialog, find **Apply Image To** property.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

3. Select one of the following value:
   - **Type and Notify** (default) - to apply image to Property type and after that show notification message on the right corner of the screen.
   - **Type** - to apply image to Property type.
   - **Property** - to apply image to Property.
4. Click **OK**.

Displaying icon or image

You can display icon or image on the symbol or instead of the symbol.

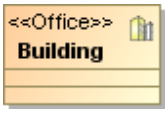
The descriptions given in the following table will help you to understand this functionality better.

Name	Description
Stereotype	Every element can be extended by applying a stereotype(see page 856). For more information about applying stereotype properties, see Stereotype properties(see page 990).
Icon	Icon of stereotype. Icon is a small image displayed in the top-right corner of shape. To assign an icon to a stereotype in the Stereotype Specification(see page 913) window, define the Icon property.
Text	Stereotype name, displayed on the symbol.
Image	Image which can be assigned to element and displayed as icon or instead of element shape. To assign image to element in the element Specification window(see page 913), assign the Image property.

To change the icon visibility mode on the element shape, do one of the following

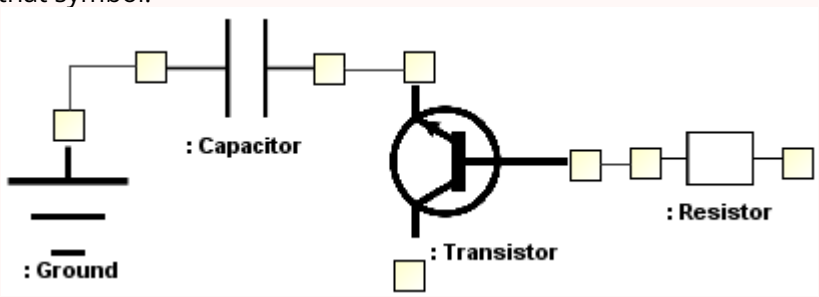
- Open the **Symbol Properties** dialog(see page 975) and in the **Show Stereotypes** property specification cell, select the icon visibility mode.
- Select an element shape and click the *Compartments* button. From the menu, select **Stereotypes** and choose the icon visibility mode.

The stereotype visibility modes are described in the following table:

Mode	Example	Icon and name visibility	Image visibility
Icon and Text		Icon of stereotype displayed. Name of stereotype displayed.	Image of element is displayed in the corner of shape.
Icon		Icon of stereotype displayed in the corner of symbol. Name of stereotype is not displayed.	Image of element is displayed in the corner of shape.
Text Only		Icon of stereotype is not displayed. Name of stereotype is displayed.	Image of element is not displayed.
Shape Image and Text*		Icon of stereotype is displayed instead of shape. Name of stereotype is displayed.	Image of element is displayed instead of shape.
Shape Image*		Icon of stereotype is displayed instead of shape. Name of stereotype is not displayed.	Image of element is displayed instead of the shape.
Do Not Display		Icon of stereotype is not displayed. Name of stereotype is not displayed.	Image of element is not displayed.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

* - When the **Shape Image and Text** or **Shape Image** property is selected, all symbol compartments should be suppressed in order to display the icon of a stereotype instead of the element symbol. If the symbol contains port or pins, these ports or pins are displayed around that symbol.



Shape Image and Text and **Shape Image** properties are not added to the **Path** element properties list.
If element has assigned both - image and stereotype icon - then image of element will be displayed on the shape.

Related pages

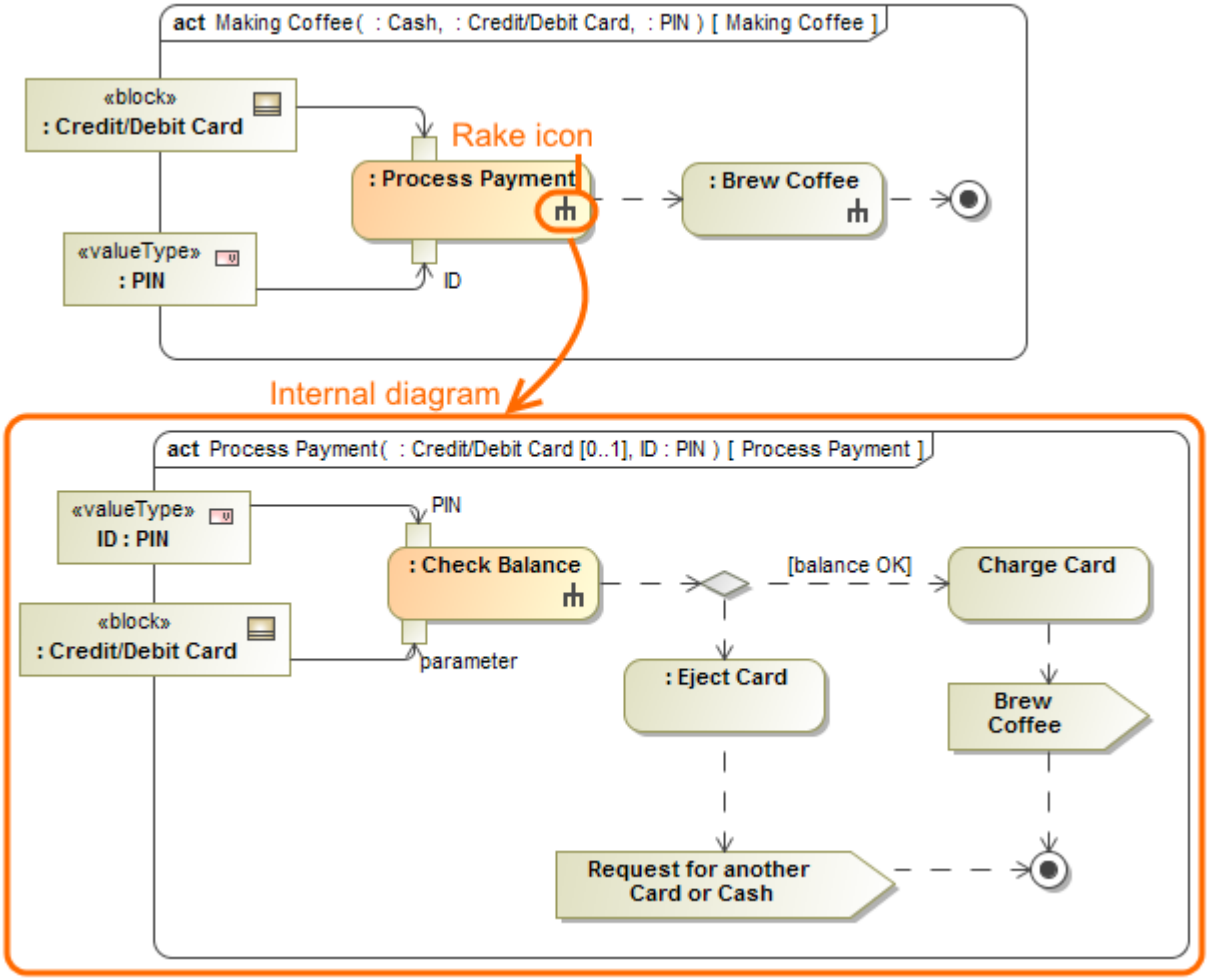
Displaying rake icon

You can use for faster navigation in the model. The rake icon on the symbol indicates that the appropriate element is decomposed and has an inner structure. Double-click the element with rake icon to open the internal diagram.

The rake icon is shown on:

- Call Behavior actions that can refer to another Activity diagram.
- Interaction Uses that can refer to another Sequence diagram.
- Packages that can refer to another Package diagrams.
- Parts that can refer to another Internal Block diagram.
- Use cases can be realized by other behavior diagrams such as Activity, State Machine and Sequence.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



To hide rake icon

1. Right-click the symbol and select **Symbol Properties** or press Alt+Enter.
2. Set the **Show Rake Icon** property value to *false*.

Symbol styles option group

All symbols in MagicDraw are created according to active properties styles. There may be more than one property style in the same project, and the whole style may be applied for the project.

Every style has its own presentation of Diagram, Shape, Path, and Stereotype that you can modify using the **Project Options** dialog in the **Symbol styles** option group. You can set your own options for every model element to the current style.

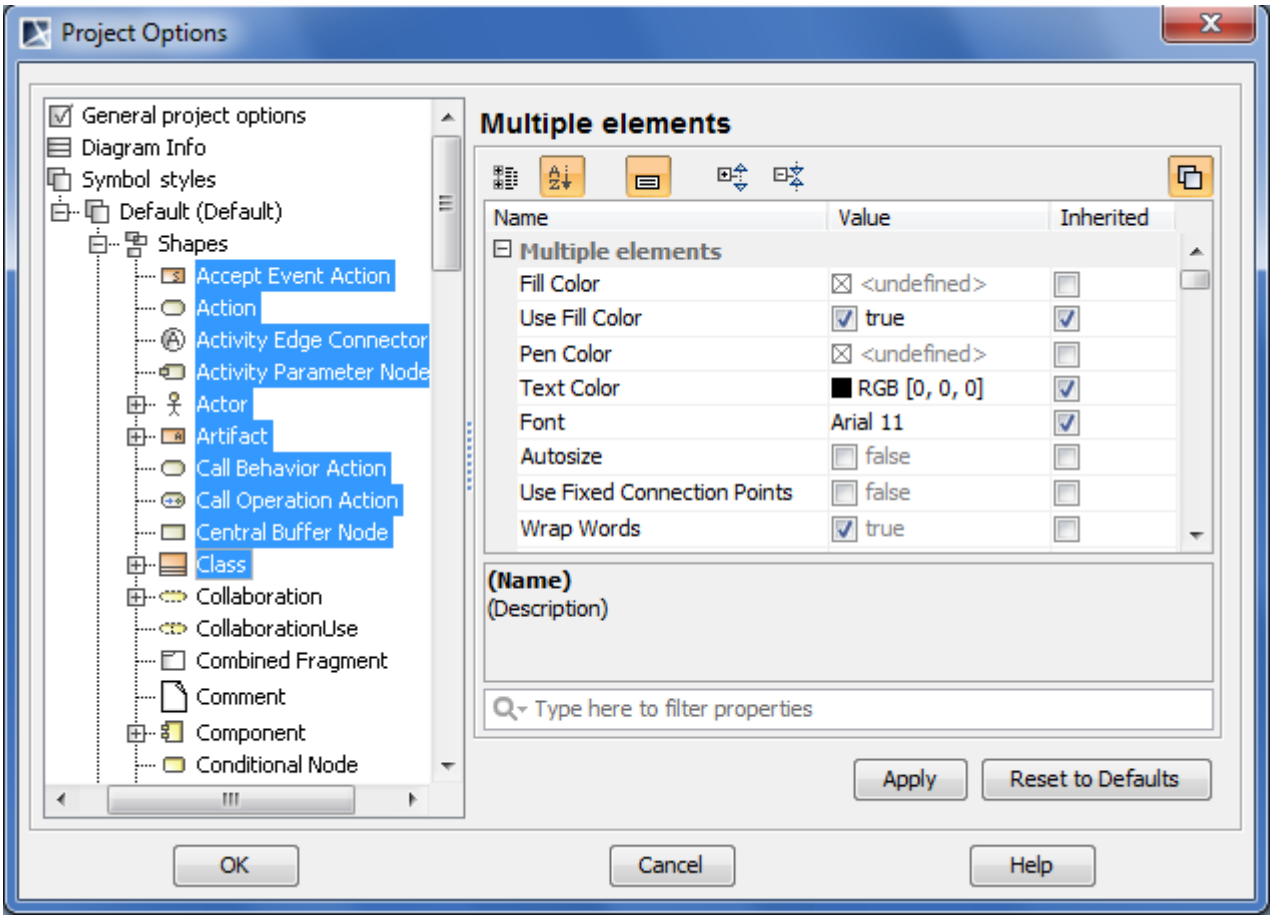
Path, **Shape**, and **Stereotype** branches have the inner structure that helps you find the model element, the representation of which must be changed. The section on the right side of the dialog contains possible choices and instruments to manage them.

The following properties are defined for the formatting symbols:

- **Shapes**. Set general options for the shapes in the right pane of the Project Options dialog. You can set options for all shapes that appear on the Diagram pane.
- **Paths**. Set general options for the paths in the right pane of the Project Options dialog. You can set options for all paths that appear on the Diagram pane.
- **Diagram**. Set general options about a diagram.
- **Stereotypes**. Set general options for the stereotypes in the right pane of the Project Options dialog. You can set options for all stereotypes that may be applied to elements on the Diagram pane.

Changing options for multiple symbols

To change properties for multiple symbols, press Ctrl or Shift and select few symbols in the **Project Options** dialog > **Symbol styles** option group.



Shape, path, and diagram options

All shapes, paths and diagrams that can be created in the project, are listed in the **Project Options** dialog. If the Shape, Path, or Diagram branches are selected in the tree, the general properties can be set in the right pane of this dialog.

When expanding any of these option groups, the style for a particular element (diagram) can be created.

To apply a new style to a previously created element symbol

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

1. In the **Project Options dialog**, change the element style properties and click the **Apply** button. The **Select Diagrams** dialog opens.
2. Select the diagrams to which the element properties will be applied and click **OK**. The **Select Properties to Apply** dialog opens.
3. Select the properties to be applied to the element symbol by moving them from the **All** list to the **Selected** list.
4. Click **OK** when you are done.

> ⚠ If a new style was set, it will be applied for all newly created elements after drawing them on the diagram pane. You can always set the default symbols style to the element by clicking the **Apply Default Symbol Style** button in the main toolbar.

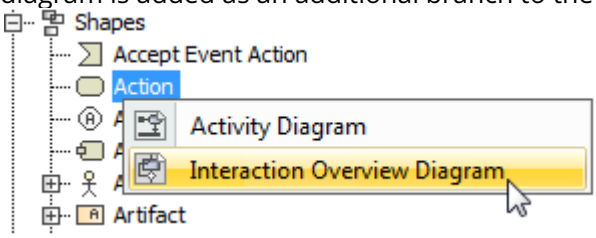
Properties extension by diagram

Diagram, shape, path, and stereotype properties can be extended by the particular diagram type. This means that presentation style options will be applied only for the specified element symbol in the specific diagram.

To specify element properties for the selected diagram type

---

1. In the **Project Options** dialog, the **Symbol Styles** option group, do one of the following
   - Selected the desired element and from its shortcut menu, select the diagram type. The diagram is added as an additional branch to the section.



   - Click the Extend by Diagram button, in the open Extend by Diagram dialog, click the Add Diagram button and select the desired diagram type.
2. Set the style properties for the element. The properties will be applied only in the specified type of diagram.

To remove the extended diagram from the tree, do one of the following

---

- In the **Project Options** dialog, select the extended diagram and from the shortcut menu, choose **Remove**.
- In the **Project Options** dialog, select the extended diagram and click the **Remove** button on the right.

On this page

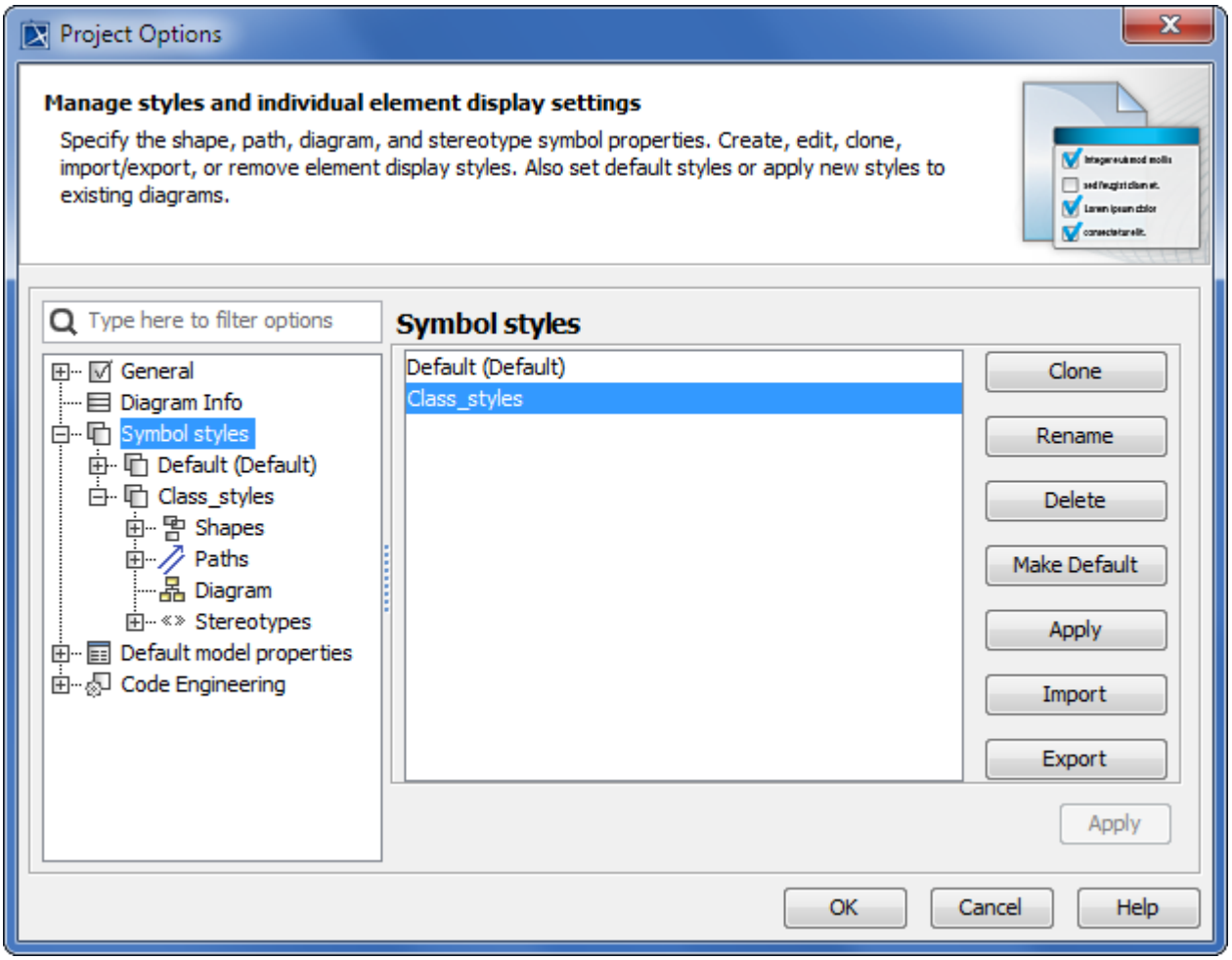
- Changing options for multiple symbols(see page 986)
- Shape, path, and diagram options(see page 986)
- Properties extension by diagram(see page 987)

Related pages

Working with Symbol styles

In the **Project Options** dialog, click the **Symbol Styles** option group, to expand the tree hierarchy of all the styles defined within the project. You may use as many of these styles as you wish.

Shape and **Path** trees have the inner structure to help you find the model element for which you want to change the symbol style. The right side of the dialog contains possible choices and the instruments to manage them.



Before changing symbol styles, we advise you to make a copy of the default style so you can come back to it if needed. You can make a copy by cloning the default style, which will create a new style. You can make changes in any selected style.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To create a new style by cloning the existing one

1. In the **Symbol styles** group, select *Default* style and click the **Clone** button.
2. Enter a name for the new style.
3. Change options of the new style.

As stated previously, you can create several symbol styles. You can set the desired style to be your default at any time.

To make a selected style your default style

- In the **Symbol styles** options group, select the style to make the default and click the **Make Default** button.

You can change Default symbol styles using the **Set Select Symbol Style as Default** button in the diagram toolbar. If you do not see this button, you need to change **Perspective** to the **Full feature** or customize perspective(see page 68).

You can select to apply one of the predefined styles for the existing diagrams.

To apply the selected style

- In the **Symbol styles** group, select the style you want to apply and click the **Apply** button.

You can also apply the desired options to selected diagram model elements. Click the **Apply** button in the specific element's pane.

You can move your earlier defined symbol styles between other projects or users. Simply export the defined style and pass it on for other projects or users to import.

To save the created style (export)

1. In the **Symbol styles** group, click the **Export** button. The **Save** dialog opens.
2. Select the location where you wish to export a style.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To import a project style

1. In the **Symbol styles** group, click the **Import** button. The **Open** dialog opens.
2. Select the style you wish to import (*.stl).

Related pages

- Symbol Properties dialog(see page 975)
- Symbol styles option group(see page 985)
- Copy and paste symbol style(see page 1000)
- Making your own default style for new projects(see page 1001)

Properties Inheritance

All element properties have the “inherited” check box. This check box indicates whether the property is derived from the base element properties or is it specific.

The **Inherited** column check box value in the element properties pane specifies if the current property is synchronized with its parent property. When the **Inherited** value is “true”, the element property is changed after changing the parent property.

If the property has no correspondent property in the upper (parent) level, the **Inherited** column check box will be cleared and not available.

If the property is modified for the specific element and the value differs from the upper level current property value, the **Inherited** column check box is cleared automatically.

Related pages

Stereotype properties

The Stereotype properties can be applied only if the stereotype properties style is created in the **Project Options** dialog.

The Stereotype properties are derived from their base class. The Stereotype base class is defined in the label of the right pane of the **Project Options** dialog.

The same element can have several stereotypes assigned. In this case, the style of the first stereotype will be applied to the element symbol. If the stereotype is removed from the element, the next (first) stereotype properties are applied. If the last stereotype is removed from the element, the base class (shape or path) properties are applied to the element symbol.

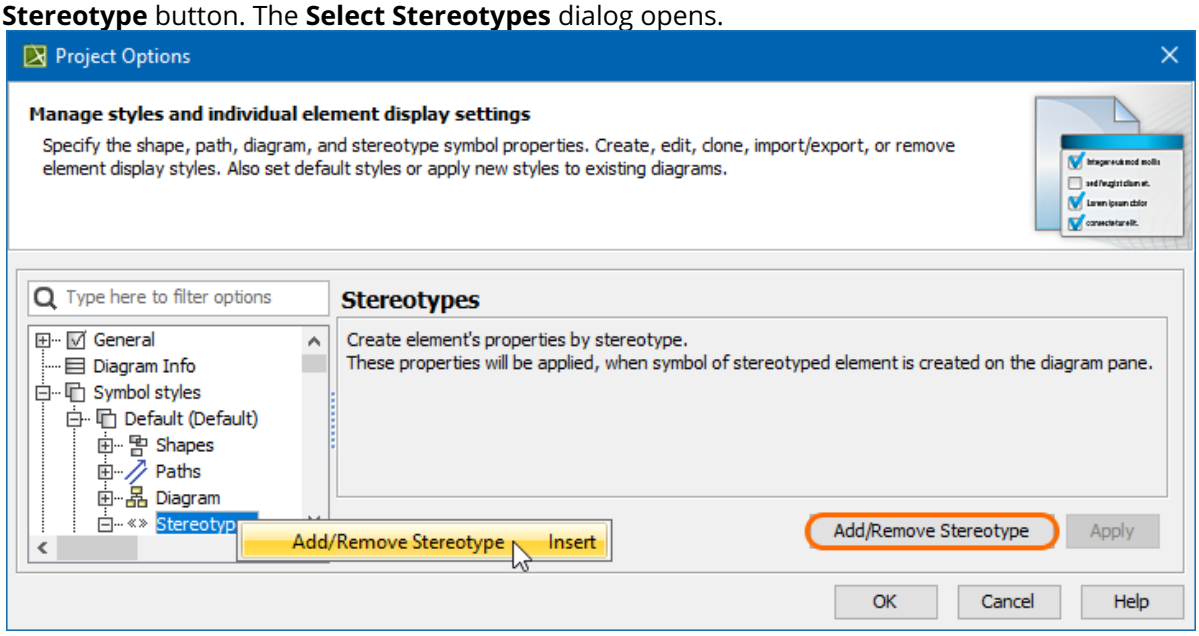
Stereotypes may be extended by diagram.

All stereotypes that have defined symbol properties are included in the **Stereotypes** property group. The default style is created for these stereotypes.

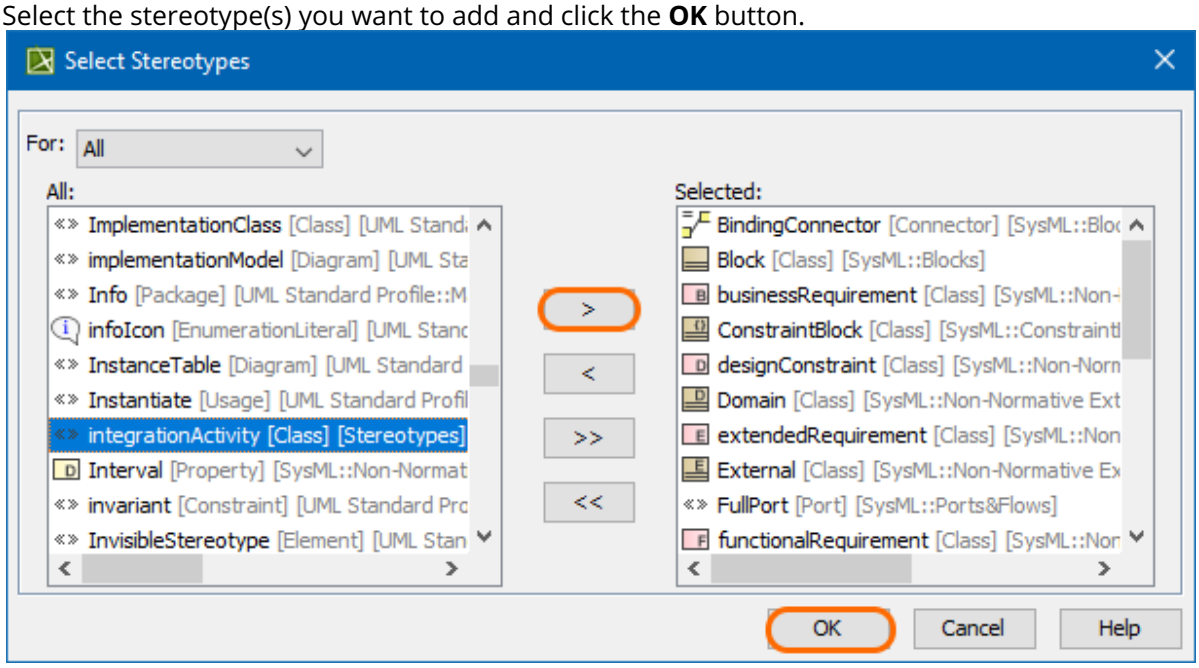
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To add a stereotype to the Stereotypes option group

1. In the **Project Options** dialog > **Symbol styles**, select **Stereotypes** and from the shortcut menu, choose **Add/Remove Stereotype**. Alternatively, you can simply click the **Add/Remove Stereotype** button. The **Select Stereotypes** dialog opens.

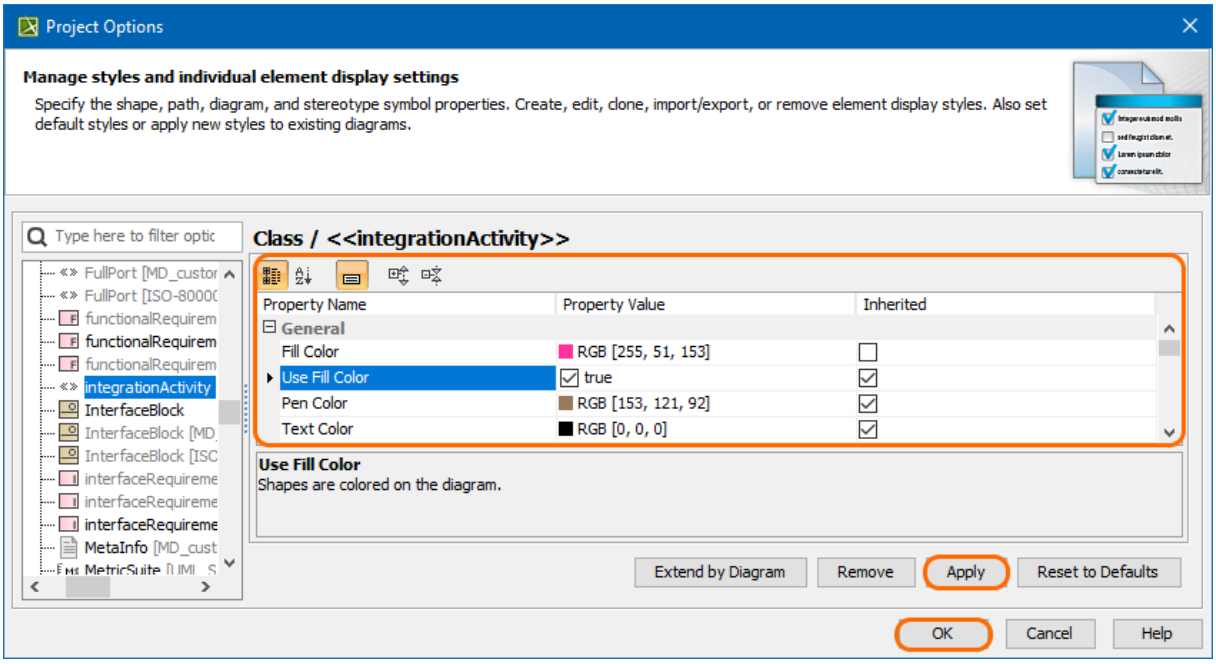


2. Select the stereotype(s) you want to add and click the **OK** button.



The stereotypes are included into the **Stereotypes** option group, and you can now set the stereotype style properties in the right pane of the **Project Options** dialog.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Once you are done customizing your stereotype style, click **Apply** to apply changes to existing shapes in the selected diagram type, or click **OK** to apply changes to future shapes.

To remove a stereotype from the options group

1. In the **Project Options** dialog > **Symbol styles**, expand the **Stereotypes** options group. The list of stereotypes opens.
2. Select a stereotype you want to remove and click the **Remove** button.

To apply the stereotype properties to a previously created symbol with the assigned stereotype

1. In the Project Options dialog, after changing the stereotype style properties, click the **Apply** button. The **Select Diagrams** dialog opens.

   Customized style
   Your customized style will only be used in the diagrams of the selected type.

2. Select the diagrams to which the stereotype properties will be applied and click **OK**.
3. In the **Select Properties to Apply** dialog, select which properties will be applied to the stereotype. Click **OK**.

   Quick search
   Use the Quick search(see page 918) **feature to find the desired property.**

« Properties Inheritance(see page 990)

Customizing stereotype style properties »(see page 993)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related pages

- Symbol Properties dialog(see page 975)
- Applying image(see page 977)
- Displaying icon or image(see page 982)
    - Displaying rake icon(see page 984)
- Symbol styles option group(see page 985)
- Working with Symbol styles(see page 988)
- Properties Inheritance(see page 990)
- Stereotype properties(see page 990)
    - Customizing stereotype style properties(see page 993)
    - Collecting symbol styles from modules[161]
- Copy and paste symbol style(see page 1000)
- Making your own default style for new projects(see page 1001)

Customizing stereotype style properties

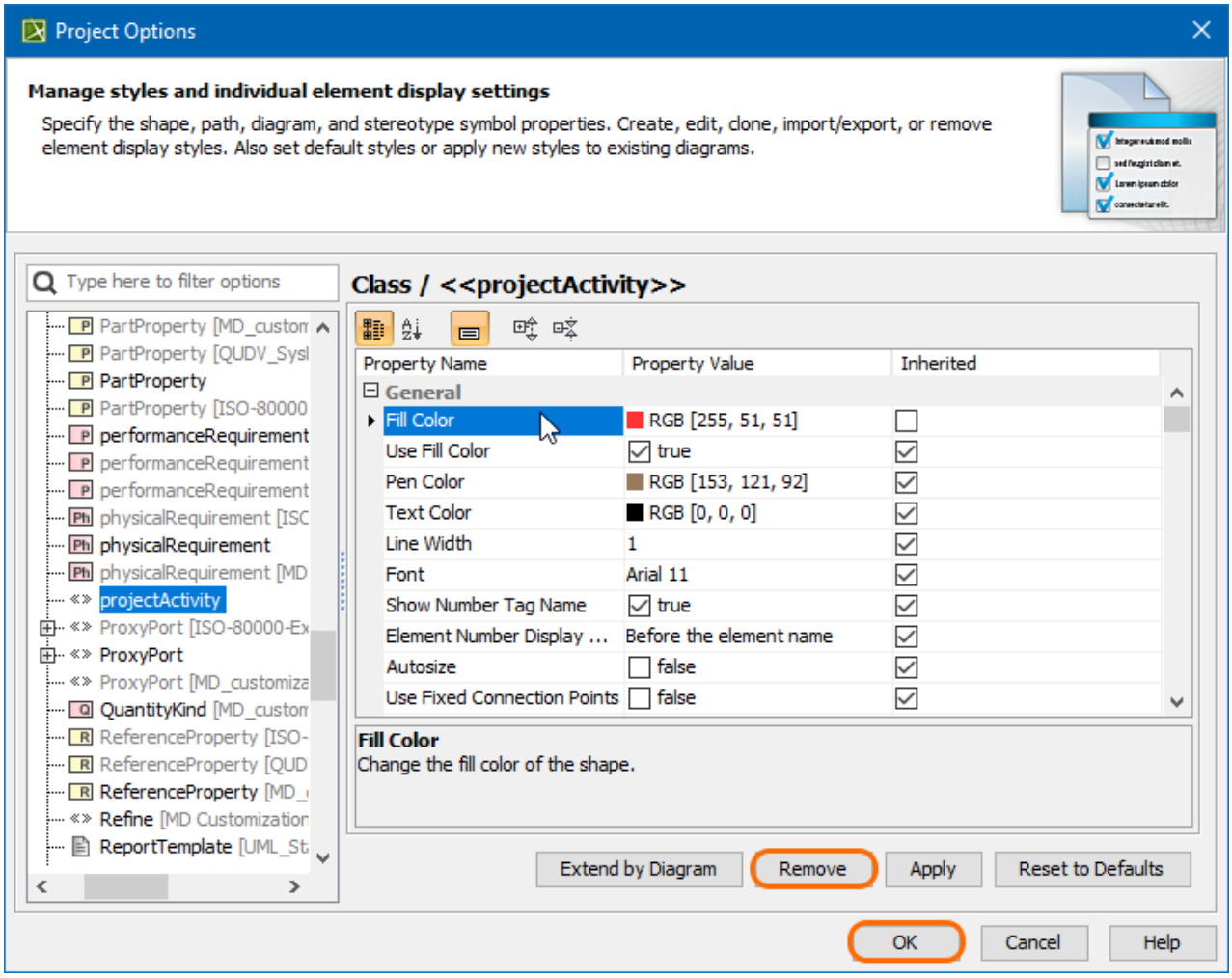
You can customize properties like colors, line styles, and line width, for your stereotypes. To be able to do it, you need to:

1. Add a stereotype to the Stereotypes option group.(see page 991)
2. Once the stereotype is added, customize stereotype style properties either via the Project options(see page 214) or Symbol Properties(see page 975) dialog.

In the **Project Options** dialog, after specifying the stereotype style properties, click **Apply** to apply changes to existing shapes or the **OK** button to apply changes to the new shapes only.

161 https://docs.nomagic.com/display/MD190SP4/Collecting+symbol+styles+from+modules

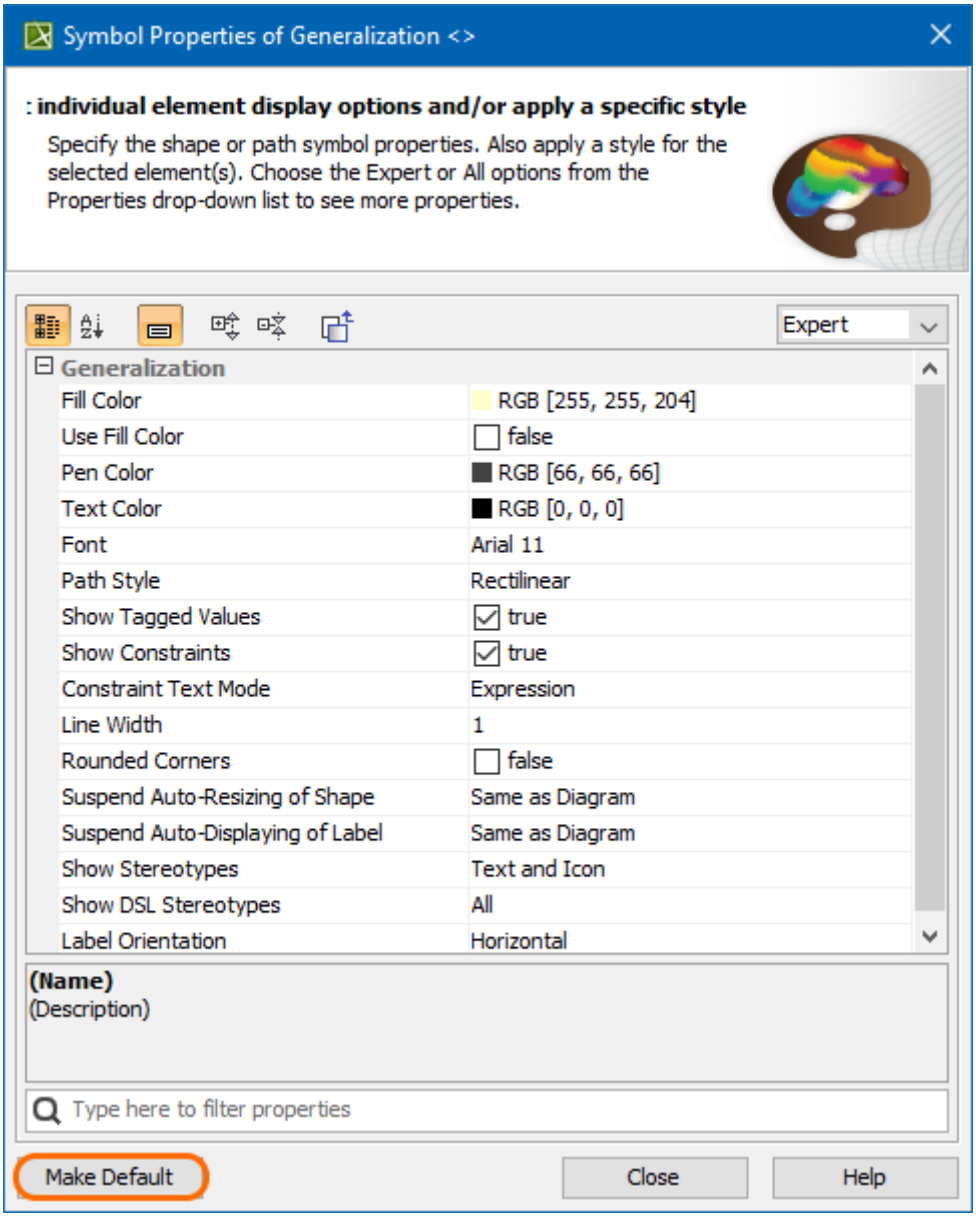
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Customizing stereotype style properties using the Project Options dialog

In the **Symbol Properties** dialog, click **Make Default** after setting the desired style properties for your stereotype.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Customizing stereotype style properties using the Symbol Properties dialog

The best practice is to create styles for stereotypes in the Profile, loading it as a project. When a project uses the Profile, styles for stereotypes will also be loaded and used. These styles will be applied right after applying the stereotype to an element and will be used in all diagrams.

Style for stereotype in custom diagram

- The custom diagram wizard allows specifying styles for stereotyped elements.

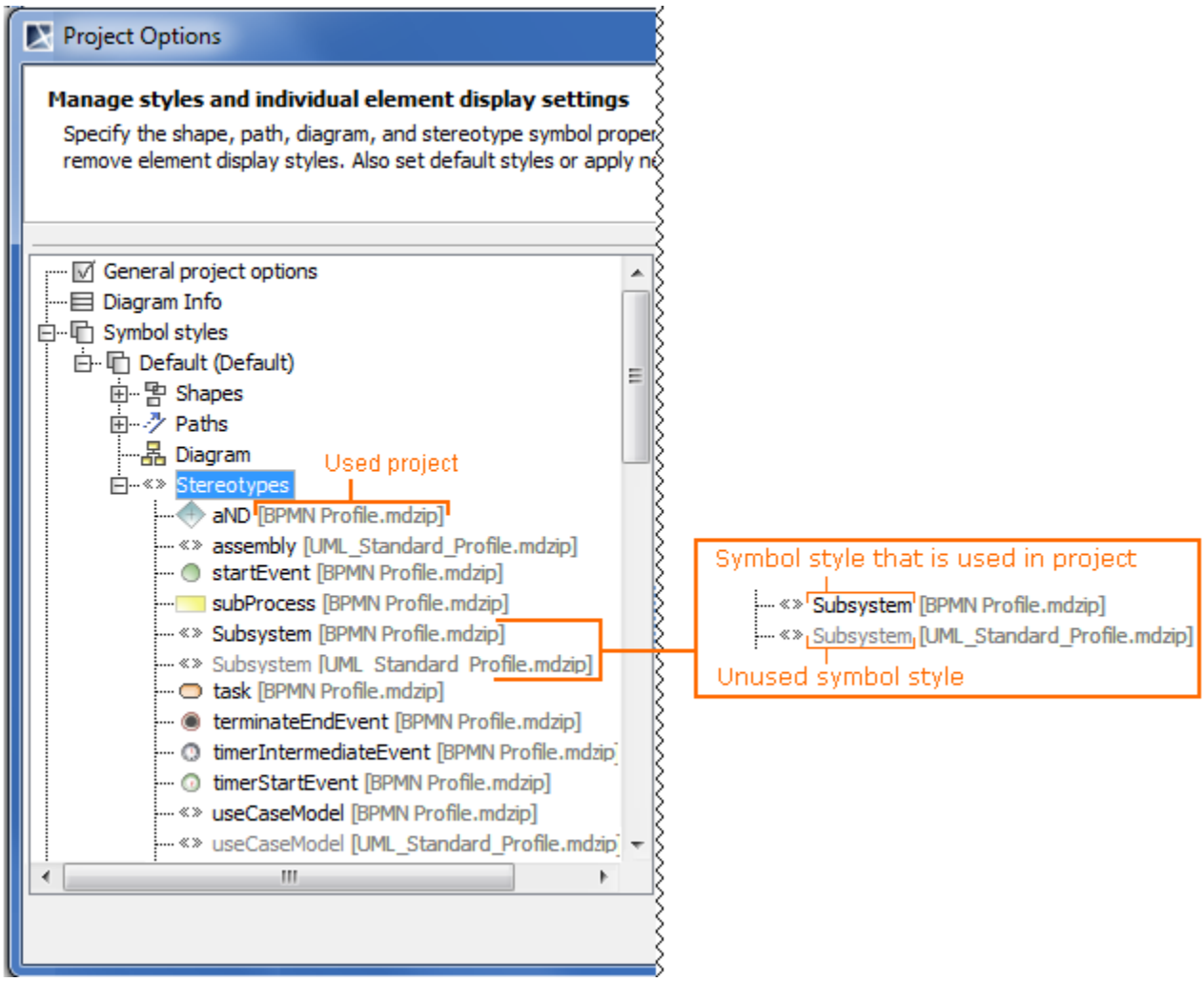
« Stereotype properties(see page 990)

Collecting symbol styles from used projects »(see page 996)

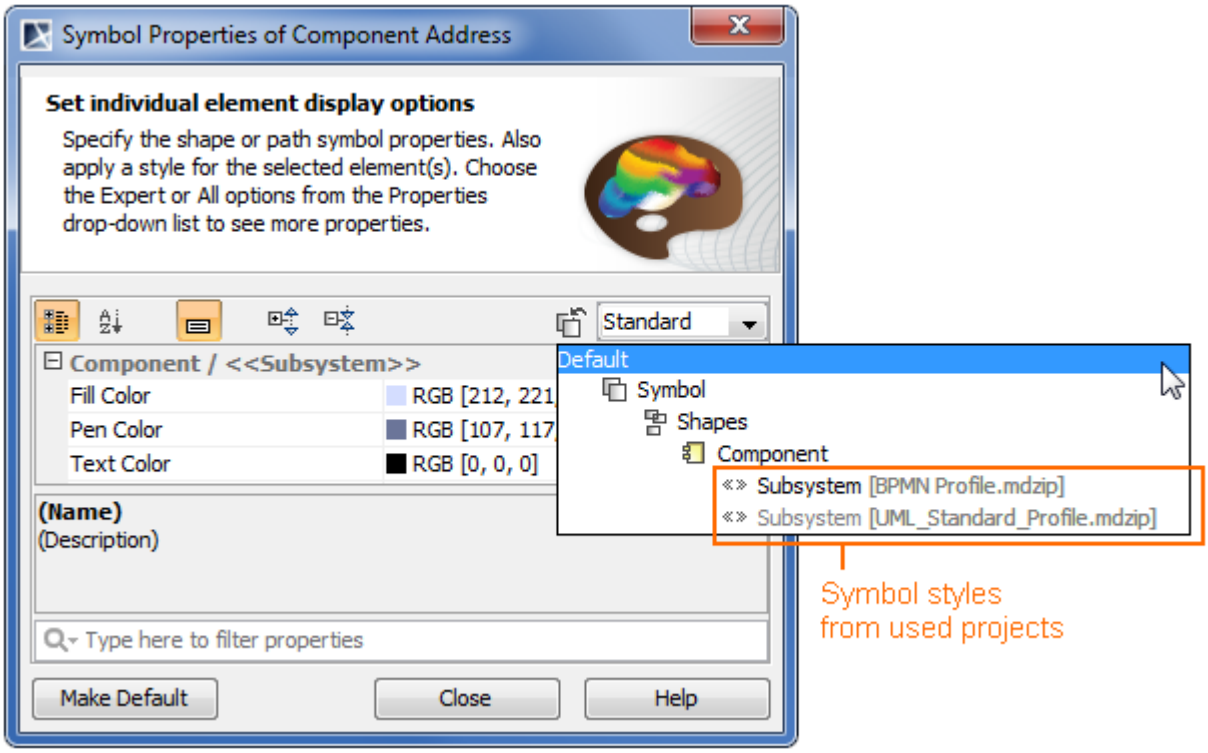
Collecting symbol styles from used projects

As of MagicDraw version 17.0.3, a used project in which the symbol style is defined is displayed next to the symbol style in the Project Options(see page 214) and **Symbol Properties** (see page 975)dialogs. This functionality also allows for copying a symbol style from a used project to the current project symbol style.

You can see the particular used project from which the symbol style is used and copy the symbol style from the used project.



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



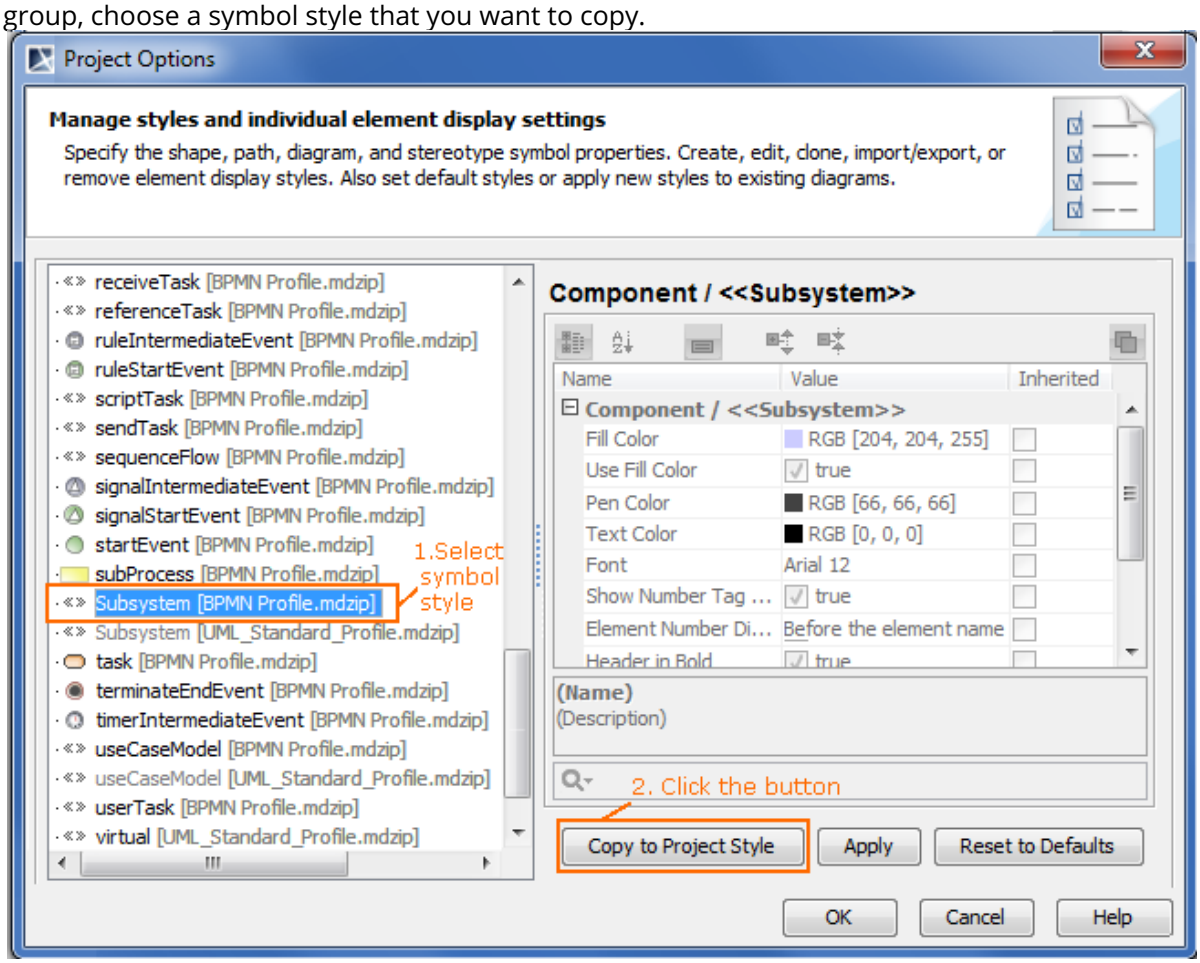
In the example, you can see that in the project there are two Subsystem symbol styles - one style from the *BPMN Profile.mdzip* used project and the other from the *UML_Standard_Profile.mdzip* used project. The Subsystem symbol style from the *BPMN Profile.mdzip* used project is used in the current project.

The symbol style from used project is not editable because the used project by itself is read-only. By copying the symbol style from the used project to the project you can create a new symbol style and then edit it.

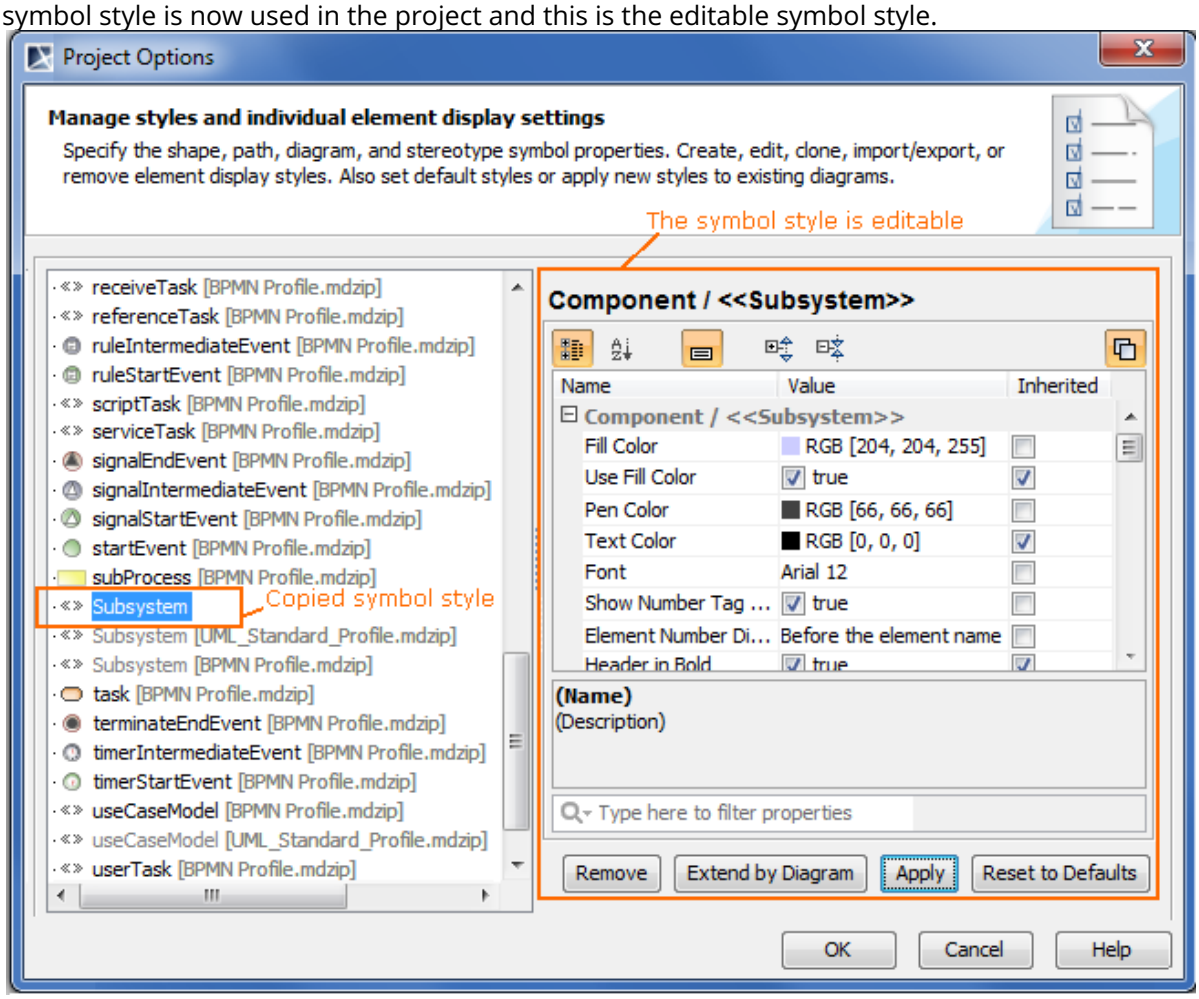
> Note that there are no rules for the symbol style usage from used projects - symbols styles are used according to used projects loading order.

To copy a symbol style to the project style by using the Project Options(see page 214) dialog

1. In the Project Options(see page 214) dialog, select **Symbol styles** > **Default** > **Stereotypes** options group, choose a symbol style that you want to copy.



2. Click the **Copy to Project Style** button. The new symbol style is created in the project. This symbol style is now used in the project and this is the editable symbol style.

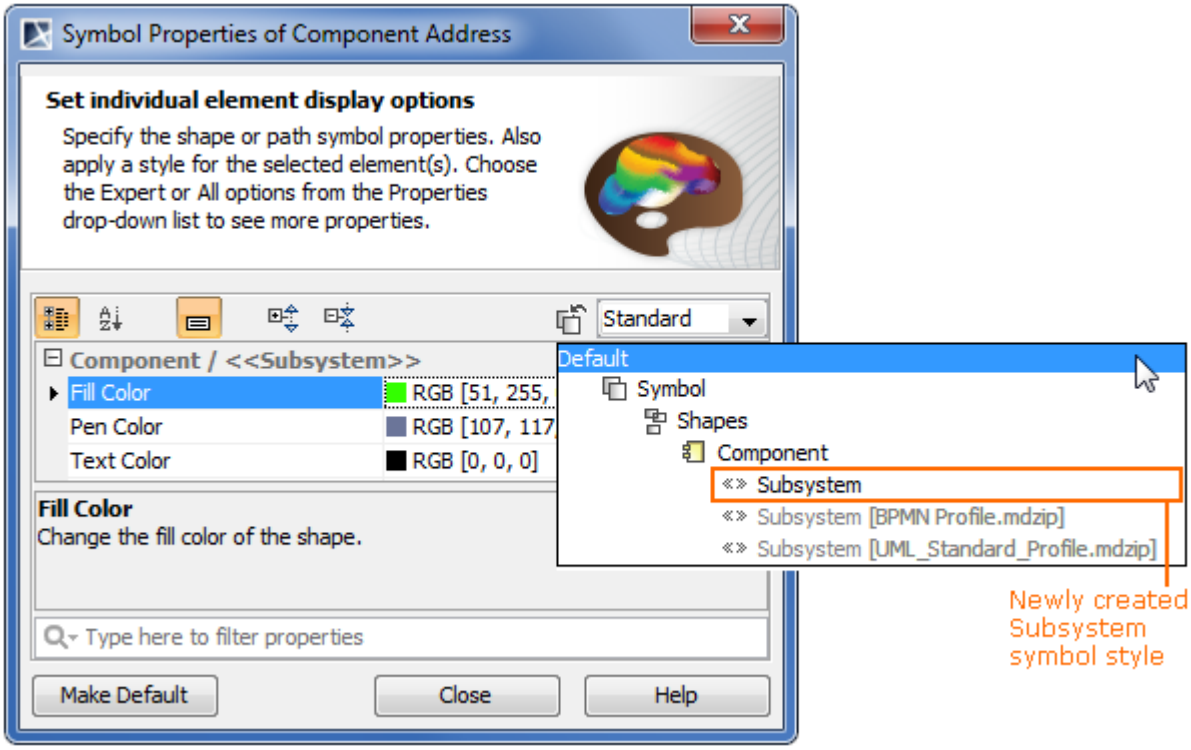


To copy a symbol style from a used project in the **Symbol Properties** dialog(see page 975)

---

1. In the **Symbol Properties** dialog(see page 975), click the Apply Style button and from the drop-down list, select a style from a used project that you want to copy.
2. Click the **Make Default** button.
3. Edit the symbol style.
4. Click **OK** to close the dialog.
5. Reopen the **Symbol Properties** dialog(see page 975). In the expanded Apply Style drop-down list, you can see the newly created symbol style. This is the copy of the selected style from the used project.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Copy and paste symbol style

A symbol style defined in symbol properties can be copied and pasted on another symbol. Copy a symbol and then select one or more other symbols of the same type where you need to paste copied properties.

To copy and paste symbol style

1. On the diagram pane, select symbol(s) which style properties you want to copy.
2. Copy style properties by doing one of the following:
   - From the **Edit** menu, select **Copy**.
   - Press Ctrl+C.
   - Click the Copy button on the diagram toolbar.
3. On the diagram pane, select the symbol(s) on which you want to paste style properties.
4. Paste style properties on the selected symbols by doing one of the following:
   - From the **Edit** menu, select **Paste Style**.
   - Press Ctrl+Shift+V.
   - Click the Paste Style button on the diagram toolbar.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

You can copy and paste symbol properties of the more than one symbol at the time. Symbol properties will be pasted according to the element type.

Related pages

Making your own default style for new projects

When you modify symbol styles in a projects, these styles will by available only in that project. However, you can make your own default symbol styles and use them for the new projects. In order to do that, you will be needing to modify the template.

To modify the template

1. From the *<MagicDraw installation directory>\templates*, open the *template.mdzip*.
2. In the **Project Options** dialog, modify symbol styles.
3. Save the template.

You can create and define your own project and use it as template.

After you define your desired symbol styles, you can use them in your new projects. All you need to do is to create project from template.

To create a project from template

1. Do one of the following:
   - From the **File** menu, select **New Project**.
   - On the main toolbar, click the **New Project** button.
   - Press Ctrl + Shift + N.
2. Select the **Project from Template** in the **Other** domain.
3. Specify name and location.
4. Select your modified template.
5. Click **OK**.

Related pages

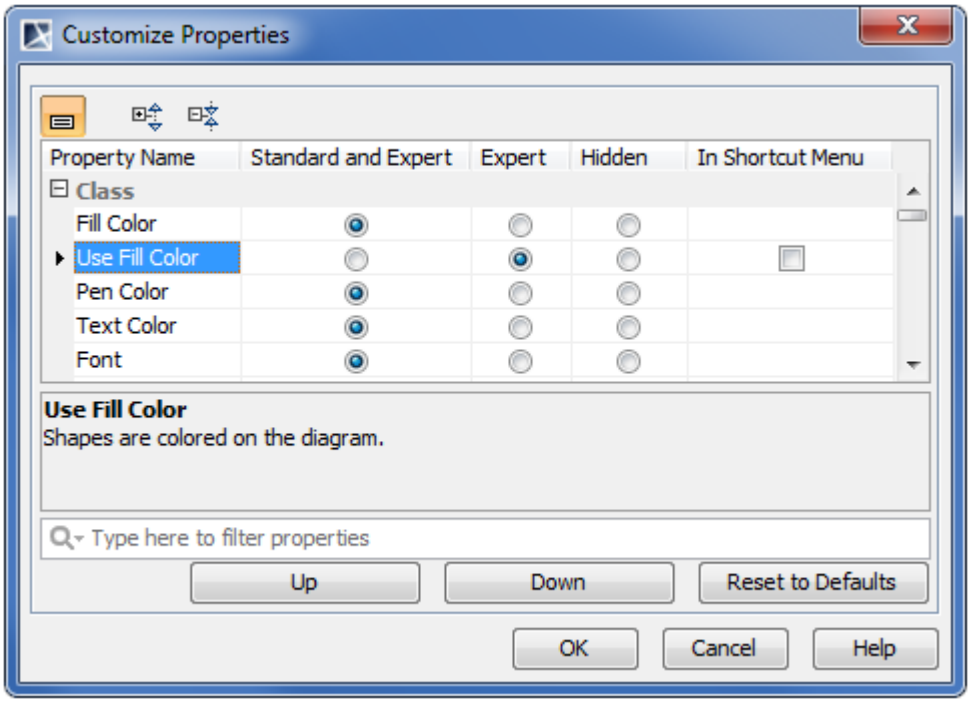
Customizing lists of properties and commands

You can customize the following:

- The list of properties in the element Specification window.
- The list of symbol properties in the Symbol Properties dialog.
- The list of properties in menus that opens after clicking the Compartments smart manipulators and also after clicking the Create Element smart manipulator.
- The list of symbol properties in the symbol shortcut menu.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

You can customize lists of properties and commands by using the Customize Properties dialog.



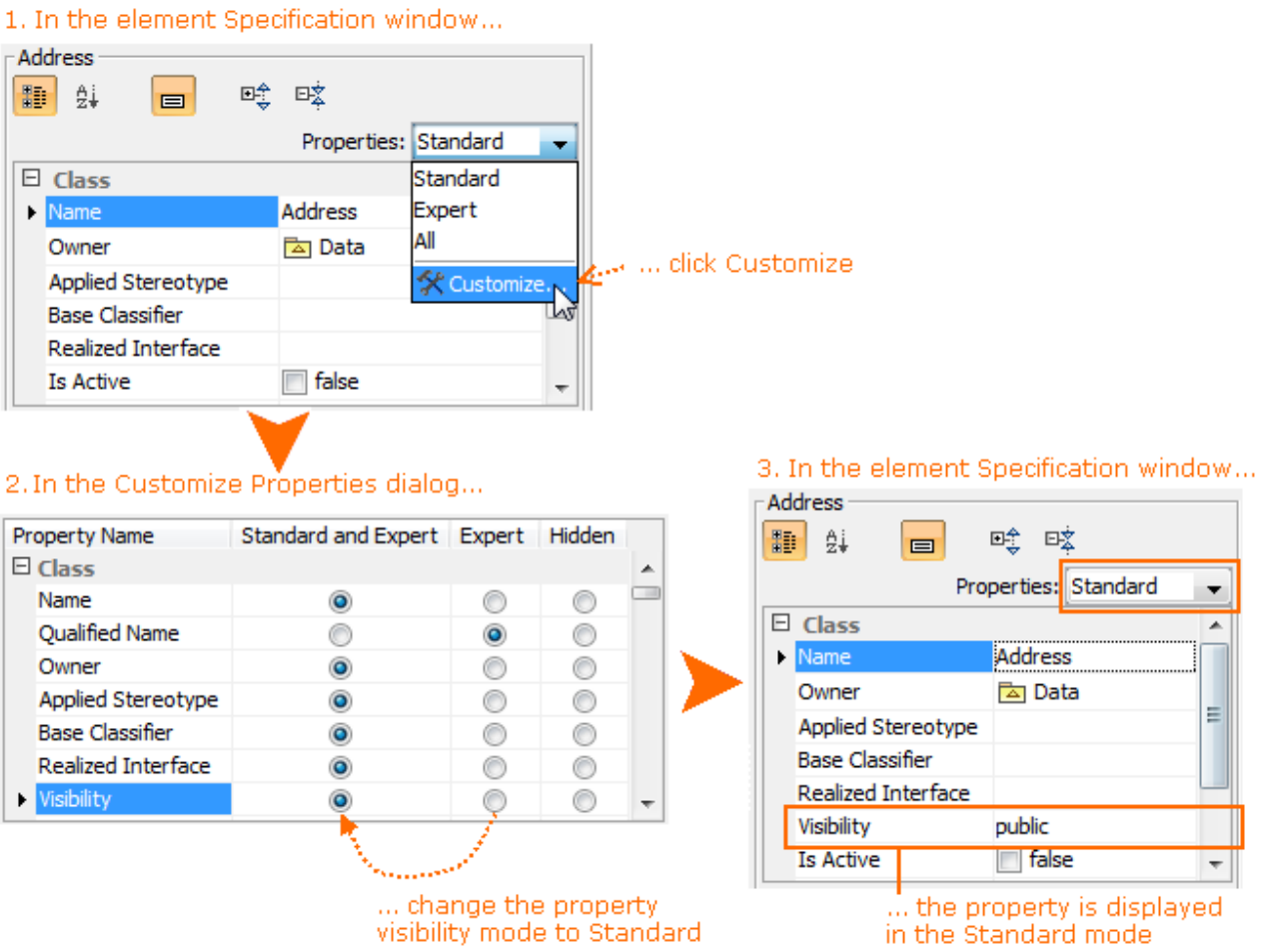
To open the **Customize Properties** dialog, do one of the following

---

- In the element **Specification window**, expand the **Properties** list and click **Customize**.
- In the Model Browser, the **Properties** > **Element** tab, at the right-top corner, click to expand the list and click **Customize**.
- On the diagram, right-click the symbol and select **Symbol Properties**. In the **Symbol Properties** dialog, at the right-top corner, click to expand the list, and click **Customize**.
- Select the symbol on the diagram. In the Model Browser, in the **Properties** > **Symbol** tab, at the right-top corner click to expand the list and click **Customize**.

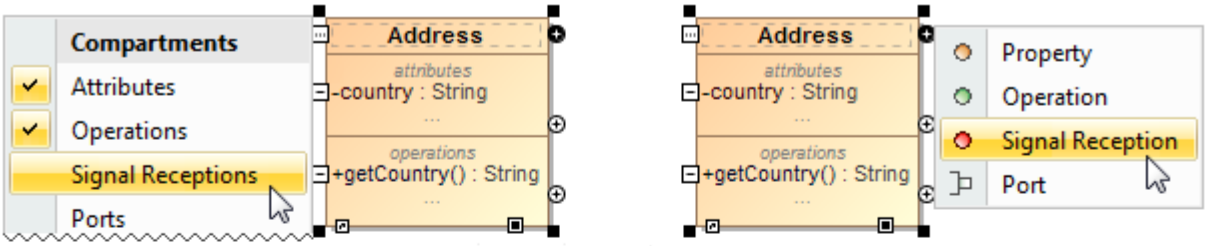
> When the **Customize Properties** dialog is open from the element Specification window(see page 913) or the **Element** tab from the Model Browser(see page 29), you can customize the list of properties in the element Specification window(see page 913).
> When the **Customize Properties** dialog is opened from the **Symbol Properties** dialog or the **Symbol** tab in the Model Browser(see page 29), then you can customize the following:
> - The list of properties in the element Specification window(see page 913).
> - The list of properties in the menus that opens after clicking the *Compartments* smart manipulator and after clicking the *Create Element* smart manipulator.
> - The list of properties in the symbol shortcut menu.

See the following examples of customizing list of properties in the element Specification window(see page 913), in the *Compartments* and *Create Element* menus, and in the symbol shortcut menu.

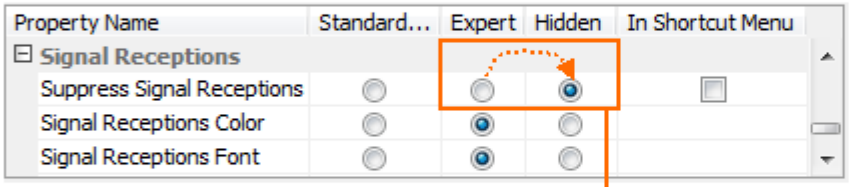


Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

1. To hide the Signal Reception property from the Compartments and Create Element menus

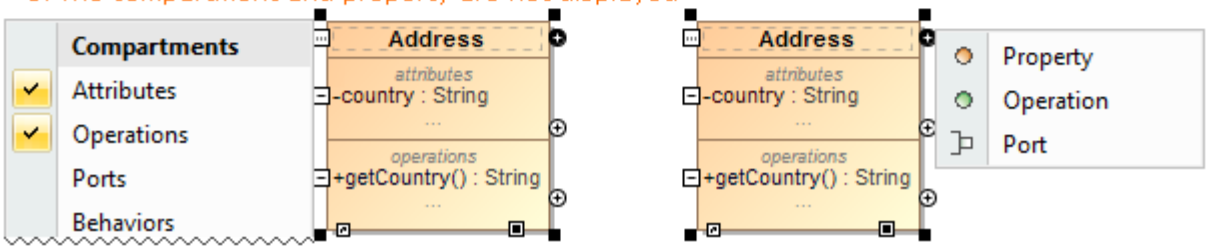


2. From the Symbol Properties dialog, open the Customize Properties dialog and ...

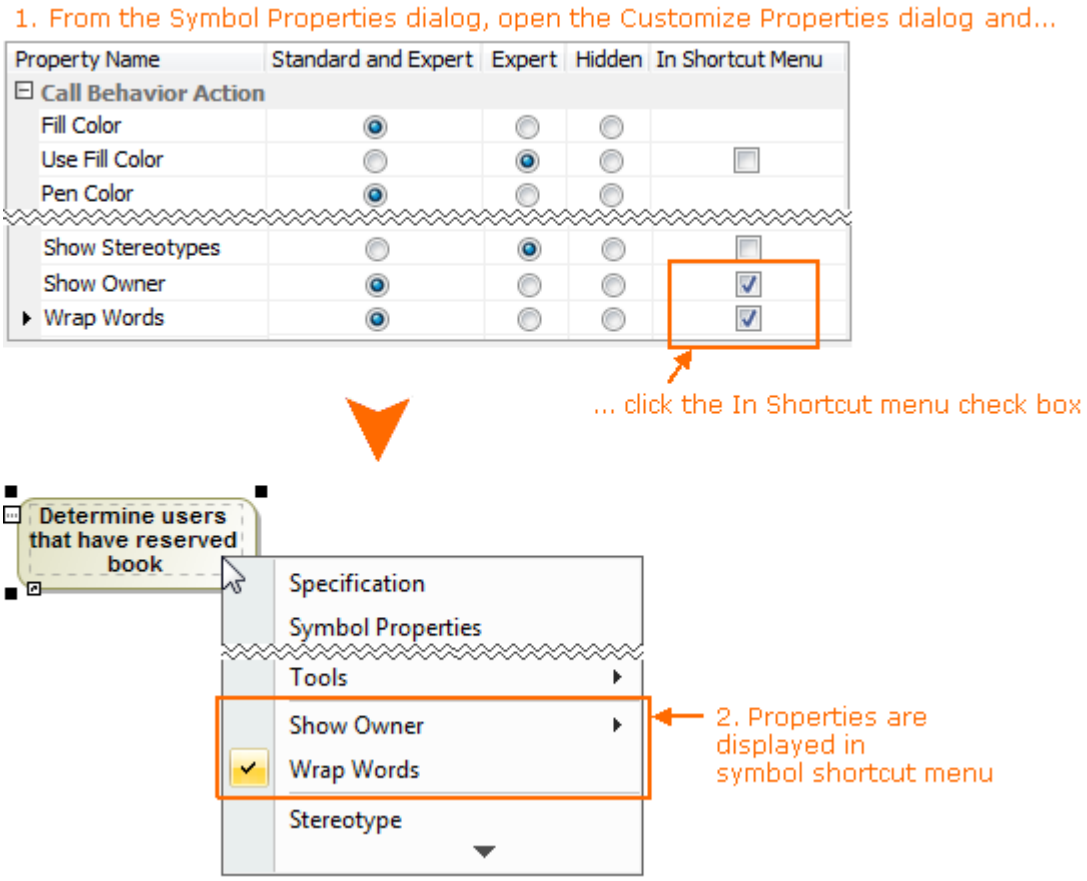


... change the mode to Hidden

3. The compartment and property are not displayed



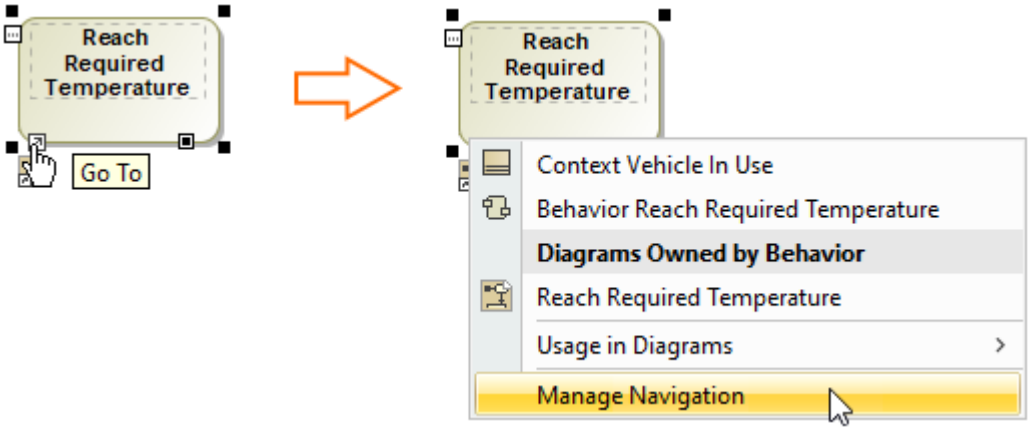
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Manage Navigation

You can define and manage navigation from the element. Here is what you can do:

- Add and manage hyperlinks(see page 1011).
- Set the active navigation target. You can choose which target should be opened when you double-click the element.
- Navigate to usages in owned diagrams and tables.
- View traceability, instances, etc.



Managing navigation

One element can have more than one target (hyperlink to web page, element, diagram, file, etc.) assigned, but you can choose which one should be set as an active target. The target which will be set as active is the one that will open when you double-click the element.

To set the target as active

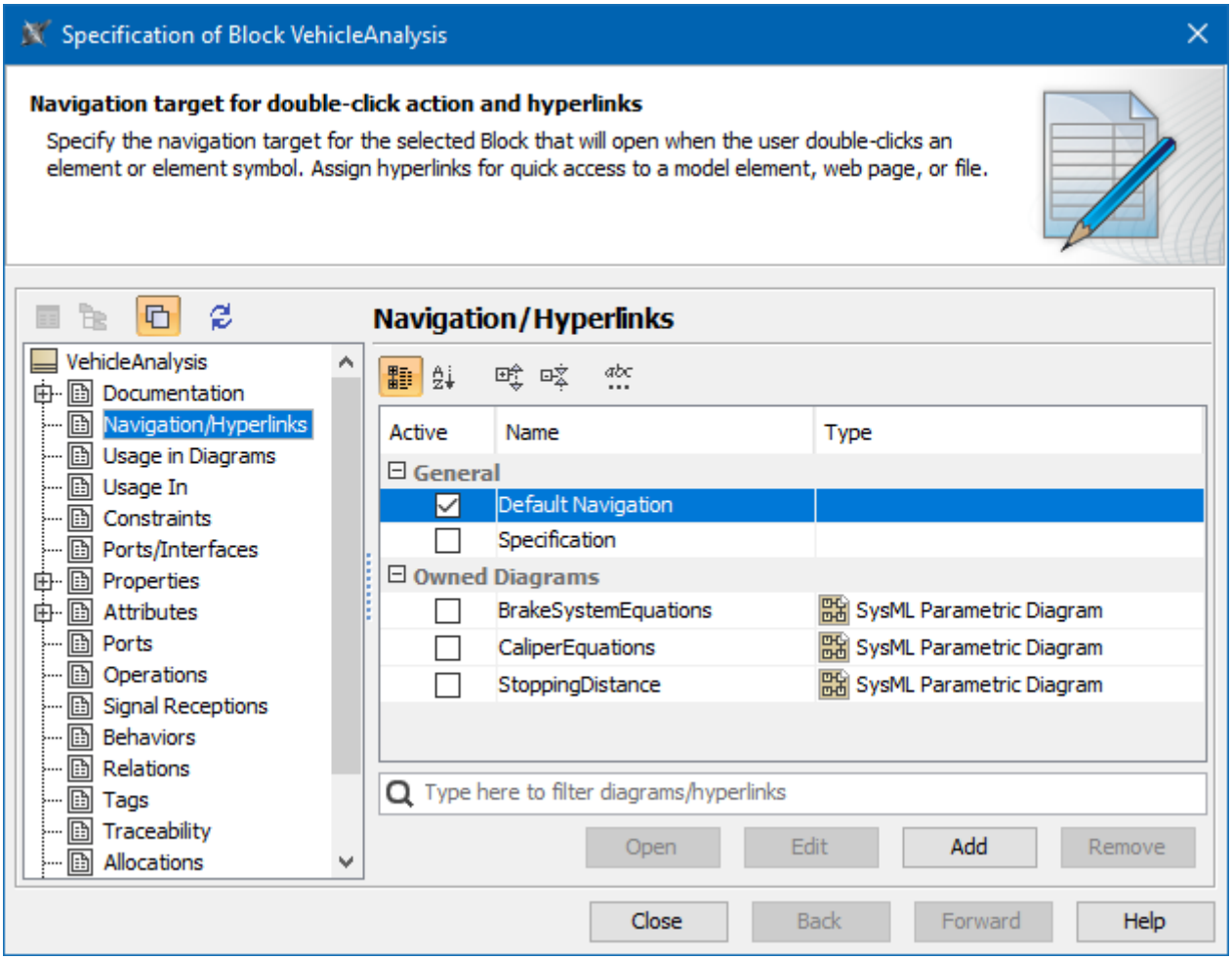
---

1. Select an element.
2. At the left bottom corner, click the Go To icon.
3. From the menu, select **Manage Navigation.** The selected element's **Specification** window opens.
4. In the **Navigation/Hyperlinks** property group, click the check box near the element which you want to set as an active target.

   ⚠ Only one element can be set as active target.

5. Click **Close** when you are done.

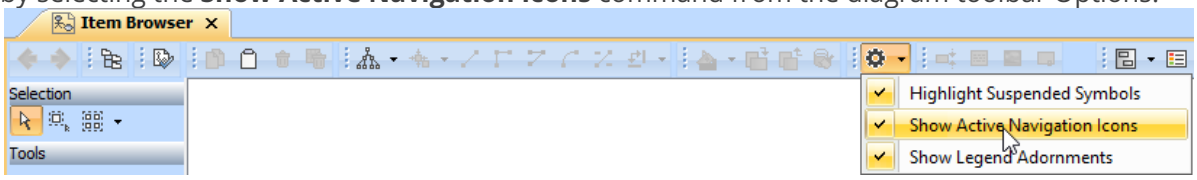
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



An example of the Navigation/Hyperlinks property group in the Specification window

If the element has an active navigation target then an active navigation target icon is show on a symbol. You can choose whether to show the icon

- by selecting the **Show Active Navigation Icons** command from the diagram toolbar Options.



- by changing a value of the Environment Option(see page 65) **Show Active Navigation Icons**. By default, this option is set to *true*.

In the **Navigation/Hyperlinks** property group, you will find groups of suggestions for the active navigation target. You can choose from the suggestions of create your own hyperlinks(see page 1008). The Default Navigation property means, that active target is the one defined by our rules. For example, for the Block the SysML Internal Block Diagram will be opened.

Related pages

- Defining hyperlinks(see page 1008)

Defining hyperlinks

On this page

- Introduction(see page 1008)
- Hyperlinks representation(see page 1009)
- Adding hyperlinks(see page 1010)

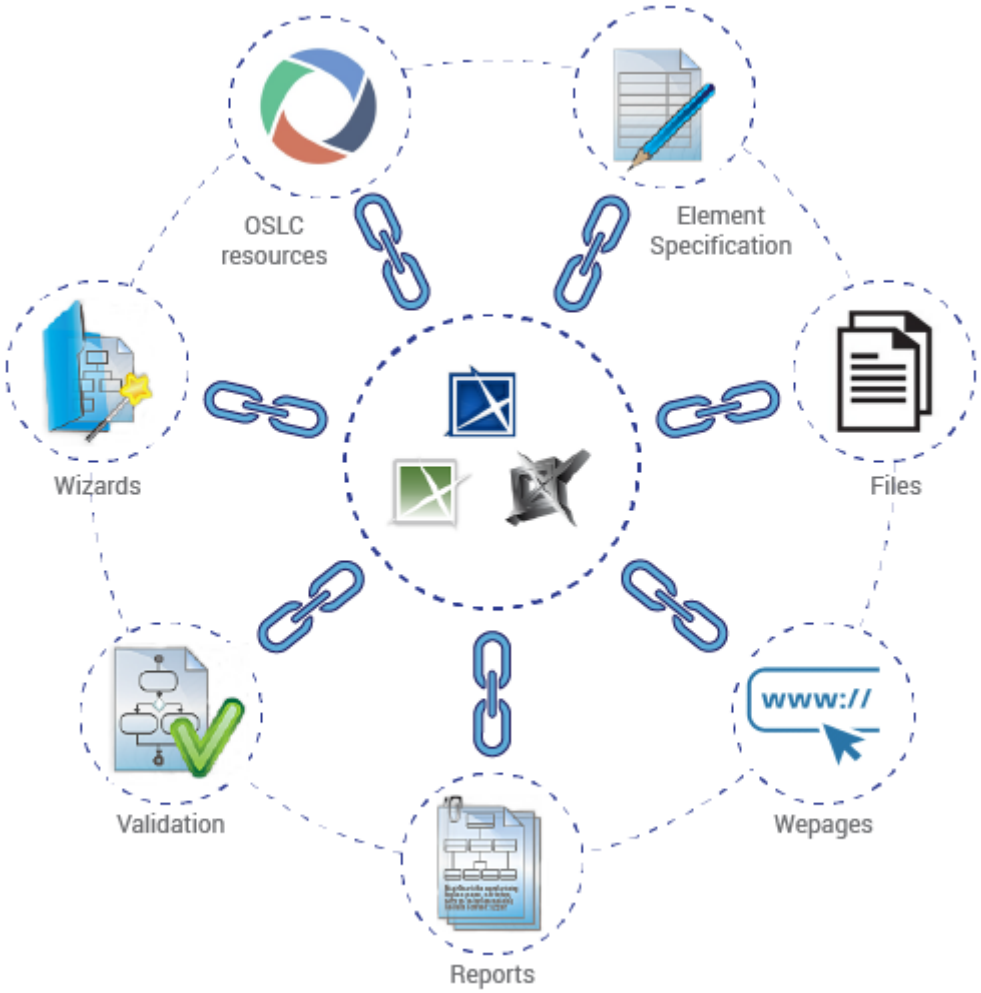
Introduction

The hyperlink is a reference to another location that you access by clicking a highlighted text or element. Hyperlinks ensure fast access of information directly from your model. You can create hyperlinks in text, on elements, or on entire diagram to link them to external or internal resources. For example, a hyperlink can open a file from your file system.The following figure illustrates all available hyperlink types.

- The hyperlinks to wizards, validation, and reports are available only if the Methodology Wizard Plugin is installed. Please see how to install the Methodology Wizard Plugin >>[162]
- The hyperlinks to OSLC resources are available only if the Cameo DataHub Plugin is installed. Please see how to install the Cameo DataHub Plugin >>[163]

162 https://docs.nomagic.com/display/NMDOC/Installing+plugins
163 https://docs.nomagic.com/display/CDH190SP4/Installation

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Hyperlink types in modeling tools.

Hyperlinks representation

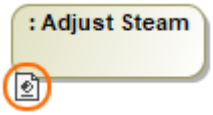
Visual representation of hyperlinks is outlined below:

- Hyperlink in text is underlined and blue. For example, OMG SysML specification .
- Hyperlink on element or diagram:
  - In the Model Browser, the

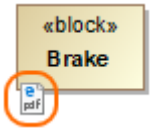
  icon appears on the element. For example, , .
  - On the diagram pane, on the symbol, the icon representing the subject to which the

hyperlink is added appears. For example:

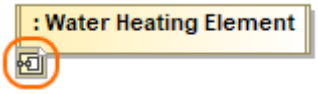
- To a webpage:

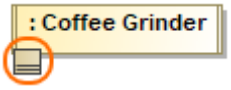


- To a file:

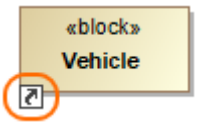


- To an element or diagram:





- To OSLC resources, wizards, validation, and reports:



> ⚠ If the element has an active navigation target then an active navigation target icon is show on a symbol. You can choose whether to show the icon by changing a value of the Environment Option(see page 65) **Show Active Navigation Icons**. By default, this option is set to *true*.

Adding hyperlinks

For more information on how to add and manage hyperlinks, please see:

- Hyperlinks extension[164]
  - Report tab[165]
  - Validation tab[166]
  - Model Wizard tab[167]
  - Model Transformation tab[168]
  - Visualizer Wizard tab[169]

- Linking model elements to OSLC resources[170]

Related pages

- Manage Navigation(see page 1005)
- Working with model elements(see page 909)

---

164 https://docs.nomagic.com/display/MWP190SP4/Hyperlinks+extension
165 https://docs.nomagic.com/display/MWP190SP4/Report+tab
166 https://docs.nomagic.com/display/MWP190SP4/Validation+tab
167 https://docs.nomagic.com/display/MWP190SP4/Model+Wizard+tab
168 https://docs.nomagic.com/display/MWP190SP4/Model+Transformation+tab
169 https://docs.nomagic.com/display/MWP190SP4/Visualizer+Wizard+tab
170 https://docs.nomagic.com/display/MD190SP4/Linking+model+elements+to+OSLC+resources

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- HTML editor(see page 1055)
- Managing hyperlinks(see page 2306)

Edit Hyperlink dialog

On this page

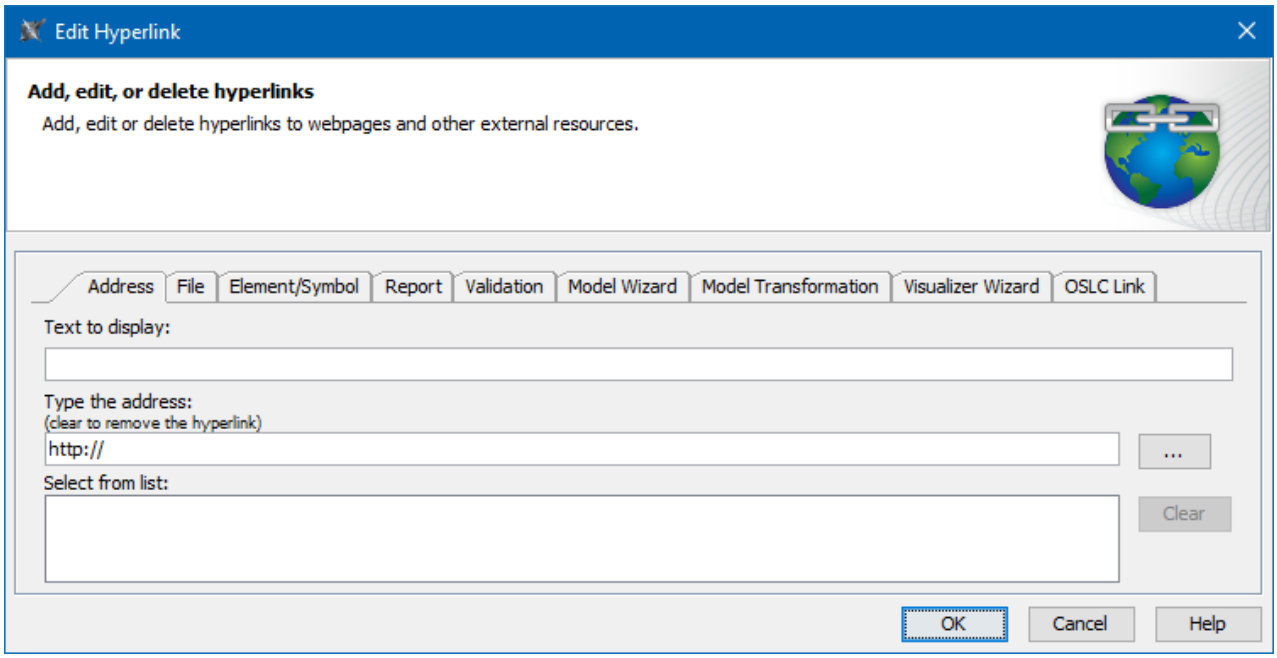
- Introduction(see page 1011)
- Hyperlink types(see page 1012)
- Opening the Edit Hyperlink dialog(see page 1013)

Introduction

You can add new or manage existing hyperlinks in the **Edit Hyperlink** dialog, consisting of:

- Three main tabs containing all modeling tools: **Address** tab, **File** tab, and **Element/Symbol** tab.
- Five tabs required to install the Methodology Wizard Plugin:[171] **Report** tab, **Validation** tab, **Model Wizard** tab, **Model Transformation** tab, **Visualizer Wizard** tab.
- One tab required to install the Cameo DataHub Plugin[172]: **OSLC Link** tab.

The following figure illustrates the tabs.



All available tabs in the Edit Hyperlink dialog.

171 https://docs.nomagic.com/display/MWP190SP4/Methodology+Wizard+Plugin
172 https://docs.nomagic.com/display/CDH190SP4/Introducing+Cameo+DataHub

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Hyperlink types

The following table provides details of available hyperlink types.

Tab in the Edit Hyperlink dialog	Hyperlink type	Required plugins	Tab descriptions
Address tab	Links to webpages or other external resources.	-	Learn more about **Address** tab >>(see page 1014)
File tab	Opens the file from your file system.		Learn more about **File** tab >>(see page 1015)
Element/Symbol tab	• In HTML text, opens the element Specification window. • In plain text, on element/symbol, or diagram, opens the diagram.		Learn more about **Element/Symbol** tab >>(see page 1017)
Model Transformation tab	Opens the Model Transformation Wizard(see page 1131) and allows you to transform your model according to a selected scope and transformation kind.	Methodology Wizard Plugin. How to install the Methodology Wizard Plugin >>[173] Learn more about hyperlinks extension >>[174]	Learn more about **Model Transformation** tab >>[175]
Model Wizard tab	Opens either the **Analysis Classes Creation Wizard** or **Use Case Model Creation Wizard** for a selected scope only if the «wizard_Capture», «wizard_Name», «wizard_Relate», «wizard_Text» stereotypes are applied on the Action (see page 629)element.		Learn more about **Model Wizard** tab >>[176]
Report tab	Opens the **Select Location** dialog to generate a selected report to your computer.	Learn more about **Report** tab >>[177]	

173 https://docs.nomagic.com/display/NMDOC/Installing+plugins
174 https://docs.nomagic.com/display/MWP190SP4/Hyperlinks+extension
175 https://docs.nomagic.com/display/MWP190SP4/Model+Transformation+tab
176 https://docs.nomagic.com/display/MWP190SP4/Model+Wizard+tab
177 https://docs.nomagic.com/display/MWP190SP4/Report+tab

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Tab in the Edit Hyperlink dialog	Hyperlink type	Required plugins	Tab descriptions
Validation tab	Starts validation of a selected validation suite and severity.	Learn more about **Validation** tab >>[178]	
Visualizer Wizard tab	Opens the Model Visualizer(see page 1346) and creates a new diagram according to a selected scope and wizard template.	Learn more about **Visualizer Wizard** tab >>[179]	
OSLC Link tab	Links to OSLC resources.	Cameo DataHub Plugin. How to install the Cameo DataHub Plugin >>[180]	Learn more about **OSLC Link** tab >>(see page 1011)

Opening the Edit Hyperlink dialog

To open the **Edit Hyperlink** dialog

- If you want to add hyperlink in text >>(see page 1021)
- If you want to add hyperlink on element or diagram >>(see page 1026)

Related pages

- Hyperlinks extension[181]
  (see page 2306)
  - Report tab[182]
  - Validation tab[183]
  - Model Wizard tab[184]

178 https://docs.nomagic.com/display/MWP190SP4/Validation+tab
179 https://docs.nomagic.com/display/MWP190SP4/Visualizer+Wizard+tab
180 https://docs.nomagic.com/display/CDH190SP4/Installation
181 https://docs.nomagic.com/display/MWP190SP4/Hyperlinks+extension
182 https://docs.nomagic.com/display/MWP190SP4/Report+tab
183 https://docs.nomagic.com/display/MWP190SP4/Validation+tab
184 https://docs.nomagic.com/display/MWP190SP4/Model+Wizard+tab

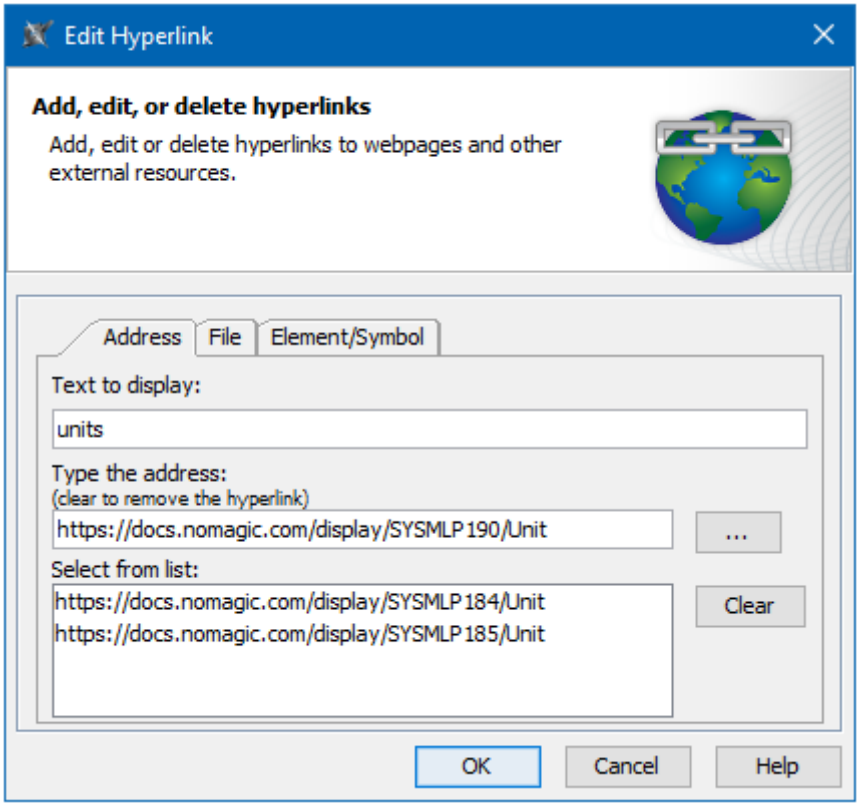
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Model Transformation tab[185]
- Visualizer Wizard tab[186]

- Linking model elements to OSLC resources[187]
- Managing hyperlinks(see page 2306)

Address tab

The **Address** tab is dedicated to adding, editing, or deleteing hyperlinks to webpages and other external resources.



The example of the Address tab in the Edit Hyperlink dialog. The text "units" links to the unit description in the NoMagic documentation.

All boxes and buttons of the **Address** tab are described in the table below.

185 https://docs.nomagic.com/display/MWP190SP4/Model+Transformation+tab
186 https://docs.nomagic.com/display/MWP190SP4/Visualizer+Wizard+tab
187 https://docs.nomagic.com/display/MD190SP4/Linking+model+elements+to+OSLC+resources

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Name	Description
Text to display box	Shows the text that will be displayed as a hyperlink. If you select the text before opening this dialog, you will see that text in this box. If you change the displayed text in this box, it will be changed in the HTML text field too. If this field is empty, you can type a text and it will be added in the HTML text field. It is available only when adding a hyperlink on text.
Type the address box	Allows you to type or paste the hyperlink address.
Select from list box	Displays all previously used hyperlinks. You can select them to fill in the **Type the address** box.
...	Opens the hyperlink in the web browser specified in the **Text to display** box.
The **Clear** button	Deletes all previously used hyperlinks from the **Select from list** box.
The **Active** check box	Activates the hyperlink if selected. It is available only when adding a hyperlink on element or entire diagram. **Active check box rule** Only one hyperlink can be active. By default the last added hyperlink is active.

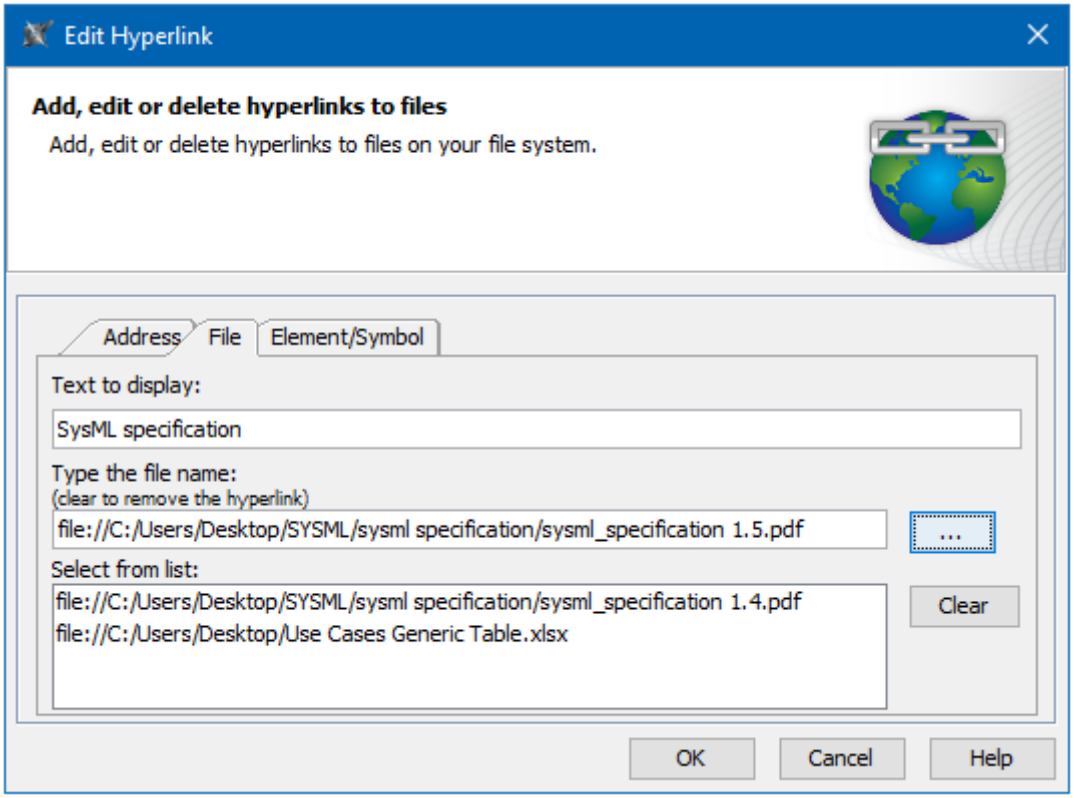
Related pages

- Hyperlinks extension[188]
  - Report tab[189]
  - Validation tab[190]
  - Model Wizard tab[191]
  - Model Transformation tab[192]
  - Visualizer Wizard tab[193]
- Managing hyperlinks(see page 2306)
- Linking model elements to OSLC resources[194]

188 https://docs.nomagic.com/display/MWP190SP4/Hyperlinks+extension
189 https://docs.nomagic.com/display/MWP190SP4/Report+tab
190 https://docs.nomagic.com/display/MWP190SP4/Validation+tab
191 https://docs.nomagic.com/display/MWP190SP4/Model+Wizard+tab
192 https://docs.nomagic.com/display/MWP190SP4/Model+Transformation+tab
193 https://docs.nomagic.com/display/MWP190SP4/Visualizer+Wizard+tab
194 https://docs.nomagic.com/display/MD190SP4/Linking+model+elements+to+OSLC+resources

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Use the **File** to add, edit, or delete hyperlinks to files on your file system.



An example of the File tab in the Edit Hyperlink dialog. The text "SysML specification" links to the SysML specification pdf document from your computer.

The table below describes the boxes and buttons found in the **File** tab.

Name	Description
Text to display box	Shows the text to be displayed as a hyperlink. If you select the text before opening this dialog, you will see that text in this box. If you change the displayed text in this box, it will be changed in the HTML text field, too. If this field is empty, you can type a text and it will be added in the HTML text field. It is available only when adding a hyperlink on text.
Type the file name box	Allows typing or pasting the directory to reference to an appropriate file, or click ... next to the box to select it from your file system.
Select from list box	Displays all previously used directories. You can select them to fill in the **Type the file name** box.

Name	Description
...	Opens the the structure of your file system.
The **Clear** button	Deletes all previously used directories from the **Select from list** box.
The **Active** check box	Activates the hyperlink if selected. It is available only when adding a hyperlink on an element or entire diagram. **Active check box rule** Only one hyperlink can be active. By default the last added hyperlink is active.

Related pages

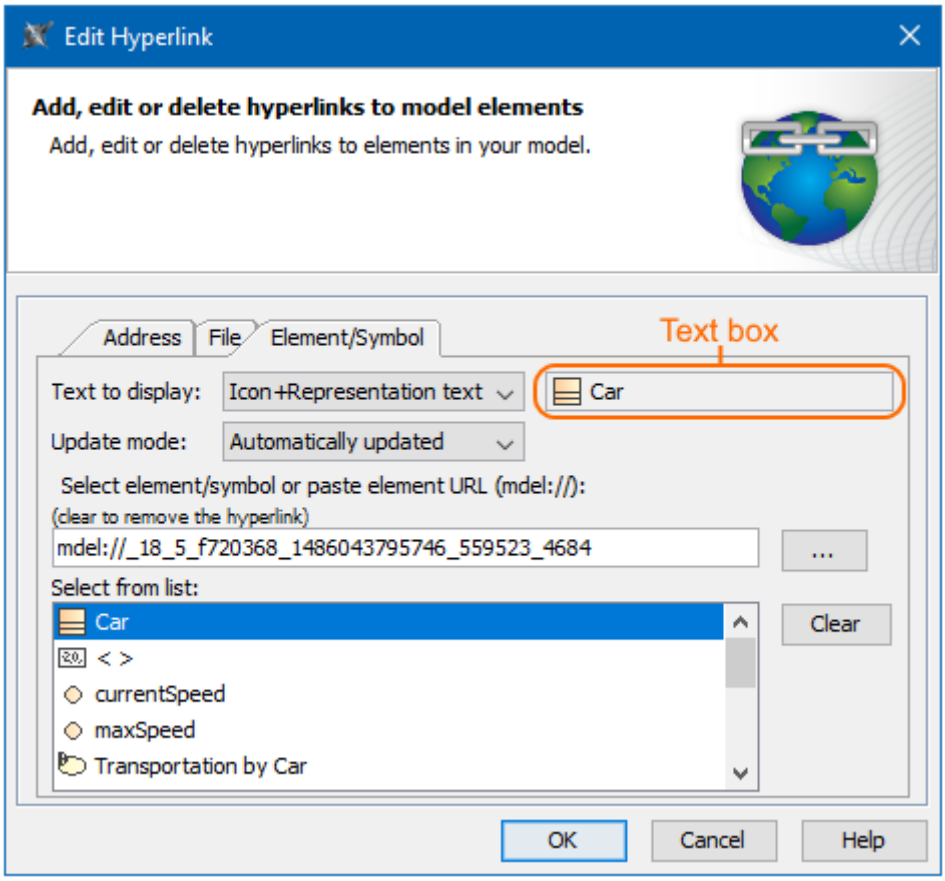
- Hyperlinks extension[195]
  - Report tab[196]
  - Validation tab[197]
  - Model Wizard tab[198]
  - Model Transformation tab[199]
  - Visualizer Wizard tab[200]
- Managing hyperlinks(see page 2306)
- Linking model elements to OSLC resources[201]

Element Symbol tab

The **Element/Symbol** tab is dedicated to adding, editing, or deleting hyperlinks to elements in your model.

195 https://docs.nomagic.com/display/MWP190SP4/Hyperlinks+extension
196 https://docs.nomagic.com/display/MWP190SP4/Report+tab
197 https://docs.nomagic.com/display/MWP190SP4/Validation+tab
198 https://docs.nomagic.com/display/MWP190SP4/Model+Wizard+tab
199 https://docs.nomagic.com/display/MWP190SP4/Model+Transformation+tab
200 https://docs.nomagic.com/display/MWP190SP4/Visualizer+Wizard+tab
201 https://docs.nomagic.com/display/MD190SP4/Linking+model+elements+to+OSLC+resources

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The Element/Symbol tab in the Edit Hyperlink dialog where the text "Car" refers to the Car Class from the model. It is represented with an icon and will be automatically updated after changes.

The table below describes the boxes and buttons of the **Element/Symbol** tab.

Name	Description
Text box (highlighted in orange in the figure)	Shows the text that will be displayed as a hyperlink. It dynamically changes according to the selected option in the **Text to display** drop-down box and the specified element/ symbol in the **Select element/symbol or paste element URL (mdel://)** box. It is available only when adding a hyperlink in text.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Name	Description
Text to display drop-down box	Allows you to select the hyperlink representation options of the specified text in the text box: • **Name**. Displays the element name from its Specification window(see page 913). • **Representation text**. Displays the full element title which is visible in Containment tree(see page 32). • **Icon+Representation text**. Displays the element icon together with the full element title which is visible in Containment tree(see page 32). • **Custom text**. Allows you to type your own text in the text box; however, then you cannot specify the **Update mode** option. It is available only when adding a hyperlink in text.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Name	Description
Update mode drop-down box	Allows you to select when the hyperlink will be updated if the referenced element and the hyperlink become inconsistent. This option depends on the option selected in the **Text to display** box and the element type specified in the **Select element/symbol or paste element URL (mdel://)** box. • **Automatically updated**. Updates the hyperlink text or the referenced element name automatically. • **Automatically checked**. Enables active validation rules that check if the hyperlink text and the referenced element are coherent. • **Manually checked**. Allows you to run validation manually to check if the hyperlink text and the referenced element are coherent. • **Do not update**. Does not update if the hyperlink text and the referenced element name become inconsistent. Validation will not find any inconsistencies. It is available only when adding a hyperlink in text. **Conditions of bidirectional update mode** If the hyperlink is inserted on text referring to LiteralString, LiteralInteger, LiteralReal, or LiteralBoolean Value Specification (e.g. default value in **Value** property), the update is bidirectional (from value to text and from text to value) when the following options are set in the **Edit Hyperlink** dialog(see page 1011) or the **Project Options** dialog(see page 1025): - the **Text to display**/**Display Mode** is set to **Representation text** or **Icon + Representation text**. - the **Update Mode** is set to **Automatically updated**, **Automatically checked**, or **Manually checked**. In all other situations, the update is only one direction: from text to value.
Select element/symbol or paste element URL (mdel://) box	Displays the unique URL of the selected element from the **Select Model Elements/Symbol** dialog.
Select from list box	Displays all previously used elements/symbols from your model. You can select them to fill in the **Select element/symbol or paste element URL (mdel://)** box.

Name	Description
...	Opens the the **Select Model Elements/Symbol** dialog where you can select an element from your model to add it as hyperlink.
The **Clear** button	Deletes all previously selected elements/symbols from the **Select from list** box.
The **Active** check box	Activates the hyperlink if selected. It is available only when adding a hyperlink on an element or entire diagram. **Active check box rule** Only one hyperlink can be active. By default the last added hyperlink is active.

Related pages

- Hyperlinks extension[202]
  - Report tab[203]
  - Validation tab[204]
  - Model Wizard tab[205]
  - Model Transformation tab[206]
  - Visualizer Wizard tab[207]
- Managing hyperlinks(see page 2306)
- Linking model elements to OSLC resources[208]

Hyperlinks in texts

On this page

- Introduction(see page 1022)
- Places to add hyperlink in text(see page 1022)
- Adding hyperlinks in text(see page 1022)
  - Adding a hyperlink in text to any hyperlink type(see page 1022)
  - Adding hyperlink in text to element/symbol(see page 1023)

202 https://docs.nomagic.com/display/MWP190SP4/Hyperlinks+extension
203 https://docs.nomagic.com/display/MWP190SP4/Report+tab
204 https://docs.nomagic.com/display/MWP190SP4/Validation+tab
205 https://docs.nomagic.com/display/MWP190SP4/Model+Wizard+tab
206 https://docs.nomagic.com/display/MWP190SP4/Model+Transformation+tab
207 https://docs.nomagic.com/display/MWP190SP4/Visualizer+Wizard+tab
208 https://docs.nomagic.com/display/MD190SP4/Linking+model+elements+to+OSLC+resources

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Introduction

You can link text directly to internal or external resources by adding hyperlinks. This enables you to open a document, wizard, diagram, or other hyperlink type(see page 1012) by clicking the hyperlink.

Places to add hyperlink in text

You can add a hyperlink in the following places:

- In all elements of the text format (e.g. Note, Anchor, Comment).
- In all element property values of the text format (e.g. **To Do**, **Documentation**, **Body**, **Body and Language**, **Text**, **Min**, **Max**).

  > You can insert a hyperlink only for LiteralString and OpaqueExpression Value Specifications(see page 899) (e.g **Default Value** property, **Value** property, and **Argument** property values).

- In all element Specification window(see page 913) > **Documentation/Hyperlinks** property groups.

Adding hyperlinks in text

Before adding a hyperlink, select the text format, either HTML or Plain text. If you select:

- HTML text - you can add any type of hyperlink(see page 1012). How to add hyperlink in text to any hyperlink type >>(see page 1022)
- Plain text - you can add a hyperlink only to an element/symbol. How to add hyperlink in text to element/symbol >>(see page 1023)

Adding a hyperlink in text to any hyperlink type
To add a hyperlink in text to any hyperlink type(see page 1012)

---

1. Select a place to add a hyperlink. All places to add hyperlinks in text >>(see page 0)
2. Click or press Ctrl+K.
3. In the **Edit Hyperlink** dialog[209], select the tab according to hyperlink type(see page 1011).
4. Specify the boxes and options in the selected tab.

   > **Descriptions of boxes and tabs**
   > Please see the following for more in-depth information about the boxes and options on each tab:
   > - **Address** tab(see page 1014).
   > - **File** tab(see page 1015).
   > - **Element/Symbol** tab(see page 1017).
   > - **Report** tab[210].
   > - **Validation** tab[211].
   > - **Model Wizard** tab[212].
   > - **Model Transformation** tab[213].

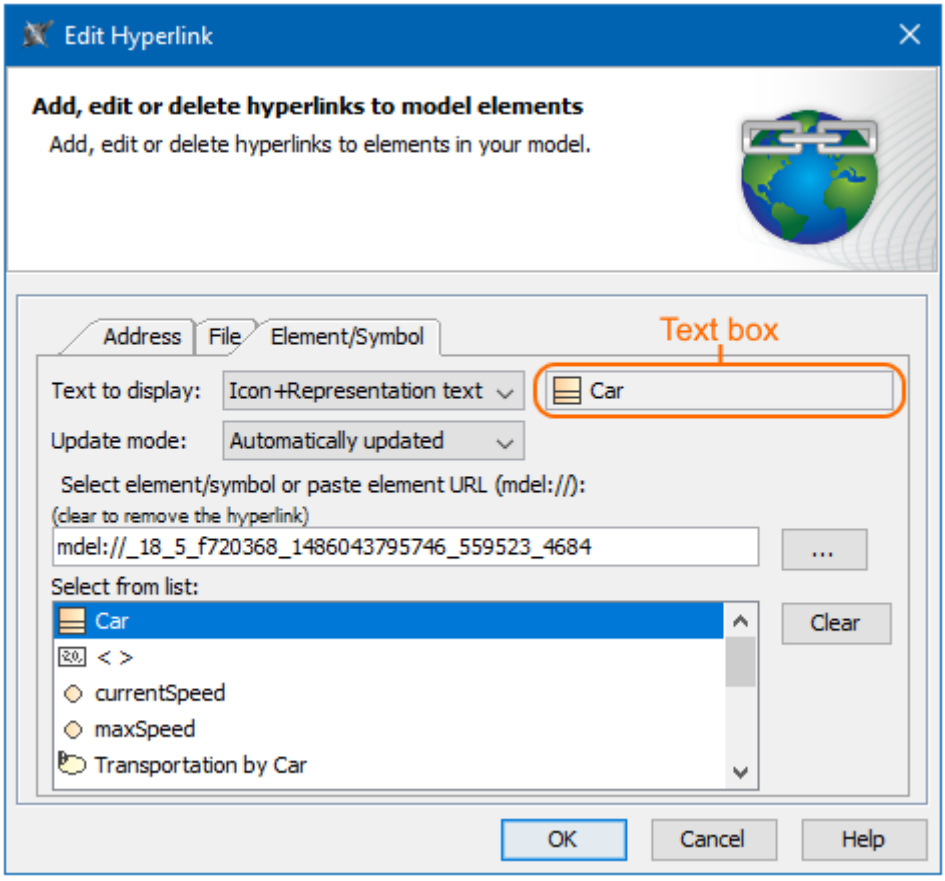
---

209 https://docs.nomagic.com/display/MD190SP4/Edit+Hyperlink+dialog_old
210 https://docs.nomagic.com/display/MWP190SP4/Report+tab
211 https://docs.nomagic.com/display/MWP190SP4/Validation+tab
212 https://docs.nomagic.com/display/MWP190SP4/Model+Wizard+tab

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- **Visualizer Wizard** tab[214].
- **OSLC Link** tab. (see page 1021)

5. Click **OK**.The hyperlink added in text becomes underlined and blue. For example, the figure below shows where the hyperlink on the text "Car" will open the Specification window of the *Car* Class. The hyperlinked text will be represented with icon and will be automatically updated after changes.



The Element/Symbol tab in the Edit Hyperlink dialog where the text "Car" refers to the Car Class from the model. It is represented with an icon and will be automatically updated after changes.

Adding hyperlink in text to element/symbol
Two additional ways to add a hyperlink in text to an element/symbol include:

- By dragging element on Note or Text box shape.(see page 1023)
- By pressing Ctrl+Space on Transition, Flow Guards, or State Invariant.(see page 1024)

Adding hyperlink to element by dragging element on Note or Text box shape
This method of adding hyperlinks in text is useful only when inserting a hyperlink to element on Note or Text box shapes.

To add a hyperlink to an element by dragging an element on a Note or Text box shape

1. In the Model Browser(see page 29) or Specification window(see page 913), select one or more elements.

214 https://docs.nomagic.com/display/MWP190SP4/Visualizer+Wizard+tab

2. Drag the selected elements onto the Note(see page 791) or the Text Box(see page 907) shapes on the diagram pane.
   The hyperlink to the element Specification window is created on the Note or Text Box. The text become underlined and blue.

   > The **Update Mode** and **Text to display** values are selected according default settings. How to change the default settings of hyperlinks to model elements >>(see page 1025)

Adding hyperlinks to elements on guard expressions and State Invariant
This method of adding hyperlinks in text is useful when inserting hyperlinks to properties on:

- A Transition Guard in a State Machine diagram.
- A Flow Guard in an Activity diagram.
- A State Invariant in a Sequence diagram.

To add a hyperlink on a guard expression and State Invariant

---

1. Select one of the following:
   - guard expression on Transition in State Machine diagram.
   - guard expression on Flow in Activity diagram.
   - State Invariant in Sequence diagram.
2. Press Ctrl+Space or Ctrl+ Backspace.

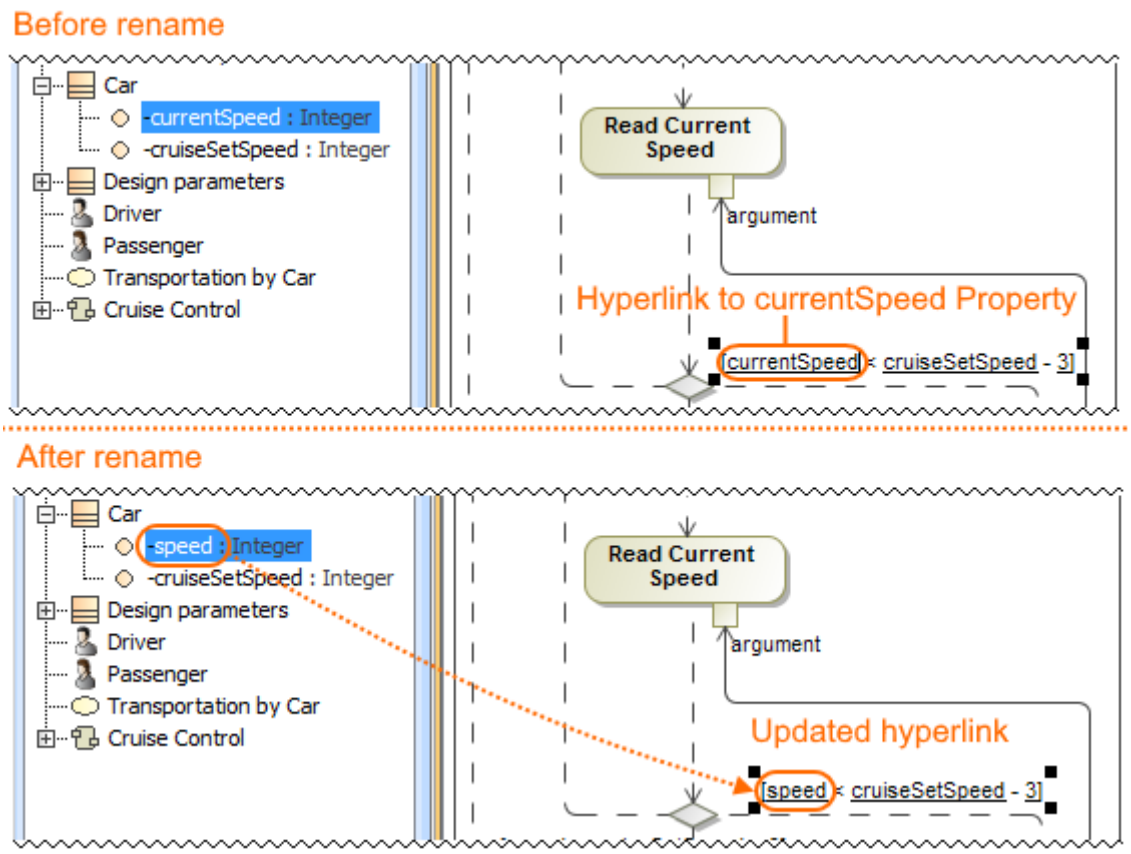
   > If you press Ctrl+K, the **Edit Hyperlink** dialog[215] opens and you can insert a hyperlink to an appropriate property. Make sure the selected property is correct.

3. Select the property from the list.

   > The list provides all parameters, structural members of behavior, and structural members of the contextual classifier of behavior.

   The hyperlinked property is added according to the default setting of the **Update Mode** and **Text to display** list boxes. How to change the default settings of hyperlinks to model elements >>(see page 1025)
   For example, in the figure below, the hyperlink to the *currentSpeed* Property is added on the guard expression. It is updated after renaming the *currentSpeed* Property to *speed*.

215 https://docs.nomagic.com/display/MD190SP4/Edit+Hyperlink+dialog_old



The hyperlink to the currentSpeed Property is added on the guard expression. It is updated after renaming the currentSpeed Property to speed in the Model Browser.

Related pages

- Hyperlinks extension[216]
  (see page 2306)
  - Report tab[217]
  - Validation tab[218]
  - Model Wizard tab[219]
  - Model Transformation tab[220]
  - Visualizer Wizard tab[221]
- Linking model elements to OSLC resources[222]
- Managing hyperlinks(see page 2306)
- HTML editor(see page 1055)

Default settings of hyperlinks to model elements

216 https://docs.nomagic.com/display/MWP190SP4/Hyperlinks+extension
217 https://docs.nomagic.com/display/MWP190SP4/Report+tab
218 https://docs.nomagic.com/display/MWP190SP4/Validation+tab
219 https://docs.nomagic.com/display/MWP190SP4/Model+Wizard+tab
220 https://docs.nomagic.com/display/MWP190SP4/Model+Transformation+tab
221 https://docs.nomagic.com/display/MWP190SP4/Visualizer+Wizard+tab
222 https://docs.nomagic.com/display/MD190SP4/Linking+model+elements+to+OSLC+resources

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

When adding hyperlinks in text to refer to model elements, the values of **Text to display box** is **Name** and **Update Mode** is **Automatically updated** are selected by default. You can change the default settings in the **Project Options** dialog.

To change the default settings of a hyperlink to model element

---

1. From the main menu, select **Options** > **Project**.
2. In the **Project Options** dialog, expand the **General** property group in the tree.
3. Select **Element References**.
4. Select the option for the value you want to change:
   - Select one of the **Default Element Hyperlink Display Mode** option values:
     - **Name** (Default). Displays the element name from its Specification window.
     - **Representation text.** Displays the full element title visible in the Containment tree.
     - **Icon+Representation text.** Displays the element icon together with the full title from the Containment tree.
     - **Custom text**. Allows you to type your own text in the text box; however, then you cannot specify the **Update mode** option.
   - Select one of the **Default Element Hyperlink Update Mode** option values:
     - **Automatically updated** (Default). Updates hyperlink text or the referenced element name automatically.
     - **Automatically checked**. Enables active validation rules which check if the hyperlink and the referenced element are coherent.
     - **Manually checked.** Allows you to run validation manually to check if the hyperlink and the referenced element are coherent.
     - **Do not update**. Does not update if the hyperlink text and the referenced element name become inconsistent. Validation will not find any inconsistencies.
5. Click **OK**.
   The default settings of hyperlinks to elements are changed.

Related pages

- Hyperlinks extension[223]
  - Report tab[224]
  - Validation tab[225]
  - Model Wizard tab[226]
  - Model Transformation tab[227]
  - Visualizer Wizard tab[228]
- Managing hyperlinks(see page 2306)
- Linking model elements to OSLC resources[229]

Hyperlinks on elements

223 https://docs.nomagic.com/display/MWP190SP4/Hyperlinks+extension
224 https://docs.nomagic.com/display/MWP190SP4/Report+tab
225 https://docs.nomagic.com/display/MWP190SP4/Validation+tab
226 https://docs.nomagic.com/display/MWP190SP4/Model+Wizard+tab
227 https://docs.nomagic.com/display/MWP190SP4/Model+Transformation+tab
228 https://docs.nomagic.com/display/MWP190SP4/Visualizer+Wizard+tab
229 https://docs.nomagic.com/display/MD190SP4/Linking+model+elements+to+OSLC+resources

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Introduction

Hyperlinked elements or diagrams allow you to quickly navigate through the model, to other resources, or between them. If an element or diagram refers to the active hyperlink(see page 1027), you can access that content by double clicking an element or an entire diagram. You can add all types of hyperlinks(see page 1012) on elements and diagrams.

Adding a hyperlink on an element or entire diagram

To add a hyperlink on an element or entire diagram

---

1. Select the element or diagram.
2. Do one of the following:
   - Click the smart manipulator **Go To** > **Manage Navigation**.
   - Open the Specification window, choose the **Navigation/Hyperlinks** property group, and click the **Add** button.
   - From the shortcut menu, choose **Go To** > **Manage Navigation**.
3. In the **Edit Hyperlink** dialog(see page 1011), select a tab according to the hyperlink type(see page 1012) you want to insert.
4. Specify the boxes and options according to the selected tab.

   The meaning of boxes and options of each tab is described in the following pages:
   - **Address** tab(see page 1014).
   - **File** tab(see page 1015).
   - **Element/Symbol** tab(see page 1017).
   - **Report** tab[230].
   - **Validation** tab[231].
   - **Model Wizard** tab[232].
   - **Model Transformation** tab[233].
   - **Visualizer Wizard** tab[234].
   - **OSLC Link** tab.(see page 1026)

   The **Text to display** box does not exist in the **Edit Hyperlink** dialog(see page 1011).

5. Select the **Active** check box to activate the hyperlink.

   Only one hyperlink can be active. By default the last added hyperlink is active.

6. Click **OK**.
   The hyperlink is added on the selected element or diagram. In the Model Browser, you will see the Hyperlinks/Go To icon on the element. On the diagram pane, on the symbol, you will see the icon representing the subject the hyperlink is added to. You can instantly open a hyperlink by double clicking an element or diagram.

You can manage hyperlinks on an element and diagram in the **Manage Hyperlinks** dialog(see page 1028).

Related pages

---

230 https://docs.nomagic.com/display/MWP190SP4/Report+tab
231 https://docs.nomagic.com/display/MWP190SP4/Validation+tab
232 https://docs.nomagic.com/display/MWP190SP4/Model+Wizard+tab
233 https://docs.nomagic.com/display/MWP190SP4/Model+Transformation+tab
234 https://docs.nomagic.com/display/MWP190SP4/Visualizer+Wizard+tab

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Hyperlinks extension[235]
  - Report tab[236]
  - Validation tab[237]
  - Model Wizard tab[238]
  - Model Transformation tab[239]
  - Visualizer Wizard tab[240]
- Linking model elements to OSLC resources[241]
- Managing hyperlinks(see page 2306)

Manage Hyperlinks dialog

On this page

- Introduction(see page 1028)
- Managing all hyperlinks in your project used on elements and diagrams(see page 1028)
- Managing hyperlinks of a specific element and diagram(see page 1029)

Introduction

The **Manage Hyperlinks** dialog is dedicated to adding, editing, or deleting hyperlinks of elements and diagrams. You can open this dialog for a specific element and diagram to manage their hyperlinks. You can also open the dialog for the entire project to manage all hyperlinks used on elements and diagrams in your model.

The procedures below explain how to manage hyperlinks:

- Introduction(see page 1028)
- Managing all hyperlinks in your project used on elements and diagrams(see page 1028)
- Managing hyperlinks of a specific element and diagram(see page 1029)

Managing all hyperlinks in your project used on elements and diagrams

To manage all hyperlinks used on elements and diagrams

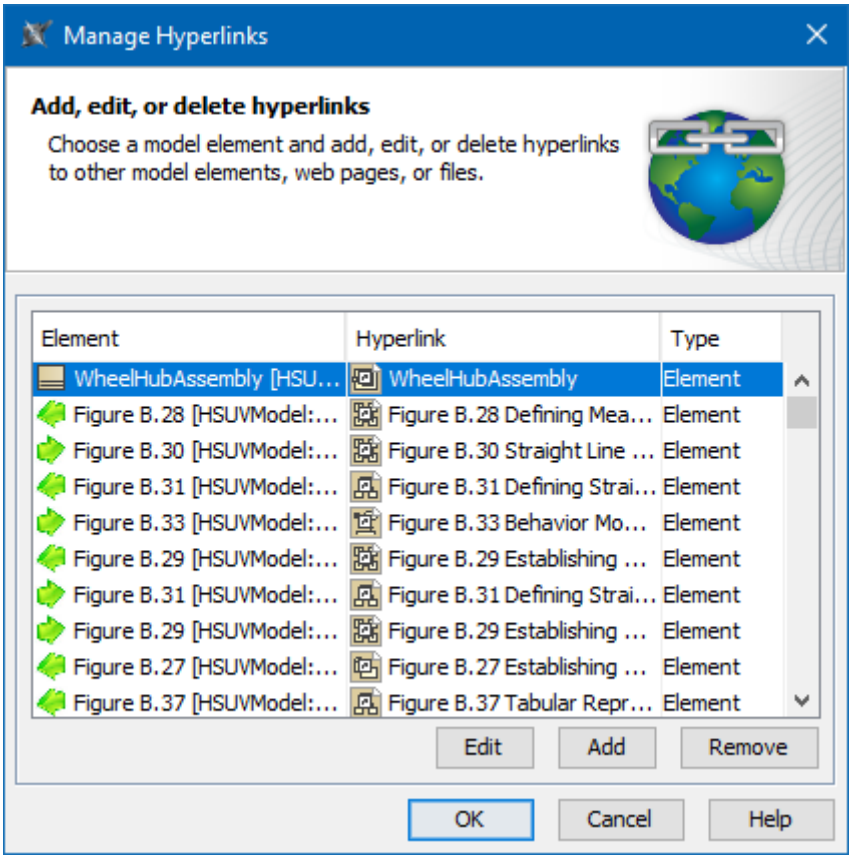
---

1. From the main menu(see page 25), select **Tools** > **Hyperlinks**.
2. In the **Manage Hyperlinks** dialog, select the hyperlink row you want to manage.
3. Manage hyperlink(s) by using the following buttons:
   - Click the **Edit** button to edit a hyperlink with the **Edit Hyperlink** dialog(see page 1011).
   - Click the **Add** button to add a hyperlink with the **Add Hyperlink** dialog.
   - Click the **Remove** button to delete the hyperlink from the **Manage Hyperlinks** dialog.
4. Click **OK**.
   The following figure shows an example of the **Manage Hyperlinks** dialog.

---

235 https://docs.nomagic.com/display/MWP190SP4/Hyperlinks+extension
236 https://docs.nomagic.com/display/MWP190SP4/Report+tab
237 https://docs.nomagic.com/display/MWP190SP4/Validation+tab
238 https://docs.nomagic.com/display/MWP190SP4/Model+Wizard+tab
239 https://docs.nomagic.com/display/MWP190SP4/Model+Transformation+tab
240 https://docs.nomagic.com/display/MWP190SP4/Visualizer+Wizard+tab
241 https://docs.nomagic.com/display/MD190SP4/Linking+model+elements+to+OSLC+resources

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The **Manage Hyperlinks** dialog of the *hybrid sport utility vehicle* project.

Managing hyperlinks of a specific element and diagram

To manage hyperlinks of specific element and diagram

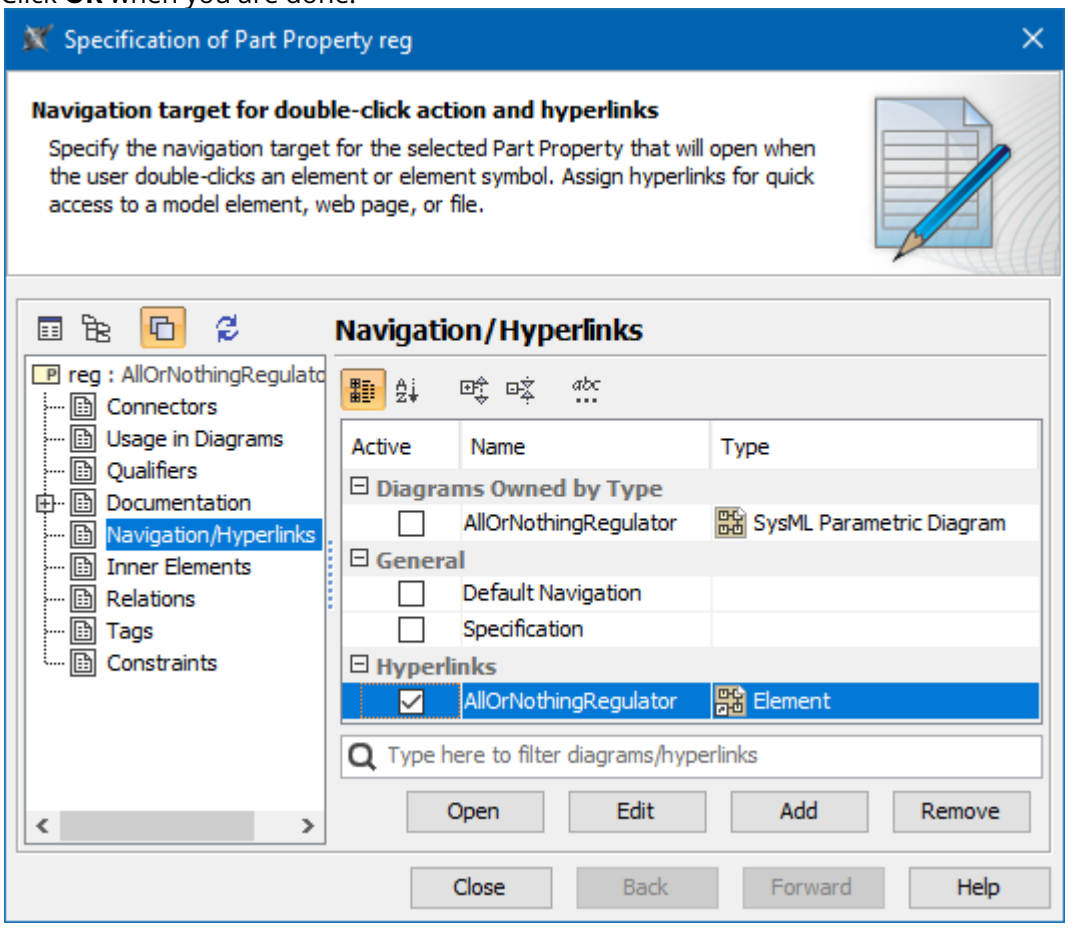
---

1. Select hyperlinked element or diagram pane.
2. Do one of the following:
   - Right click it, and select **Go To** > **Manage Navigation**.
   - Click the smart manipulator
     Go To > **Manage Navigation**.
3. The selected element **Specification window** opens. In the **Navigation/Hyperlinks** property group, manage hyperlink(s) by using the following buttons:
   - Select the **Active** check box to make the diagram or hyperlink active.

     Only one hyperlink can be active. By default the last added hyperlink is active.
   - Click the **Open** button, to open the content of the hyperlink.
   - Click the **Edit** button, to edit hyperlink with the **Edit Hyperlink** dialog(see page 1011).
   - Click the **Add** button, to edit hyperlink with the **Edit Hyperlink** dialog(see page 1011).
   - Click the **Remove** button, to delete the hyperlink from the **Manage Hyperlinks** dialog.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

4. Click **OK** when you are done.



Related pages

- Hyperlinks extension[242]
  - Report tab[243]
  - Validation tab[244]
  - Model Wizard tab[245]
  - Model Transformation tab[246]
  - Visualizer Wizard tab[247]
- Managing hyperlinks(see page 2306)
- Linking model elements to OSLC resources[248]

Opening hyperlinks

To open hyperlinks in text

242 https://docs.nomagic.com/display/MWP190SP4/Hyperlinks+extension
243 https://docs.nomagic.com/display/MWP190SP4/Report+tab
244 https://docs.nomagic.com/display/MWP190SP4/Validation+tab
245 https://docs.nomagic.com/display/MWP190SP4/Model+Wizard+tab
246 https://docs.nomagic.com/display/MWP190SP4/Model+Transformation+tab
247 https://docs.nomagic.com/display/MWP190SP4/Visualizer+Wizard+tab
248 https://docs.nomagic.com/display/MD190SP4/Linking+model+elements+to+OSLC+resources

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

1. Hover your mouse pointer at the hyperlinked text.
2. Do one of the following:
    - Click the link.
    - Press Ctrl and click the link.
      The content of the hyperlink is opened.

To open hyperlinks on elements

---

1. Select the hyperlinked element on the diagram pane or in the Model Browser.
2. Do one of the following:
    - Double click the element.
    - Click the icon which indicates that the element has a hyperlink.
    - Press and hold Ctrl and double click the element (or click the icon which indicates that the element has a hyperlink). If the hyperlink leads to a diagram, the diagram is opened in the new window.
      The content of the hyperlink is opened.

Selecting referenced element in Containment tree

You can select a referenced element in the Containment tree[249] directly from the hyperlinked text or element.

To select a referenced element in the Containment tree

---

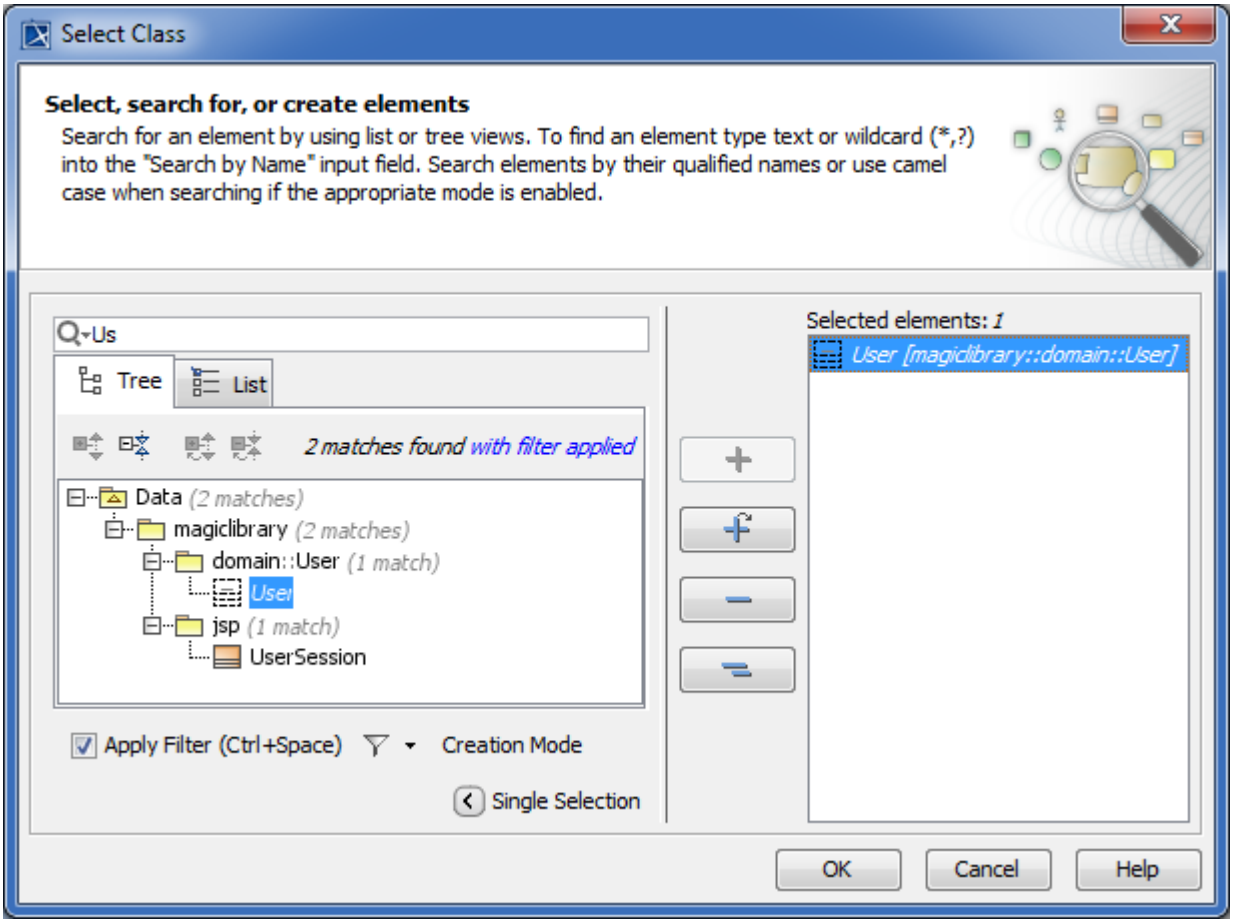
1. Select the hyperlinked element on the diagram pane(see page 24) or in the Model Browser(see page 29), or hower your mouse pointer at the hyperlinked text.
2. Do one of the following:
   - Press Alt and double click the hyperlinked text or element.
   - Press Ctrl+Alt and double click the hyperlinked text or element.
   The referenced element is selected in the Containment tree.

Selecting elements

The element Selection dialog is used for selecting elements from available contents and adding them to a certain destination scope. The title of the dialog varies depending on the way it is invoked. Depending on the content, the element Selection dialog is used for selecting an element, diagram, or owner.

249 https://docs.nomagic.com/display/SYSMLP190SP4/Containment+tab

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The element Selection dialog opens when assigning a property value - the particular element or elements. For example, the element Selection dialog opens when assigning the base classifier in the Class Specification window(see page 913).

In the element Selection dialog, you can perform the following actions:

- Select one or more elements to assign them as property values.
- Search for an element.
- Create elements.
- Clone a selected element.

Related pages

- Elements multiple selection(see page 1032)
- Searching for elements in the Element Selection dialog(see page 1033)
- Creating new elements(see page 1035)
- Cloning selected element(see page 1036)

Elements multiple selection

> The **Multiple Selection** button is available only when it is allowed to select more than one element.

To select more than one element

1. Click the **Multiple Selection** button. The *Selected elements* area appears on the right side of the element selection dialog.
2. In the model tree or list, select one or more elements at a time by holding the Ctrl or Shift keys.
3. Move selected elements to the *Selected elements* area.

To return to the single selection

- Click the **Single Selection** button.

When more than one element is selected in the tree or in the list in the multiple selection mode, the selection will be reduced to one element (the first one based on alphabetical order), when switching to the single selection mode.

Related pages

- Elements multiple selection(see page 1032)
- Searching for elements in the Element Selection dialog(see page 1033)
- Creating new elements(see page 1035)
- Cloning selected element(see page 1036)

Searching for elements in the Element Selection dialog

When searching for an element, type the search phrase in the Search By Name box. The following table describes the search types that can be used.

Search Type	Description
Simple search in the **List and Tree views**	In the **Search by Name** box, type a phrase to search for in element names. Elements matching the search phrase will be displayed in the **List** or **Tree** view. When switching between element selection views, the selected elements are remembered. That is, if the element is selected in the **Tree** view, it will be selected in the **List** view as well.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Search Type	Description
Search using wild cards	In the **Search by Name** box, type a phrase using the following wildcard characters: • "*" substitutes any range of characters. So if your search phrase is "*dd", all the elements containing "dd" in their names can be found. For example, "Address" and "ThreadDeath". • "?" substitutes exactly one character. So if your search phrase is "?dd", only the elements with "dd" in their names starting from the second character can be found. For example, "Address". "ThreadDeath" cannot be found in this case, because it contains more than one character before the search phrase "dd". **Match Text Anywhere command** Be sure the **Match Text Anywhere** command is unchecked, when using wildcard characters! Click to find the command.

To optimize the search, you can customize search options. For this, click on the **Search By Name** box. The following table describes the commands for the search customization.

Command	Description
Match Text Anywhere	• To search for matching phrase in any part of element name, click the command to add the check mark. • To explicitly search from the beginning of element name, click the command to remove the check mark.
Use Camel Case	• To search for matching phrase typed in camel case, click the command to add the check mark. • To search for matching phrase typed in regular case, click the command to remove the check mark. You can use camel case as follows: • Type only the first letters (either capitalized or not) of each word, for example, "and" or "AND" to find "Analysis and Design". • Type skipping spacers, for example, "cv3" to find "CV-3". • Perform partial search, for example, "ibd" to find SysML Internal Block diagram. Be sure the **Match Text Anywhere** command is checked!
Search in Qualified Name	• To search for matching phrase in qualified names of elements, click the command to add the check mark. • To search for matching phrase in element names only, click the command to remove the check mark.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To narrow the search scope, you can customize scope filter options. For this, click filter below the Tree or List view. Be sure the **Apply Filter** check box is selected! The following table describes the commands for the search scope filter customization.

Command	Description
Hide Uncommon Elements	Reduces the search results list by hiding uncommon elements, keeping elements such as Interface, Association Class, Class, and Component in the list.
Exclude Auxiliary Resources	Reduces the search results list by excluding elements marked as auxiliary. By default, elements in the standard/system profiles are marked as auxiliary.
Filter by Package Imports(see page 704)	Reduces the search results list by showing elements from the imported packages only.
Search in Favorites	Performs search only in your specified favorite package(s).

Creating new elements

You can create a new element directly in the element selection dialog. To create a new element, turn on the element creation mode. The Tree view is opened automatically, when the element creation mode is turned on.

To turn on the element creation mode in the element selection dialog, do one of the following

- Click the **Creation Mode** button.
- In the **Tree** tab, right-click the empty space near the model tree and from the shortcut menu, select **Creation Mode**.

To create a new element in the element selection dialog

1. In the **Tree** tab, turn the element creation mode on.
2. In the model tree, select the owner for a new element and do one of the following:
    - Click the **Create** button.
    - Right-click the selected element or empty space near the model tree and from the shortcut menu, select **Create**.
3. In the list of available elements, select the element you want to create.
4. The Specification window(see page 913) of a new element opens. Specify desired element properties and click **Close**.

Related pages

- Elements multiple selection(see page 1032)
- Searching for elements in the Element Selection dialog(see page 1033)
-

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Creating new elements(see page 1035)
- Cloning selected element(see page 1036)

Cloning selected element

In the element selection dialog, you can clone an existing element. It means you can create a new element on the basis of the selected element having the same data as the selected element.

To clone a selected element

---

1. In the **Tree** tab, click the **Creation Mode** button.
2. In the model tree, select the element which you want to clone and do one of the following:
   - Click the **Clone** button.
   - From the selected element's shortcut menu, select **Clone**.
3. The Specification window of a new element opens with all properties derived from the cloned element. By default, the number is added to the element name.
4. Change or define desired properties and click **Close** when you are done. The new element is included to the model tree.

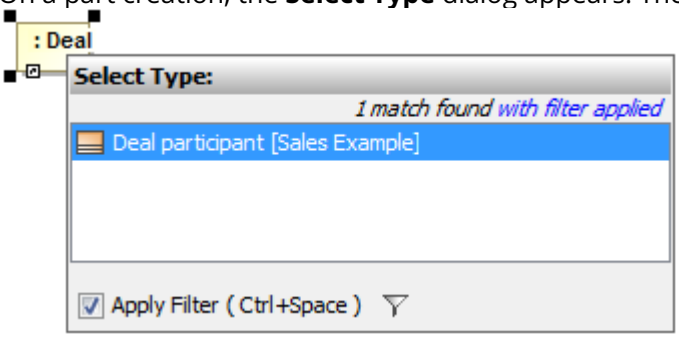
Related pages

- Selecting elements(see page 1031)
- Working with model elements(see page 909)

Selecting port and part types

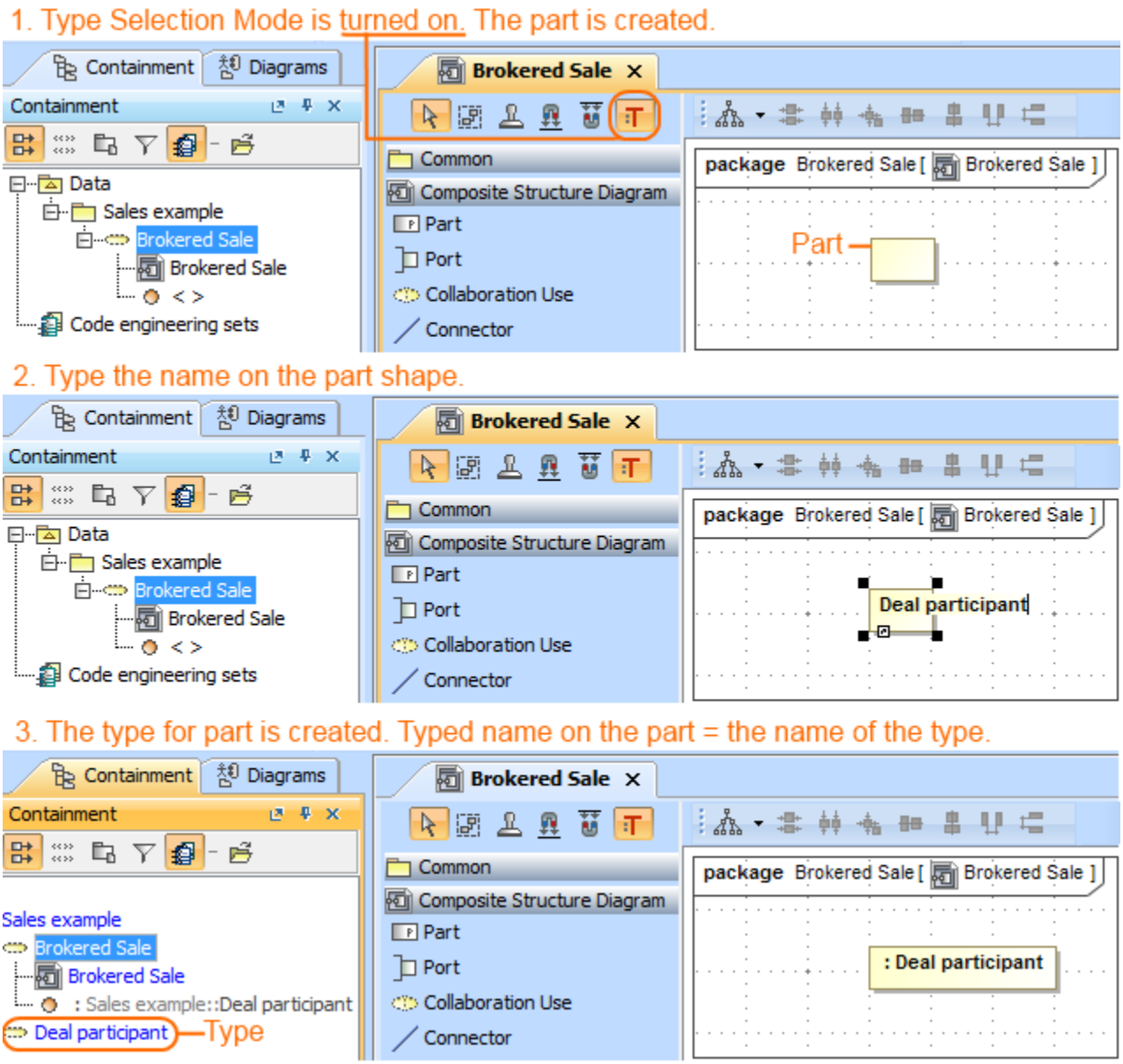
The **Type Selection Mode** turns on the automatic port and part type selection. When the function is turned on:

- On a part creation, the **Select Type** dialog appears. The same is valid when creating ports.



- If a part does not have a specified type, then on a nested port creation, a new type for the part will be created. A type for the new port will be created too.
- If a part is without name, type the name on the part shape on the diagram and the type will be created automatically. The typed name will be the name of the part type (note, that part will be without name).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



To turn on the **Type Selection Mode**

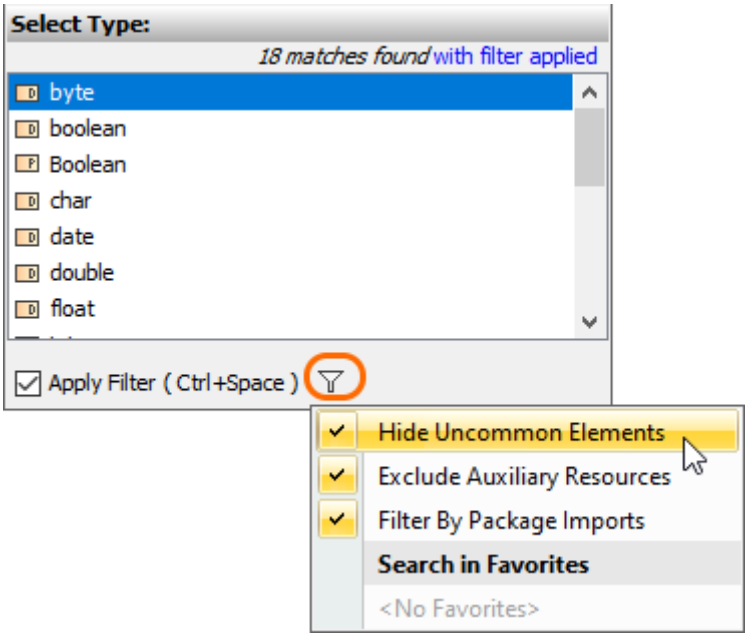
- On the Composite Structure, Class, Component, or Package diagrams palette(see page 24), click T .

  ⚠ The **Type Selection Mode** works separately for each project.

ⓘ You can save the time spent looking for elements in the type selection list by using package imports(see page 704).

Hide Uncommon Elements Filter

You can enable the *Hide Uncommon Elements* filter in the *Select Type* dialog to leave out uncommon element types and thus find the ones you need easier.

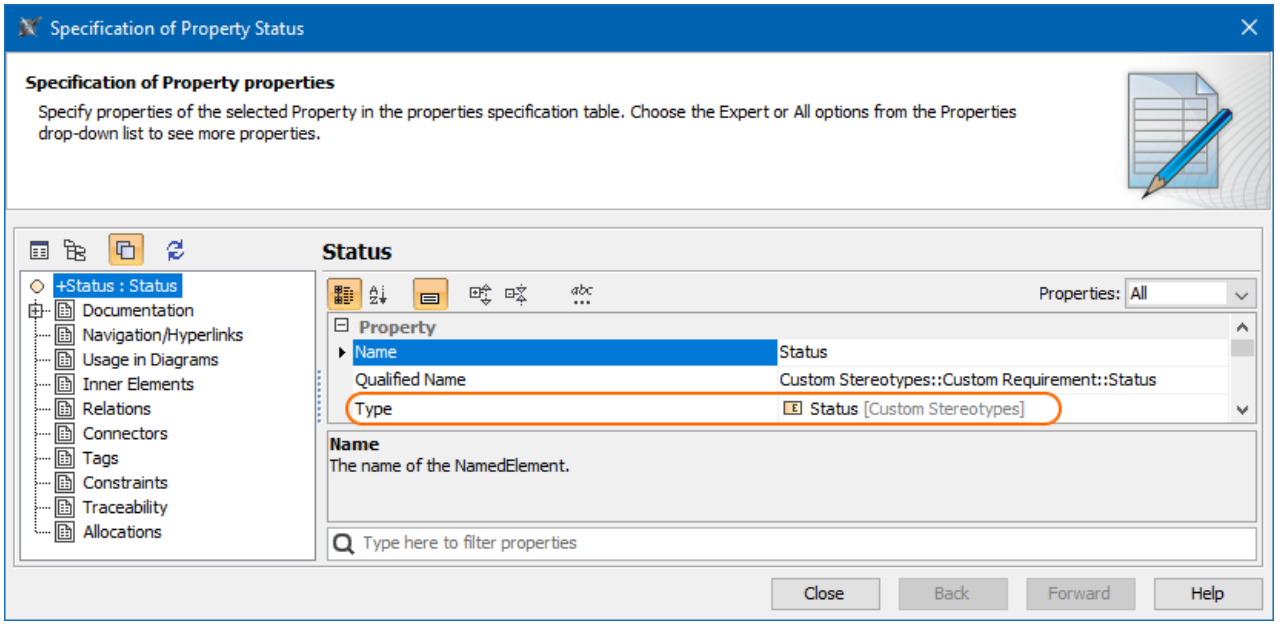


Hide Uncommon Elements filter in the Select Type dialog

Once *Hide Uncommon Elements* is enabled, the filter returns only those elements whose types are based on:

- Class (except Metaclass and Term)
- Component
- Signal
- Interface
- DataType
- Enumeration
- PrimitiveType
- AssociationClass

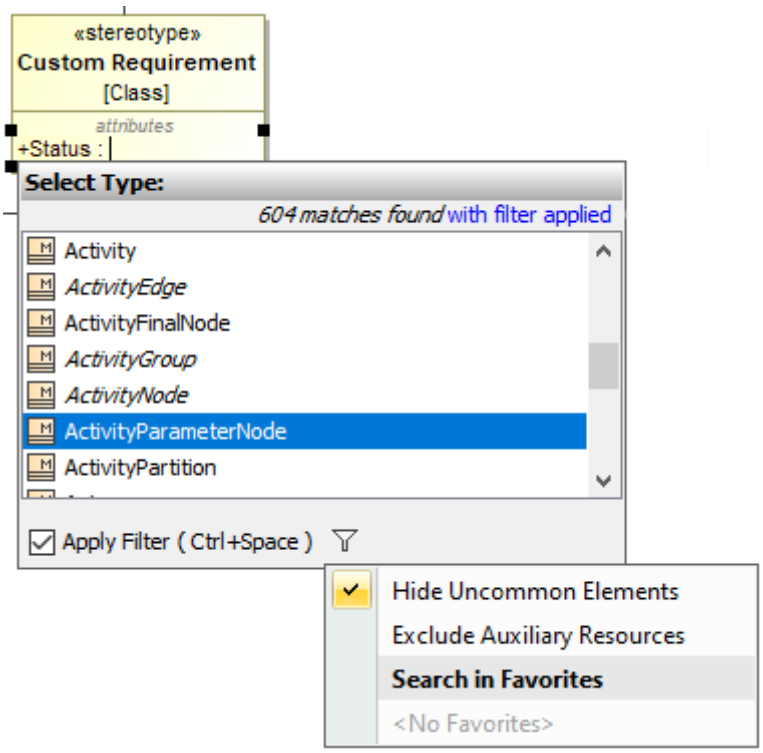
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Specified property type

If the SysML profile is loaded, only elements of *Block* and *ValueType* types are listed in the *Select Type* dialog.

If the Stereotype tag types are being specified, metaclasses are not filtered out.



Metaclasses in the *Select Type* dialog

Related Pages:

- Selecting port and part types(see page 1036)

Generic numbering mechanism

Modeling tools developed by No Magic Inc. provides you with a generic numbering mechanism that can be applied to any elements created in the modeling tool. The generic numbering mechanism allows you to:

- Automatically assign a unique number to the elements in the model when they are created, moved, or your model is refactored.
- Easily identify and find an element in the model when communicating with colleagues or stakeholders.
- Show an element's place in a hierarchy.

With this feature you can easily:

- Customize currently applied numbering formats.
- Create and apply your own numbering formats.
- Change the numbering format while creating numbered elements.

The Generic numbering mechanism feature description is organized in two sections:

- Working with Generic numbering mechanism(see page 1041) describes how to number elements, apply numbering formats, and modify numbers.
- Element Numbering dialog(see page 1046) explains GUI elements, used while numbering elements.

Before stating to describe the generic numbering mechanism, get acquainted with used concepts:

Concept	Description
Numbering customization	A customization class which specifies element types that will be numbered, numbering format that will be used to number elements, and the property to which the created number will be assigned. The numbering format should be defined prior to numbering elements. Usually the system administrator creates the numbering customization(see page 2132).
Numbering property	Indicates an element property wherein the element number will be stored and defines a numbering scheme that will be used for the element numbering.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Concept	Description
Numbering scheme	A set of rules describing what numbering formats will be applied on the concrete elements. Numbering scheme is defined while creating the numbering customization.
Prefix	An affix which is placed before the element number.
Separator	A symbol which is used to separate of the different level numbers. Separator is defined in the numbering scheme.

Related pages

- Working with generic numbering mechanism(see page 1041)
  - Automatic numbering(see page 1041)
  - Manual numbering(see page 1042)
- Element Numbering dialog(see page 1046)

Working with generic numbering mechanism

With the generic numbering mechanism you can number the elements by setting its ID. There are two element numbering modes: automatic (see page 1041)and manual(see page 1042). If you choose automatic numbering mode, a number for an element will be created automatically according to predefined numbering format. Using the manual numbering mode you can create a custom number format for the element.

Make sure that numbering customizations(see page 2132) are already created.

Related pages

Automatic numbering

Make sure that numbering customizations(see page 2132) are already created.

With an automatic numbering feature, you can number the elements of the selected type upon the element creation or modification. You can specify the number display location after the elements are numbered.

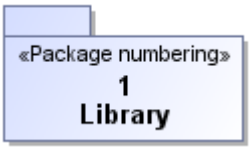
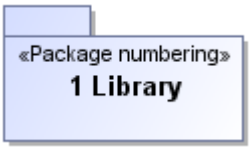
To enable automatic element numbering

1. From the main menu, select **Options** > **Project**. The **Project Options** dialog(see page 214) opens.
2. In the options group list, select or expand the **General** option and click **Numbering** either in the options list or in the options group list. The numbering properties opens.
3. Set the **Use Element Auto-numbering** property value to *true*.
4. Click **OK**.

To specify the element number display location

---

1. Do one of the following:
   - On the selected element's shortcut menu, click **Symbol Properties**. The **Symbol Properties** dialog opens.
   - From the main menu, select **Options** > **Project**. Then, in the opened **Project Options** dialog(see page 214), select the **Symbol styles** > **Default** group.
2. Select one of the following **Element Numbering Display Mode** property value:

Property value	Visualization on the element shape
Above the element name	
Before the element name	
Do not display on symbol	

3. Click **OK** after you have selected.

> You can edit automatically applied element numbers(see page 1042).

Manual numbering

Using the manual numbering, you can:

- Set the ID for one or more elements.
- Increase, decrease, or change a number for an element.
- Add a prefix for the element number.
- Change a numbering format.
- Update the last ID.

To number the element

---

1. On the selected element's shortcut menu, click **Element Numbering**. The **Element Numbering** dialog opens.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

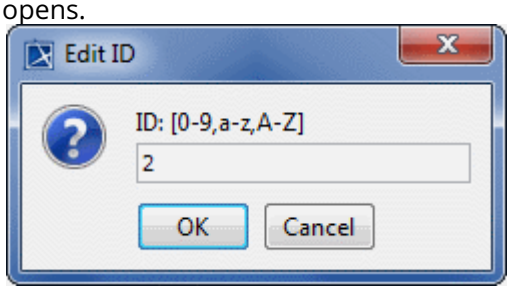
2. In the element list on the right side of the dialog, select an element you want to number.
3. Click **Create**.

To change the element's numbering symbol

---

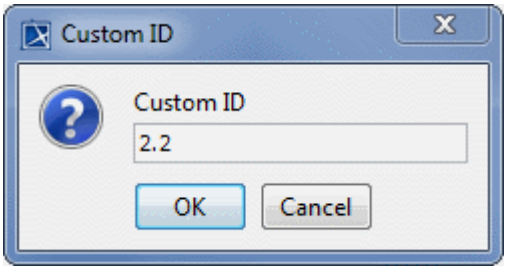
1. Open the selected element's shortcut menu and click **Element Numbering**. The **Element Numbering** dialog opens.
2. Click the **Edit** button.

   - If you need to change just the last number of the element ID, click **Edit**. The **Edit ID** dialog opens.



   > You can enter only the number, if the selected element's numbering value is numerical.
   > You can enter only a new literal symbol, if the selected element's numbering value is a literal symbol.

   - If you need to change the whole element ID (including the prefix), click **Set Custom ID**. The **Custom ID** dialog opens.



3. Enter a new numbering symbol.
4. Click **OK**.

To remove the element's number

---

1. Open the selected element's shortcut menu and click **Element Numbering**. The **Element Numbering** dialog opens.
2. Select the element from which you want to remove the number and click the **Remove** button.
3. You can select either to remove the number only from the selected element or to remove numbers from all elements owned by the selected elements. Do one of the following:

   - Click the **Remove** button. Only the selected element's number is removed.
   - Click the **Remove Recursively** button. Numbers are removed from the selected element and all elements owned by the selected element.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

4. Click **OK** when you are done.

To increase the element number

1. Open the selected element's shortcut menu and click **Element Numbering**. The **Element Numbering** dialog opens.
2. Select the element which number you want to increase.
3. Click the **Increase** button. The element number is increased by one.
4. Click **OK** when you are done.

To decrease the element number

1. Open the selected element's shortcut menu and select **Element Numbering**. The **Element Numbering** dialog opens.
2. Select the element which number you want to decrease.
3. Click the **Decrease** button. The element number is decreased by one.

   The **Decrease** button is available, when the selected element's number is not the first in the list.

4. Click **OK** when you are done.

To renumber the elements

1. On the selected element's shortcut menu, click **Element Numbering**. The **Element Numbering** dialog opens.
2. Click **Details** to extent the dialog with the renumbering commands.
3. You can select either to renumber only elements listed in the element list or to renumber elements listed in the element list including their owning elements. Do one of the following:

   - Click the **Renumber** button. All the elements in the elements list are renumbered successively.
   - Click the **Renumber Recursively** button. All the elements and elements owned by the selected element are renumbered successively.

4. Click **OK** when you are done.

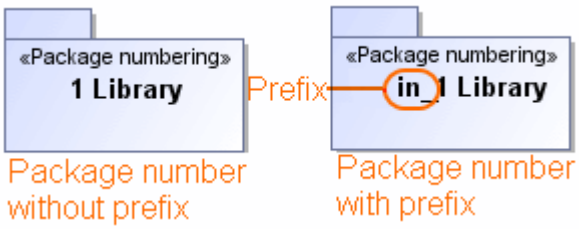
If the **Renumber from Initial Value** is selected, the whole element number changes. Otherwise just the number of the selected level changes.

To add a prefix to the element number

1. On the element owner's shortcut menu, select **Element Numbering**. The **Element Numbering** dialog opens.
2. In the **Prefix** column, click the appropriate cell (according to the used numbering property and numbering scheme).
3. Type the prefix. It can be any symbol.
4. Click **OK**.

The following example illustrates how numbers with prefixes and without prefixes are depicted on the element shape.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



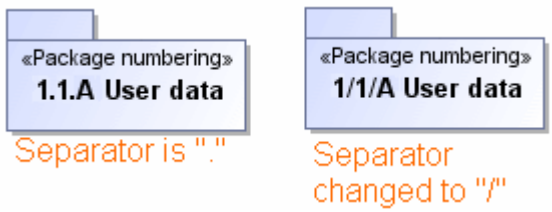
To change a separator in the element number

1. Open the element owner's shortcut menu and click **Element Numbering**. The **Element Numbering** dialog opens.
2. In the **Separator** column, click the appropriate cell (according to the used numbering property and numbering scheme).
3. Type the separator symbol.

   A new separator will override the earlier predefined separator.

4. Click **OK**.

The following example illustrates the number separator changing from "." to "/".



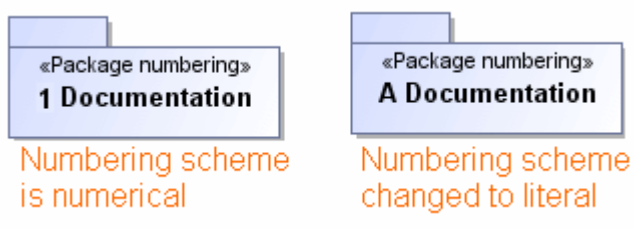
To change a numbering scheme

1. On the selected element's shortcut menu, click **Element Numbering**. The **Element Numbering** dialog opens.
2. In the appropriate **Numbered property** row, click the **Numbering Scheme** cell to open the list of the defined numbering schemes.
3. Select the desired numbering scheme in that list.
4. Click **OK**.

When you change the numbering scheme for the selected element, the new numbering scheme is applied to all elements owned by the selected element.

The following example illustrates how numbers are depicted on the element shape after changing the numbering scheme from numerical to literal.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



To update the last ID

1. Right-click an element numbered in the sequence, whose last ID you want to update.
2. From the shortcut menu, select **Element Numbering**. The appropriate dialog opens.
3. Click **Details** and then click the **Update Last ID** button.
4. Wait while the message informs you that the last ID is set to the particular value. This value is the highest ID in the sequence.
5. Click **OK** to close the **Element Numbering** dialog.

As a result, all subsequently created elements are numbered continuing from the the highest ID. This is especially useful after you delete one or more elements from the end of the numbered sequence. In such cases, the last ID remains higher than it is expected to be, causing a numbering gap in the sequence. Updating the last ID prevents the emergence of that gap.

However, if a numbering gap is not at the end of the sequence, it remains even after the update of the last ID.

For better understanding the functionality, let's study a few cases.

1. If you have a sequence from 1 to 5, and delete elements with ID = 4 and ID = 5, the ID of a subsequently created element by default is 6. But if you update the last ID in advance, the one of the subsequently created element is 4 as it should be.
2. If you have a sequence from 1 to 3 and from 5 to 6, the last ID after the update is 6. As you can see, ID = 4 is skipped, because it isn't the highest ID in the sequence; the highest ID is 6.

Element Numbering dialog

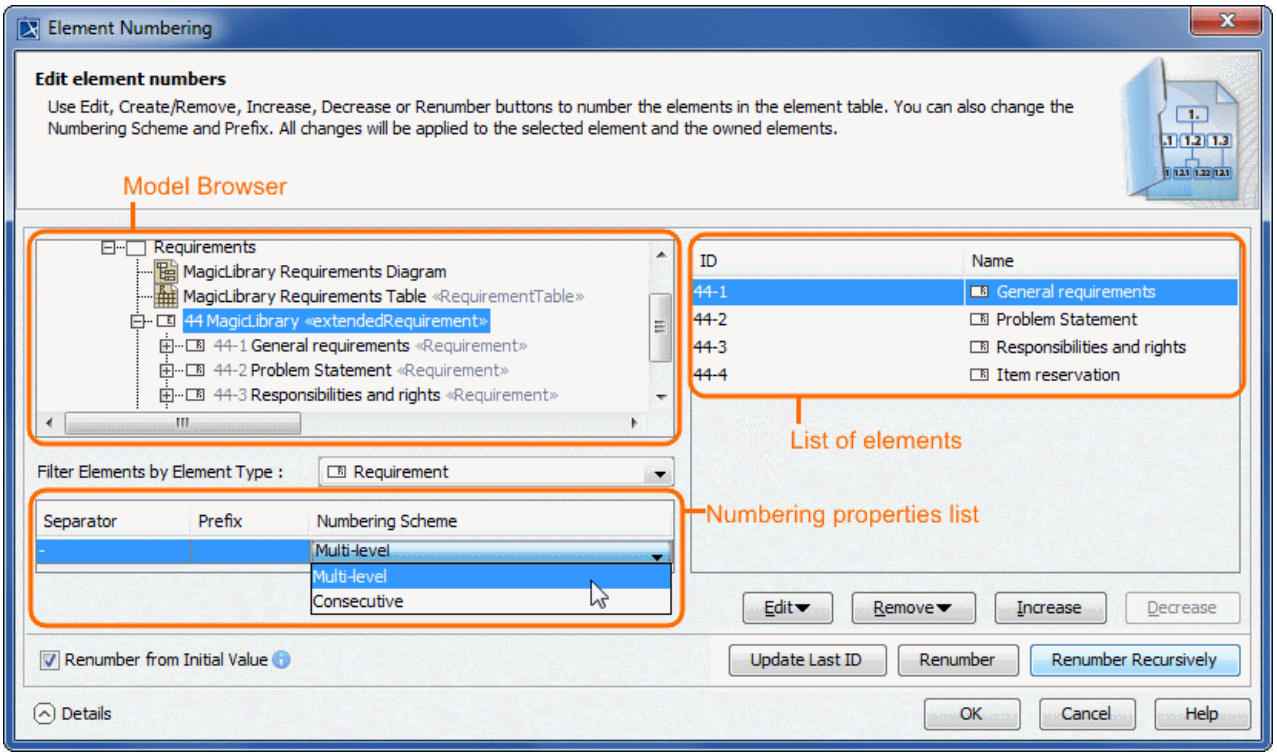
The **Element Numbering** dialog is designed to number elements by setting the ID, select different numbering schemes, or modify numbering properties that are defined by the numbering customization. Using this dialog, you can also change, remove, and update the element ID, as well as you can change the number separator, prefix, and the numbering scheme.

Numbering changes you make for the selected element will by applied to all elements owned by the selected element.

To open the **Element Numbering** dialog

1. Select the model element you want to number.
2. Open the selected element's shortcut menu and click **Element Numbering**.

The command **Element Numbering** is available, if numbering customization(see page 2132) for the selected element type is created.



Your model is represented in the Model browser on the left side of the dialog. If the element of the type that is filtered in the **Filter Elements by Element Type** list is selected in the Model browser, owned elements are listed on the right side of the dialog.

In the **Filter Elements by Element Type** list, you can select which type of elements you want to number. The selected element type separator, prefix, and numbering scheme are displayed in the numbering properties list.

The usability of numbering properties list is described in the following table:

Item name	Item type	Description
Separator	Text box	Enter your custom separator which can be any symbol. The separator is changed for all numbers of the selected element type. All previously defined separators will be overridden.

Item name	Item type	Description
Prefix	Text box	Enter your custom prefix which can be any symbol. Just the one prefix can be entered for the selected numbering scheme. The defined prefix is displayed before the element number. All previously defined prefixes will be overridden.
Numbering Scheme	Drop-down list	Select one of the available numbering schemes for the selected element type. Numbering schemes are defined in numbering customization.
Renumber from Initial Value	Check box	Select if you want elements to be renumbered from the initial value after you have changed it. Available in the renumbering options area.

For more information on numbering properties, see Creating your first numbering customization(see page 2135).

You can have several numbering schemes defined for one type of elements.
All defined numbering schemes for the selected element type are displayed in the numbered properties list.

The **Element Numbering** dialog buttons are described in the following table.

Button name	Description
Edit	Click to change the numbering symbol for the selected element. The button is available, when the selected element has a number and the element is editable. It has two options: • Click **Edit** to change the last ID symbol of the selected element. • Click **Set Custom ID** to change the whole ID (including the prefix) of the selected element.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

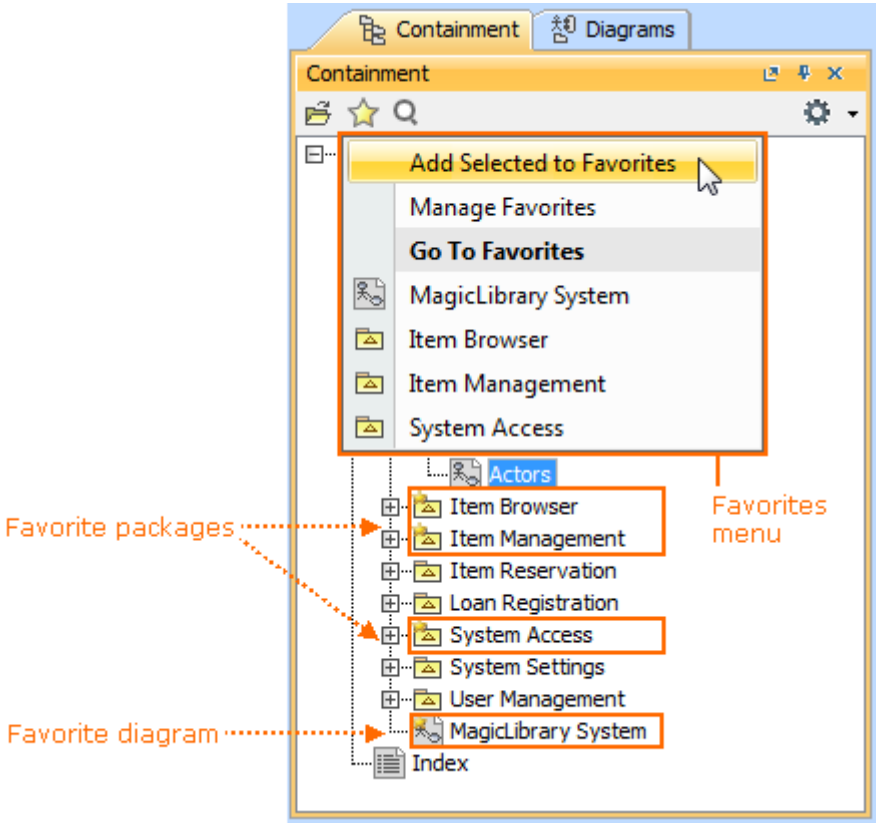
Button name	Description
Create / Remove	The **Create** button is available, if a selected element in the element list is not numbered. Click the **Create** button to number the selected element. A unique successive number will be created for the selected element. The **Remove** button is available, if a selected element in the element list is numbered and editable. This button has two options: • Click **Remove** to delete the selected element's number. • Click **Remove Recursively**. Numbers are removed from the selected element and all elements owned by the selected element.
Increase	Click to increase an element number. The button is available, if the selected element is numbered and editable. The element's place in the list changes after increasing the number.
Decrease	Click to decrease an element number. The button is available, if the element is editable, is numbered, and that number is not the first one. The element's place in the list changes after decreasing the number.
Details	Click to open the renumbering options area.
Update Last ID	Click to set the maximum ID in the numbered sequence as the last one.
Renumber	Click to update the element numbers. The numbering is updated for all elements in the list.
Renumber Recursively	Click to update all element numbers recursively. The numbering is updated for all elements in the element list, including their owning elements.

Related pages

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Favorites

You can mark as favorite any element, including diagrams, types, packages, and so forth that you regularly use in your model and need to access quickly.



Using favorites saves your time as you are able to

- Navigate quickly to any desired place in your model.
- Easily narrow the search scope to get rid of unneeded elements and thus find desired elements faster.
- Easily narrow the scope for model analysis.

In a server project, each user has his own list of favorites.

Related pages

Adding, removing, and changing order of favorites

There are a few ways of adding an element to favorites:

- By using the **Add Selected to Favorites** command from the **Favorites** menu. This way is especially handy for the quick addition to favorites.

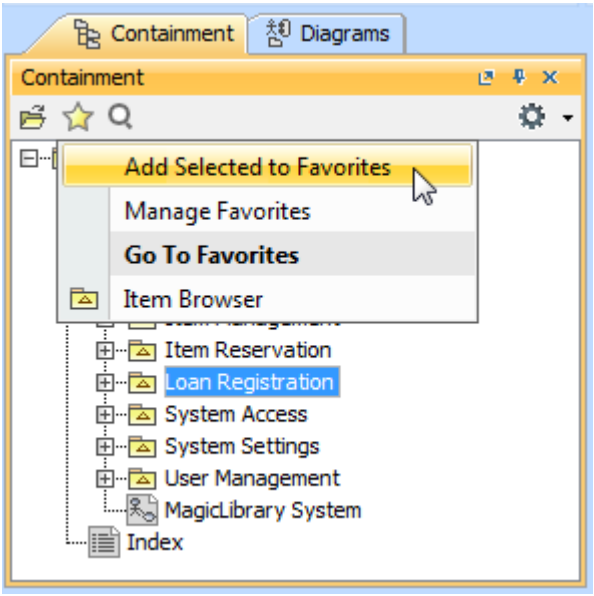
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- By using the Manage Favorites(see page 1053) dialog. Choose this way, when you need to add and then perform another manipulations with favorites.

To quickly add an element to favorites

---

1. Select the element in the Model Browser and click on the toolbar of the open tab.



2. Click **Add Selected to Favorites** on the menu.

To add an element to favorites in the Manage Favorites(see page 1053) dialog

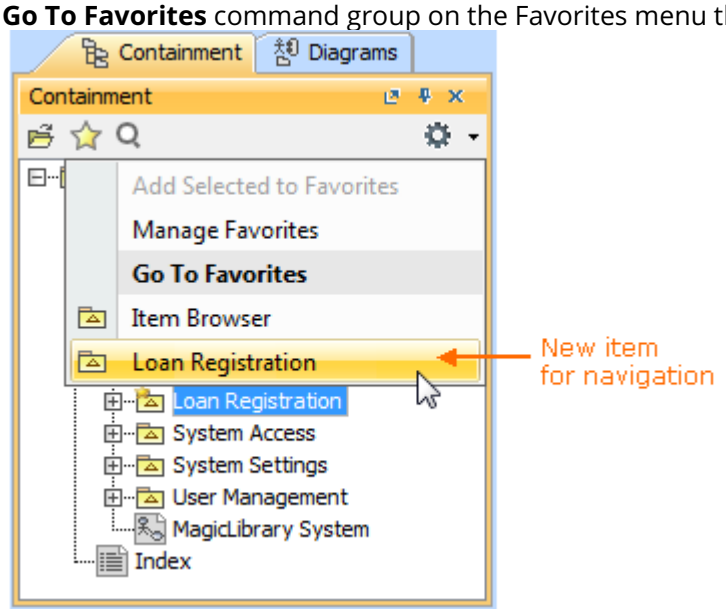
---

1. Open the Manage Favorites(see page 1053) dialog.
2. In the Manage Favorites(see page 1053) dialog, select the element and click the button, which is between two lists of the dialog. The element appears in the list of favorites.
3. Click **OK** to confirm the addition and close the dialog.

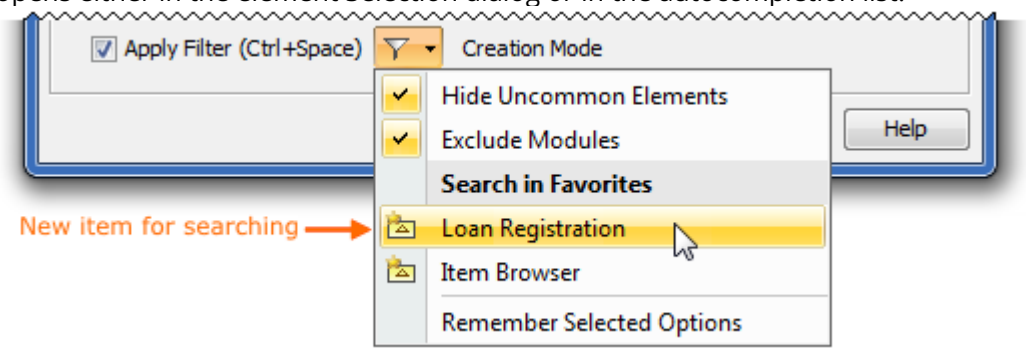
Once the element is added to favorites, the following changes occur in your project:

- The element becomes marked as favorite in all the trees and lists of the model.

- You can easily navigate to the element in the Model Browser. The element name is added to the **Go To Favorites** command group on the Favorites menu that opens in the Model Browser.



- You can easily specify the element (in case it is a diagram or a package) as the search scope. The element name is added to the **Search in Favorites** command group on the Favorites menu that opens either in the element Selection dialog or in the autocompletion list.



To remove an element from favorites

---

1. Open the Manage Favorites(see page 1053) dialog.
2. In the list of favorites (on the right side of the dialog), click the element you need to remove and then click the button.
3. Click **OK** to confirm the removal and close the dialog.

Once the element is removed from favorites, it is no longer appropriately marked. Commands for navigation to the element and for searching in it are no longer on the Favorites menu.

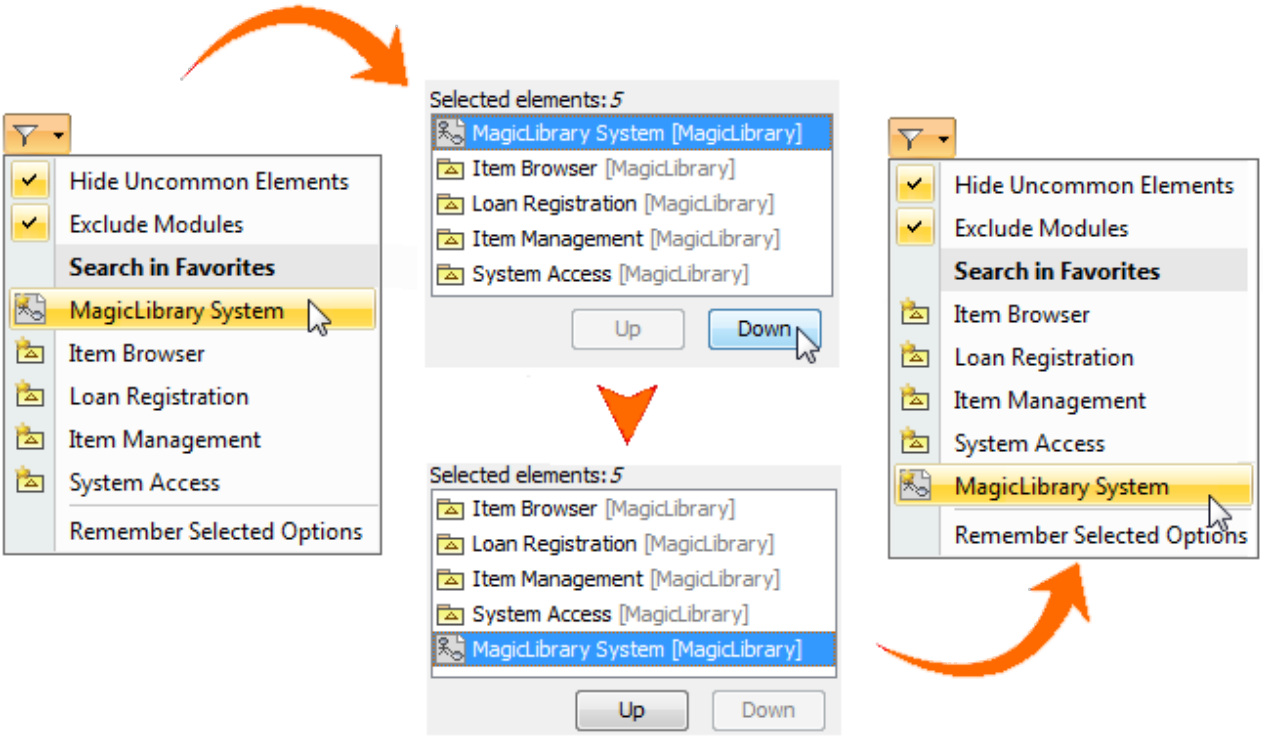
The following procedure describes how to change the order of favorite elements on the Favorites menu. To change the order of favorites on the Favorites menu, you need to make appropriate order changes in the list of favorites on the Manage Favorites(see page 1053) dialog.

To change the order of favorites

---

1. Open the **Manage Favorites** (see page 1053)dialog.

2. In the list of favorites (on the right side of the dialog), click an element, whose order on the Favorites menu you need to change and then do either:
    - If you need to move the element up, click the **Up** button. Click again, if you need to move the element up by one more position and so on.
    - If you need to move the element down, click the **Down** button. Click again, if you need to move the element down by one more position and so on.
3. Click **OK** to confirm the reorder and close the dialog.



Related pages

- Favorites(see page 1050)

Manage favorites

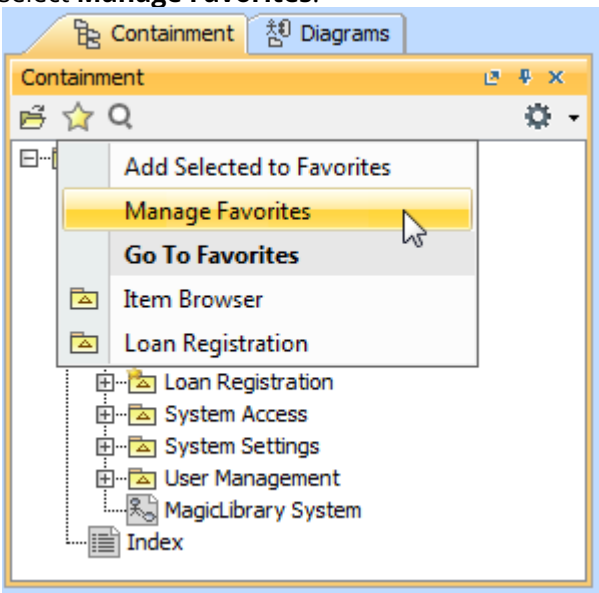
The Manage Favorites dialog allows for

- Adding to favorites
- Removing from favorites
- Changing the order of favorites

To open the **Manage Favorites** dialog

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

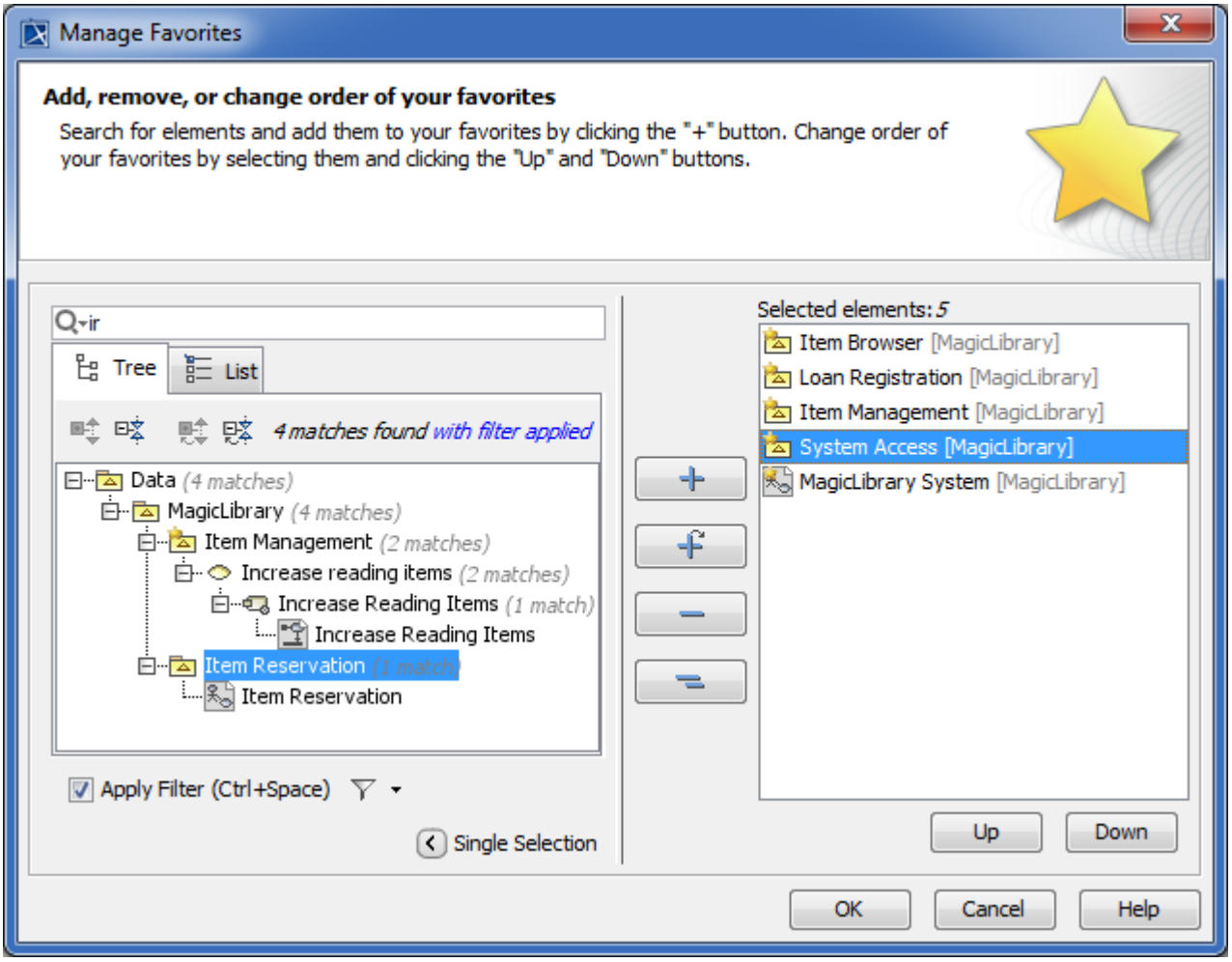
- In the Model Browser, click the button, which is on the toolbar of the open tab, and then select **Manage Favorites**.



The **Manage Favorites** dialog is a modification of the element Selection dialog. The dialog displays

- All elements (in tree or list view on the left side of the dialog)
- Favorite elements (in the list on the right side of the dialog)
- Buttons for adding and removing from favorites
- Search modes and scope filters area
- Buttons for reordering favorites

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The order of favorites in the list of the **Manage Favorites** dialog meets the order of favorites on the Favorites menu. Thus if you need to change the order of favorites on this menu, make appropriate order changes in the list of favorites on the **Manage Favorites** dialog. Buttons below the list will help you to change the order of favorites.

Related pages

- Favorites(see page 1050)

HTML editor

You can easily add or edit HTML text using the HTML editor. It also allows you to preserve the text format when copying formatted text.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To set the note, text box, or separator text as HTML

Do one of the following:

- Select a note(see page 791)/text box(see page 907) shape and click the **HTML** button when **Switch To HTML Text** appears on the lower-left corner of the shape.

- Draw the note/text box, using the **Note(HTML text)** or **Text Box(HTML text)** buttons on the diagram palette.

Click the text area on the selected shape or start typing to open the HTML editor toolbar. The buttons on the HTML editor toolbar have the same functions as a normal editing toolbar.

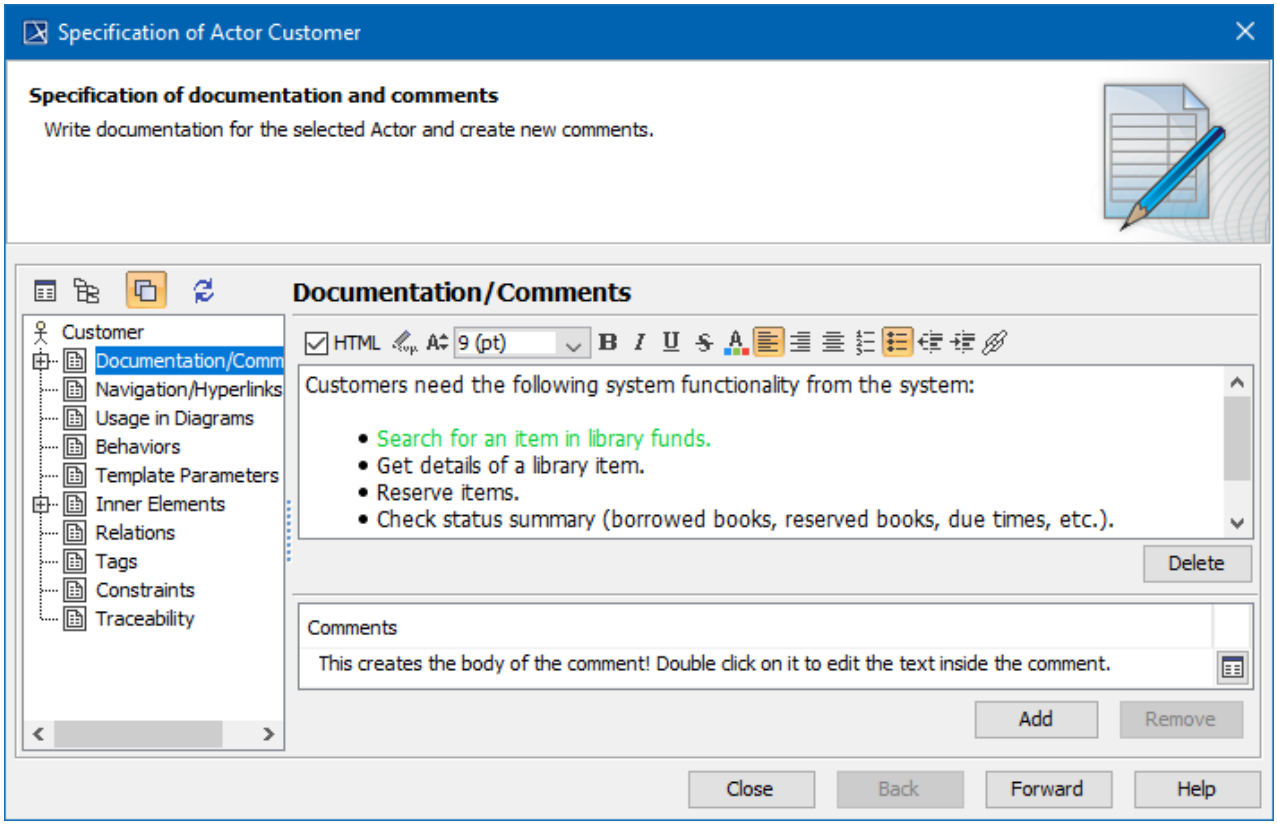
HTML editor toolbar

You can write HTML text in various dialogs, but you must turn on the HTML mode first.

To turn on the HTML mode in a dialog

- Simply click to select the **HTML** check box. The following scenarios show the various ways of reaching the check box:
  - Element Specification window(see page 913):
    i. Open an element's specification window.
    ii. Click the **Documentation/Comments** tab.
    iii. Check HTML. Click **Close**. See the figure below.

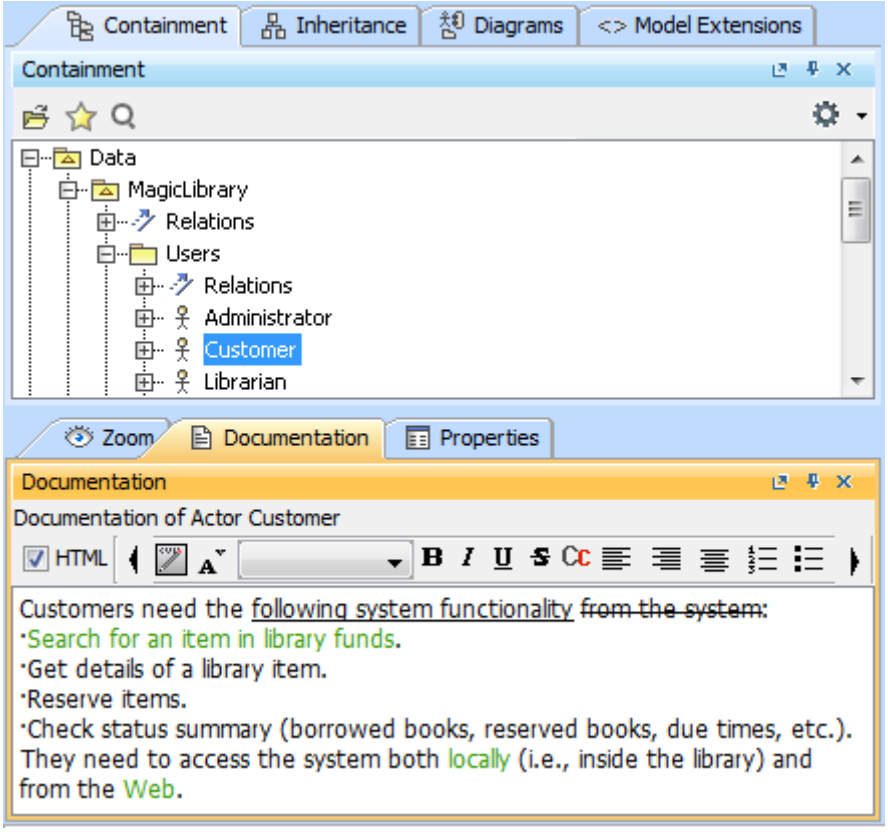
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



HTML editor toolbar in Documentation/Hyperlinks tab of element Specification window

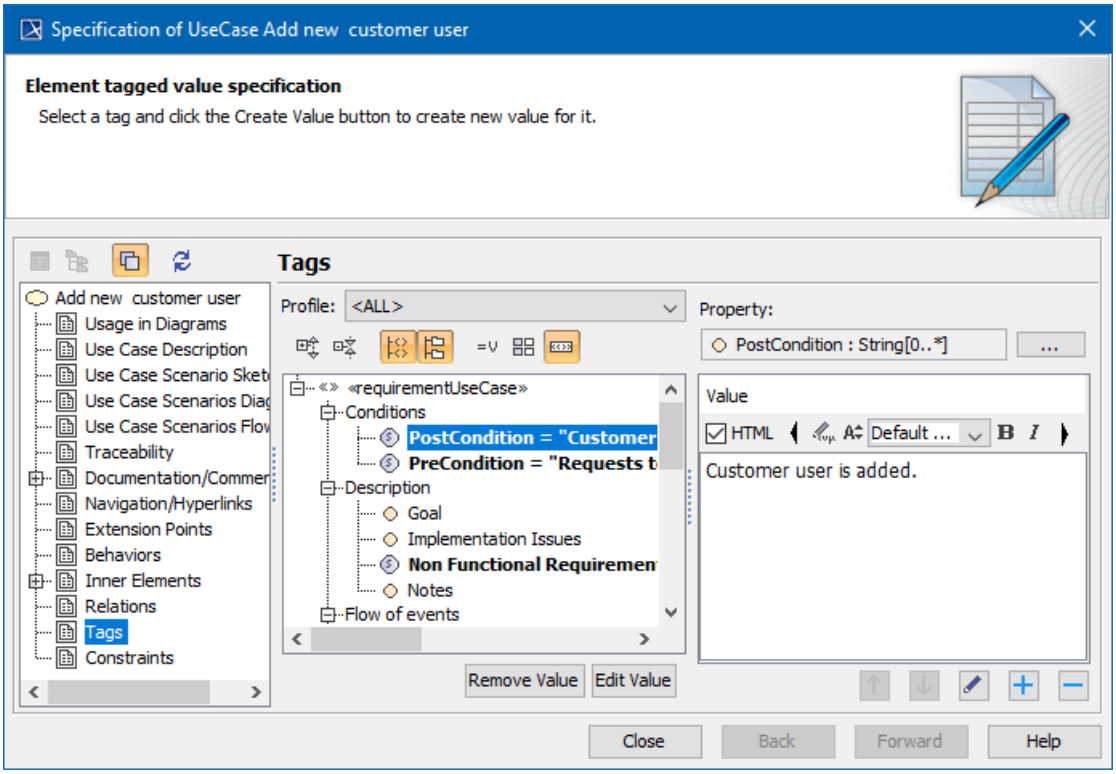
- In the Model Browser:
  a. Click the element in the Containment Tree(see page 693).
  b. Click the **Documentation** tab.
  c. Click **HTML**, then click anywhere on the diagram pane. See the figure below.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



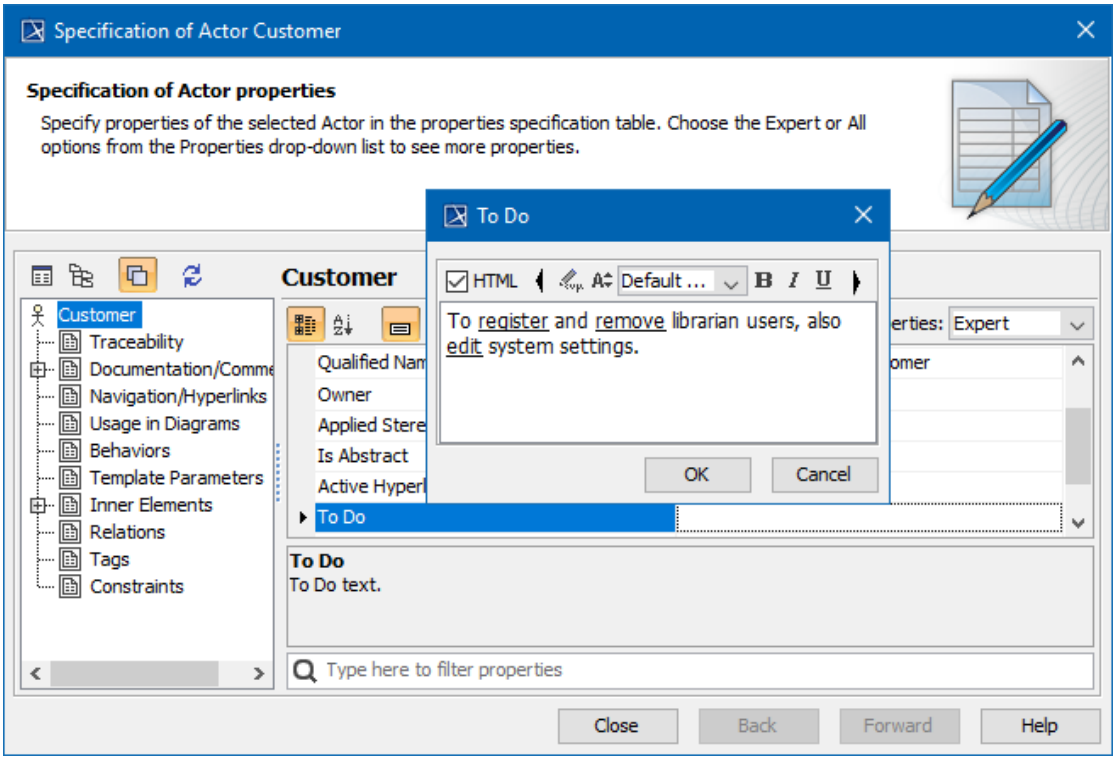
HTML editor toolbar in Documentation panel on Model Browser

- Element Specification window:
  a. Open an element specification window.
  b. Click the **Tags** property group, when editing a tagged value.
  c. Click **HTML**. Then click **Close**. See the figure below.



HTML editor toolbar in Tags tab of element Specification window when editing tag value

- In the **To Do** field:
  a. Open an element specification window.
  b. In the dialog, click the **To Do** field to edit it.
  c. Click the ··· button to open the **To Do** text dialog.
  d. In the **To Do** text dialog, click **HTML**. Edit the text as you wish, then click **OK** and click **Close**.



HTML editor toolbar in To Do dialog

You can also use the HTML editor toolbar when editing a tagged value directly on the element shape.

To use the HTML editor toolbar for editing a tagged value

---

1. Click the **HTML** button on the element shape to switch to the HTML mode.
2. On the element shape, click the tagged value. The HTML editor toolbar opens for editing the tagged value.

Related Pages:

Advanced HTML Editor dialog

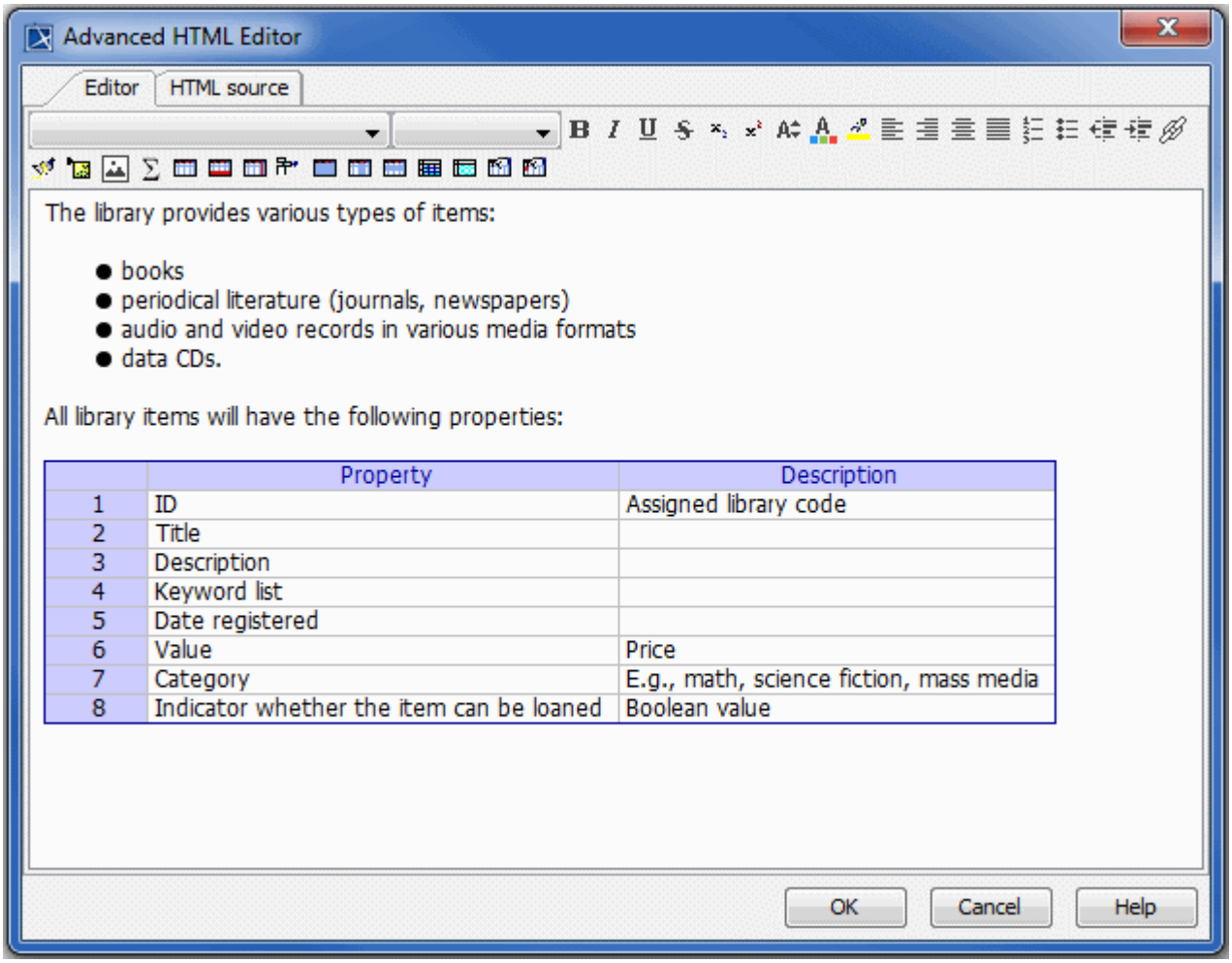
he Advanced HTML Editor allows you to change text style, insert images(see page 1062), insert symbols and tables, and perform other formatting actions.

To open the **Advanced HTML Editor** dialog

- Click the Advanced HTML Editor button on the HTML editor toolbar.

For more information, please see how to turn on the HTML mode(see page 1055) in a dialog.

Click the **HTML source** tab to view the HTML source.



Advanced HTML Editor dialog

To insert an image into HTML text

1. Click the **Insert image** button, as shown in the following picture.

2. In the **Picture Properties** dialog, specify the image location.
   Do one of the following:
   - Type the path to the image location.
   - Click **Browse** and select the image you want to insert.

   > If you are working on a server project, make sure that the path to the image location is accessible from any computer that has an Internet connection. Otherwise, the image does not display when the project opens on another computer.

   To insert an image as an attached file, simply paste its URL to the **Picture Source** box by pressing Ctrl+V (Cmd+V on OS X). For more information, refer to Inserting into HTML text(see page 1062).
3. Click OK when you are done.

The image is inserted into HTML text.

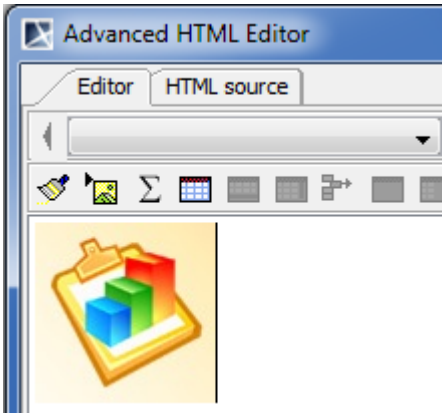


Image in HTML text.

Inserting images into HTML text

To insert an image into HTML text

From the Containment tree

- When the image is already an AttachedFile, please refer to Working with attached files(see page 212).

By using the Advanced HTML Editor

1. Open the Advanced HTML Editor(see page 1060).
2. On the toolbar, click **Insert Image** button , then click one of the following:

   - **From File** – to browse the image in your file system or paste the attached image file URL into the **Picture source** field.
   - **From Attached File** – to open the **Select Elements** dialog and select the appropriate parent element to which the AttachedFile image should be attached.
   - **From Image Library** – to use images from a bundled image collection or your own added sets of images.

3. Click **OK** twice to close both dialogs.

By pasting or dragging the image file from file system

1. Open the image file location, select the file(s) you want to insert, then do either:

   - copy/paste them to the HTML Editor. In the **Paste Special** menu, select **Image** to paste image and attach it to the project or **Image (without attaching)** to paste it only for referencing in the diagram.
   - drag them to the HTML Editor. In the **Select Image Owner**, select the appropriate parent element to which the AttachedFile image should be attached. If you click **Cancel**, the image will be displayed in the diagram, but not attached to the project.

2. Click **OK** to close the dialogs.

   If you chose to not attach the image, it is displayed in the HTML Editor, but when you delete the file from your file system, it is no longer displayed. That also means, other users that you want to share the project with, won't be able to see the image if they open the project from another device. This option is very useful when the file you want to insert is large and you don't want to increase the project size.

   This feature works on both HTML Editor and Advanced HTML Editor.

By pasting or dragging an image from a web browser

1. Do either:

   - copy/paste the image to the HTML Editor. In the **Paste Special** menu, select **Image** to paste image and attach it to the project or **HTML Text** to paste it only for referencing in the diagram.
   - drag the image to the HTML Editor.

2. Click **OK** to close the dialogs.

   This feature works only on HTML Editor.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

> This feature works only with Google Chrome and Mozilla Firefox.

By pasting or dragging rich text with an image

1. Do either:
   - copy/paste content to the HTML Editor. In the **Paste Special** menu, select **HTML Text** to choose whether paste it only for referencing in the diagram or attach it to the project, or select **Image** to paste and attach it to the project, or select **Plain Text** to paste only text without image.
   - drag the image to the HTML Editor. In the **Select Image Owner**, select the appropriate parent element to which the AttachedFile image should be attached. If you click **Cancel**, the image will be displayed in the diagram, but not attached to the project.
2. Click **OK** to close the dialogs.

> This feature works only on HTML Editor.

> Only rich text from MS Office files is supported.

> If the image from the rich text document is not stored in your file system, you are not asked to choose whether to attach the file or not.

To insert multiple image files

---

1. Open the image files location, select the files you want to insert, then do either:
   - copy/paste them to the HTML Editor. In the **Paste Special** menu, select **Image** to paste images and attach them to the project or **Image (without attaching)** to paste them only for referencing in the diagram.
   - drag them to the HTML Editor. In the **Select Image Owner**, select the appropriate parent element to which the AttachedFile images should be attached.
2. Click **OK** to close the dialogs.

> This feature works on both HTML Editor and Advanced HTML Editor.

Parameters synchronization

The parameters synchronization keeps your model valid and synchronized after making changes. How the synchronization works? After you have created a new element or edited a property the associated elements and properties are updated according to your changes.

The following table lists the pairs of elements that are synchronized.

Synchronized Elements		The backward synchronization is available
Behavior (Parameter)	Operation (Parameter)	Yes
Operation (Parameter)	Call Operation Action (Pin)	Yes
Behavior (Parameter)	Call Behavior Action (Pin)	Yes
Signal (Attribute)	Send Signal Action (Pin)	Yes
Signal (Attribute)	Accept Event Action (Pin)	Yes
Activity (Parameter)	Activity (Activity Parameter Node)	Yes
Behavior (Parameter)	Signal Reception (Parameter)	Yes
Operation (Parameter)	Message (Argument)	No
Interaction (Parameter)	Interaction Use (Argument)	No
Signal (Attribute)	Message (Argument)	No

The following table lists the synchronized element properties and conditions on which these properties are synchronized.

Synchronized Elements		Properties			
		Name	Type	Direction	Multiplicity
Behavior (Parameter)	Operation (Parameter)	always	always	always	always
Operation (Parameter)	Behavior (Parameter)	always	always	always	always
Operation (Parameter)	Call Operation Action (Pin)	when source and target have the same values	when source and target have the same values	always	when source and target have the same values
Behavior (Parameter)	Call Behavior Action (Pin)				
Call Operation Action (Pin)	Operation (Parameter)	when target is undefined or unnamed	when target is undefined or unnamed	always	when target is ndefined or nnamed

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Call Behavior Action (Pin)	Behavior (Parameter)				
Signal (Attribute)	Send Signal Action (Pin)	when source and target have the same values	when source and target have the same values	-	when source and target have the same values
Signal (Attribute)	Accept Event Action (Pin)				
Send Signal Action (Pin)	Signal (Attribute)	when target is undefined or unnamed	when target is undefined or unnamed	-	when target is undefined or unnamed
Accept Event Action (Pin)	Signal (Attribute)				
Activity (Parameter)	Activity (Activity Parameter Node)	when target is undefined or unnamed	always	always	-
Activity (Activity Parameter Node)	Activity (Parameter)	when target is undefined or unnamed	always	not synchronized	-
Behavior (Parameter)	Signal Reception (Parameter)	always	always	always	always
Signal Reception (Parameter)	Behavior (Parameter)	always	always	always	always

To turn on the parameters synchronization

1. On the **Options** menu, click **Project**.
2. In the **Project Options** dialog, the **General** project options group, click to select the **Synchronize Parameters and Arguments** option.

Related pages

Resolving not synchronized elements

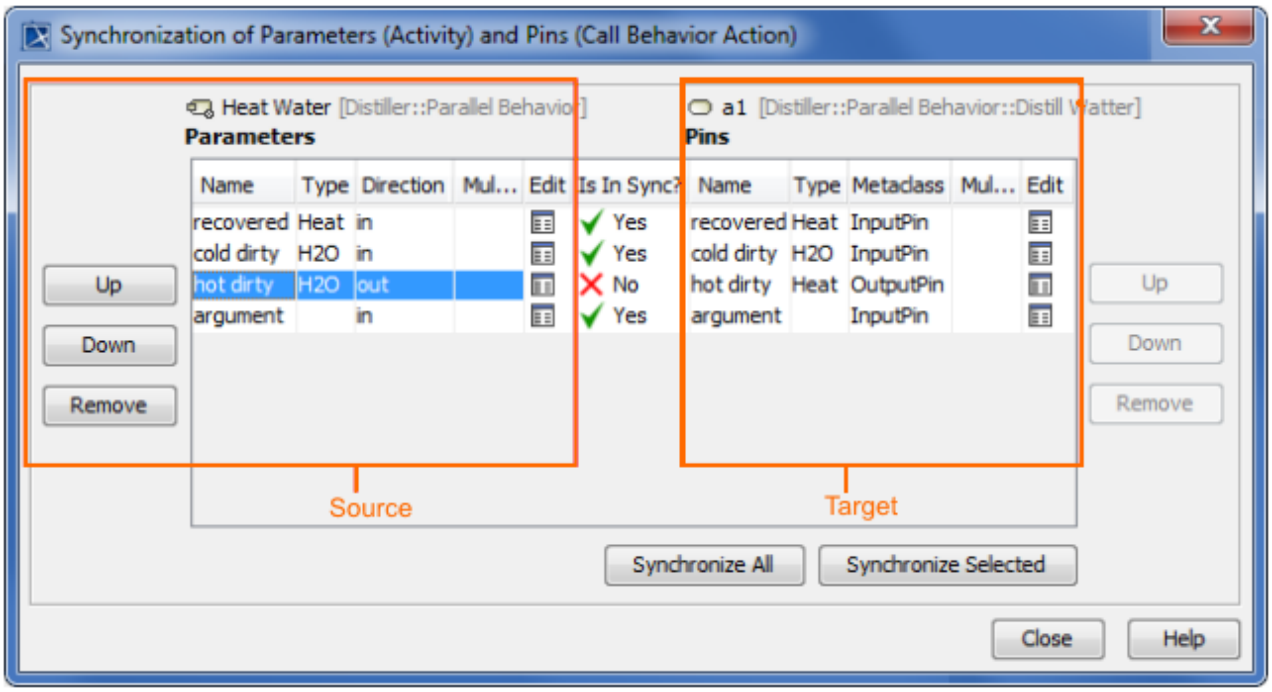
The active validation functionality marks the not synchronized elements in the Model Browser and highlights the shapes on the diagram pane. For more information about the active validation and solving the incorrect model, see Validation(see page 1306).

In addition, you can solve the broken parameters synchronization by using the **Synchronization** dialog.

Synchronization dialog environment

The **Synchronization** dialog represents the elements those parameters are not synchronized and provides the possibility to restore the synchronization.

In this section, you will find the brief information on how to open the **Synchronization** dialog and on each button that can be useful while working in the dialog.



To open the Synchronization dialog, do one of the following

- In the Model Browser, right-click the invalid element, on the shortcut menu, point to **Validation**, then to the validation group, and click the **Synchronize Manually** command.
- On the diagram pane, select the invalid shape, on the smart manipulator toolbar, click the button, and then click the **Synchronize Manually** command.
- In the **Validation Results** window, click the **Solve** button and then click the **Synchronize Manually** command. For more information on how to open the **Validation Results** window, see Validation(see page 1306).

The source and target elements area

The following table describes the buttons in the source and target areas of the Synchronization dialog.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Button	Description
Source elements area:	
Up	Click to shift the selected source element up a row. In this way you can change the order of the source elements.
Down	Click to shift the selected source element down a row. In this way you can change the order of the source elements.
Remove	Click to remove the selected source element from the project.
Edit	Click to open the Specification window of the selected source element.
Target elements area:	
Up	Click to shift the selected target element up a row. In this way you can change the order of the target elements.
Down	Click to shift the selected target element down a row. In this way you can change the order of the target elements.
Remove	Click to remove the selected target element from the project.
Edit	Click to open the Specification window of the selected target element.

Buttons in the Synchronization dialog

The following table describes the Synchronization dialog buttons.

Button	Description
Synchronize All	Click the button to synchronize all the not synchronized elements. Then from the menu select the straightforward or backward synchronization. For example, when synchronizing Parameters (Activity) and Pins (Call Behavior Action) you can select to synchronize one of the following: • Pins (Call Behavior Action) by Parameters (Activity) • Parameters (Activity) by Pins (Call Behavior Action)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Button	Description
Synchronize Selected	Click to synchronize the selected parameter. Note that the button is disabled if no parameters are selected in the list. After you click the Synchronize Selected button, the following menu appears: **Create ->** - **Click** to create the target element with the same properties as the source element. **<-Create** - Click to create the source element with be create with the same properties as the target element. **Update ->** - Click to update the not synchronized properties of the target element according to the properties of the source element. **<-Update** - Click to update the not synchronized properties of the source element according to the properties of the target element. Note that the Update command exists only if there are properties to update.

On this page

- The source and target elements area(see page 1067)
- Buttons in the Synchronization dialog(see page 1068)

Related pages

- Parameters synchronization(see page 1064)
- Working with model elements(see page 909)

Copying and opening element URLs

Note
Copying and opening element URLs do not work with the no install version (.zip) of a modeling tool.

You can copy a project element URL to a clipboard and share it with others as a quick reference to model elements.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To copy a project element URL, do one of the following

- Select **Copy URL** from the element shortcut menu in the Containment tree to copy the URL to a model element.
- Select the element symbol in a diagram and click **Edit** > **Copy URL** on the main menu to copy the URL to element symbol.

You can open any element through its URL by clicking the **Open Element from URL** command. The element will then be highlighted in the Containment tree or in the diagram. Custom URL "mdel://" is registered into windows registry. Activating the URL in other applications will allow you to start MagicDraw, open the project (if possible), and select one or more elements. You can paste URLs from the clipboard to any MagicDraw diagram. Hyperlinks also can hold URLs of any model elements.

Note
You cannot open elements through their URLs on Chrome or Opera browser to start a modeling tool.

Related page

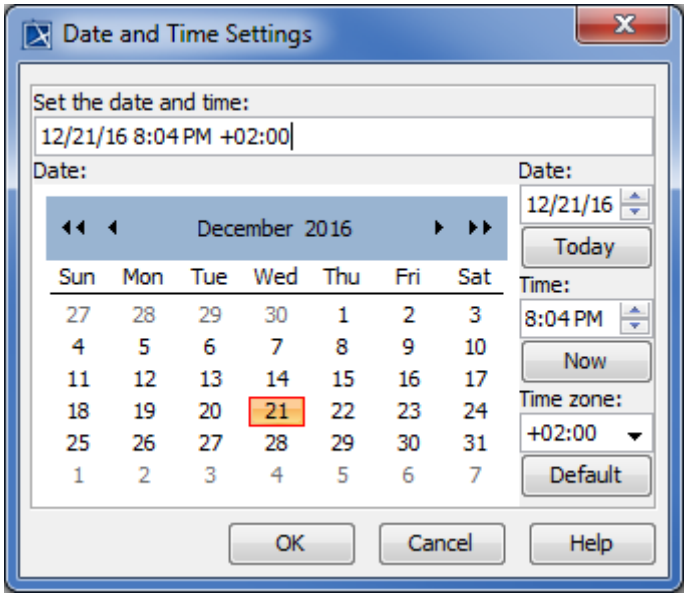
- Working with model elements(see page 909)

Setting date and time

Date and time are set in the **Date and Time Settings** dialog. This dialog is always used when specifying the properties that have a date type value. It allows you to select a date, time, and a time zone using a visual calendar and/or the **Date**, **Time**, and **Time zone** boxes.

Date/time format information
The **Date and Time Settings** dialog uses your computer's locale to set the date and time format. You can still enter date and time values of another format manually in the **Set date and time** box, but the dialog will not recognize them; i.e., you won't be able to use a calendar or the **Date** and **Time** boxes. Using them would automatically change the date and time format to the one that is specified in your computer's locale.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The figure displays the Date and Time Settings dialog with specified date, time and time zone values.

The **Date and Time Settings** dialog can be opened from:

- The Specification window(see page 913) of an element that has a property with a date type value
- A Generic Table(see page 441) that has a column of date type values.

The Date and Time Settings dialog elements

Dialog element	Element type	Description
Set the date and time	Text box	Allows you to enter date and time values in the following order: • Date • Time (optional) • Time zone (optional) Values should be separated by spaces.
Date	Visual calendar	To select a date, click a day on a calendar. To switch between months, use the arrows on the left and on the right side of the title section of the calendar. To switch between years, use the double-arrows on the left and on the right side of the title section of the calendar. The selected date is automatically entered in the **Set the date and time** box using the format that is specified in your computer's locale.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Dialog element	Element type	Description
Date	Spin box	Allows you to select a date value in the format specified in your computer's locale. The default value is the current date. To change a date value, use the up or down arrow to increase or decrease the current value. The selected value is automatically entered in the **Set the date and time** box.
Time	Spin box	Allows you to select a time value in the format specified in your computer's locale. The default value is the current time. To change a time value, use the up or down arrow to increase or decrease the current value. The selected value is automatically entered in the **Set the date and time** box.
Time zone	Drop-down list	Click the arrow on the right side of the box to select the desired time zone.
Today	Button	Click the button to set the **Date** box value to the current date of your computer.
Now	Button	Click the button to set the **Time** box value to the current time of your computer.
Default	Button	Click the button to set the **Time zone** box value to the value set in your computer.

On this page

- The Date and Time Settings dialog elements(see page 1071)

Related pages

- Working with model elements(see page 909)
  - Specification window(see page 913)
  - Default property values(see page 919)
  - Editing property values(see page 921)
  - Refactoring(see page 936)
  - Extracting(see page 947)
  - Formatting symbols(see page 975)
  - Customizing lists of properties and commands(see page 1001)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Owner of model elements

Model elements and diagrams belong to a package, model (system boundary), subsystem, or other appropriated model element, which is called owner.

The name of the owner is displayed in the model element name compartment in parentheses.

To add a model element to a package, model (system boundary), or subsystem, do one of the following

- Drag a model element to the desired package on the diagram pane or in the Model Browser.
- Open the Package, Model , or Subsystem Specification window(see page 913), the Inner Elements property group. Click **Create** and select a model element or diagram you want to add. Define a model element or diagram in the open Specification window and click **OK**.
- From the selected owner shortcut menu in the Model Browser, select **Create Element**. From the list, select the desired model element. The element is created.

To display/hide the package/system boundary/subsystem name (the owner of an actor) on a model element

- From the symbol shortcut menu, select **Symbol Properties**. The Symbol Properties dialog (see page 975)opens. In the Show Owner drop down list, select one of the owner display mode: Do Not Display, Below Element Name, In Same Line With Name, Above Element Name. For more information about the owner display mode, see " Owner display mode".
- From the Options menu, select Project. The Project Options dialog opens. Select the desired model element and in the Show Owner drop down list, choose one of the owner display mode. If you want to apply changes for previously created model elements, click Apply.

Related pages

- Working with model elements(see page 909)
- Owner display mode(see page 1073)
- Qualified name display style(see page 1074)

Owner display mode

You can change the display position of the qualified name on the element shape.

To change the qualified name position

1. On the diagram, right-click the shape and, from the shortcut menu, select the **Symbol Properties** command or select the shape and press Alt+Enter.
2. In the open **Symbol Properties** dialog, change the property **Show Owner**. Select one of the four property modes:

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Show Owner Property Mode	Shape	Description	Notation
Do Not Display	MagicLibraryforCustomer	Only element name is displayed on the element shape. This is the default value.	-
Below Element Name	MagicLibraryforCustomer (Code Level.com.nomagic.magiclibrary.jsp)	Owner is displayed below the element name.	This is MagicDraw style notation. The owner name is constructed from the names of the containing namespaces starting at the root of the hierarchy and ending with the owner of the NamedElement itself. Containing namespaces are separated by dot and owner is displayed in brackets.
In Same Line With Name	Code Level::com::nomagic::magiclibrary::jsp::MagicLibraryforCustomer	Element owner is displayed in the same line as the element name.	This is a notation from UML specification. The qualified name is constructed from the names of the containing namespaces, starting at the root of the hierarchy and ending with the name of the NamedElement itself. Containing namespaces are separated by double colons. The double colon is shown to separate containing namespaces and element name.
Above Element Name	Code Level::com::nomagic::magiclibrary::jsp::MagicLibraryforCustomer	Owner is displayed above the element name.	Notation is the same as **In Same Line With Name** option notation.

Qualified name display style

There is an option to show the owner hierarchy starting from the model library as the root. *Model Library* is a package with the stereotype *«modelLibrary»* applied.

This option is called **Qualified name display style**.

To change the **Qualified name display style** value

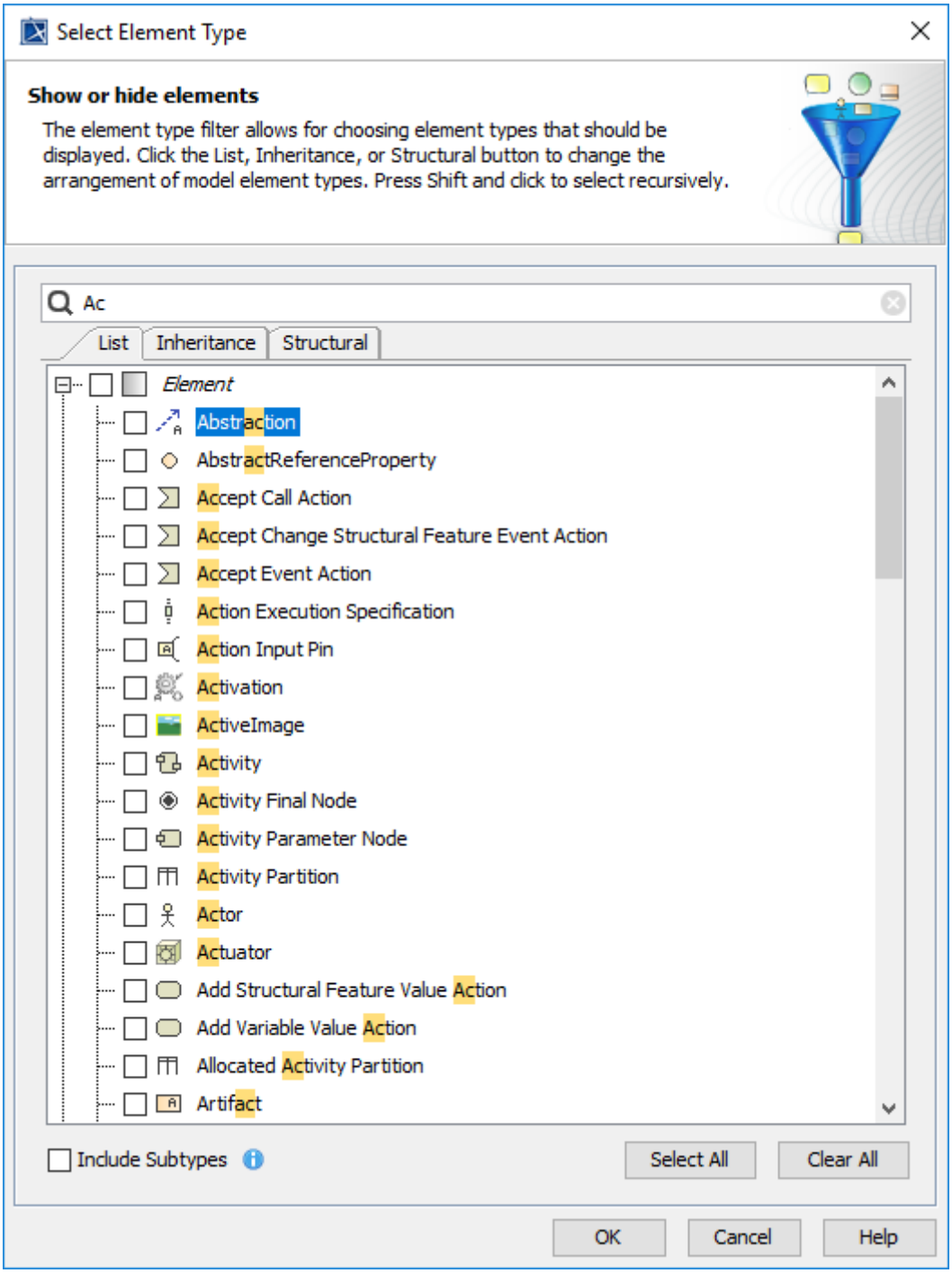
1. From the **Options** menu, select the **Project** command. The **Project Option** dialo(see page 214)g opens.
2. In the option group list, select **General** > **General**, and then select the **Qualified name display style** option.
3. For the option, set **Model Library Relative** (default value for a new project) to display the full qualified name hierarchy of the element starting from the model library as a root element. The model library itself is not displayed. The **Qualified name display style** property allows for having the relative path for library items used in the project.

Selecting element types

In the **Select Element Type** dialog, you can select the check box next to the model element type to be filtered – displayed or searched for. The dialog opens when filtering items in the **Containment** tab(see page 32) and when searching or filtering for elements:

- When searching(see page 216) for items, the purpose of the **Select Element Type** dialog is to select the type of element to search.
- When filtering(see page 233) items in the **Containment** tab, the purpose of the **Select Element Type** dialog is to provide a possibility to hide elements which you do not want to see in the **Containment** tab.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

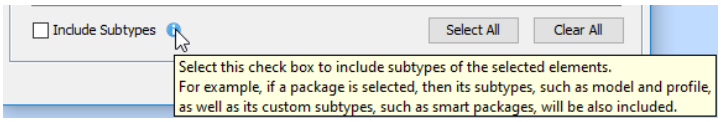


The Select Element Type dialog.

The following table explains the functions of the tabs, option, and buttons in the **Select Element Type** dialog.

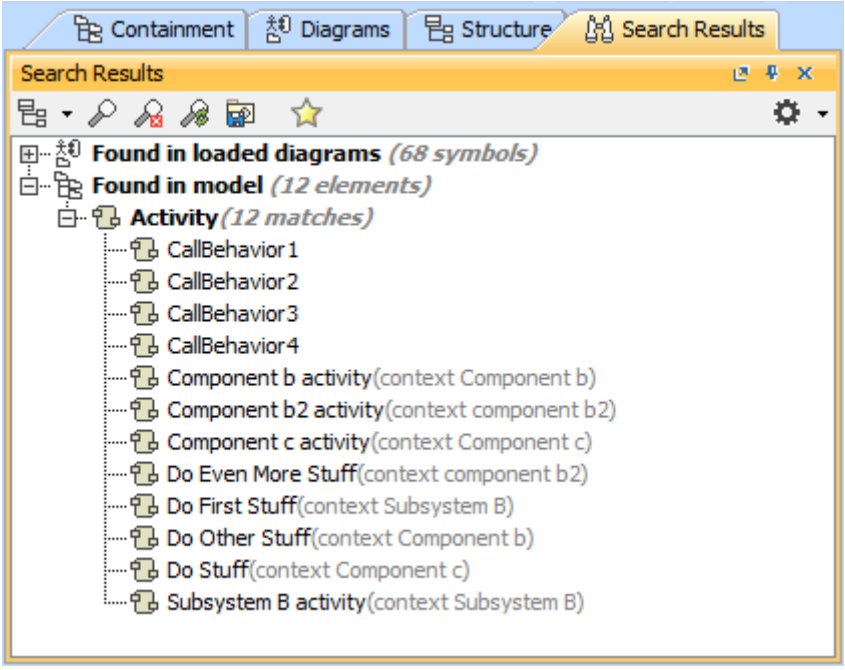
Button/Tab	Function
Search text box	Specifies a searching word to filter the search results of element types in the **List**, **Inheritance**, and **Structural** tabs. **Note** The appearances of the searching word in the search results are highlighted in yellow.
List	Click the **List** button to list the types of elements in one alphabetic list. There are two groups in the list: *Element* and *Symbol*.
Inheritance	Click the **Inheritance** button to group the types of elements according to the inheritance. For example, types of elements are grouped to the following inheritances: *Named Element*, *Relations*, and others. The *Symbol* group is listed also.
Structural	Click the **Inheritance** button to group the types of elements according to the following groups: *DSL Customization*, *MagicDraw Profile*, *UML2 Metamodel*, *Symbol*.
Select All	Selects all types of elements in the list – all elements will be displayed after filtering or searching.
Clear All	Clears all types of elements in the list – no elements will be displayed after filtering or searching.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Button/Tab	Function
Include Subtypes (Available only when opening the **Select Element Type** dialog from the **Find** dialog(see page 217))	Includes subtypes of the selected elements, e.g., models and profiles of Packages.  The tooltip of the Include Subtypes option in the Select Element Type dialog.
OK	Applies the selected/cleared element types to display or hide them in the **Containment** tab when you have opened the dialog from the **Containment** tab. Applies the selected/cleared element types to filter and display them in the **Search Results** tab in the Model Browser when you have opened the dialog from the **Find** dialog(see page 217).
Cancel	Cancels the selecting/clearing or filtering of the element types and closes the dialog.
Help	Opens the **Selecting element types** page with the description of the dialog.

The **Search Results** tab(see page 47) opens in the Model Browser to display the search results according to the searching criteria you have created from selecting/clearing types of elements in the **Select Element Type** dialog.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The Search Results tab in the Model Browser to display the search results.

Related pages

- Filtering(see page 233)
- Model Browser(see page 29)
- Containment tab(see page 32)
- Search Results tab(see page 47)

Changing the tooltips style

You can specify the tooltips display mode of the symbols on the diagram pane(see page 24) by selecting to hide it, to show only element name, or its documentation text. The element tooltip in the Model Browser(see page 29) displays only documentation text, if it is specified for an element. Otherwise, the tooltip is not shown in the Model Browser(see page 29).

> **Writing documentation for an element**
> You can write a documentation for an element by using the:
> - Documentation tab(see page 54).
> - Element Specification window(see page 913) > **Documentation/Hyperlinks** property group.
> - Element Specification window(see page 913) > **Documentation** property.

To change the tooltips style of the symbol on the diagram pane

1. On the main menu, select **Options** > **Project**.
2. In the **Project Options** dialog(see page 2221) find the **Tooltips Style** option.

3. Select one of the following value:
    - **Do Not Show** (default) - the tooltip is not shown when hovering a mouse on any element.
    - **Show Element Name** - The tooltip displays the name of the element when hovering a mouse on it.
    - **Show Element Documentation** - The tooltip displays the documentation text of the element when hovering a mouse on it.
4. Click **OK**.
   The symbol tootips display mode is changed in the entire project.

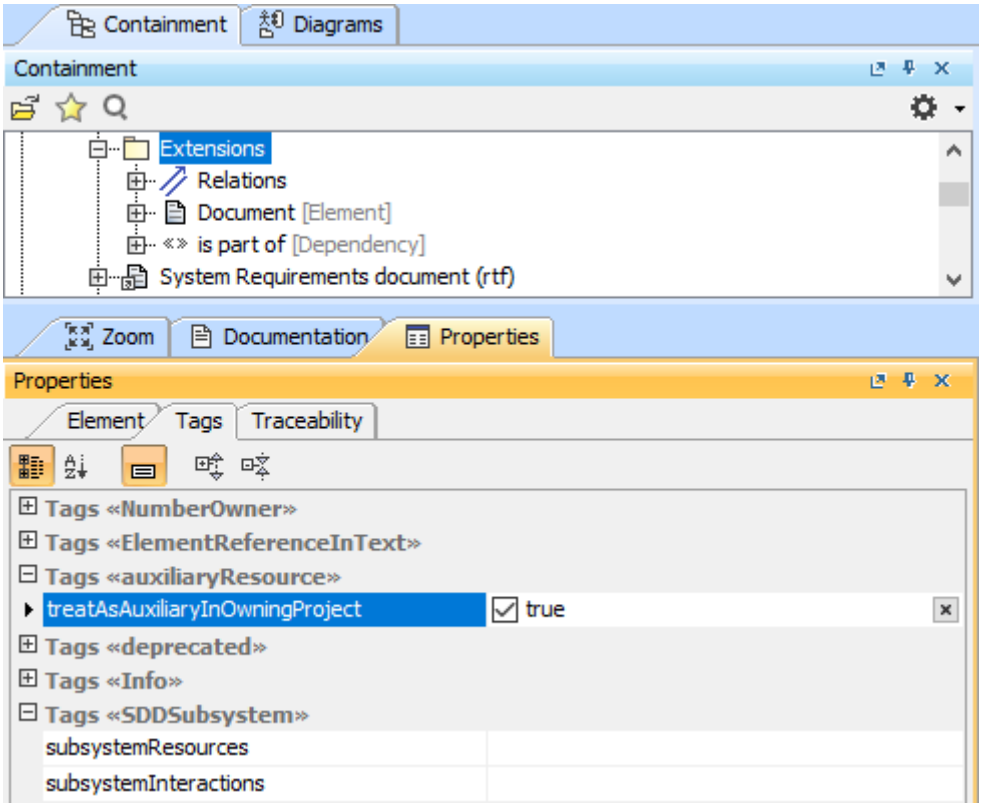
Marking elements as auxiliary

An auxiliary element is as element needed in the model but not necessary to be visible in the Model Browser(see page 29). Usually some profiles are marked as auxiliary and are not displayed by default in the model. You can mark the element in your model as an auxiliary as well.

To mark an element as auxiliary

---

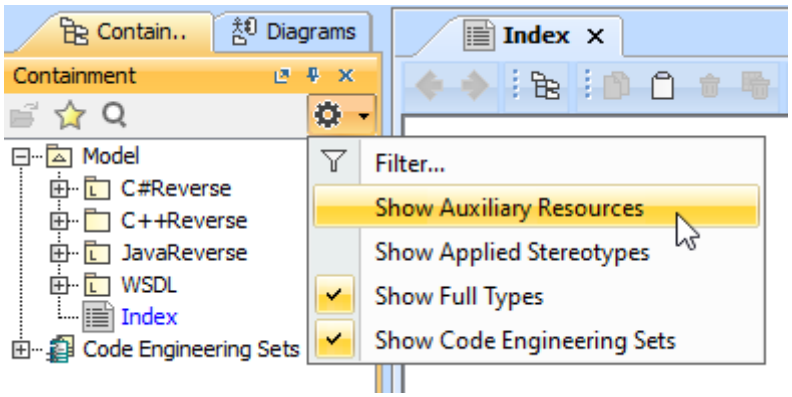
1. Select the element in the Model Browser.
2. In the **Properties** tab, select the **Tags** tab.
3. Set the **treatAsAuxliaryInOwningProject** tag value to *true*.



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Setting the tag value.

The element marked as auxiliary is visible in the Model Browser in an option **Show Auxiliary Resources** is selected in the Options.



The Options drop-down list.

You can specify all auxiliary resources are shown in the model by default by specifying the Environment Options(see page 65) of your tool.

To show auxiliary resources by default in your model

---

1. On the main menu, click **Options** > **Environment**.
2. In the search box, type *auxiliary*. All auxiliary resources related properties appears in the options list.
3. Set the **Show Auxiliary Resources** property value to *true*.

Related pages

- Model Browser(see page 29)
- Customizing environment options(see page 65)

Reordering elements

You can change the order of elements/properties presented in a list, tree, and Specification window, so if you need to move them, you can easily do so using one of the following methods:

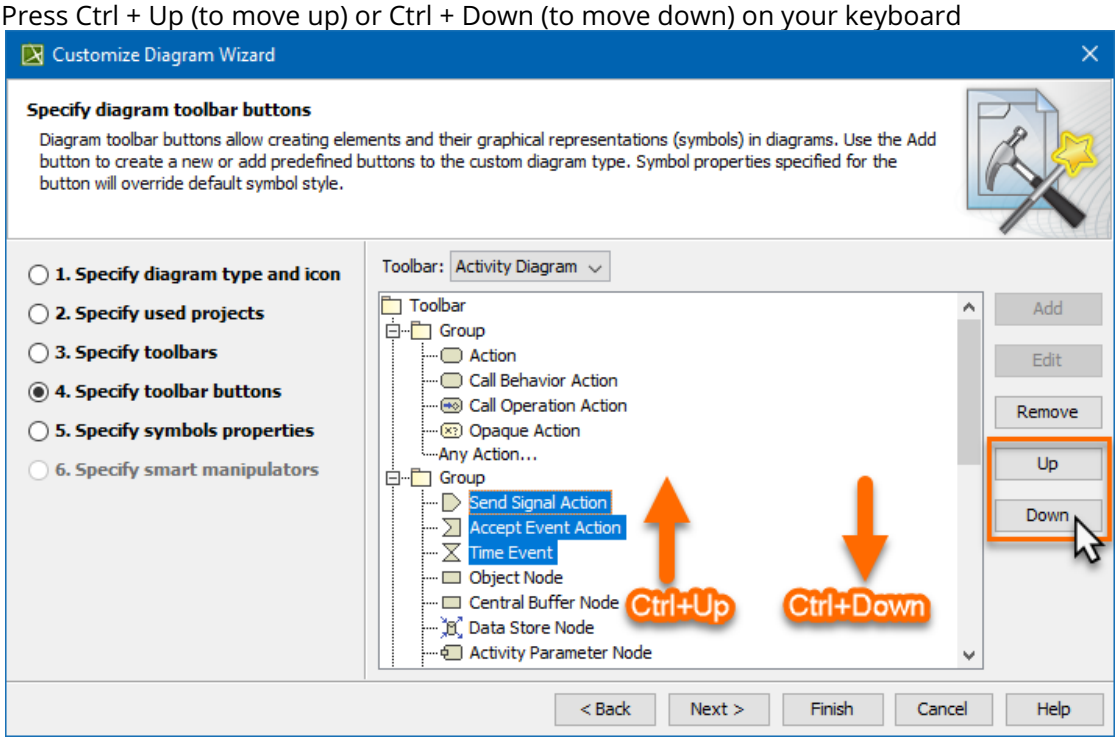
To reorder elements in dialogs

---

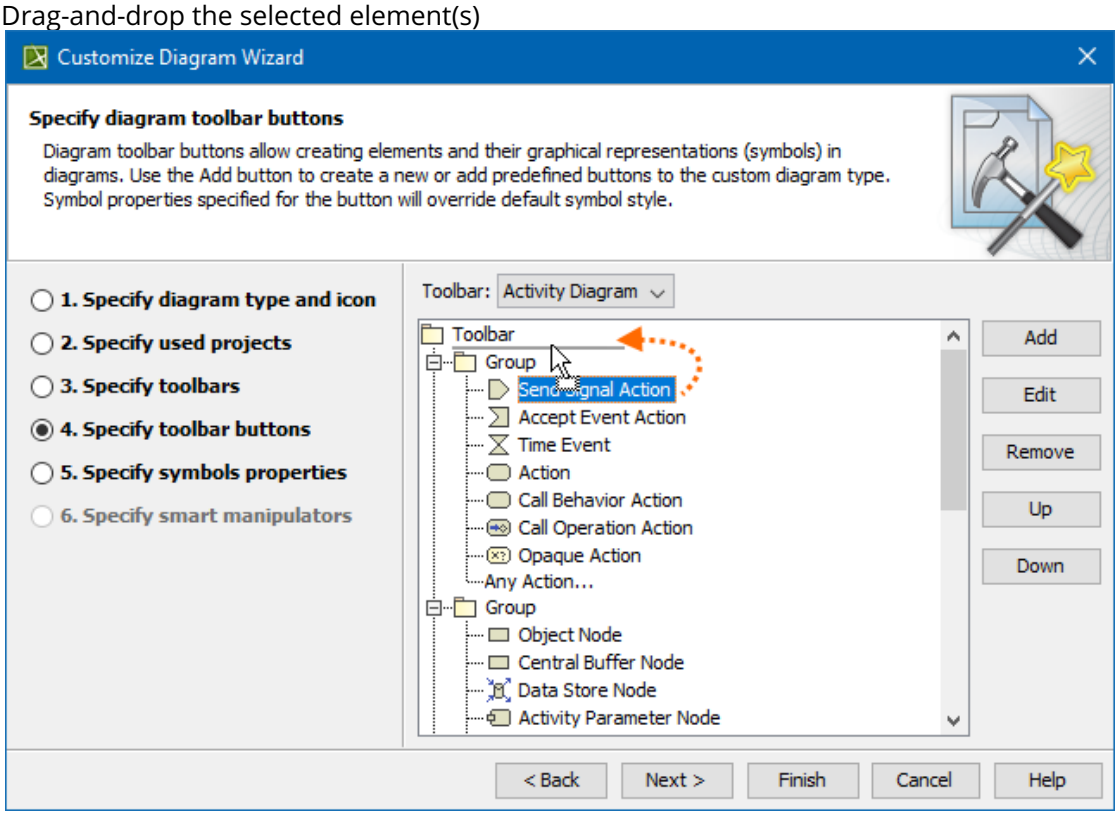
1. Select one or more element(s) to move to the desired place.
2. Do one of the following:

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Click the **Up** (to move up) and **Down** (to move down) buttons on the right side of the dialog
- Press Ctrl + Up (to move up) or Ctrl + Down (to move down) on your keyboard



- Drag-and-drop the selected element(s)



If you need to move multiple elements, press Ctrl on your keyboard and click to select elements to reorder.

Tools

MagicDraw provides a number of tools and wizards to help you quickly and easily perform design and analysis tasks. The following links provide detailed instructions for using them:

Model Merge

Model Merge allows copying changes between different project versions. This functionality is usually needed when there are several branches that reflect different releases or versions of the product, for example, when certain fixes have to be copied from a release branch to the mainstream development

You can merge different versions of both local and server projects.

Projects that use other projects can be compared and merged in one single operation. You do not need to merge each used project and finally the main project one-by-one.

Basic concepts

For a better understanding of the further material, learn the following concepts.

Concept	Description
Source	A project version from which changes are copied.
Target	A project version into which changes are copied.
Ancestor	A common parent project version of both source and target.
Contributor	General description for a project version that participates in the merge as a source or a target.

Related pages

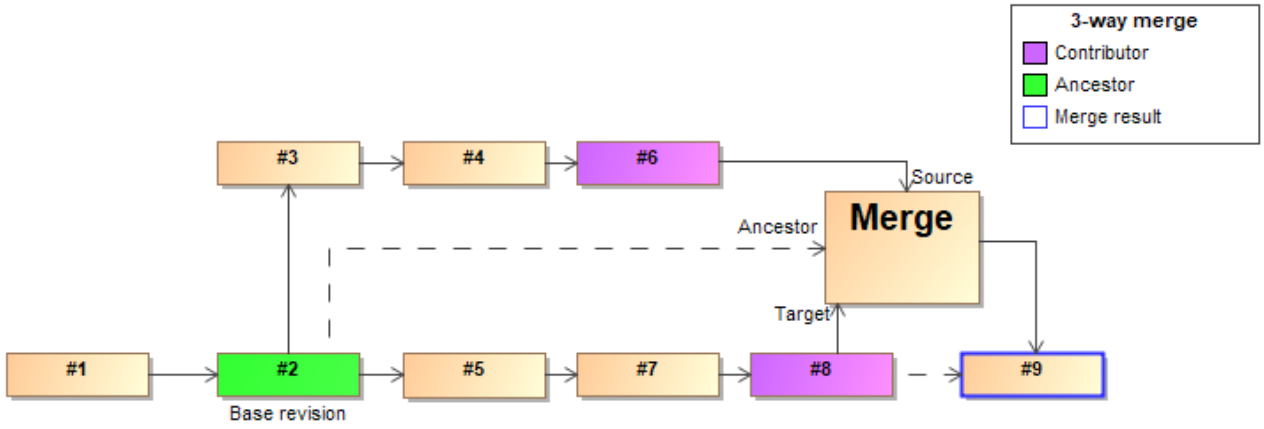
Understanding merge types

There are two types of model merge that can be used to merge two selected project versions (source and target):

- 3-way merge
- 2-way merge

3-way merge

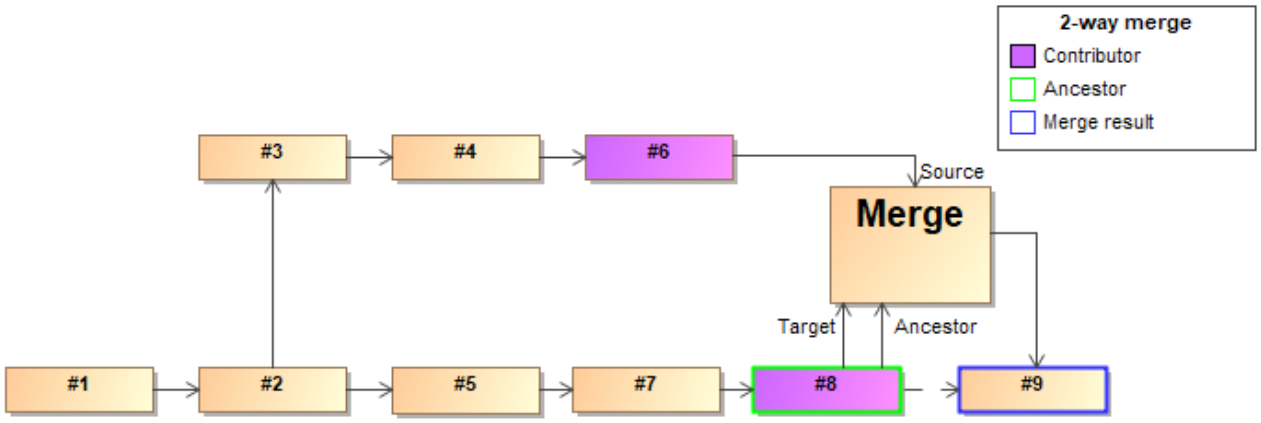
The 3-way merge allows merging both local and server projects. To merge two selected project versions, the common ancestor (a common project version for both target and source) is used. The common ancestor is calculated automatically when working with Teamwork Cloud projects; however, you have to select a correct common ancestor manually for local and Teamwork Server projects. This is the recommended way to merge project versions because the 3-way merge is able to identify the changes made in the source and target project versions separately by comparing changes between the ancestor and both contributors. This allows merging the selected project versions more smoothly.



Visual representation of the 3-way merge

2-way merge

The 2-way merge compares two local or Teamwork Server project versions selected by the user. Typically, the target version of the project is considered to be the common ancestor (a common project version for both target and source). As a result, only the differences between the source and the target project versions are identified.



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

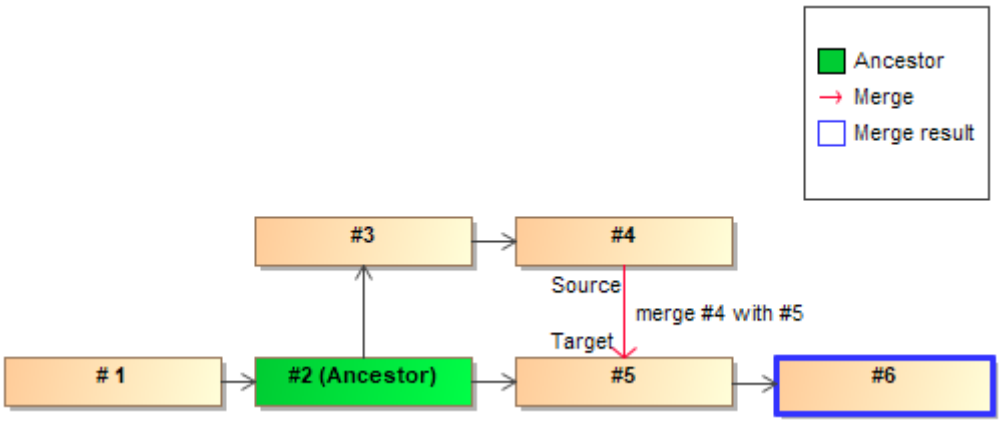
Visual representation of the 2-way merge

Merge in Teamwork Cloud

The 3-way merge is the default merging method used to merge the selected Teamwork Cloud projects. It means that the correct common ancestor version is specified automatically and there is no need to select one manually.

This is how the common ancestor is calculated:

- The lowest version of the trunk a branch was created from (it contains the latest changes both from the source and the target project), e.g., version #2 is considered the common ancestor version of the projects being merged. Once the correct common ancestor for the two selected project versions is calculated, a merge dialog is opened. In the merge dialog, you can review changes and accept/reject them. Once you are done, click Finish Merge and commit the changes to Teamwork Cloud. A new version, e.g., version #6 is created.

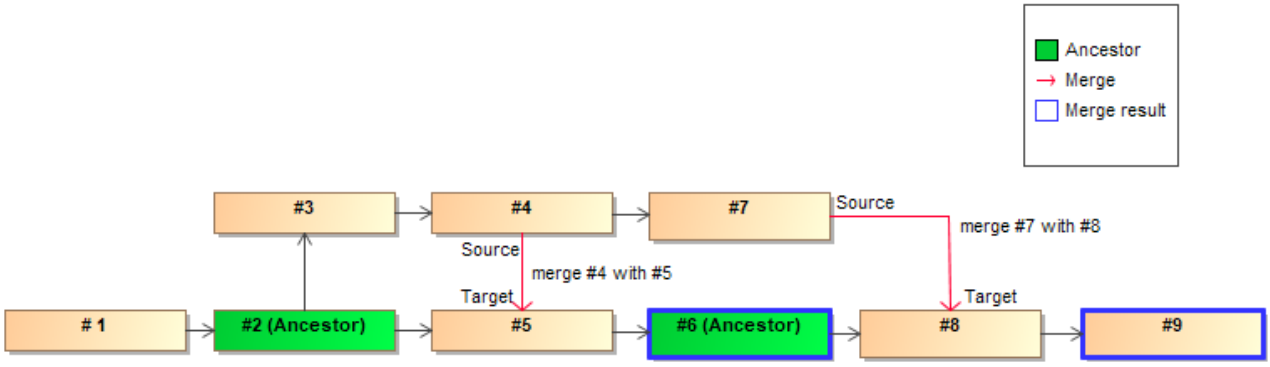


Automatic common ancestor calculation in Teamwork Cloud

Repetitive merge

When two selected branches are merged several times, i.e., it is not the first merge, e.g., version #4 is merged with version #5 and then version #7 is merged with version #8, the latest common version (merge result) is considered a common ancestor. In this case, version #6, which was produced during the very last merge (when version #4 was merged with version #5), is used as the common ancestor of the source and target project versions because it contains data from both selected branches. This helps to prevent repeated merge decisions made in previous merges.

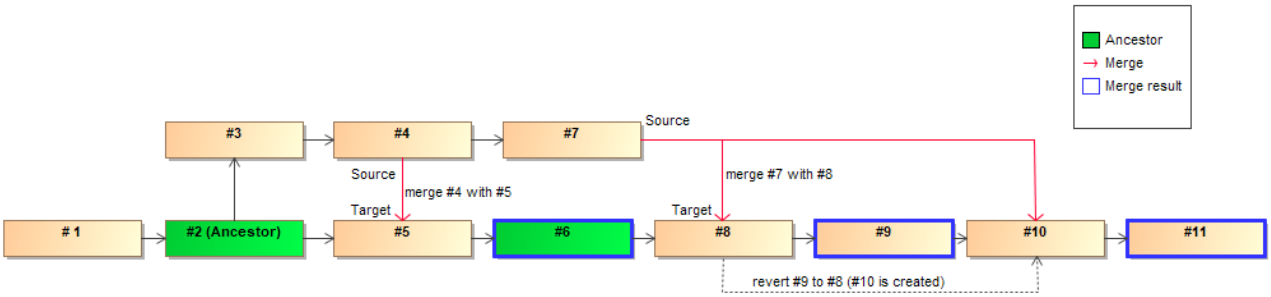
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Automatic common ancestor calculation when there are repetitive merges

Set as latest

When the earlier version of the project is set as the latest version, e.g., version #9 is reverted to version #8 and then the result of two merged versions (e.g., version #10) is selected for merge with another project version, the common version that is closest to that specific reverted project version is considered the common ancestor.



Automatic common ancestor calculation when Set as latest is used

> ⚠ Please note that these common ancestor calculation rules are used only if the modeling tool and the Teamwork Cloud server are both of 19.0 SP3 or higher version.

Related pages

- Understanding change concept(see page 1086)
- Preparing for merge(see page 1089)
- Starting Model Merge(see page 1090)
- Analyzing and managing merge results(see page 1095)
- Finishing and canceling merge(see page 1112)

Understanding change concept

Change is a difference, found between the ancestor and a contributor. Every change can be accepted or rejected, as well as have dependent changes. Changes can conflict with each other or be equivalent.

Merging begins with building a composite change tree, which consists of a model, diagramming, and non-model changes.

Change types

Read the following definitions to get familiar with different change types.

Addition change
If an element has been added to a contributor, an addition change occurs.

Deletion change
If an element has been removed from a contributor, a deletion change occurs.

Modification change
If an element property in a contributor has been modified, a modification change occurs.

> If the **IsAbstract** property value of a class in the ancestor had the default value *false* and the same property value in a contributor has been changed to *true*, a modification change occurs.

There are three types of modification changes:

- **Addition modification change** that occurs when a value is added to a property.
- **Deletion modification change** that occurs when a value is removed from a property.
- **Replacement modification change** that occurs when one value is replaced with another. This type of modification change occurs only for properties that have multiplicity less or equal to 1.

Movement change

If an element owner has been changed in a contributor, a movement change occurs.

> Let's say package A contains some class in the ancestor and package B contains the same class in a contributor. This means that the class has been moved from package A to package B in the contributor. This case is recognized as a movement change.

> Another case of the movement change is when an attribute or an operation that has been owned by class A in the ancestor, becomes the attribute or an operation of class B in a contributor.

Order change
If the order of elements has been changed in a contributor, an order change occurs. Order changes can occur on elements such as attributes, operations, and other ordered elements. Even if a single element in a collection has changed its place, the order change is applied to the entire collection.

Since an element can have several ordered collections, several order changes can occur on a single element.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Let's say class A has attributed a, b, and c in the ancestor. The attribute c has been moved up and placed above attribute a in a contributor. This means that the order of attribute collection in class A has changed in the contributor. This is a case of the order change.

Turn off order change detection

This is the optional phase of the merge procedure. You may not need to perform it.

Order changes can be skipped while merging. For this, you need to specify names of properties wherein order changes should not be detected.

To turn off the order changes detection in specificproperties

1. Fromthe **Options** menu select **Environment**. The **Environment Options** dialog opens.
2. Find the Do Not Detect Order Changes for the option under the **Merge** category in the **General** options group.
3. In the option value cell, specify names of properties wherein order changes should not be detected while merging.

Property names must be written in camel case, for example, ownedAttribute, ownedElement, and so on.

Change states

Every change, whether it is addition, modification, deletion, movement, or order change, can be either accepted or rejected. It is a change state.

All accepted changes will be incorporated into the target. Alternatively, they can be rejected and excluded from the target.

Dependent changes

In some cases, other changes have to be accepted or rejected before accepting or rejecting a selected change. In other words, a selected change sometimes depends on other changes and is called a dependent change.

For a better understanding of the concept of dependent changes, study the following examples.

Let's say a class attribute type has been changed to a type that had been created by another change. In consequence, the attribute type change depends on the change that has created the type. This means that type creation change must be accepted before accepting type modification change.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Let's suppose there is an attribute type change in a contributor. An old type has been deleted and a new type has been added to the contributor. In this case, three changes occur:

- deletion change (for the old type)
- addition change (for the new type)
- modification change (for the property type)

These are also ownership changes, but they are accepted together with deletion and addition changes.
The modification change depends on the addition change, and the deletion change depends on the modification change.Thus accepting the deletion change means also accepting the addition change, the modification change and the deletion change itself.

Conflicting changes

Conflict occurs when two changes are incompatible, i.e., the changes cannot be accepted together. Conflicts can only occur when using 3-way merge.

The following situations will result in a conflict:

- Each contributor has changed a class name or any other element property value.
- One contributor has added an operation to a class and the other contributor deleted the class.
- One contributor has moved a class into one package while the other contributor moved it to another package.

Equivalent changes

Equivalent change is a pair of identical changes that are detected in both source and target when performing a 3-way merge.

Let's say package A contains some class in the ancestor and both contributors contain the same class in package B. This means that the class has been moved from package A to package B in both source and target. This is a case of the equivalent change.

Another case of equivalent changes is when the same element A becomes renamed to B in both contributors.

Related pages

- Understanding merge types(see page 1083)
- Preparing for merge(see page 1089)
- Starting Model Merge(see page 1090)
- Analyzing and managing merge results(see page 1095)
- Finishing and canceling merge(see page 1112)

Preparing for merge

One option to customize the model merge procedure is turning off the order changes detection. This means you can skip the order changes of specified properties while merging.

For server projects

- Partial locking of a project occurs when merging changes of a project in the Teamwork Cloud server. Only changed/updated and merging elements are locked in the project, not the whole project. Thus, actions with other server project elements are not interrupted for other users by locking and merging activities.
- While a user performs the merge process in a server project on the Teamwork Cloud server, committing of project changes to the server is unavailable for other users.

To turn off the order changes detection in specific properties

1. From the **Options** menu, select **Environment**. The **Environment Options** dialog opens.
2. Expand the **General** option group and select **Merge and Compare**.
3. In the value cell of the **Do Not Detect Order Changes for** option, specify the names of the properties for which order changes should not be detected

Property names must be written in camel case, e.g., ownedAttribute, ownedElement, etc. For information about editing these sorts of option values, see Editing long textual values(see page 922).

Related pages

- Understanding merge types(see page 1083)
- Understanding change concept(see page 1086)
- Starting Model Merge(see page 1090)
- Analyzing and managing merge results(see page 1095)
- Finishing and canceling merge(see page 1112)

Starting Model Merge

On this page

- Model merge for Teamwork Cloud(see page 1091)
- Model merge for Teamwork server (see page 1092)
- Local model merge(see page 1094)

Start the merge using either of the following:

- From **Collaborate** > **Merge From**:

You can apply this method of starting a merge exclusively to Teamwork server and Teamwork Cloud projects. For Teamwork Cloud projects, you do not need to specify the ancestor; it will be instantly identified after the source is selected. For Teamwork server projects, the 3-way merge will be applied if you select project versions from different branches. Please note that the 2-way merge will be applied if you select an older version from the same branch or trunk.

- From **Tools** > **Project Merge**.

  You can apply this method of starting a merge to both local and server projects, although used projects can be included in a merge only for server projects. You must specify both the source and the ancestor, and manually choose the 3-way or 2-way merge.

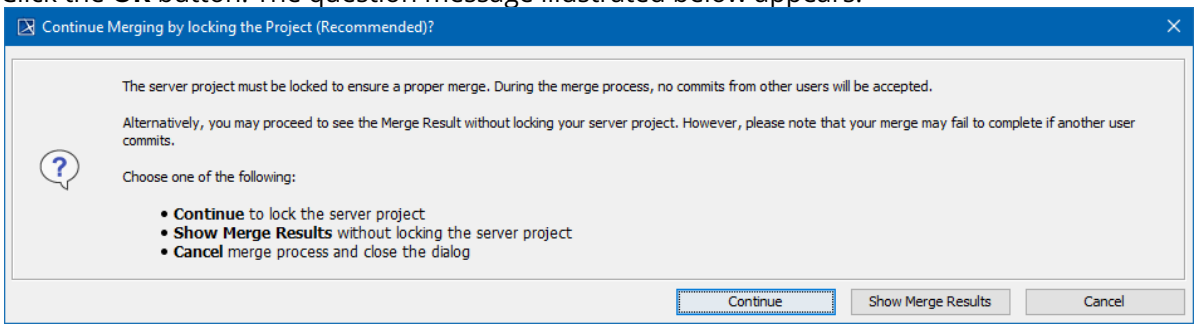
- For Teamwork Cloud projects, the modeling tool automatically applies the 3-way merge, as a merge is only available between project versions from different branches.
- Partial locking of a project occurs when merging changes of a project in the Teamwork Cloud. Only changed/updated and merging elements are locked in the project, not the whole project. Thus, actions with other server project elements are not interrupted for other users by locking and merging activities.
- While the merge is in process in a server project on the Teamwork Cloud server, other users will not be able to commit project changes to the server, as it will be unavailable.

Conflicting changes detected during merge are automatically resolved by accepting changes either from the source or the target version. To specify a contributor, go to **Options** > **Environment** > **General** and then next to **Automatically Resolve Conflicts by Choosing**, select either **Target** or **Source**.

Model merge for Teamwork Cloud

To start a merge of the from **Collaborate** > **Merge From**

1. Open a server project that will be the target(see page 1083).
2. From the **Collaborate** menu, select **Merge From**.
3. In the opened dialog, select a server project that will be the source(see page 1083).
4. Click the **OK** button. The question message illustrated below appears.



5. Do one of the following:
   - Click **Continue** to lock the server project. Other users will not be able to commit changes.
   - Click **Show Merge Results** to proceed directly to the Merge Results, without locking the server project.
   - Click **Cancel** merge process and close the dialog.
6. The project merge will start. If changes are found, the **Merge** window will open.

7. Proceed to Analyzing and managing merge results(see page 1095).

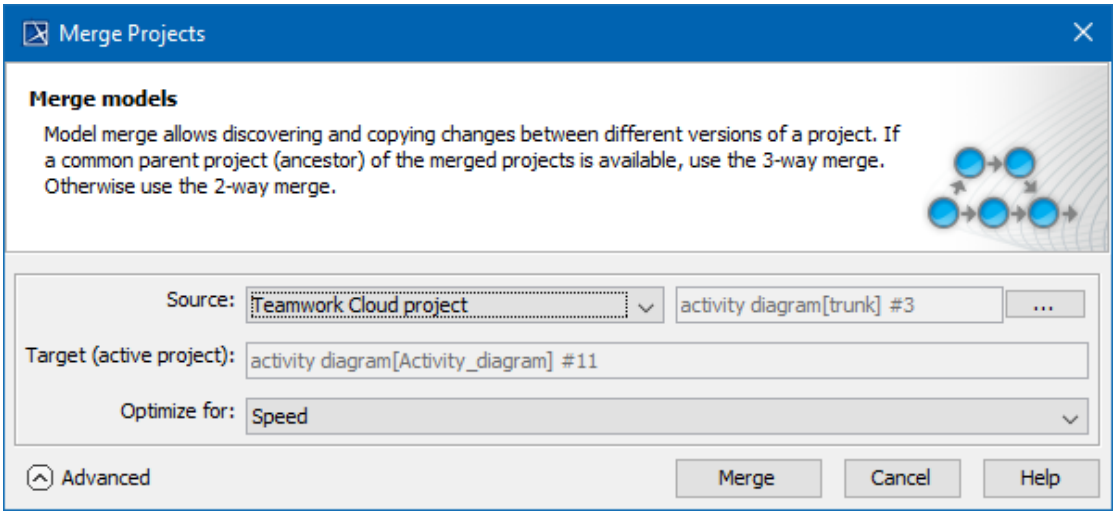
To start merge from **Tools** > **Project Merge**

---

1. Open a project that will be the target(see page 1083) .
2. From the **Tools** menu, select **Project Merge**. The **Merge Projects** dialog will open within the active project selected as the target.
3. Select a project that will be the source(see page 1083).
4. Click the **Advanced** button to see more merge options. Otherwise, go to step #6(see page 1092) .
5. In the **Optimize for** drop-down list, select either:

   - **Speed**, to merge the projects faster. This will require more memory.
   - **Memory,** to decrease memory usage while merging the projects if your computer does not have enough memory. This will slow down the merge procedure. It will not include the differing diagrams/elements count. If you need that information, select optimize for **Speed**.

   > You can also specify the **Optimize for** option in the **Environment Options** dialog. Find this option under the **Merge and Compare** category in the **General** options group.



6. Click the **Merge** button. The question message illustrated in the previous section will appear.
7. Do one of the following:
   - Click **Continue** to lock the server project and continue the merge process. Other users will not be able to commit changes.
   - Click **Show Merge Results** to proceed directly to the Merge Results.
8. The project merge will start. If changes are found, the **Merge** window will open. Proceed to Analyzing and managing merge results(see page 1095).

Model merge for Teamwork server

To start a merge from **Collaborate** > **Merge From**

---

1. Open a server project that will be the target (see page 1083).
2. From the **Collaborate** menu, select **Merge From**.
3. In the opened dialog, select a server project that will be the source(see page 1083).

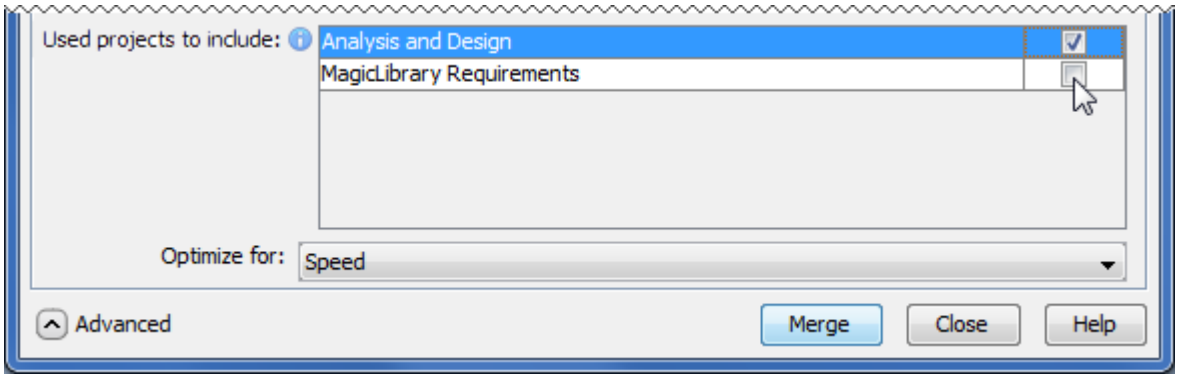
4. In the same dialog, click the **Advanced** button to see more merge options. Otherwise, go to step #7(see page 1093) .
5. In the **Optimize for** drop-down list, select either:

   - **Speed**, to merge the projects faster. This will require more memory.
   - **Memory,** to decrease memory usage while merging the projects if your computer does not have enough memory. This will slow down the merge procedure. It will not include the differing diagrams/elements count. If you need that information, select optimize for **Speed**.

   > You can also specify the **Optimize for** option in the **Environment Options** dialog. Find this option under the **Merge** category in the **General** options group.

6. In the **Used projects to include** list, select the used projects of the target you want to merge with the appropriate used projects of the selected source. To select a used project, click the check box next to the used project name (see the following figure).

   > The list includes only the projects that:
   > - are used by the target in the read-write accessibility mode
   > - have the same composition in both contributors
   > - do not have private parts

7. Click the **Merge** button. The project merge will start. If changes are found, the **Merge** window will open. Proceed to Analyzing and managing merge results(see page 1095).

To start merge from **Tools** > **Project Merge**

---

1. Open a project that will be the target(see page 1083) .
2. From the **Tools** menu, select **Project Merge**. The **Merge Projects** dialog will open within the active project selected as the target.
3. Select the type of merge.

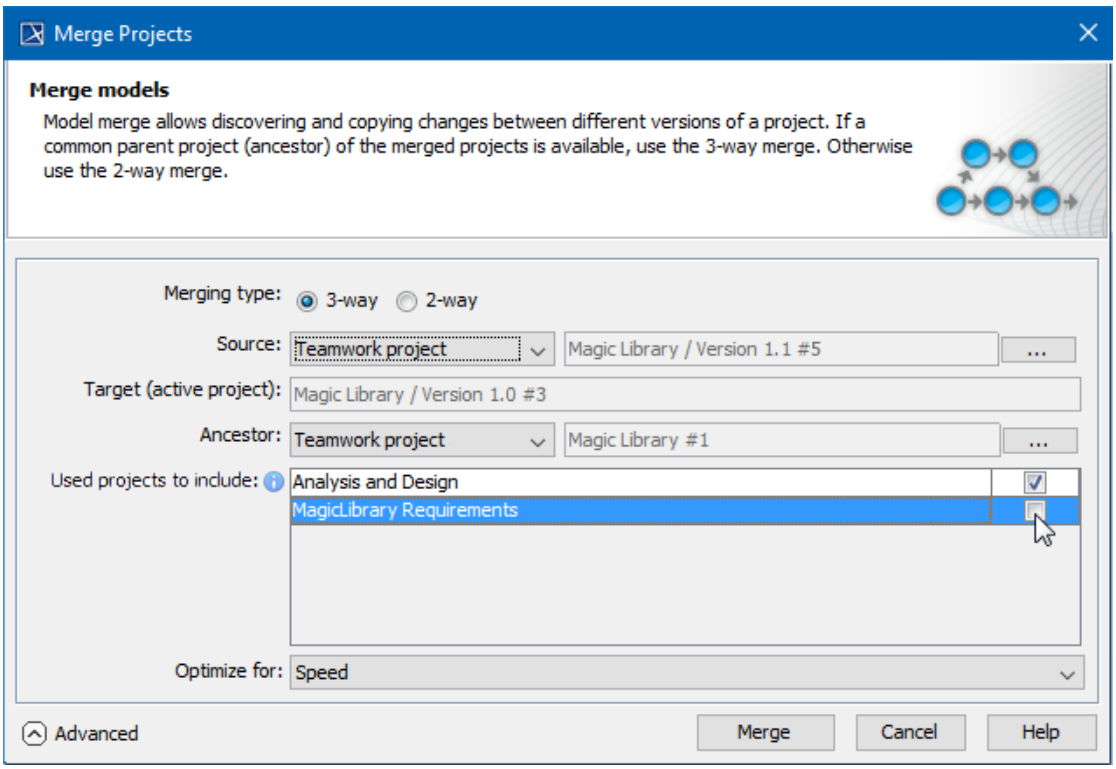
   > Learn about 3-way and 2-way merge in Choosing merge type.(see page 1083)

4. Select a project that will be the source(see page 1083) .
5. If you selected 3-way merge, choose a project that will be the ancestor.
6. Click the **Advanced** button to see more merge options. Otherwise, go to step #9(see page 1094) .
7. In the **Optimize for** drop-down list, select either:

   - **Speed**, to merge the projects faster. This will require more memory.

- **Memory,** to decrease memory usage while merging the projects if your computer does not have enough memory. This will slow down the merge procedure. It will not include the differing diagrams/elements count. If you need that information, select optimize for **Speed**.

> You can also specify the **Optimize for** option in the **Environment Options** dialog. Find this option under the **Merge and Compare** category in the **General** options group.



8. In the **Used projects to include** list, select the used projects of the target you want to merge with the appropriate used projects of the selected source. To select a used project, click the check box next to the used project name (see the following figure).

> The list is available only for server projects and includes only the projects that:
> - are used by the target in the read-write accessibility mode
> - have the same composition in both contributors
> - do not have private parts

9. Click the **Merge** button. The project merge will start. If changes are found, the **Merge** window will open. Proceed to Analyzing and managing merge results(see page 1095).

Local model merge

To start merge from **Tools** > **Project Merge**

---

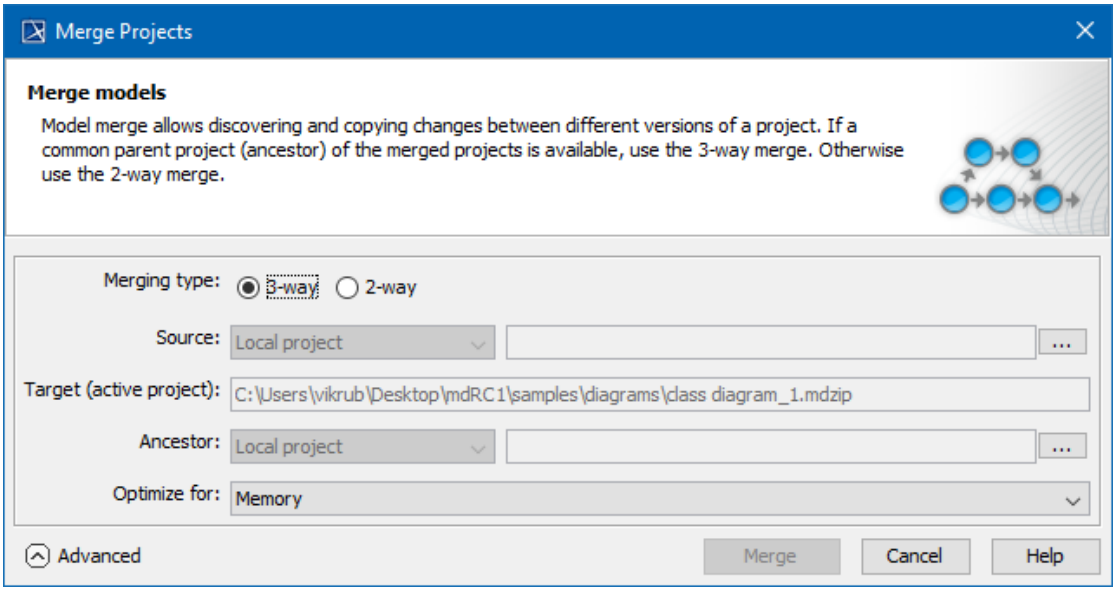
1. Open a project that will be the target(see page 1083) .
2. From the **Tools** menu, select **Project Merge**. The **Merge Projects** dialog will open within the active project selected as the target.
3. Select the type of merge.

> Learn about 3-way and 2-way merge in Choosing merge type (see page 1083).

4. Select a project that will be the source(see page 1083).
5. If you selected 3-way merge, choose a project that will be the ancestor.
6. Click the **Advanced** button to see more merge options. Otherwise, go to step #8(see page 1094).
7. In the **Optimize for** drop-down list, select either:

   - **Speed**, to merge the projects faster. This will require more memory.
   - **Memory,** to decrease memory usage while merging the projects if your computer does not have enough memory. This will slow down the merge procedure. It will not include the differing diagrams/elements count. If you need that information, select optimize for **Speed**.

   > You can also specify the **Optimize for** option in the **Environment Options** dialog. Find this option under the **Merge and Compare** category in the **General** options group.



8. Click the **Merge** button. The project merge will start. If changes are found, the **Merge** window will open. Proceed to Analyzing and managing merge results(see page 1095).

Related pages

- Understanding merge types(see page 1083)
- Understanding change concept(see page 1086)
- Preparing for merge(see page 1089)
- Analyzing and managing merge results(see page 1095)
- Finishing and canceling merge(see page 1112)

Analyzing and managing merge results

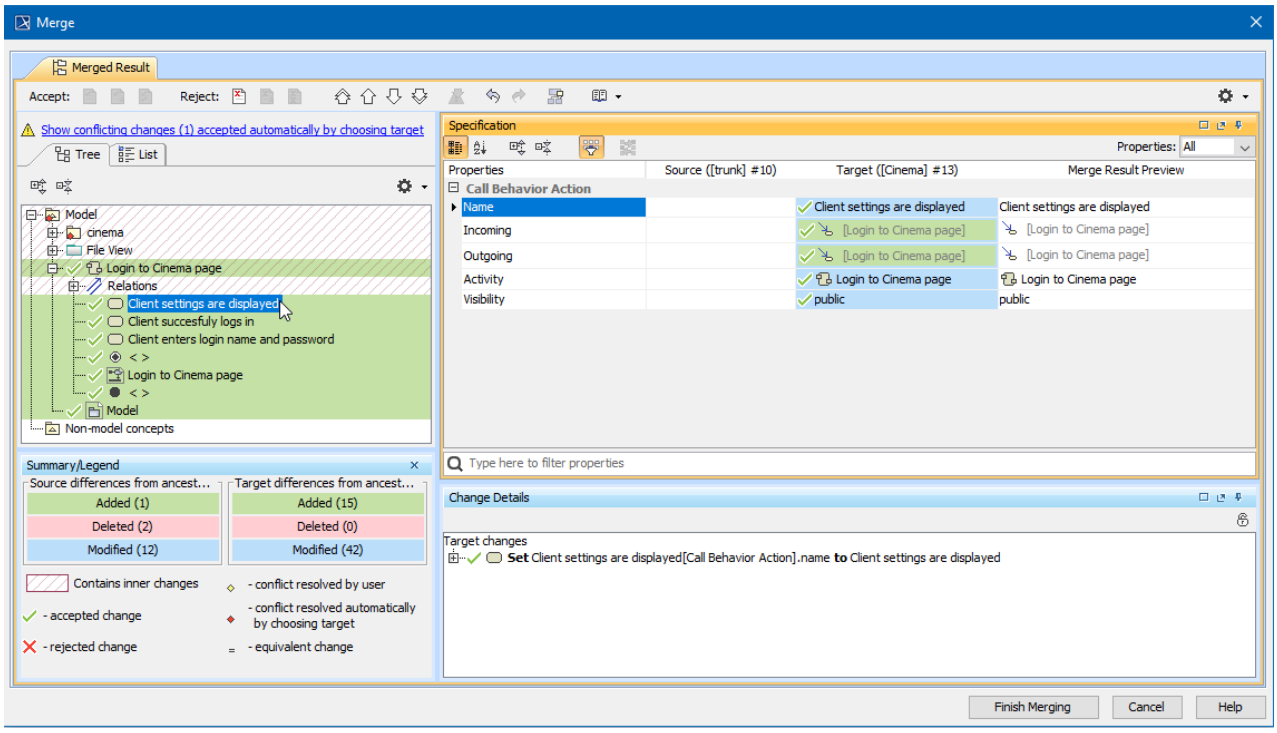
Merge results are displayed in the **Merge** window. Read about analyzing and managing model merge results in

- Getting merge summary information and changes legend(see page 1096)
- Understanding content of the Merged Result panel(see page 1097)
- Inspecting element property changes in the Specification panel(see page 1101)
- Inspecting changes in the Change details panel(see page 1103)
- Displaying and navigating through changes(see page 1104)
- Quickly navigating through conflicting changes(see page 1106)
- Accepting and rejecting changes using shortcut menu(see page 1107)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Viewing and analyzing changes of modified diagrams(see page 1108)
- Generating a differences report(see page 1111)

If you do not need to perform the analysis and management of merge results, proceed to Finishing and canceling merge(see page 1112).

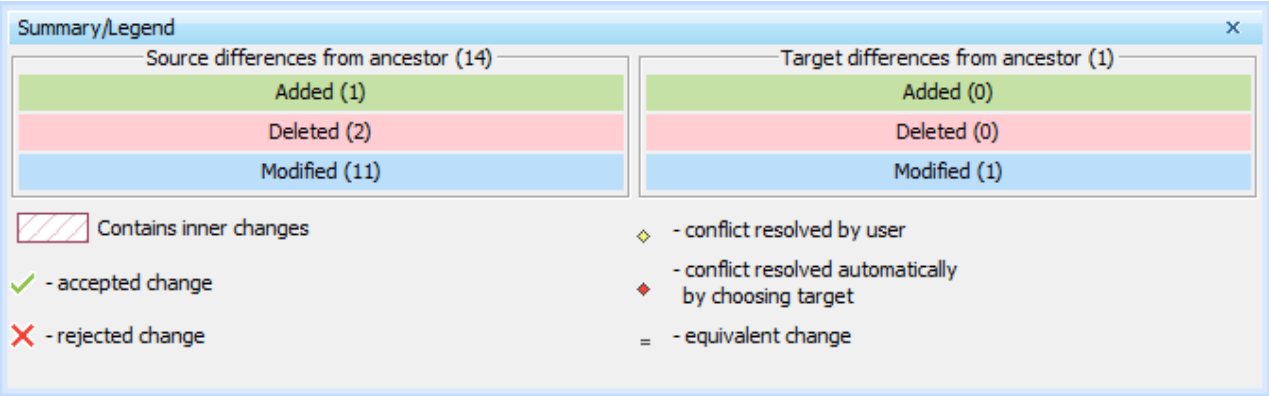


Example of the Merge window.

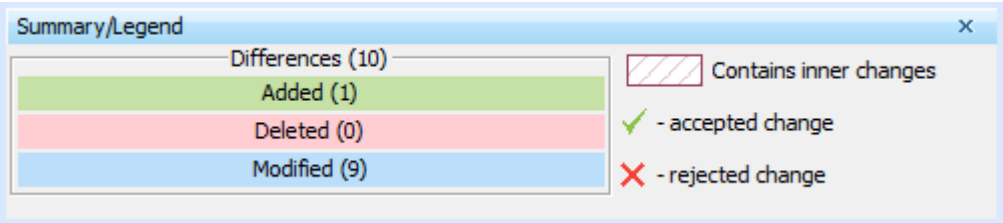
Getting merge summary information and changes legend

The **Summary/Legend** panel is located at the bottom left of the **Merge** window. The panel helps to understand the marking and highlighting of elements in both the **Merged Result** tree/list and the **Specification** panel.

On the **Summary/Legend** panel, for 3-way merge, you can see how many differences have been detected between the ancestor and the source, as well as between the ancestor and the target.



For 2-way merge, you can see how many differences have been detected between two versions, and what kind of changes were detected.



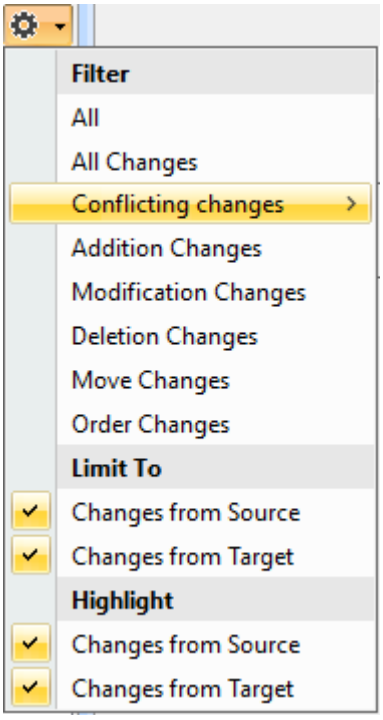
Related pages

- Analyzing and managing merge results(see page 1095)
  - Understanding content of the Merged Result panel(see page 1097)
  - Inspecting element property changes in the Specification panel(see page 1101)
  - Inspecting changes in the Change details panel(see page 1103)
  - Displaying and navigating through changes(see page 1104)
  - Quickly navigating through conflicting changes(see page 1106)
  - Accepting and rejecting changes using shortcut menu(see page 1107)
  - Viewing and analyzing changes of modified diagrams(see page 1108)
  - Generating a differences report(see page 1111)

Understanding content of the Merged Result panel

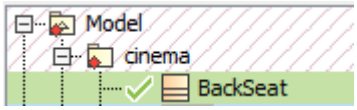
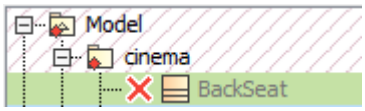
Elements with all change types are displayed in the **Merged Result** tree/list that is located at the top left of the **Merge** window.

You can also filter the elements displayed in the **Merged Results** tree/list by type of changes. Use the **Changes filter** drop-down list to filter out a specific type of changes.



Understanding element marking

When a change occurs on a single element, the ticks and crosses before element icon indicate the state of the change which can be either accepted or rejected.

Marking	Change state	Example
BackSeat (green tick before the element icon)	Accepted	 A new element had been created in one of the contributors and its addition was accepted.
BackSeat (red cross before the element icon)	Rejected	 A new element had been created in one of the contributors and its addition was rejected.

To understand the concept of change states, refer to Change states(see page 0).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

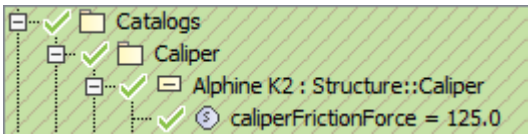
Elements with resolved conflicting changes are marked as shown in the following table. Owners of these elements are also marked.

Marking	Change applied by	Example
Pad:Pad (red diamond on the lower-left corner of the element icon)	System	Model Pad:Thin The element had a conflicting modification change which was resolved automatically.
Pad:Pad (yellow diamond on the lower-left corner of the element icon)	User	< > Pad:Pad The element had a conflicting modification change which was resolved by the user.

Recognizing addition changes

For the addition change description, refer to Addition change(see page 0).

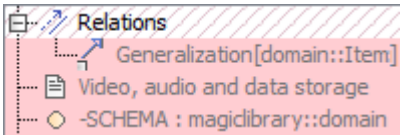
Added elements are highlighted in green background.



Recognizing deletion changes

For the deletion change description, refer to Deletion change(see page 0).

Deleted elements are highlighted in red background.

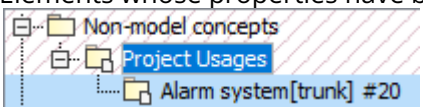


Recognizing modification changes

For the modification change description, refer to Modification change(see page 0).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Elements whose properties have been changed are highlighted in blue.



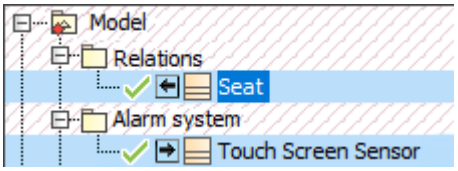
To see what properties have been modified, click the element and see changes in the **Specification** panel.

For more information about using the Specification panel, see Inspecting changes in the Change details panel(see page 1103).

Recognizing movement changes

For the movement change description, refer to Movement change(see page 0).

Elements with movement changes are highlighted in the same blue background that is used to highlight modification changes. Since movement changes are some kind of modification changes. Icon of an element with movement changes is represented with an arrow symbol on the left.

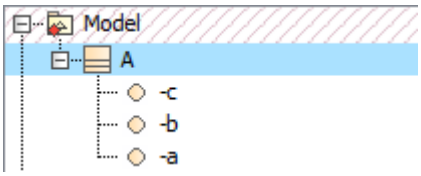


You can navigate from the element original location to the new one (and vice versa) using the commands on the element shortcut menu.

Recognizing order changes

For the order change description, refer to Order change(see page 0).

Elements with order changes are highlighted in the same blue background that is used to highlight modification changes, since order changes are some kind of modification changes.
After accepting the change, all the elements in the collection are reordered so that it would be the same as in one of the contributors.

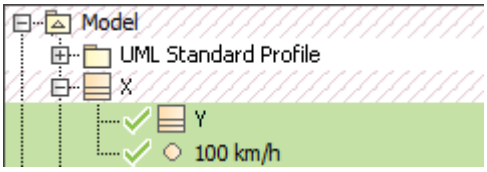


You can turn off the order changes detection. For more about it, see Turning on showing merged diagram annotations.(see page 0)

Recognizing elements with changed owned elements

Elements whose owned elements (in any nesting level) have changes are highlighted using red diagonals.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



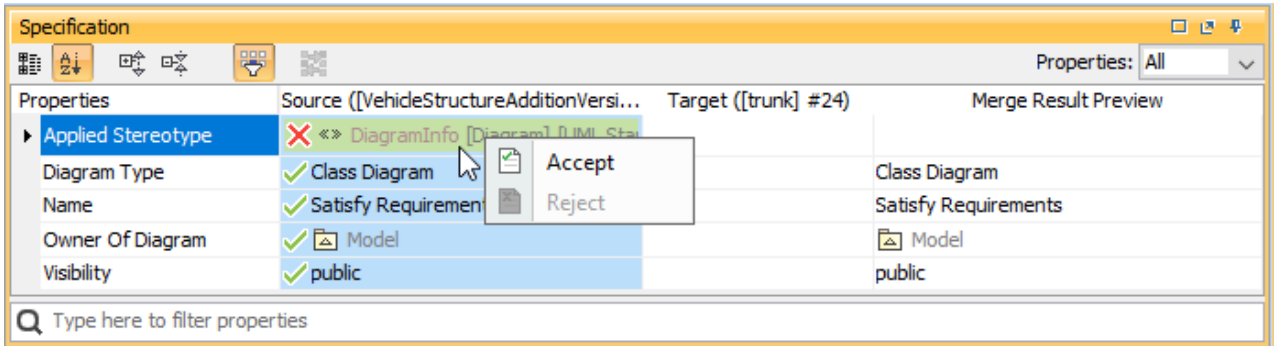
Related pages

- Getting merge summary information and changes legend(see page 1096)
- Inspecting element property changes in the Specification panel(see page 1101)
- Inspecting changes in the Change details panel(see page 1103)
- Displaying and navigating through changes(see page 1104)
- Quickly navigating through conflicting changes(see page 1106)
- Accepting and rejecting changes using shortcut menu(see page 1107)
- Viewing and analyzing changes of modified diagrams(see page 1108)
- Generating a differences report(see page 1111)

Inspecting element property changes in the Specification panel

You can see all the details of the element property changes in the **Specification** panel that is located at the top right of the **Merge** window. All you need to do is selecting the element in the **Merged Result** tree or list.

As you can see in the following figure, the first column of the panel contains a list of element property names, and other columns display their value changes that have been detected in both source and the and target. Third column Merge Results Preview shows the final merge result. Properties changes can be accepted or rejected in this panel.

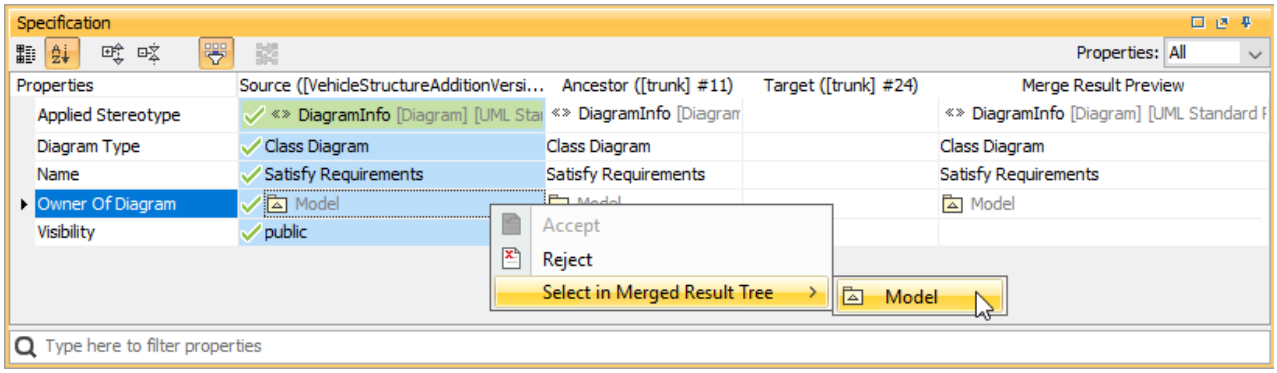


To navigate from the **Specification** panel to the **Merged Result** tree, do either

---

- In the **Properties** column, right-click the name of a property, which references other elements. Then on its shortcut menu, point to **Select in Merged Result Tree** and choose an element to which you want to navigate.

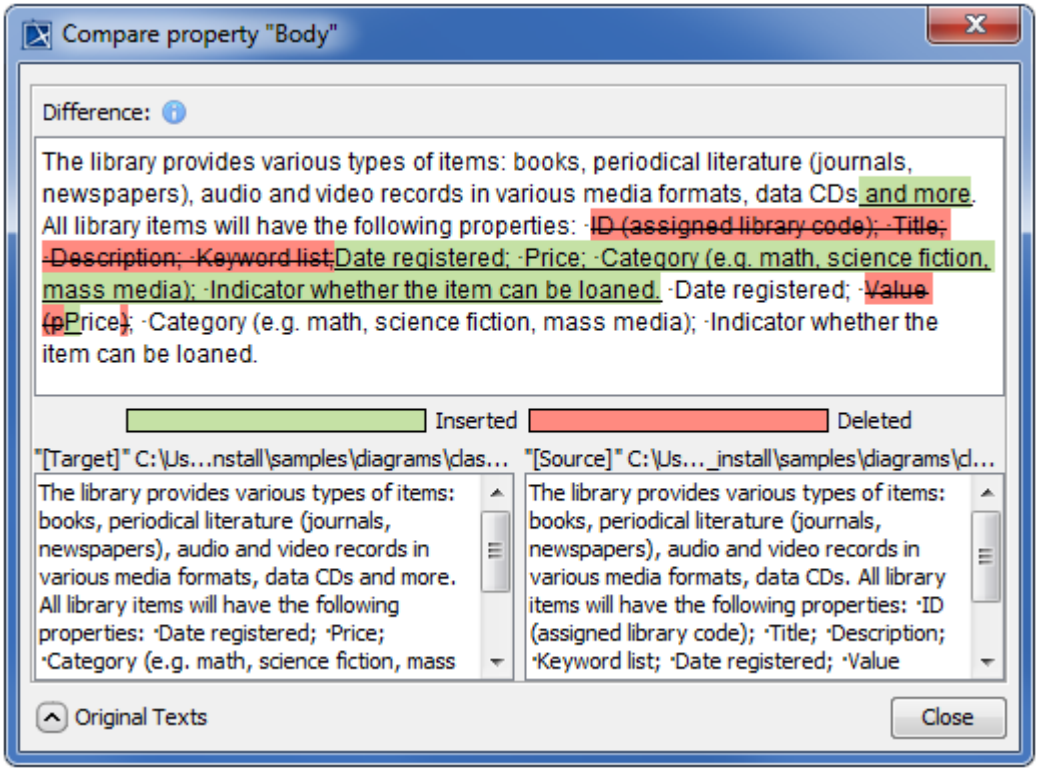
- In the **Source**, **Ancestor**, **Target** or **Merge Result Preview** column, right-click the value of a property, which references other elements. Then on its shortcut menu choose an element to which you want to navigate.



The **Ancestor** column by default is hidden. To show it in the **Specification** panel, click on the **Merge** toolbar and select **Show Ancestor**.

Inspecting differences between textual values

You can see exact changes between textual values of element properties, such as element documentation, comments, pre- and post-conditions of a use case. Differences of compared texts can be displayed in a single dialog, where inserted and deleted parts of the text are highlighted.



To see the differences of compared texts, do either

- Select a property with a modified text on the **Specification** panel and then click on the toolbar.
- Double-click the property on the **Specification** panel. The **Compare property** dialog opens.

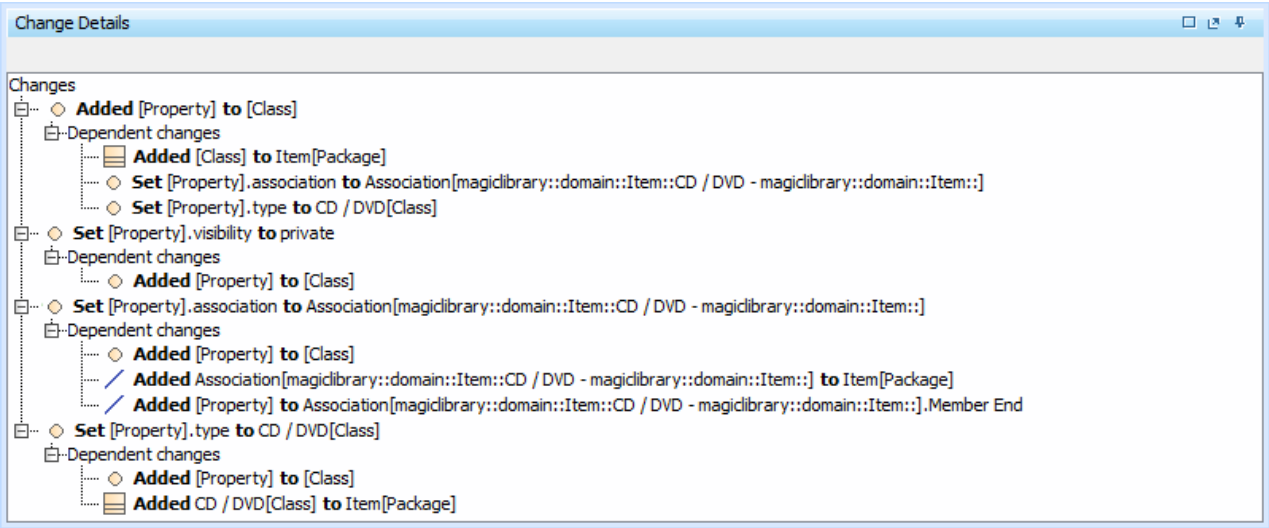
Related pages

- Understanding merge types(see page 1083)
- Understanding change concept(see page 1086)
- Preparing for merge(see page 1089)
- Starting Model Merge(see page 1090)
- Analyzing and managing merge results(see page 1095)
- Finishing and canceling merge(see page 1112)

Inspecting changes in the Change details panel

The **Change details** panel displays a tree structure reflecting the changes that occurred either on the element selected in the **Merged Result** tree/list or on the property selected in the **Specification** panel. The panel is located under the **Specification** panel on the **Merge** window.

The panel contains a tree structure that has two top-level branches – Source changes and Target changes – in 3-way merge, and 1 top-level branch – Source changes – in 2-way merge. In this panel, you can accept or reject changes using the shortcut menu.



In this panel, you can see dependent changes. In order to see them, expand the **Dependent changes** branch under an element with dependent changes.

For the dependent changes description, refer to Dependent changes(see page 0).

Related pages

- Getting merge summary information and changes legend(see page 1096)
- Understanding content of the Merged Result panel(see page 1097)
- Inspecting element property changes in the Specification panel(see page 1101)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Displaying and navigating through changes(see page 1104)
- Quickly navigating through conflicting changes(see page 1106)
- Accepting and rejecting changes using shortcut menu(see page 1107)
- Viewing and analyzing changes of modified diagrams(see page 1108)
- Generating a differences report(see page 1111)

Displaying and navigating through changes

You can customize the content of the **Merged Result** tree/list as well as navigate through changes in both the **Merged Results** tree/list and the **Specification** panel using the toolbar that is located at the top of the **Merge** window.

The toolbar buttons for customizing the **Merged Results** tree/list and navigating through changes are described in the following tables.

Button / Shortcut keys	Tooltip text	Description
	Accept the selected change and go to the next change	Click to accept the selected element change and go to the next change (might be a property or the element).
	Accept the selected change, its property changes and go to next change	Click to accept the selected element change together with the element property changes and go to the next changed element.
	Accept the selected change, its property changes, all subelement changes and go to next change	Click to accept all changes from both contributors starting from the selected element and go to the next changed element (which is not a subelement). For example, if the root package *Model* selected, then all changes for the whole project will be accepted.
	Reject the selected change and go to the next change	Click to reject the selected element change and go to the next change (might be a property or the element).
	Reject the selected change, its property changes and go to next change	Click to reject the selected element change together with the element property changes and go to the next changed element.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

	Reject the selected change, its property changes, all subelement changes and go to next change	Click to reject all changes from both contributors starting from the selected element and go to the next changed element (which is not a subelement). For example, if the root package *Model* is selected, then all changes for the whole project will be rejected.
Alt + Home	Go To First Change	Click to select the first change. The button is not available if the first change is already selected.
Alt + Up arrow	Go To Previous Change	Click to select the previous change.
Alt + Down arrow	Go To Next Change	Click to select the next change.
Alt + End	Go To Last Change	Click to select the last change. The button is not available if the last change is already selected.
Alt + Page Down	Go To Next Conflict	Click to select the next conflict. The button is not available if there are no conflicts.
	Annotate Merged Diagram	Click to turn on showing merged diagram annotations.
	Report	Click to generate a Project Merge differences report in Microsoft Word format.
Options menu		
Icon	**Option**	**Description**
>	Filter	Click to open the **Select Element Type** dialog. Clear the checkbox next to an element that you want to hide in the **Merged Result** tree/list.

>	Show Auxiliary Resources	Click to show or hide in the **Merged Result** tree the profiles with the «auxiliaryResources» stereotype applied (for example, UML Standard Profile).
	Show Elements from Used projects	Click to show or hide in the **Merged Result** tree/list the elements from used projects.
	Show Source	Click to show or hide **Source column** in the **Specification** panel. By default, this column is shown.
	Show Ancestor	Click to show or hide **Ancestor column** in the **Specification** panel. By default, this column is hidden.
	Show Target	Click to show or hide **Target column** in the **Specification** panel. By default, this column is shown.
	Show Merge Result Preview	Click to show or hide **Show Merge Result Preview** column in the **Specification** panel. This column shows final changes in the merged project and by default, this column is shown.
	Summary/Legend	Click to show or hide the **Summary/Legend** panel.
	Reset Windows Configuration	Select to place all tabs and windows to their default positions.

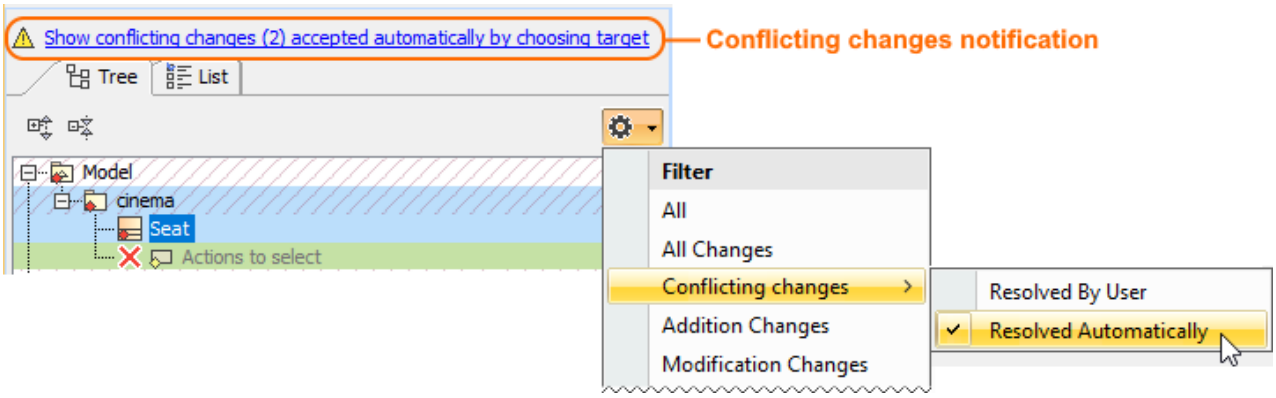
Related pages

- Analyzing and managing merge results(see page 1095)
- Getting merge summary information and changes legend(see page 1096)
- Understanding content of the Merged Result panel(see page 1097)
- Inspecting element property changes in the Specification panel(see page 1101)
- Inspecting changes in the Change details panel(see page 1103)
- Quickly navigating through conflicting changes(see page 1106)
- Accepting and rejecting changes using shortcut menu(see page 1107)
- Viewing and analyzing changes of modified diagrams(see page 1108)
- Generating a differences report(see page 1111)

Quickly navigating through conflicting changes

You can quickly navigate through the conflicting changes in the **Merged Result** tree/list. Use the filter on the right of the **Merged Result** tree/list to filter conflicting changes by type. You can select to filter

conflicting changes resolved by the user or resolved automatically. Moreover, click conflicting changes notification and it will filter conflicting changes resolved automatically.



Related pages

- Getting merge summary information and changes legend(see page 1096)
- Understanding content of the Merged Result panel(see page 1097)
- Inspecting element property changes in the Specification panel(see page 1101)
- Inspecting changes in the Change details panel(see page 1103)
- Displaying and navigating through changes(see page 1104)
- Accepting and rejecting changes using shortcut menu(see page 1107)
- Viewing and analyzing changes of modified diagrams(see page 1108)
- Generating a differences report(see page 1111)

Accepting and rejecting changes using shortcut menu

You can accept and reject changes using the toolbar that is located at the top of the **Merged Result** window or using the shortcut menu in all panels.

The following table lists the commands of the shortcut menu that can be opened when an element is selected in the **Merged Result** tree/list.

Command	Description
Accept	Click to accept the selected change. If the change has equivalent changes, they are accepted too.
Reject	Click to reject the selected change. If the change has equivalent changes, they are rejected too.
Accept With Properties	Click to accept the selected change and its property changes.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Command	Description
Reject With Properties	Click to reject the selecting change and its property changes.
Accept Recursively	Click to accept the selected change, its property changes, and all owned element changes in the **Merged Result** tree view.
Reject Recursively	Click to reject the selected change, its property changes, and all owned element changes in the **Merged Result** tree view.
Mark as Resolved	Click to mark the conflicting change as resolved by the user, but not automatically.
Select Original Location(s)	If the selected instance of a moved element is in the new location, the **Select Original Location(s)** command appears on the shortcut menu. Click it to navigate to and select the original location(s) of the moved element in the **Merged Result** tree.
Select New Location(s)	If the selected instance of a moved element is in its original location, the **Select New Location(s)** command appears on the shortcut menu. Click it to navigate to and select a new location(s) of the moved element in the **Merged Result** tree.

Related Pages:

- Analyzing and managing merge results(see page 1095)
  - Getting merge summary information and changes legend(see page 1096)
  - Understanding content of the Merged Result panel(see page 1097)
  - Inspecting element property changes in the Specification panel(see page 1101)
  - Inspecting changes in the Change details panel(see page 1103)
  - Displaying and navigating through changes(see page 1104)
  - Quickly navigating through conflicting changes(see page 1106)
  - Viewing and analyzing changes of modified diagrams(see page 1108)
  - Generating a differences report(see page 1111)

Viewing and analyzing changes of modified diagrams

The different diagram versions are displayed in separate **Difference Viewer** tabs in the **Merge** window next to the **Merged Result** tree (see the following figure).

To open diagram difference viewers of a modified diagram

- Double-click the modified diagram in the **Merged Result** tree.

One, two or three new tabs open in the **Merge** window. In 3-way merge, the ancestor's tab is opened by default, the source's tab is opened, if changes have been detected in the source, and the target's tab is opened, if changes have been detected in the target. In 2-way merge, the target tab is opened by default. If a diagram is only available in source, only the source tab opens in the difference viewer .

All the diagram difference viewers are used to display the following changes:

- One or more symbols were moved or modified on the diagram.
- A new symbol was added to the diagram.
- A symbol was deleted from the diagram.
- The number of inner model elements has changed.
- A model element has been modified.

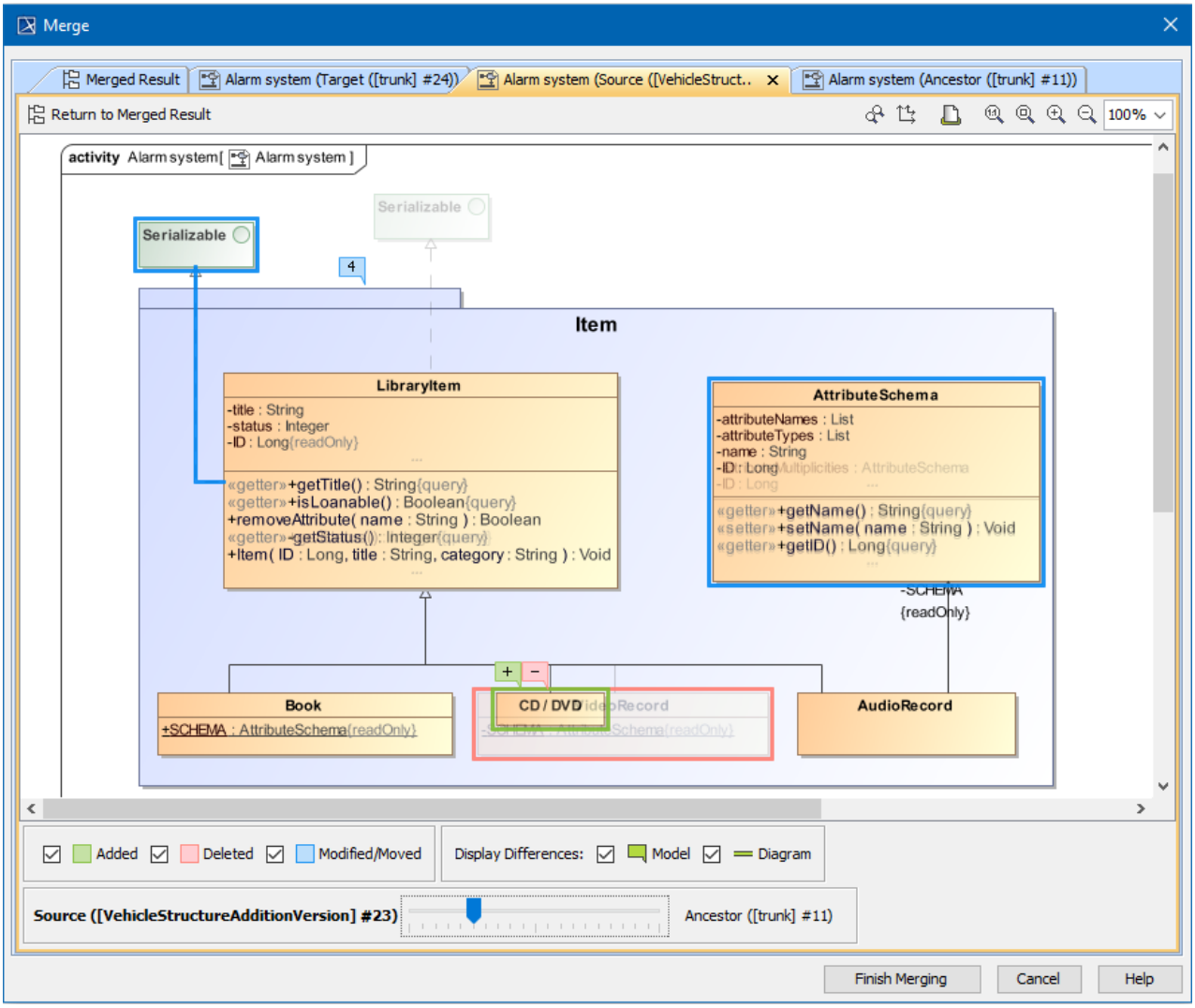
> Be advised that symbols with property value changes that actually have no effect on their appearance are highlighted in a diagram difference viewer as well. For example, changing the **Show Stereotypes** property value in the **Symbol Properties** dialog of a class makes the class symbol highlighted as changed in a diagram difference viewer.

As you can see in the following figure, all these changes are marked by color coded lines and flags allowing the quick distinguishing between different types of changes. Numbers on flags indicate the number of changes made to that element. Clicking a flag switches to the **Merged Result** tab and selects the flagged element in the **Merged Result** tree.

The colors of lines and flags correspond to the color system explained in the difference legend:

- Blue flags indicate the number of modification/movement changes made to the element, blue lines represent modified/moved relations, and blue borders around symbols mark modified/moved elements in the diagram.
- Green flags indicate the number of addition changes made to the element, green lines represent added relations, and green borders around symbols mark added elements in the diagram.
- Red flags indicate the number of deletion changes made to the element, red lines represent deleted relations, and red borders around symbols mark deleted elements in the diagram.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

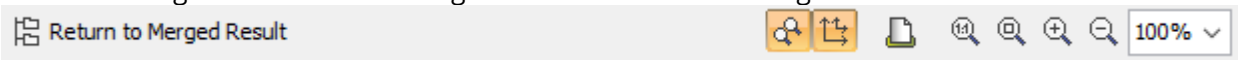


You can customize the types of changes you want to be displayed in the difference viewer by using filtering options available in the legend. Simply select or clear selection from the appropriate check boxes and view the differences that are relevant to you.

If you would like to see the diagram version from other project version without switching between tabs, use the slider available at the bottom of the **Difference Viewer**. Moving the slider makes the current diagram symbols to fade, and the symbols from another diagram version start appearing. When the slider is set to the middle position, different symbols from both diagram.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

You can manage the content of a diagram difference viewer using the toolbar that is located above it.



See the button descriptions in the following table.

Button	Tooltip text	Description
Return to Merged Result	Return to Merged Result	Click to open the **Merged Result** tree with the appropriate diagram selected within.
	Synchronize Zooming	Click to turn on or off the zooming synchronization in all difference viewers of the same diagram.
	Synchronize Scrolling	Click to turn on or off scrolling synchronization in all difference viewers of the same diagram.
	Print Diagram	Click the button to print a diagram. The diagram will be printed with highlighted areas on it, if the highlighting of the diagram changes is turned on.
100%	Zooming	Use the buttons to control the diagram size.

Related pages

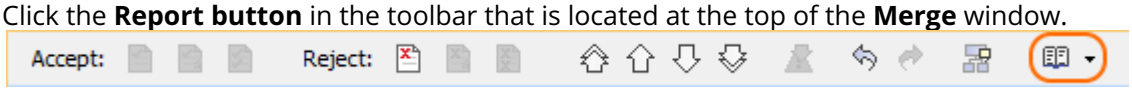
- Getting merge summary information and changes legend(see page 1096)
- Understanding content of the Merged Result panel(see page 1097)
- Inspecting element property changes in the Specification panel(see page 1101)
- Inspecting changes in the Change details panel(see page 1103)
- Displaying and navigating through changes(see page 1104)
- Quickly navigating through conflicting changes(see page 1106)
- Accepting and rejecting changes using shortcut menu(see page 1107)
- Generating a differences report(see page 1111)

Generating a differences report

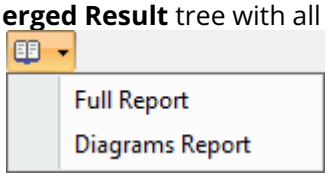
When you're done reviewing differences and wish to see them *on paper*, you can simply export all the diagrams with visible changes to a Microsoft Word document using one of the predefined templates or your custom report template.

To generate a differences report using a predefined template

1. Click the **Report button** in the toolbar that is located at the top of the **Merge** window.



2. Choose the preferred template – Full Report for a more detailed overview, including the entire **Merged Result** tree with all changes, or **Diagrams Report**.



3. The **Save Report** dialog opens. Specify the location in your file system for your report to be stored, type the file name and click **Save**.

If you need to slightly modify the report, for example, to show less data or add your company's information, you can edit the generated Microsoft Word document as per your needs.

However, if you require more control over what's represented in reports, you can customize report templates and generate differences reports of any desired depth and scope.

Creating custom report templates

All the .docx files located in *<configuration files directory>\data\merge reports* are treated as report templates, therefore you can copy and edit any of the existing templates, then save it as your custom report template with a different name.

You will see your saved template in the drop-down list of available templates when you click the **Report** button .

You can read more about creating and modifying report templates in the **Report Wizard Template Creation Tutorial**(see page 1916).

Related pages

- Analyzing and managing merge results(see page 1095)
- Configuration files(see page 73)

Finishing and canceling merge

To finish the merge procedure

- On the **Merge** window, click the **Finish Merging** button and all accepted changes will be copied to the target.

After the merge results are copied to the project, do not forget to either save or commit the project to the server.

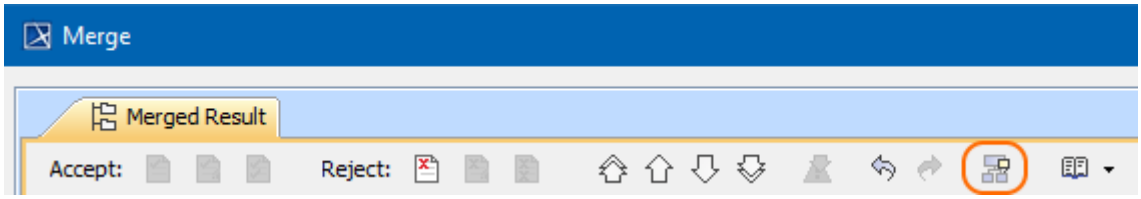
To cancel the merge procedure

- On the **Merge** window, click the **Cancel** button. The question dialog box opens asking you if you want to cancel the merge procedure.
  - Click **Yes** if you are sure that you want to cancel merge procedure.
  - Click **No** and you will be returned back to the Merge window.

Also you can cancel merge procedure by clicking **Close** button on the **Merge** window.

Turning on merged diagram annotations

Before finishing the merge procedure, you can show merged diagram annotations. For this, click the **Annotate Merged Diagram** button on the toolbar located at the top of the **Merge** window.

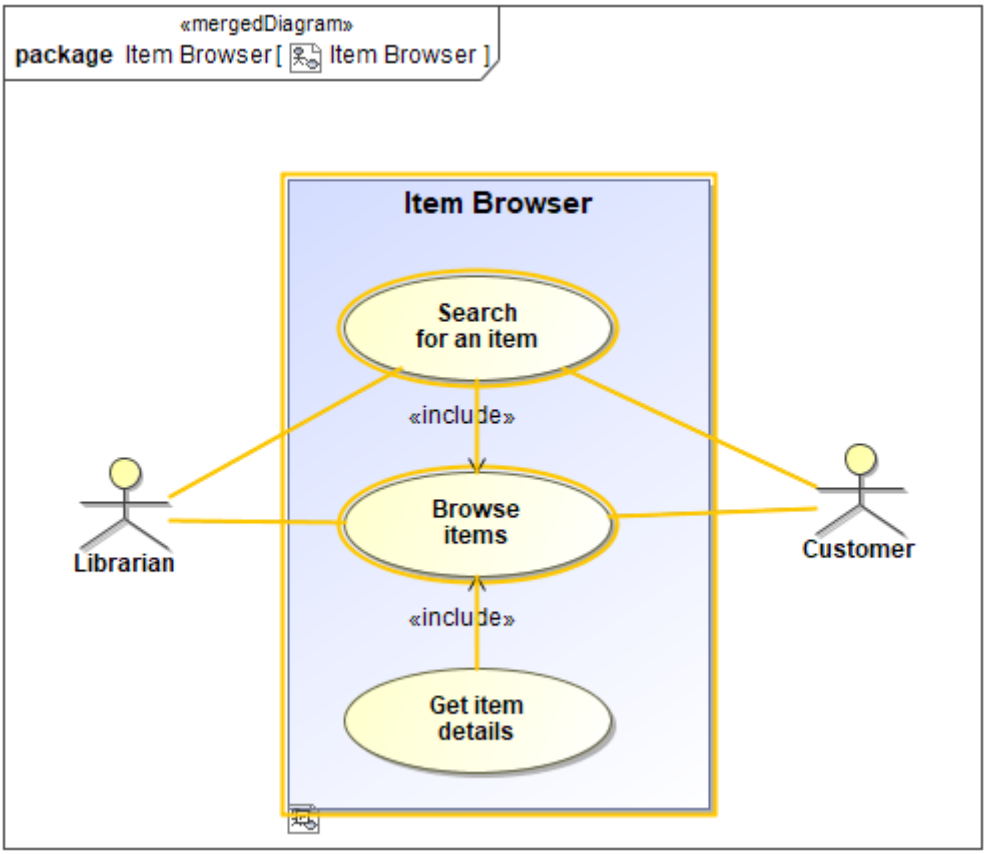


The Annotate Merged Diagram button in the Merge Results window toolbar.

Availability
This button is available if any positions of symbols in diagrams have been changed in both contributors.

The purpose of the annotation is to mark symbols that have different positions on the same diagrams in ancestor and source projects on a merged diagram. If the Annotate Merged Diagram button is enabled, those symbols are annotated in yellow, as shown below.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Annotated symbols in a merged diagram.

Click the selected symbol to open a smart manipulator and select a resolution for symbol merging.

Related pages

- Model Merge(see page 1083)

Metric Suites

On this page

- Creating Metric Suites(see page 1116)

A metric is the item of information about the specific aspect of a model. The collection of various metrics can be used to evaluate the current state of the model. Calculated at regular intervals they can help to track the evolution of model development.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

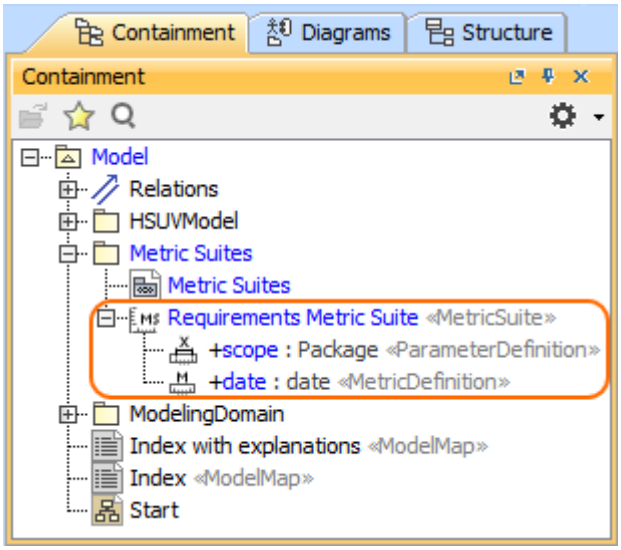
Sorry, the widget is not supported in this export.
But you can reach it using the following URL:

https://www.youtube.com/watch?v=T_1Bpu18KXI

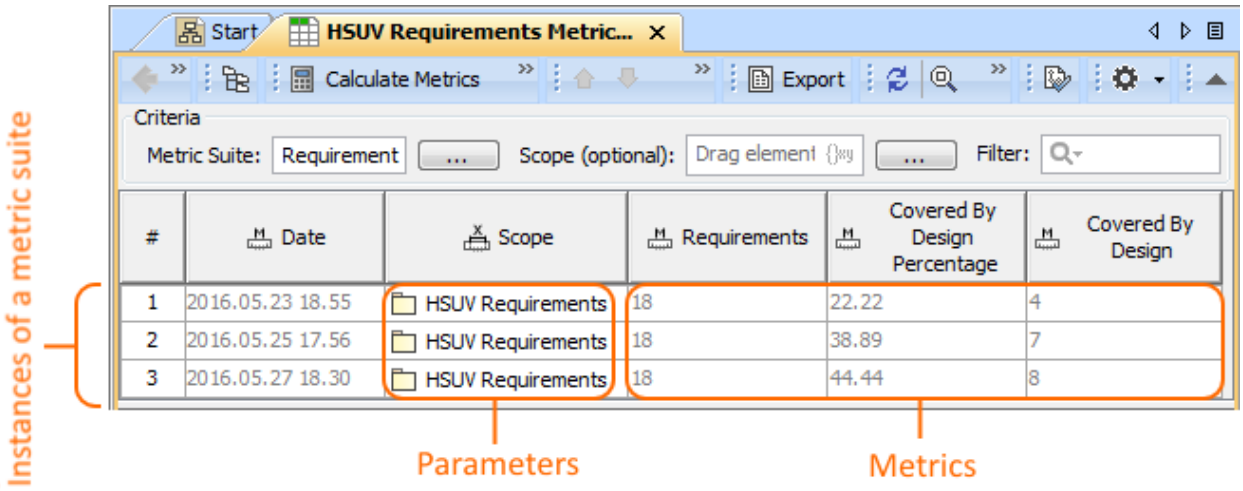
For better understanding of further material, get acquainted with the following concepts

Concept	Description
Metric Suite	A collection of parameter and metric definitions used to calculate the metrics that evaluate specific aspects of your model.
Parameter definition	The variable that is used to calculate metrics. It can be an element type (including both relationship and DSL element types), particular element, data type, or data type value.
Metric definition	The attribute of a Metric Suite defining the formula for calculating metrics according to the parameter values.
Validation-based metric definition	The attribute of a Metric Suite specifying the validation rule for calculating metrics according to the parameter values.

In general, a Metric Suite is a Class with the «MetricSuite» stereotype applied. Both parameter and metric definitions are attributes of a Metric Suite.



One metric is the result of a single calculation of a metric definition. The metrics of all the metric definitions of a Metric Suite together with relevant parameters are stored in a single instance of a Metric Suite (as slot values). To analyse and manage instances of a Metric Suite with a modeling tool, use the Metric Table(see page 468).



Creating Metric Suites

There are several conventions that you must know about Metric Suites before creating one:

- Metric Suite must have a target. A target specifies model elements, for which metrics of the particular metric suite can be calculated. The target value can be any type of elements or even a particular element of your model.
- Metric Suite must have one parameter definition declared as context. The element, which is selected for calculating metrics, is automatically set as the value of the context parameter definition.
- Metric Suite must have a date tracker, so that you could see the timestamp of each metric.

To create and define a Metric Suite, complete the following steps:

1. Add a new Metric Suite to your model(see page 1116).
2. Specify the target for the Metric Suite(see page 1118).
3. Specify parameter definitions of the Metric Suite(see page 1119) (including the context parameter definition).
4. Build metric definitions(see page 1121) and/or validation-based metric definitions(see page 1122).

> It's recommended to have a single person per project, who is responsible for creating and managing all the Metric Suites of the project.

Related Pages

- Specific use cases(see page 1127)

Adding a new Metric Suite

There are two ways to add a new Metric Suite to your model:

- Create a Metric Suite using the *BaseMetricSuite* (recommended).
- Create a Metric Suite from scratch.

The *BaseMetricSuite* is a predefined Metric Suite stored in the *DSL Customization* profile of the *UML Standard Profile*.

You are free to choose either way, but we highly recommend the first one. It aids the creation of a new Metric Suite, which inherits the *BaseMetricSuite* attributes that are necessary for calculating metrics (e.g., the context parameter definition and the metric definition for date tracking). This way you don't have to specify them on your own.

To create a Metric Suite using the *BaseMetricSuite*

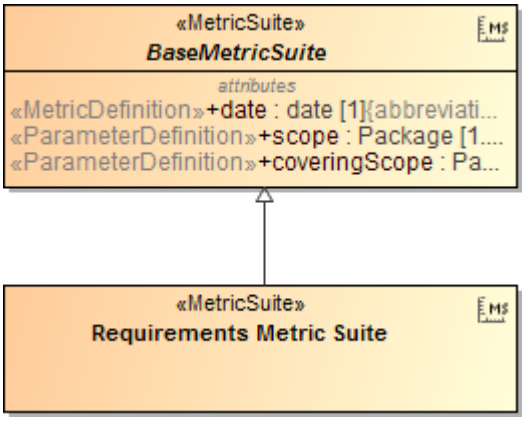
---

1. Make sure that your project uses the *UML Standard Profile*. If not, use it(see page 242) by going to **Options** > **Project Usages**.
2. Create a Package to store all the Metric Suites of your model.
3. In the Package, create a Profile Diagram. If you can't find it in the list of available diagrams, enable the **Expert** mode.
4. Press Ctrl+Alt+F and in the **Quick Find** dialog type "BMS".

   ⚠ Make sure you clear the **Apply Filter** check box.

5. Click the *BaseMetricSuite* in the list of search results. The *BaseMetricSuite* is selected in the Containment tree.
6. Drag the *BaseMetricSuite* from the Containment tree to the diagram pane. Its shape appears on the diagram.
7. In the diagram palette, click **MetricSuite**, and then click a free space on the diagram. A new metric suite is created and its shape appears on the diagram.
8. Create a Generalization relationship from the newly created Metric Suite to the *BaseMetricSuite.*

Now the newly created Metric Suite inherits all the parameter and metric definitions of the *BaseMetricSuite.*

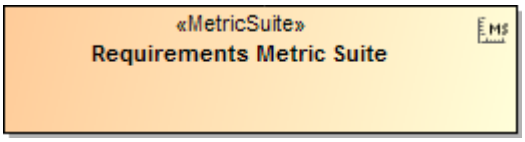


Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To create a Metric Suite from scratch

1. Create a Package to store all the Metric Suites of your model.
2. In the Package, create a Profile Diagram. If you can't find it in the list of available diagrams, enable the **Expert** mode.
3. On the diagram palette, click **MetricSuite**, and then click a free space on the diagram.

A new Metric Suite is created and its shape appears on the diagram.



To specify the description of a Metric Suite

1. Double-click the shape of a Metric Suite to open its Specification window.
2. Click **Documentation/Hyperlinks** on the left side of the window.
3. In the text box on the right side of the window, type the description.
4. Click **Close**.

Related pages

- Metric Suites(see page 1114)
  - Specifying the target for a Metric Suite(see page 1118)
  - Specifying parameter definitions(see page 1119)
  - Building metric definitions(see page 1121)
  - Building validation-based metric definitions(see page 1122)
  - Specific use cases(see page 1127)

Specifying the target for a Metric Suite

A Metric Suite must have a target. It specifies model elements, for which the metrics of a particular Metric Suite can be calculated. The target value can be any type of elements or even a particular element of your model. For example, if the target of a Metric Suite is the Package type, you can calculate metrics for any package in your model by creating a Metric Table(see page 468).

> Although the target of a Metric Suite can be any type of element or even a particular element of your model, we recommend selecting the required Package or the Package type.

A target needs to be specified as the property of a Metric Suite

To specify the target of a Metric Suite

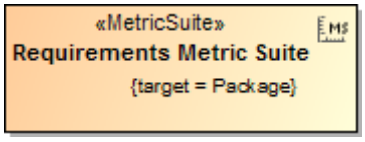
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

1. Double-click the shape of the newly created Metric Suite to open its Specification window.
2. Click the cell of the **Target** property value and then click . The **Select Elements** dialog opens.
3. In the tree or list on the left side of the dialog, select the element that you want to use as a target.

   If you want to select the type of elements (e.g., Package), which is stored in a standard/system profile used in your project, make sure you clear the **Apply Filter** check box. Besides, you can select more than one element. For more information about the manipulations in the element selection dialog, see Elements multiple selection(see page 1032).

4. Add the element to the **Selected elements** list by clicking or (if you want to add elements recursively).
5. Click **OK.**
6. In the Specification window, click **Close**.

The target is specified.



A Metric Suite with the Package type target.

If you need to calculate metrics for the whole model, select the *Model* Package as the target of your Metric Suite.

Related pages

- Metric Suites(see page 1114)
  - Adding a new Metric Suite(see page 1116)
  - Specifying parameter definitions(see page 1119)
  - Building metric definitions(see page 1121)
  - Building validation-based metric definitions(see page 1122)
  - Specific use cases(see page 1127)

Specifying parameter definitions

On this page

- Declaring the context(see page 1120)

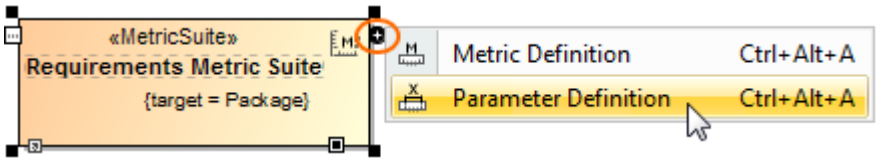
The parameter definition of a Metric Suite can be specified:

- As an attribute with the «ParameterDefinition» stereotype.
- In the body of a metric definition.

If you need to track parameter values, the parameter definition should be specified as the attribute of a Metric Suite. In this case, each value of the parameter becomes the slot of an instance created when running the Metric Suite. If a Metric Suite is created using the *BaseMetricSuite*, it already has a few parameter definitions inherited through the Generalization relationship, such as *scope* and *coveringScope*. If you have created a Metric Suite from scratch or the inherited parameters don't meet your needs, you should specify your own parameter definitions.

To specify a parameter definition as the attribute of a Metric Suite

1. Select the shape of a Metric Suite.
2. Click the **Create Element** button on the right side of the shape and select **Parameter definition**. The attribute with the «ParamterDefinition» stereotype is created.



3. Type its name directly on the shape, and then click a free space on the diagram.
4. Select the attribute on the shape and click .
5. Select the type of the parameter.

   This step is mandatory. A parameter must have a type.

The parameter definition is specified.

Instead of defining the parameters from scratch, you can copy them from the *BaseMetricSuite* and then modify according to your needs.

Declaring the context

One of the parameter definitions must be declared as the context of a Metric Suite. If a Metric Suite has the context declared, then any element, for which you select to calculate appropriate metrics, becomes the value of the context parameter definition of this Metric Suite and is used for calculating metrics. Otherwise, calculating metrics for the selected element makes no sense (unless the context is declared in the body of a metric definition).

Make sure the type of the parameter you are going to declare as the context is the same as the type of the target of a Metric Suite.

To declare the context for a Metric Suite

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

1. On the shape of a Metric Suite, double-click the parameter you need to define as the context for the Metric Suite. The Specification window of the parameter opens.
2. Set the **Set As Context** property value as *true*.
3. Click the **Close** button.

The parameter is declared as the context of the Metric Suite.

> If your Metric Suite is created using the *BaseMetricSuite*, it already has the context parameter definition - the *scope*.

Related pages

- Metric Suites(see page 1114)
  - Adding a new Metric Suite(see page 1116)
  - Specifying the target for a Metric Suite(see page 1118)
  - Building metric definitions(see page 1121)
  - Building validation-based metric definitions(see page 1122)
  - Specific use cases(see page 1127)

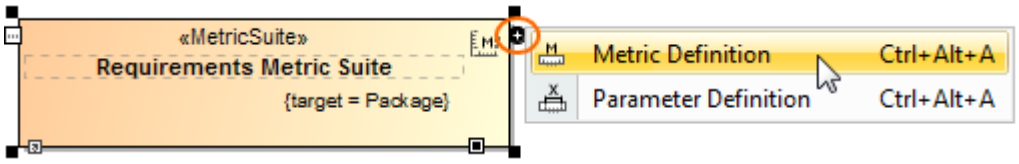
Building metric definitions

Metric definition is the attribute of a Metric Suite with the «MetricDefinition» stereotype applied.

To build a metric definition

---

1. Select the shape of a Metric Suite.
2. Click the **Create Element** button on the shape, and then select **Metric Definition**. An attribute with the «MetricDefinition» stereotype is created.



3. Type its name directly on the shape and then click a free space on the diagram.
4. Select the attribute on the shape and click .
5. Select the type of the result values that should be calculated according to the formula specified in a metric definition.

   > This step is mandatory. A metric definition must have a result type, e.g., Real or Integer.

6. Press **Enter** to open the Specification window of the metric definition.
7. Click the cell of the **Default Value** property value. and appear on the cell.
8. Click and select **Value Specification** > **Opaque Expression**.
9. Click the cell of the **Default Value** property value again, and then click . The **Default Value** dialog opens.
10. From the **Language** drop-down list, select a language.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

> To build a structured expression, select **StructuredExpression**.

11. In the **Body** box, specify the formula for the metric definition. For more information about building structured expressions, see Specifying criteria for querying model(see page 1227).
12. Click **OK** and then click **Close** in the Specification window.

The metric definition is specified.

> If your Metric Suite is created using the *BaseMetricSuite*, it already has the metric definition for date tracking.

One metric definition can refer to another metric definition of either the same Metric Suite or a more general one (in case there is the Generalization relationship between these Metric Suites).

Related pages

- Metric Suites(see page 1114)
  - Adding a new Metric Suite(see page 1116)
  - Specifying the target for a Metric Suite(see page 1118)
  - Specifying parameter definitions(see page 1119)
  - Specific use cases(see page 1127)
  - Building validation-based metric definitions(see page 1122)

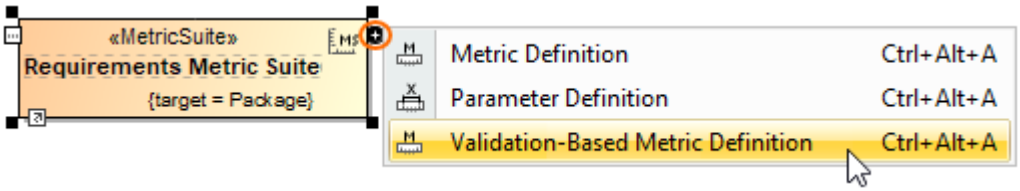
Building validation-based metric definitions

A validation-based metric definition is the attribute of a Metric Suite with the «ValidationBasedMetricDefinition» stereotype applied. In this attribute, you can specify a predefined validation rule for calculating metrics according to the parameter values.

To build a validation-based metric definition

---

1. Select the shape of a Metric Suite, click the smart manipulator on the shape, and select **Validation-Based Metric Definition**. An attribute with the «ValidationBasedMetricDefinition» stereotype is created.



2. Type the name of the attribute directly on the shape and then click a free space on the diagram.
3. Select the newly created validation-based metric definition on the Metric Suite shape and click .

4. Select the type of the result values that should be calculated according to the validation rule specified in the validation-based metric definition.

   > **Specifying a result type**
   > This step is mandatory. A validation-based metric definition must have a result type, e.g., Real or Integer.

5. Open the Specification window of the validation-based metric definition.
6. Select the specification cell of the **Validation Rule** property and click ...
7. In the **Select Constraint** dialog, select the validation rule that will be used to calculate metrics. Click **OK**.
8. Select the specification cell of the **Metric Type** property. In the drop down list select the type of the metric you want to calculate.
9. Optionally, select the specification cell of the **Failing Elements Definition** property and enter the name of the menu item that displays failing elements according to the validation rule selected in step 7.

   > **Displaying failing elements**
   > To find the menu item that displays failing elements, right-click the Metric Table cell displaying the metric according to the selected validation rule. By default the menu item is **Display Failing Elements**, unless you give it a different name as described in step 9.

The validation-based metric definition for a Metric Suite is now specified and you can start calculating Metrics by using a Metric Table(see page 468).

Related pages

- Metric Suites(see page 1114)
  - Adding a new Metric Suite(see page 1116)
  - Specifying the target for a Metric Suite(see page 1118)
  - Specifying parameter definitions(see page 1119)
  - Building metric definitions(see page 1121)
  - Specific use cases(see page 1127)

Creating a validation dashboard

If you have several Metric tables where you calculate metrics using Metric Suites with validation-based metric definitions, you can display the validation results from these tables in a validation dashboard. To create a validation dashboard follow these steps:

1. Create Metric Suites for validation rules you want to track and create Metric Tables to calculate validation results(see page 0).
2. Create a validation dashboard that displays metric values from the Metric Tables created in the previous step(see page 1124).

Creating Metric Suites and Metric Tables to track in a validation dashboard

You can calculate model validation results in Metric Tables based on Metric Suites with validation-based metric definitions. Metrics calculated in these Metric Tables can later be displayed in a validation dashboard to track all validation results in one place. Follow the steps below to create Metric Suites and Metric Tables for a validation dashboard.

To create Metric Suites for a validation dashboard

1. Create a Package to store your Metric Suites.
2. Create a Profile Diagram and name it (e.g., *Metric Suites*).
3. In the Profile diagram, Create a Metric Suite using the *BaseMetricSuite*(see page 1116) for each validation rule you want to track.
4. Create validation-based metric definitions(see page 1122) for the Metric Suites created in the previous step.

   Example of a Metric Suite that tracks a validation rule
   For example, if you want to track the Unused Constraint Parameter validation rule, you can create a Metric Suite named *Unused Constrained Parameter* with the following validation-based metric definitions:
   - Validation-based metric definition named *Constrained Parameter* with the **Type** property set to **Integer**, **Validation Rule** property set to **Unused Constrained Parameter**, and the **Metric Type** property set to **Element Quantity**.
   - Validation-based metric definition named *Unused Constrained Parameter* with the **Type** property set to **Integer**, **Validation Rule** property set to **Unused Constrained Parameter**, and the **Metric Type** property set to **Failed Element Quantity**.
   - Validation-based metric definition named *Unused Constrained Parameter Percentage* with the **Type** property set to **Real**, **Validation Rule** property set to **Unused Constrained Parameter**, and the **Metric Type** property set to **Failed Element Percentage**.

5. Create a Metric Table(see page 470) for each Metric Suite created in step 3 and calculate metrics(see page 471).

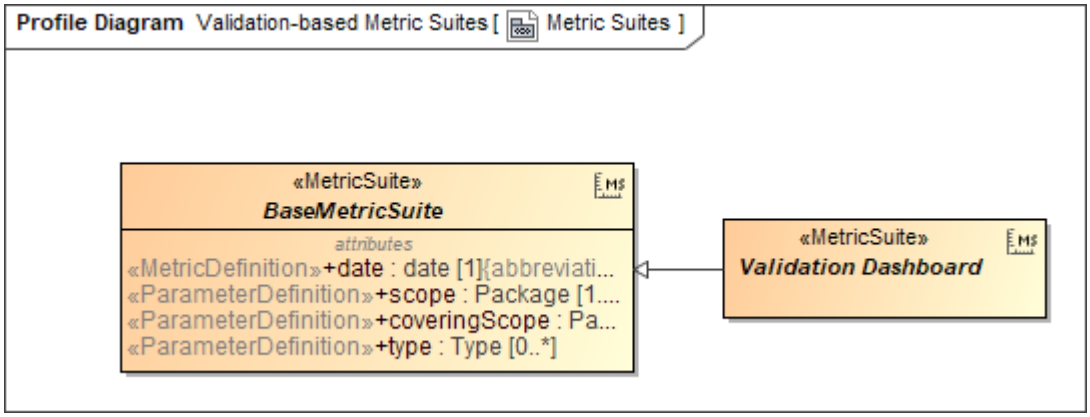
Creating a validation dashboard

A validation dashboard is a convenient way to track model validation results in one place. It displays metric values from multiple Metric Tables created from validation-based Metric Suites.

To create a validation dashboard

1. Open the Profile Diagram in which you created Metric suites with validation based metric definitions (see step 2 in Creating Metric Suites and Metric Tables for a validation dashboard(see page 0)).
2. In the Profile diagram, Create a Metric Suite using the *BaseMetricSuite*(see page 1116) and name it (e.g., *Validation Dashboard*).

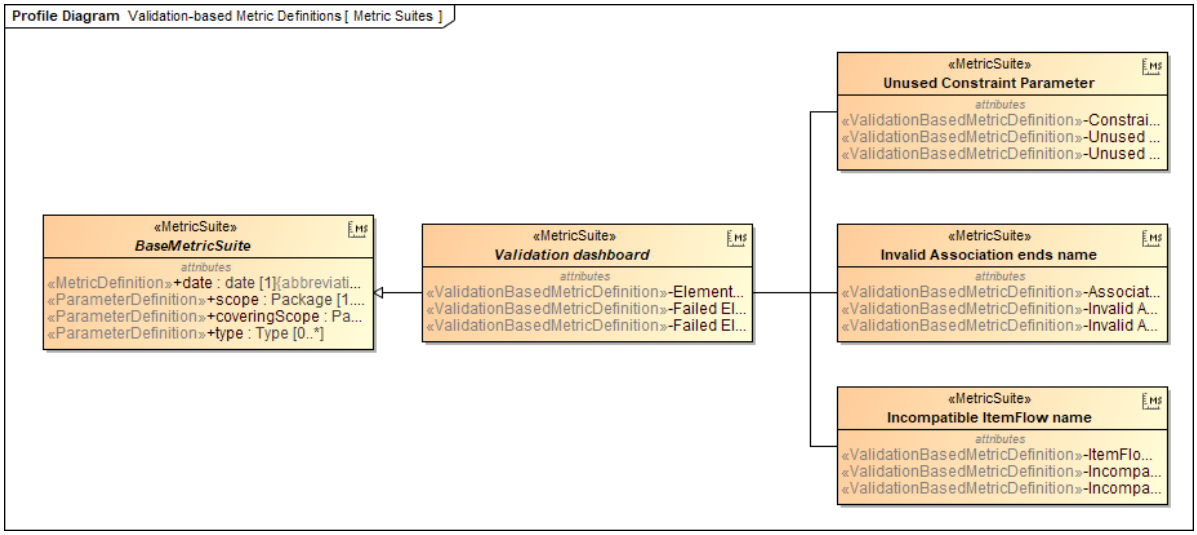
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



3. Open the Specification window of the newly created Metric Suite (e.g., *Validation Dashboard* Metric Suite, as displayed above) and set the **Is Abstract** property to *true*.
4. Build validation-based metric definitions(see page 1122) for the Metric Suite to define the structure of the dashboard. Each validation-based metric definition represents a column in the validation dashboard.

> **Example of validation-based metric definitions**
> For example, if you want to create a validation dashboard with the *Elements*, *Failed Element Quantity*, and *Failed Element Percentage* columns, create the following validation-based metric definitions:
> - Validation-based metric definition named *Elements* with the **Type** property set to **Integer** and the **Metric Type** property set to **Element Quantity**.
> - Validation-based metric definition named *Failed Element Quantity* with the **Type** property set to **Integer** and the **Metric Type** property set to **Failed Element Quantity**.
> - Validation-based metric definition named *Failed Element Percentage* with the **Type** property set to **Real** and the **Metric Type** property set to **Failed Element Percentage**.

5. In the same Profile diagram, create a Generalization relationship from each Metric Suite that you created to track a validation rule (step 3 in Creating Metric Suites and Metric Tables to track in a validation dashboard(see page 1123)) to the abstract Metric Suite created in step 2.



6. Link each validation-based metric definition that calculates the metric you want to display in the validation dashboard to the metric definition representing a specific validation dashboard column (see step 4), as described below:
   a. Open the Specification window of the validation-based metric definition that calculates the metric you want to display in the validation dashboard.
   b. Select the specification cell of the **Redefined Property** property and click ...
   c. On the left side of the **Select Property** window, find the abstract Metric Suite created in step 2 and select its metric definition representing the validation dashboard column in which you want to display the metric.
   d. Click + to add the metric definition to the **Selected elements** area. Click **OK**.

      Example of linking a validation-based metric definition
      Let's say you have the *Unused Constraint Parameter* Metric Suite with the *Constraint Parameter* validation-based metric definition that calculates the number of constraint parameters. If you want to display the number of constraint parameters in the *Elements* column of the validation dashboard, you must open the Specification window of the the *Constraint Parameter* validation-based metric definition, and select the *Elements* validation-based metric definition of the abstract Metric Suite (created in step 2) as the value of the **Redefined Property** property.

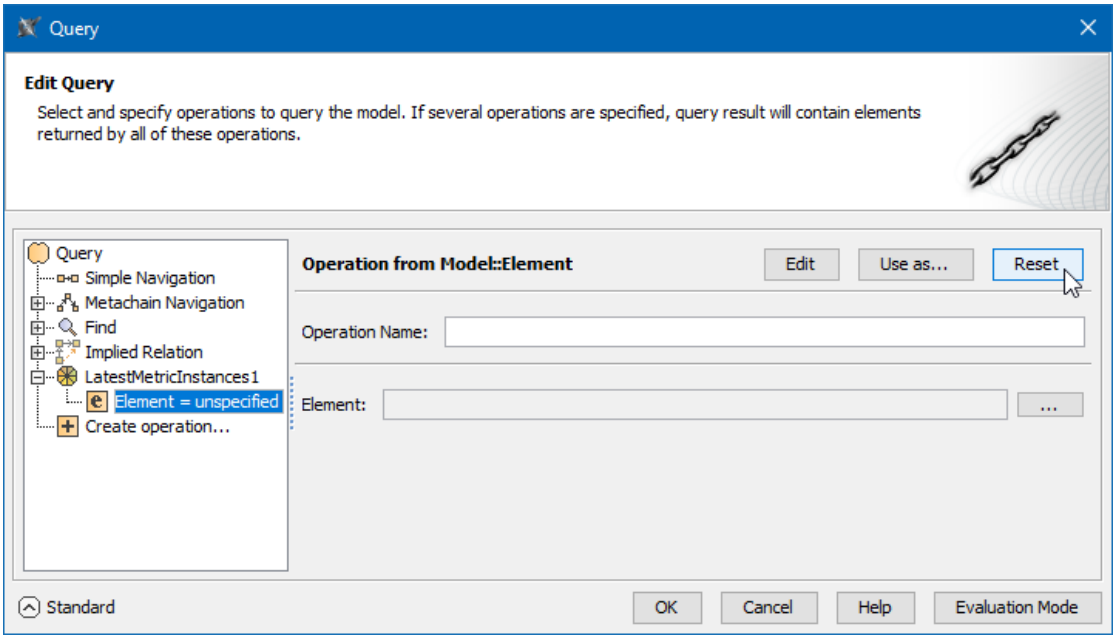
   e. Repeat steps a through d for each validation-based metric definition calculating the metric you want to display in the validation dashboard.
7. Build the validation dashboard:
   a. Create a Metric Table(see page 470) that will be your validation dashboard.
   b. In the Metric table, click ... next to the **Metric Suite** box.
   c. In the **Select Metric Suite** dialog, select the abstract Metric Suite created in step 2. Click **OK**.
   d. Open the Specification window of the Metric table.
   e. Select the specification cell of the **Scope** property and click ...
   f. On the left side of the **Select Elements** dialog, select the Metric Tables you want to track in the validation dashboard.

      Selecting multiple elements
      If you cannot select multiple elements, click the **Multiple Selection** button on the bottom of the **Select Elements** dialog.

   g. Click + to add the selected Metric tables to the **Selected elements** area. Click **OK**.
   h. In the Specification widow of the Metric Table, select the specification cell of the **Query** property and click ...
   i. On the left side of the **Query** dialog, select **Create operation** and click the **Operation from Model** operation on the right side of the dialog.

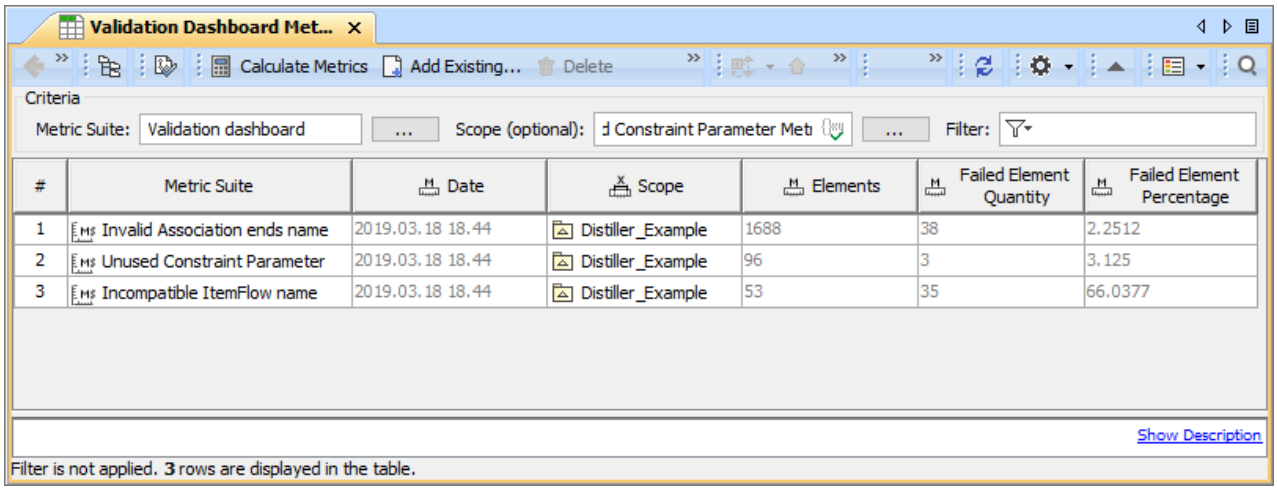
      Enabling the Expert mode
      The **Operation from Model** operation is only available in the Expert mode. To enable the Expert mode, click the **Expert** button on the bottom left corner of the **Query** dialog.

   j. In the **Select Opaque Behavior** dialog, select the *LatestMetricInstances* Opaque Behavior. Click **OK**.
   k. On the left side of the **Query** dialog, select **Element = unspecified** located under the *LatestMetricInstances1* operation and click the **Reset** button.



l. On the right side of the **Query** dialog, select the **Contextual Variable** operation in the **Other** operation group and click **OK**.
m. Close the Specification window of the Metric Table.

The latest instances from the specified Metric Tables are now collected and displayed in your validation dashboard, as shown below.



This sample validation dashboard tracks the latest metrics from three Metric Tables.

Specific use cases

On this page

- How to build a metric definition for date and time tracking?(see page 1128)
- How to access parameter values in a metric definition?(see page 1129)
- How to build a metric definition that takes the result of another metric definition?(see page 1130)

How to build a metric definition for date and time tracking?

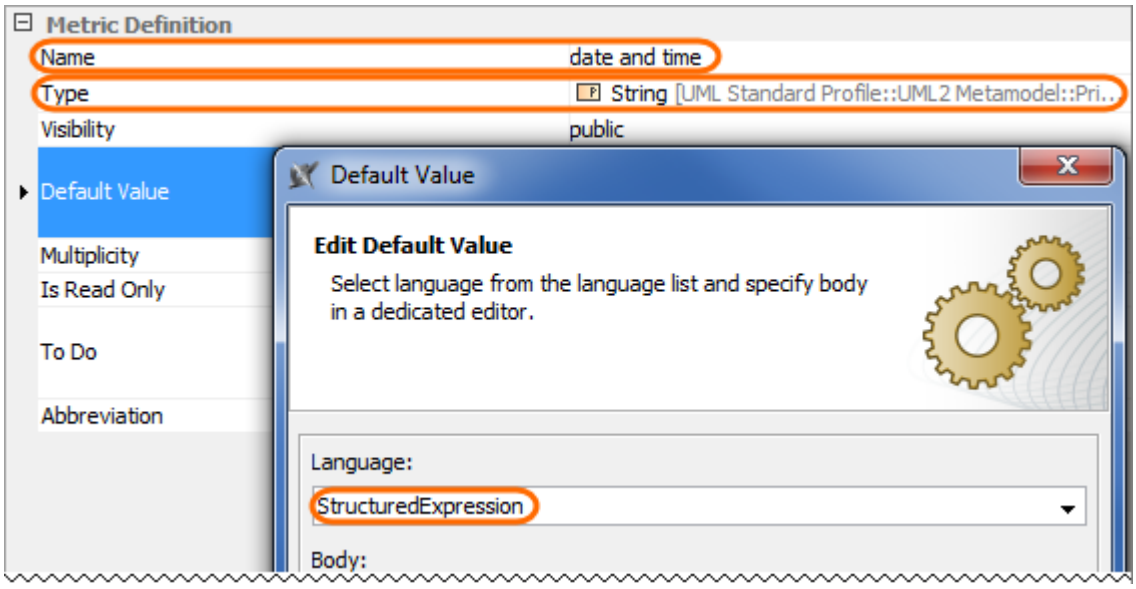
If your metric suite isn't created using the *BaseMetricSuite*, you must build the metric definition for date and time tracking on your own.

To build the metric definition for date and time tracking

---

1. Add a new metric definition with the following characteristics to your metric suite:

    - *date and time* as name
    - **String** as the result type
    - **StructuredExpression** as the language of the metric definition body

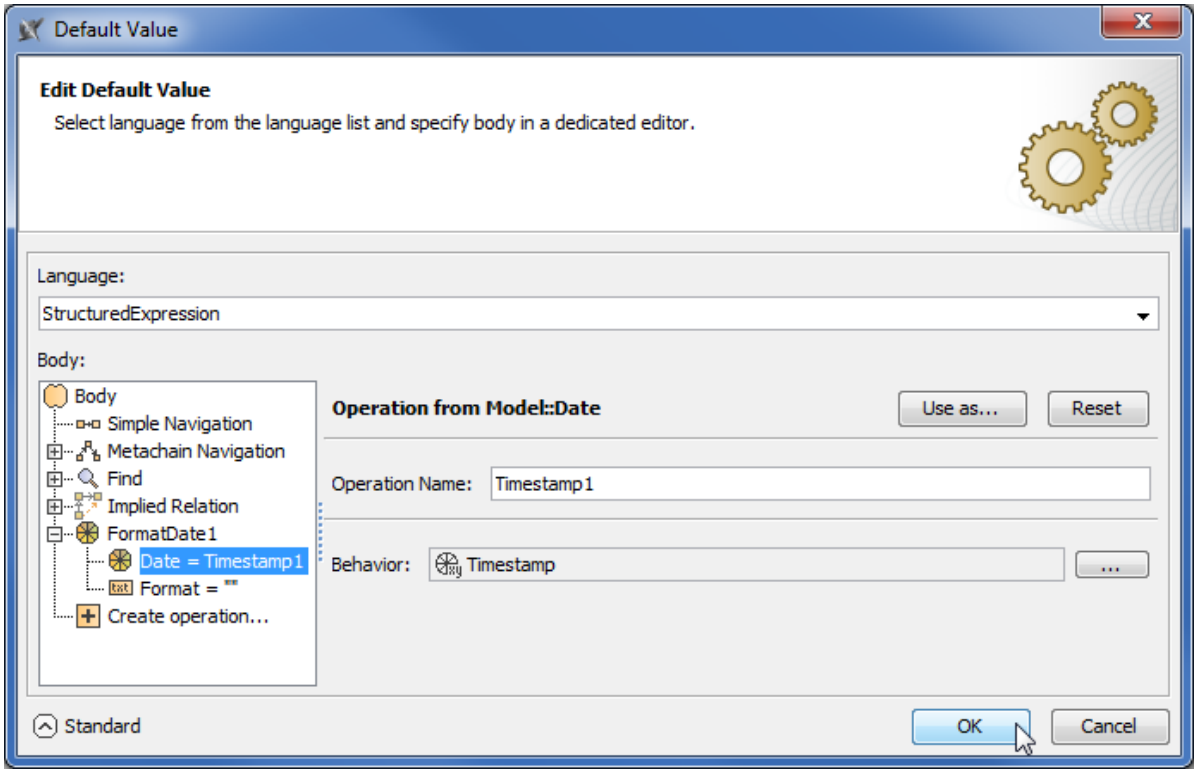


2. In the **Body** area of the **Default Value** dialog, click **Create operation**, and then select **Operation from Model** on the right side of the dialog.
3. In the open dialog, select the **FormatDate** built-in operation which is stored in the *UML Standard Profile* and click **OK**.

    ⚠ Make sure you clear the **Apply Filter** check box in the **Select Opaque Behavior** dialog.

4. In the **Body** area of the **Default Value** dialog, select the **Date** parameter of the **FormatDate** operation and click The **Reset** button.
5. Select **Operation from Model** on the right side of the dialog.
6. In the open dialog, select the **Timestamp** built-in operation which is stored in the *UML Standard Profile* and click **OK**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



7. In the **Body** area of the **Default Value** dialog, select the **Format** parameter of the **FormatDate** operation and in the **Value** box type "dd.MM.yyyy hh:mm aaa".
8. Click **OK**, and then close the metric definition Specification window.

Now you can track the date and time of each instance of the metric suite.

How to access parameter values in a metric definition?

If you need to use the values of a parameter for specifying calculations in a metric definition, you can easily access them with the help of the functionality provided by the StructuredExpression language. Let's overview the case when you need to use the value of the *scope* parameter inherited from the *BaseMetricSuite* to calculate your metrics.

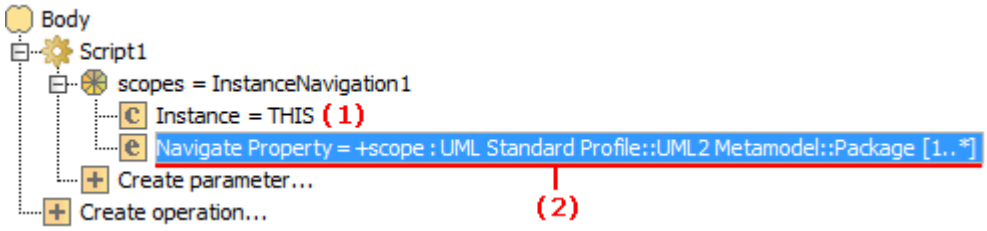
To access the value of the *scope* parameter inherited from the *BaseMetricSuite*

---

1. Add a new metric definition to your metric suite.
2. Select the **StructuredExpression** language for specifying the calculations.
3. In the structured expression, create a script operation with single parameter named *scopes*.
4. Select the **InstanceNavigation** built-in operation as the *scopes* parameter value.
5. Specify the following parameter values of that operation:

    - **Contextual Variable** as the **Instance** parameter value.
    - The *scope* parameter of the *BaseMetricSuite* as the **Navigate Property** parameter value.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



6. In the script operation, refer to the *scopes* parameter simply by its name as the following Groovy script sample illustrates:

```
import static com.nomagic.magicdraw.modelmetrics.ScriptHelper.*;
import static com.nomagic.requirements.util.SysMLProfile.*;

getElementsRecursively(scopes)
findAll {el -> isRequirement(el)}
size()
```

An alternative way to access parameters is calling the *getValue()* method from MagicDraw Open API on the *valueContext* globally defined value, as the following Groovy script sample illustrates:

```
def scopes = valueContext.getValue("scope")
```

How to build a metric definition that takes the result of another metric definition?

Let's say, you need to calculate the percentage of requirements that are covered by blocks. To do this, you must have the following values:

- Overall requirements count
- Requirements covered by blocks count

Let's say, these values are the results of two already existing metric definitions: *requirementsCount* and *requirementsCoveredByBlocksCount*. You need another metric definition, which takes the results of the latter metric definitions and calculates the percentage. To access these values and make sure they are already calculated, we must call the *getOneValue()* method from MagicDraw Open API on the *valueContext* globally defined value for both metric definitions, as the following Groovy script sample illustrates:

```
int all = valueContext.getOneValue("requirementsCount");
int covered = valueContext.getOneValue("requirementsCoveredByBlocksCount");
com.nomagic.magicdraw.modelmetrics.ScriptHelper.calcPercentage(all, covered)
```

Related pages

- Metric Suites(see page 1114)
  - Adding a new Metric Suite(see page 1116)
  - Specifying the target for a Metric Suite(see page 1118)
  - Specifying parameter definitions(see page 1119)
  - Building metric definitions(see page 1121)
  - Building validation-based metric definitions(see page 1122)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Model Transformation Wizard

Sorry, the widget is not supported in this export.
But you can reach it using the following URL:

https://www.youtube.com/watch?v=Ls4f12ieG44

Transformation engine itself is available in MagicDraw editions from Standard and up. However, only MagicDraw Architect and Enterprise editions bring any particular transformations. So for MagicDraw Standard and Professional edition users, transformations are not available.
Plugins can bring additional transformations regardless of MagicDraw edition. For example, users, who have the Cameo Data Modeler plugin, can use the transformation engine to run ER to SQL(Generic/ Oracle) transformations, even if they do not have the MagicDraw Architect/ Enterprise editions.

The Model Transformation Wizard enables running one of the predefined transformations. When using this wizard to run a chosen transformation, you have to perform the following steps:

1. Choose a transformation type.
2. Specify both the transformation source model (or a part of it) and destination package.
3. Select a type map.
4. Set custom transformation properties.

Each transformation converts the chosen model part into another model according to the rules of this transformation. Transformations are usually used for converting models between different modeling domains, e.g., UML and SQL.

All transformations follow a similar approach. They take a part of a model as the transformation source and copy it to the destination model, establishing traces between the transformation source and target elements. Then each transformation performs the specific model reorganizations, which are necessary for each
transformation type according to the transformation options specified by the user in the transformation wizard. Transformation can also be performed in-place, i.e., the source model is not copied to the destination model, but transformation works directly on it instead.

Transformations also perform the so-called type remapping. During the transformation between the different modeling domains, such as UML and SQL, it is necessary to go through the data types used in the source model and change the types from the source domain into the equivalent types in the target domain, for
example, changing String type usages in the UML model into the varchar type usages in the SQL model.

Available Transformations

Transformations are usually used for converting models between different modeling domains. Transformations are named by the types of their source models and their destination models. These are the available transformations:

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- **Any to Any**. This transformation copies all your model or part of it to another package without making any changes. You can also remap types in the destination model by applying some type mapping rules.
- **Profile Migration**. Helper transformation for migrating models using one profile to models using another profile (usually - between different versions of the same profile - old and new).

> UML to SQL(Generic/Oracle), SQL to UML, UML to XML Schema, XML Schema to UML transformations are available with the separately-installed Cameo Data Modeler plugin (which comes free of charge with MagicDraw Architect and Enterprise Editions and is separately purchseable for MagicDraw Standard and Professional editions).

> Additional custom types of transformations can be defined by MagicDraw plugins.

Working with Model Transformation Wizard

The wizard can be opened from several places.

To start the **Model Transformation Wizard**, do one of the following

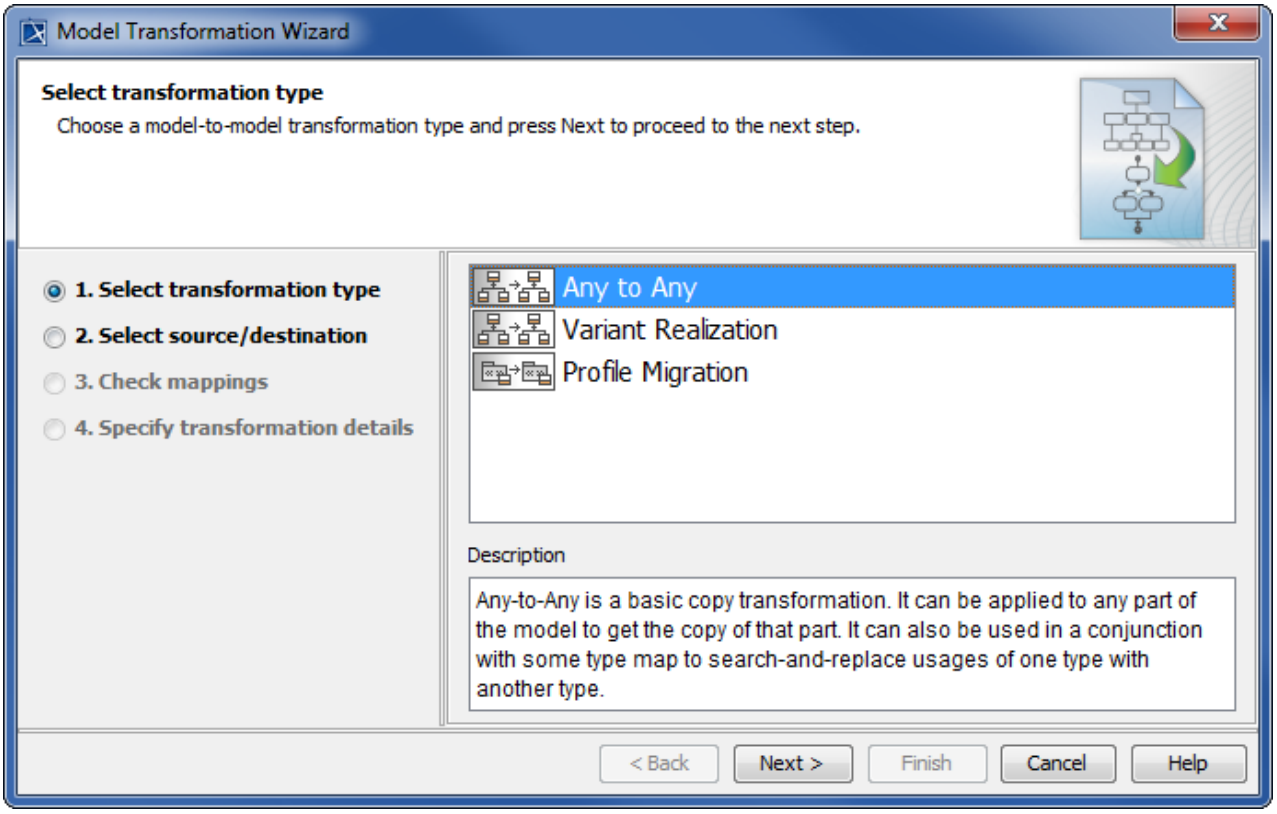
---

- From the **Tools** menu, choose **Model Transformations**.
- Right-click one or more packages and select **Tools** > **Transform**.

Related pages

Selecting transformation type

In the first step of the Model Transformation Wizard, a list of the available transformation types is displayed.



Transformation types are displayed in the list window.

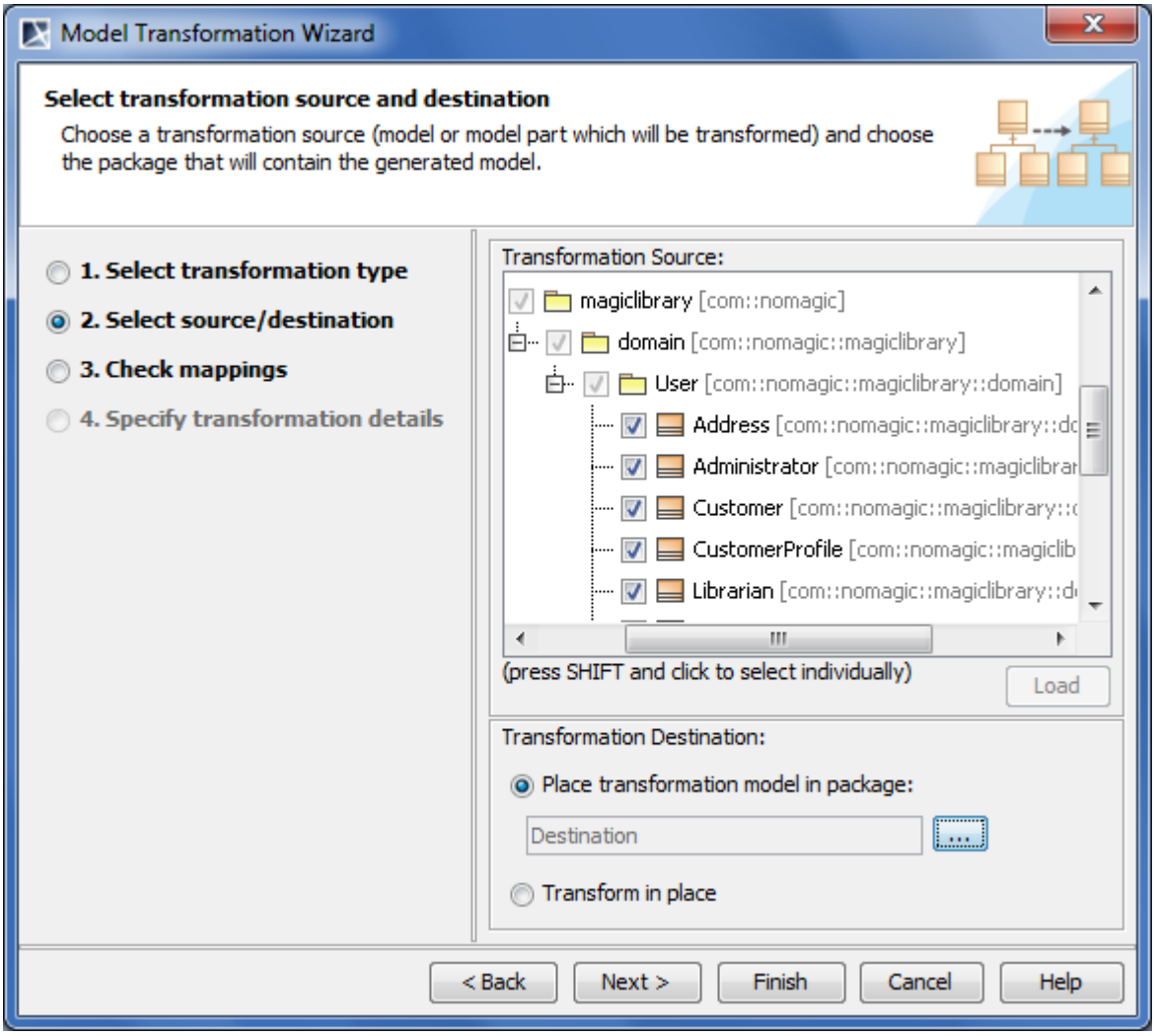
The following operations are available in the Select transformation type window:

Button	Function
Next	Proceed to the next step (in this case, Select source/destination).

Selecting source and destination models

In the second step of the wizard, the Transformation Source tree displays all project data, i.e. the packages and their inner elements, that can be selected as a transformation source. Transformation will take the selected elements as input data.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Select the **Place transformation model in package** option button to specify the package into which the source will be transformed. Click the **...** button to display the **Destination Package** dialog. Select an existing package from the **Packages** tree or create a new one.

Select the **Transform in place** option button, if you want the source model to be edited.

> If you choose the in-place transformation, the model part selected as the transformation source, will be edited directly, and you will not retain your original model. So, please, be careful with this option.
>
> If you choose the destination package, the source model will be copied to it and the transformation will be performed on this copy. Hence you will retain your source model and get a resulting model and traces will be established between elements in these model parts.

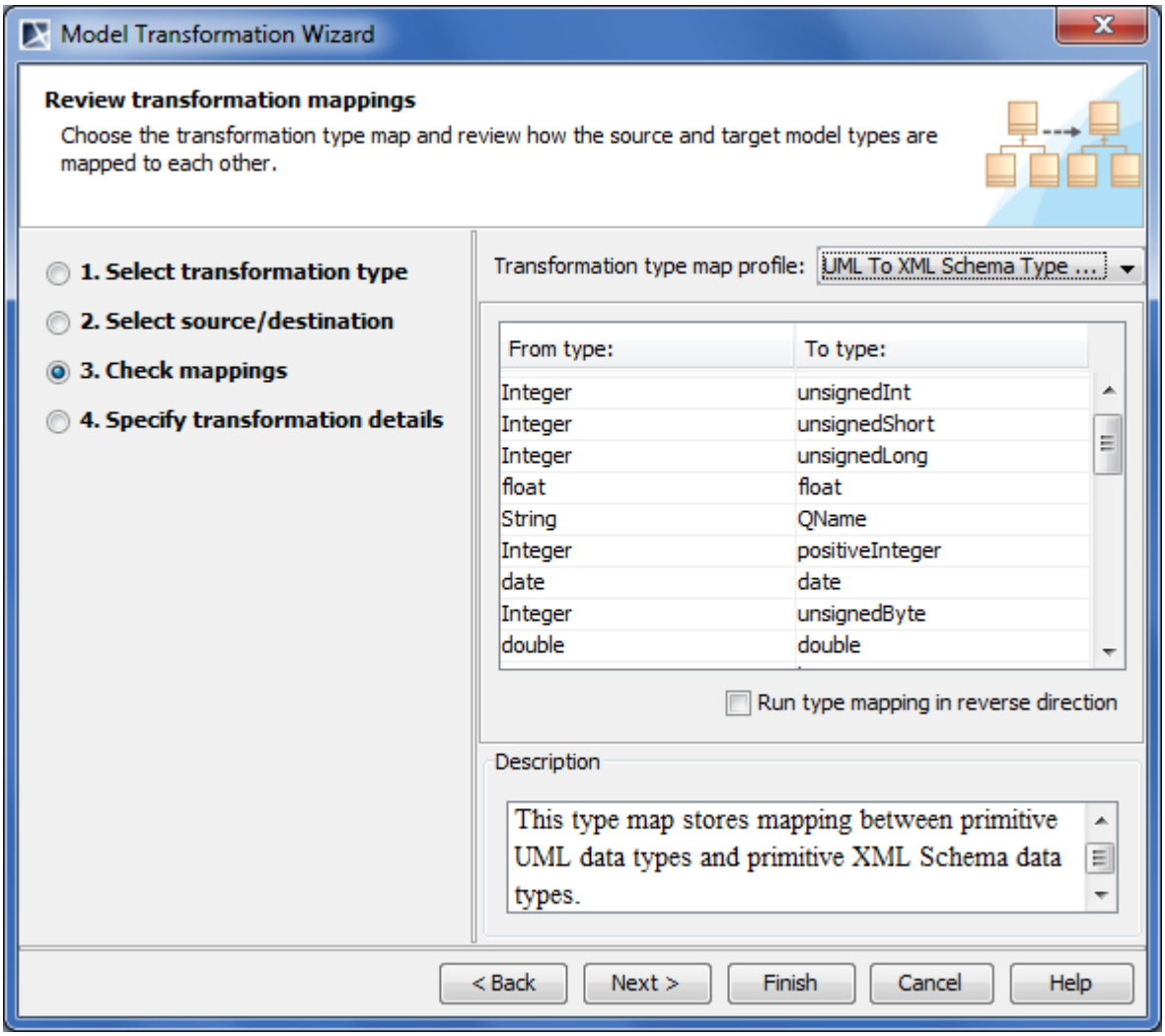
The following operations are available in the **Select source/destination** window:

Button	Function
Back	Return to the previous dialog box.

Button	Function
Next	Proceed to the next step (in this case, Select type mappings).
Finish	Finish the transformation configuration. All other options will be set by default. The Model Transformations Wizard exits and transformation results appear in the project.

Selecting type mappings

The third step in the wizard allows for selecting a type map that will be applied during the transformation. Usually transformation has and brings in some predefined type map, but if you want, you can specify another type map.



A type map can be regarded as a collection of rules of the form “Replace the usage of type X in the used project with the usage of type Y”.

A type map(see page 1137) is a model object, i.e. a package with a collection of dependencies, hence all model manipulation operations can be performed on it. In particular, it can be refactored into a used project and mounted into any project,
which needs it. It can be a simple package in your project as well, if you need a custom, one-off type map. A predefined type map can be taken from the MagicDraw module and edited.

To see a list of the type maps available in your model, click the down arrow in the Transformation type map(see page 1137) combo box. These type maps specify the mapping rules that will be applied to the model during the transformation.

When you select a particular map, its contents are displayed in a table below. Each row in the table is a rule to remap one particular type to another. The **From type** and **To type** columns in the table show the source and target types.

The **Run type mapping in reverse order** check box creates the opposite type mapping. Type maps can be bidirectional, e.g., the same type map is reused both in the UML to XML schema and XML schema to UML transformations. This checkbox governs the direction in which the type map should be used.

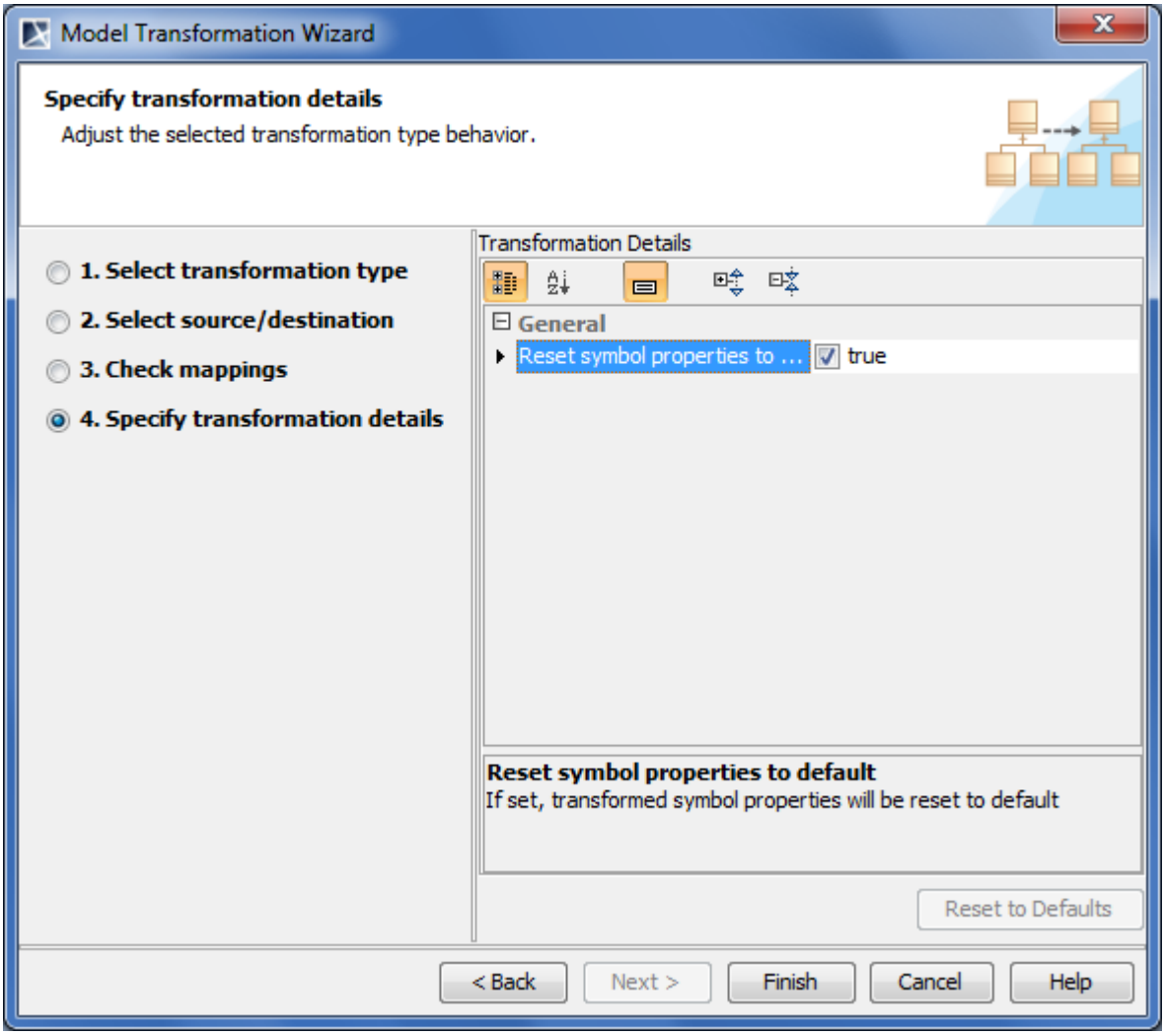
The following operations are available in the **Select type mapping** window:

Button	Function
Back	Return to the previous dialog box.
Next	Proceed to the next step (in this case, Specify transformation details). **NOTE:** This button is disabled during the **Any to Any** transformations.
Finish	Finish the configuration of the transformation. The **Model Transformations Wizard** exits and the transformation results appear in the project.

Setting custom transformation properties

The **Transformation Details** table displays the various properties of a specific transformation, selected in the first step of the **Model Transformation Wizard**. Each transformation type has its own set of options, which govern functionality of that transformation.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



To change the transformation properties to the default values, click **Reset to Defaults**.

The following operations are available in the **Specify transformation details** window:

Button	Function
Back	Return to the previous dialog box.
Finish	Finish the transformation configuration. The **Model Transformations Wizard** exits and the transformation results appear in the project.

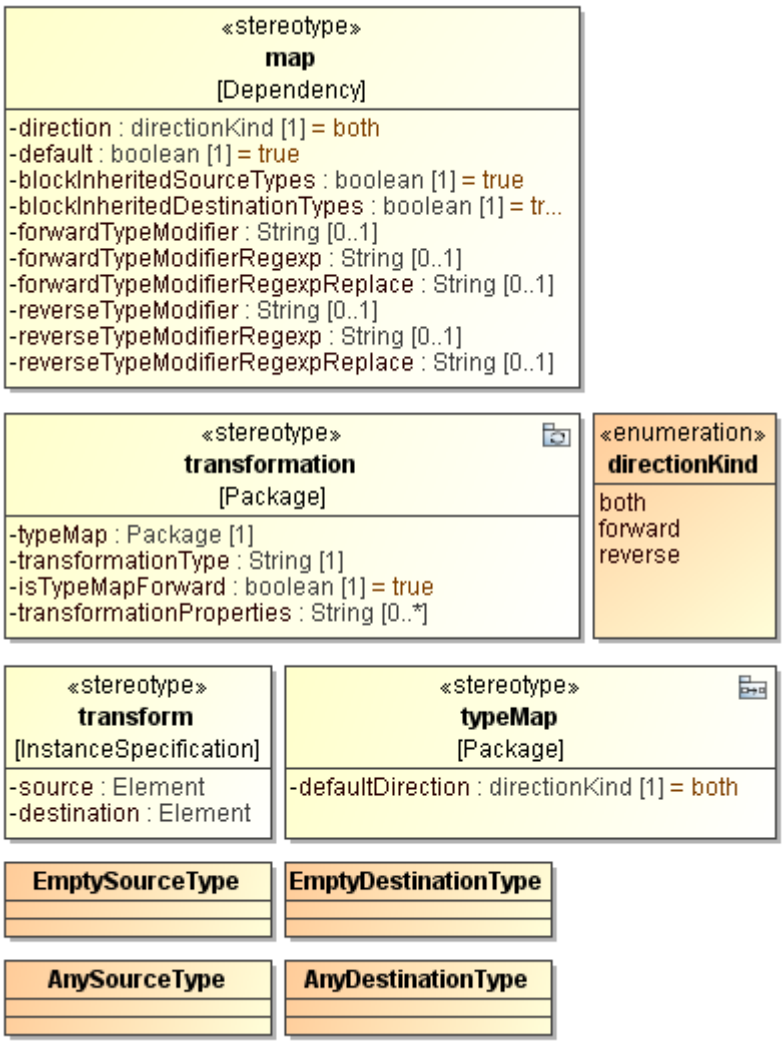
Transformation Type Mapping

During the transformation between the different modeling domains, such as UML and SQL, it is necessary to go through data types used in the source model and change the types from the source domain into the equivalent types in the target domain, for example, changing String type usages in the UML model into varchar type

usages in the SQL model. This is achieved by establishing a type map and then supplying it for the transformation (many transformations have default, predefined type maps).

A type map can be regarded as a collection of rules of the form “Replace the usage of type X with the usage of type Y”. A type map is created by modeling means and is a model object, hence all model manipulation operations can be performed on it. In particular - it can be refactored into a module and mounted into any project, that needs it. It can be a simple package in your project as well, if you need a custom, one-off type map. Predefined type map can be taken from the MagicDraw module and edited.

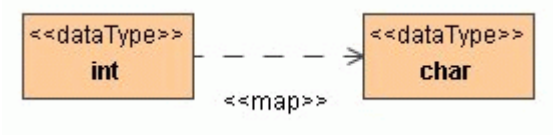
A type map is a stereotyped package, holding a collection of stereotyped dependencies. Stereotypes for building type maps are stored in the *Model Transformation Profile*.



To create a transformation type map

1. Use or import *Model_Transformation_Profile.xml.zip*.
2. Create a package, which will represent your type map.
3. Apply a stereotype *«typeMap»* to the created package.
4. Choose types (data types, classes, enumerations) in your source domain and their corresponding types in your target domain.

5. Create the desired dependency relationships between the corresponding types.
6. Apply a stereotype named *«map»* to these dependencies.



> Be sure to place dependencies in the type map package (MagicDraw is prone to placing dependencies in or near the dependent model element, so you may need to relocate them).

In the example above, after the transformation, all *int* types will be transformed to *char*.

Each of thus created dependencies represents one type remapping rule. The package represents the complete type map.

Type mapping rule behavior can be further customized by setting various tags on the rules(see page 1139).

> Transitive type mapping (of the form type1 > type2 > type3) is not supported.

Controlling Type Mapping Rule Behavior

There are several different tags that can be set on a type map or an individual rule to change its behavior slightly.

Controlling direction

By default, the same type map can be applied in two directions: forward and backward. The backward direction can be set by selecting the **Run type mapping in reverse order** check box in the third step of the Model Transformation Wizard. This is useful, when there are two related opposite transformations for some domain; for example, the same type map is used for both UML to XML schema and XML schema to UML transformations.

If you want to limit the directions, in which type map can be used, you can set the **defaultDirection** tag for your type map package. Possible values are **forward**, **reverse**, and **both** (default).

The direction can also be limited on a per-rule basis. This is controlled by setting the **direction** tag on the type map dependency. Then the mapping rule is excluded from the rule set when the type map is run in a different direction than specified for this rule.

Multiple rules for the same source type

There can be multiple mapping rules for the same source type. For example, *String* > *varchar* and *String* > *nvarchar*. In this case, one of the rules must be marked as default by setting the **default** tag value on it to true.

> The type map having several rules for the same type and without any one set as default cannot be used.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

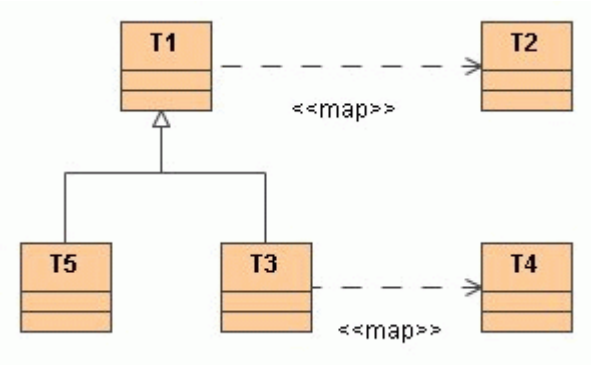
During the initial transformation, only the default rules for each source type come into play. E.g., if the user has a property with the *String* type, this will be transformed to property having the *varchar* type set.

However, during the transformation update, all rules come into play. If the destination type is one of the acceptable types according to the map, it is not changed. Otherwise it is replaced with the default mapping.

Regarding the example above, let's say that after the initial transformation, the user changes the type of the property in the destination model from *varchar* to *nvarchar* (as a post-transformation refinement process). If the user now runs a transformation update, this change will not be overwritten, since *nvarchar* is an acceptable
type as there is a *String* > *nvarchar* mapping in the type map as well. If on the other hand the user sets the type of this property to number, this would be reset the during transformation update, and the type will be forced back to *varchar*, as there is no *String* > *number* mapping.

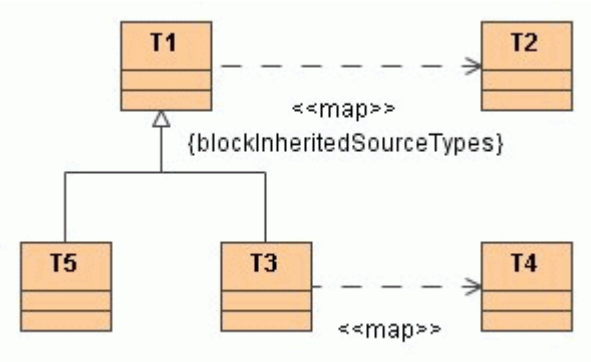
Controlling type inheritance, any, and empty types

You can also control mapping behavior for the type inheritance. By default, derived subtypes are also mapped by the rule governing the parent type (unless, of course, they have their own rules for mapping). If the **blockInheritedSourceTypes** tagged value is set, derived types are not affected by this rule. Let's review the following example:



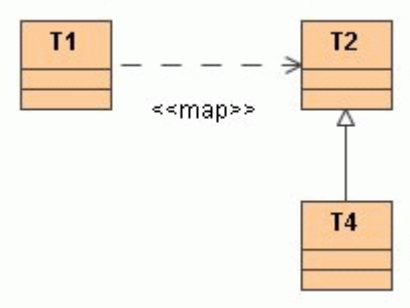
Here T1, T2, and T5 are types in the source domain, while T2 and T4 are types in the destination domain. Given these two mappings (T1 > T2 and T3 > T4), the following statement is true: T1, and all types derived from it (such as T5), are mapped to the T2 type, except T3 and any of the types derived from it. These types are
mapped to T4.

Now consider an example where blockInheritedSourceTypes is set:



In this case, T3, along with the types derived from it, are still mapped to T4. T1 is still mapped to T2. However, unlike the previous example, T5 and all the types inherited from T1 are NOT mapped to T2.

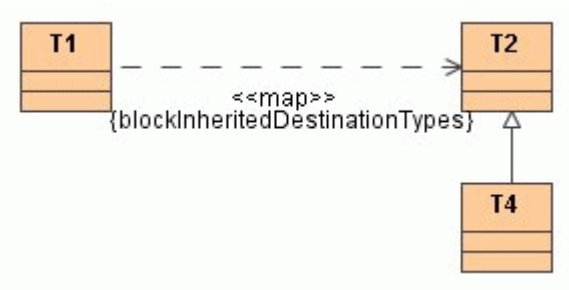
You can also control the mapping behavior of the type inheritance in the destination model. This is only effective on the transformation updates, the second (and successive) reapplications of the transformation. By default, derived subtypes in the destination model are not overwritten, since they are considered suitable substitutes of their parent. Let's review the following example:



Here T1 is a type in the source domain, while T2 and T4 are types in the destination domain. Given this mapping (T1 > T2), on the first application of the transformation, type T1 residing in the source model will be mapped to type T2 in the destination model.

Now let us look at a case, where the user refines the destination model by changing the type on the destination model attribute from T2 to T4. This situation is quite common, for example, the user refines an attribute type from string to basic URI in the XML schema, or from Integer to *nonNegativeInteger*, and so forth. The essence is that the mapping for inherited types of T2 is performed as if there was a mapping T1 > T2 (default), T1 > T4, T1 > <any_other_type_inherited_from_T2>.

Now consider what happens, when we apply the **blockInheritedDestinationTypes** tagged value:



In this case, type T4 has no special treatment. If the user applies the transformation, T1 is mapped to T2. Afterwards the user refines the destination model, changing the attribute type from T2 to T4. If the user now updates the transformation, the attribute type is overwritten: T4 is reset back to T2.

When the user loads the type map in the reverse direction, the roles of the **blockInheritedSourceTypes** and **blockInheritedDestinationTypes** are transposed (unless of course the **direction** tag mandates that this mapping is not used in the reverse direction).

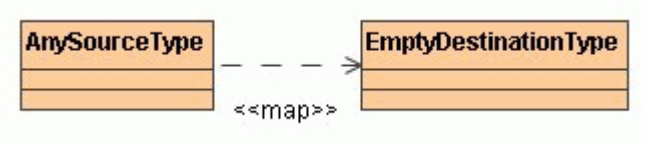
The special type **EmptySourceType** (residing in the *Model_Transformation_Profile.xml.zip*) is used in type maps to indicate that the attributes with no type should be mapped with this dependency.

The special type **EmptyDestinationType** (residing in the *Model_Transformation_Profile.xml.zip*) is used to indicate that the attributes in the destination classes should have no type after remapping (type removal).

The special type **AnySourceType** is a template that matches any type in the source model (see mapping rules for type inheritance). By using this type, together with the inheritance mapping rules, the user can specify that any other types not defined by the mapping should be interpreted by this mapping.

The special type **AnyDestinationType** is a template that matches any type in the destination model (see mapping rules for type inheritance).

Here is an example of template type usage:



According to this rule, any types in the source model for which there are no other mapping rules should be stripped in the destination model.

Type modifiers

Type mapping rules can also affect type modifiers during the type replacement. Type modifier is a small string, which modifies type usage in the typed element. They are used, for example, for specifying arrays during the modeling (e.g., property type = char and type modifier = [30] gives property:char[30]). Type modifiers are
extensively used in SQL models for specifying number field widths and varchar field lengths. For example, phone:varchar"(100)", where varchar is a type of phone property and "(100)" is a type modifier.

Each type mapping rule can carry a triple <modifier, regexp, replacementregexp> for setting type modifiers during the type replacement. These are specified in the tags on the mapping rule <**forwardTypeModifier, forwardTypeModifierRegexp, forwardTypeModifierRegexpReplace**> triple for controlling modifiers during
the forward application of type map and correspondingly the <**reverseTypeModifier, reverseTypeModifierRegexp, reverseTypeModifierRegexpReplace**> triple for controlling modifiers during reverse application of type map.

Any of the components of the triple can be missing, i.e., not specified.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

If no tags are specified, then type modifiers are not changed during the type remapping operation (whatever modifier was in the source model, it will be copied into the target model).

If just the modifier is specified for the mapping rule, then modifiers are set during the application of this type rule. This can be used for setting the fixed type modifiers. For example, mapping boolean in the UML model to number(1) in the SQL models (in this case the modifier="(1)" is used in the type map).

If all three are specified, a modifier, regexp, and regexp replacement, modifier remapping is performed as follows: during the transformation, the existing type modifier is matched against the given regexp. If it does not match, the type modifier is overwritten with the value, specified in the modifier field of the rule. If it does match
regexp, the replacement is run on the match result and produces a type modifier to be set as a result. This allows quite complex rules to be written and executed, however this mandates good knowledge of regexp.

Let's review the following live example: in the char -> varchar type mapping rule for the UML to SQL transformation, the following triple can be used: modifier="(255)", modifierRegexp="^[\(\[]([0-9]*)[\)\]] $", and modifierRegexpReplace="($1)". This causes the char[20] type usages (type=char, modifier="[20]") in the source be changed to varchar(20); char (without modifier) would be remapped to varchar(255).

If regexp replacement is not specified, it is treated as if "$0" was specified: the type modifier is copied from the source, if it does match the regexp.

Transformation Traces and Update

When a transformation is performed, it establishes traces between the transformation source model elements and the transformation result model elements. These traces carry information of what was transformed into what. Traces are stored in an auxiliary package under the transformation destination package carrying the
«transformation» stereotype and containing a lot of instance specifications (since this is implementation specific data, please, do not edit the internals). If you want to remove trace information, simply delete this package.

Trace information can be used for navigating between the model layers. This is done with the traceability features of MagicDraw. To navigate in the forward direction, i.e., from the transformation source model element to the destination model element, right-click that element and choose **Go To** > **Traceability** > **Model**
Transformations > **Transformed To** > <element>. To navigate in the backward direction, i.e., from the transformation destination model element to the source model element, right-click that element and choose **Go To** > **Traceability** > **Model Transformations** > **Transformed From** > <element>.

Traceability information is also visible in the element's Specification window(see page 913), the **Traceability** tab; in the Properties panel, **Traceability** tab; it can also be depicted in the Relation Map diagram or in the custom dependency matrix.

Traces can be used for running the transformation update. The transformation update reapplies the transformation with the same source and target for the purpose of carrying additional changes from the source (which occurred after transformation was made) into the destination.

During the transformation update, presence of unmapped model elements in the source model indicates that these are newly added elements. Usual rules and the same behavior for the transformation are used for these elements as if this were the first application of the transformation.

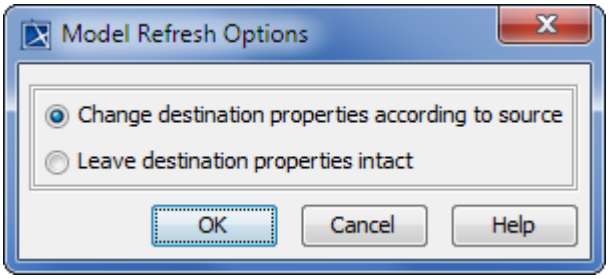
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

During the transformation update, if model elements already contain mapping, and the source and destination does not match, the question arises - which properties to use. This is fundamental problem for all updating operations. The general solution is to have some kind of merge between the source and target. But merge is
clumsy and expensive. MagicDraw implements a simplistic approach meaning that during the transformation update a user can choose, wherever he/she wants the source model element properties to win (destination model element properties are overwritten) or wherever he/she wants the destination model properties to win (destination model properties are not changed).

To update a transformed model

- Right-click the destination package and select **Tools** > **Update Transformed Model**.

The Model Refresh Options dialog opens.



The **Change destination properties according to source** option causes overwriting of element properties in the destination model with properties from the source model (only for the elements connected with mapping dependencies).

The **Leave destination properties intact** option leaves the destination model properties unchanged but different from the source model, in other words retains changes made to the destination model while it ignores changes made to the source model.

In Oracle DDL transformation updates, only changing of destination properties according to the source is available. It means, all changes are overwritten when updating a transformed Oracle DDL model, and the **Model Refresh Options** dialog does not appear in this case.

> If new elements are added to the source, once the update occurs, copies of the new elements will be created in the destination model. If an element is deleted
> from the source, it will not be removed from the destination after the update.

Profile Migration Transformation

Transformation provides a way to migrate your model according specified mapping rules.

Profile migration transformation search-and-replaces the following elements:

- Applied stereotypes
- Tagged values
- Usages of one type with another type

Element symbols are updated according to model changes.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Before starting transformation you have to define transformation mapping.

Profile Migration Transformation mapping

Before starting transformation you have to create transformation mapping rules.

To create mapping rules you need to create Dependency relationship between elements you want to transform.

Mapping rules can be created (dependency relationship can be created) between the following elements:

1. Stereotypes
2. Tags
3. Types

Creating mapping rules for Stereotype transformation

This type of transformation is used to replace Stereotype.

To create Stereotype transformation mapping rule

---

1. Create Dependency relationship between Stereotypes which you want to transform.
2. Apply *ReplaceStereotype* stereotype to Dependency.
3. Perform transformation.

Tag values of old stereotype are preserved when tag name and type of tag value is the same. For tag values with different names create mapping rule for tag transformation.

ReplaceStereotype stereotype has the following tags:

- **disableNewTypeCreation** tag. By default false. Set this tag to true if you do not want to perform transformation when target and source metaclasses are not compatible. For example, if you do not want that Class would be changed to Use Case.
- **disableReplaceWhereSaveAsElementValue** tag. By default false. Set this tag to true if you do not want that stereotype would be changed where it is used as Tag value (tag value is stereotype itself, for which you perform transformation).

Creating mapping rules for Tag transformation

This type of transformation is used to replace Tag (when tag names differs). For example, source stereotype has author Tag and target stereotype has name Tag.

To create Tag transformation mapping rule

---

1. Create Dependency relationship between Tags which you want to transform.
2. Apply *ReplaceTaggedValue* stereotype to Dependency.
3. Perform transformation.

> ⚠ To create mapping rule correctly, you have to create Dependency relationship not only between Tags, but also between Stereotypes of these tags (see mapping between stereotypes, which is described above).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

This type of transformation is used to replace type. For example, to replace type of Attribute.

To create types transformation mapping rule

1. Create Dependency relationship between Types which you want to transform.
2. Apply *ReplaceType* stereotype to Dependency.
3. Perform transformation.

Starting Profile Migration Transformation

To start the **Model Transformation Wizard**, do one of the following

- From the **Tools** menu, choose **Model Transformations**.
- Right-click one or more packages and select **Tools** > **Transform** .

To start the **Profile Migration Transformation**

- In the **Model Transformation Wizard**, select the **Profile Migration** transformation.

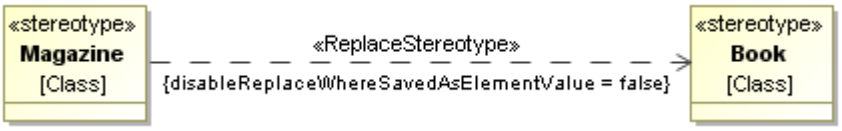
The **Next** step is disabled in the wizard, if there are no mapping defined.

Sample of the Profile Migration Transformation

This sample describes step-by-step instructions how to create profile migration mapping rules and perform transformation. In this sample we will change one stereotype to other.

1. Create *Book* stereotype with Class metaclass.
2. Create your model, for example, create *Source* package with *Source Class* diagram. Draw the *Sample Class* and apply the *Book* stereotype.
   To change the *Book* stereotype to other, for example to *Magazine* stereotype, you have to create profile migration transformation mapping rules. Follow next steps for creating the mapping.
3. Create the stereotype *Magazine* with *Class* metaclass.
4. From the *Magazine* stereotype to the *Book* stereotype draw Dependency relationship.
5. Apply *ReplaceStereotype* stereotype to the Dependency relationship.
   Profile Migration transformation mapping rule is created. Now you can start transformation.
6. To start the transformation, from the **Tools** menu, select **Model Transformations**. The Model Transformation Wizard(see page 1132) opens.
7. Select the **Profile Migration** transformation and click **Next**.
8. Select the *Source* package in the **Select source/destination** step. Click **Next**.
9. The **From** and **To** fields display the mappings of the selected transformation profile in the **Check mappings** step. Click **Finish** .

After this transformation stereotype of Sample Class will be changed to Magazine stereotype.



Executable concepts

We call executable such concepts that can be specified by some expressions and calculated. In modeling tools developed by No Magic, executable are:

- Validation rules(see page 1306) - that are predefined or custom constraints having a purpose to validate a model or particular model elements.
- Derived properties - properties that are automatically calculated from the other properties.
- Smart packages(see page 824) - that are special collections of model elements.
- Executable opaque behaviors - reusable expression fragments callable form other executable concepts.
- Metric definitions - calculate aggregate parameters of models.
- Macros(see page 2700) - a script that is created using a particular script language. With the help of macros, you can control items that are allowed in Developer Guide(see page 2156).
- Opaque behaviors(see page 799) - a behavior with implementation-specific semantics.

In a modeling tool, executable concepts are defined in one of the languages that our modeling tool can evaluate:

- StructuredExpression(see page 1228) - a graphical way to construct expressions out of predefined blocks.
- Object Constraint Language(see page 1147).
- Binary - an expression that is written in Java, compiled and added to our modeling tool with a plugin.
- One of the following scripting languages:
    - Groovy
    - JRuby
    - JavaScript
    - JavaScript Rhino
    - Jython

> **Additional scripting languages**
> Additional scripting languages can be plugged into our modeling tool, if they have JSR-223-compliant implementation.

You can test and evaluate your expressions with the help of the Expression Evaluation tool(see page 1273).

Related pages

Object Constraint Language

Object Constraint Language (OCL) is a formal language used to express constraints. OCL typically specify the invariant conditions that must hold for the system being modeled.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Expressions can be used in a number of places in the UML model:

- To specify the initial value of an attribute or association end.
- To specify the derivation rule for an attribute or association end.
- To specify the body of an operation.
- To indicate an instance in a dynamic diagram.
- To indicate a condition in a dynamic diagram.
- To indicate the actual parameter values in a dynamic diagram.

There are four types of constraints:

- An *invariant* is a constraint that states a condition that must always be met by all instances of the class, type, or interface. The invariant is described using an expression that evaluates to true if the invariant is met. Invariants must be true all the time.
- A *precondition* to an operation is a restriction that must be true at the moment the operation is going to be executed. The obligations are specified by the postconditions.
- A *postcondition* to an operation is a restriction that must be true at the moment the operation has just been executed.
- A *guard* is a constraint that must be true before a state transition discharged.

You can use derived properties in the OCL expressions. Scripts having multiple parameters can now be defined using OCL in structured expressions or Opaque Behaviors(see page 799)

Invariants on attributes

The simplest constraint is an invariant on an attribute. Suppose a model contains a class *Customer* with an attribute age, then the following constraint restricts the value of the attribute:

```
context Customer inv :
age >= 18
```

Invariants on associations

One may also put constraints on the associated objects. Suppose a model contains the class Customer that has an association to the class Salesperson with the role name salesrep and multiplicity 1, then the following constraint restricts the value of the attribute knowledge level of the associated instance of Salesperson:

```
context Customer inv :
salesrep.knowledgelevel >= 5
```

Collections of objects

In most cases the multiplicity of an association is not 1, but more than 1. Evaluating a constraint in these cases will result in a collection of instances of the associated class. Constraints can be put on either the collection itself, e.g. limiting the size, or on the elements of the collection. Suppose in a model the association between
Salesperson and Customer has the role name clients and multiplicity 1..* on the side of the Customer class, then we might restrict this relationship by the following constraints:

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

```
context Salesperson inv :
clients->size() <= 100 and clients->forAll(c: Customer | c.age >= 40)
```

Pre- and postconditions

In the pre- and postconditions the parameters of the operation can be used. Furthermore, there is a special keyword result which denotes the return value of the operation. It can be used in the postcondition only. For example an operation sell was added to the Salesperson class.

```
context Salesperson::sell( item: Thing ): Real
pre : self.sellableItems->includes( item )
post: not self.sellableItems->includes( item ) and result = item.price
```

Derivation Rules

Models often define derived attributes and associations. A derived element does not stand alone. The value of a derived element must always be determined from other (base) values in the model. Omitting the way to derive the element value results in an incomplete model. Using OCL, the derivation can be expressed in a derivation rule. In the following example, the value of a derived element usedServices is defined to be all services that have generated transactions on the account:

```
context LoyaltyAccount::usedServices : Set(Services)
derive : transactions.service->asSet()
```

Initial Values

In the model information, the initial value of an attribute or association role can be specified by an OCL expression. In the following examples, the initial value for the attribute points is 0, and for the association end transactions, it is an empty set:

```
context LoyaltyAccount::points : Integer
init: 0
context LoyaltyAccount::transactions : Set(Transaction)
init: Set{}
```

Body of Query Operations

The class diagram can introduce a number of query operations. The query operations are operations that have no side effects, i.e. do not change the state of any instance in the system. The execution of a query operation results in a value or set of values without any alterations in the state of the system. The query operations can
be introduced in the class diagram, but can only be fully defined by specifying the result of the operation. Using OCL, the result can be given in a single expression, called a body expression. In fact, OCL is a full query language, comparable to SQL. The use of body expressions is an illustration thereof.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

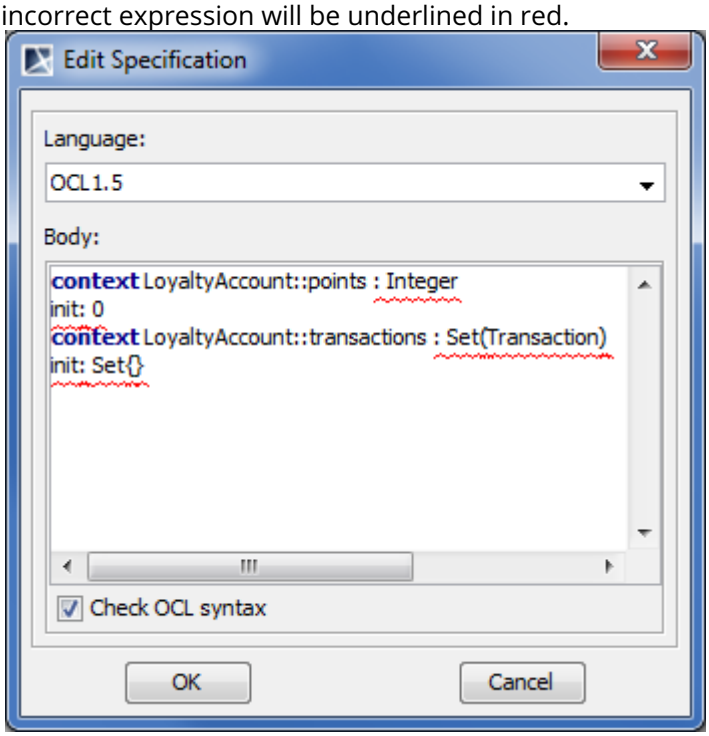
The next example states that the operation *getCustomerName* will always result in the name of the card owner associated with the loyalty account:

```
context LoyaltyAccount::getCustomerName() : String
body: Membership.card.owner.name
```

To check OCL syntax according to OCL grammar

---

1. In the constraint Specification window, click the Specification property value.
2. Click the **...** button in the property value cell. The **Edit Specification** window opens.
3. In the Language list, click **OCL** and select the **Check OCL syntax** check box. In the Body box, incorrect expression will be underlined in red.



Resource Manager

Resource Manager functionality allows you to manage local resources (installed with a modeling tool or downloaded separately) and resources available on the web.

With Resource Manager you can manage different types of resources, such as Profiles, Plugins, Templates, Language resources, Case studies/examples, Custom diagrams, and others.

The Resource Manager functionality allows you to:

- Download and install the resources
- Update the resources

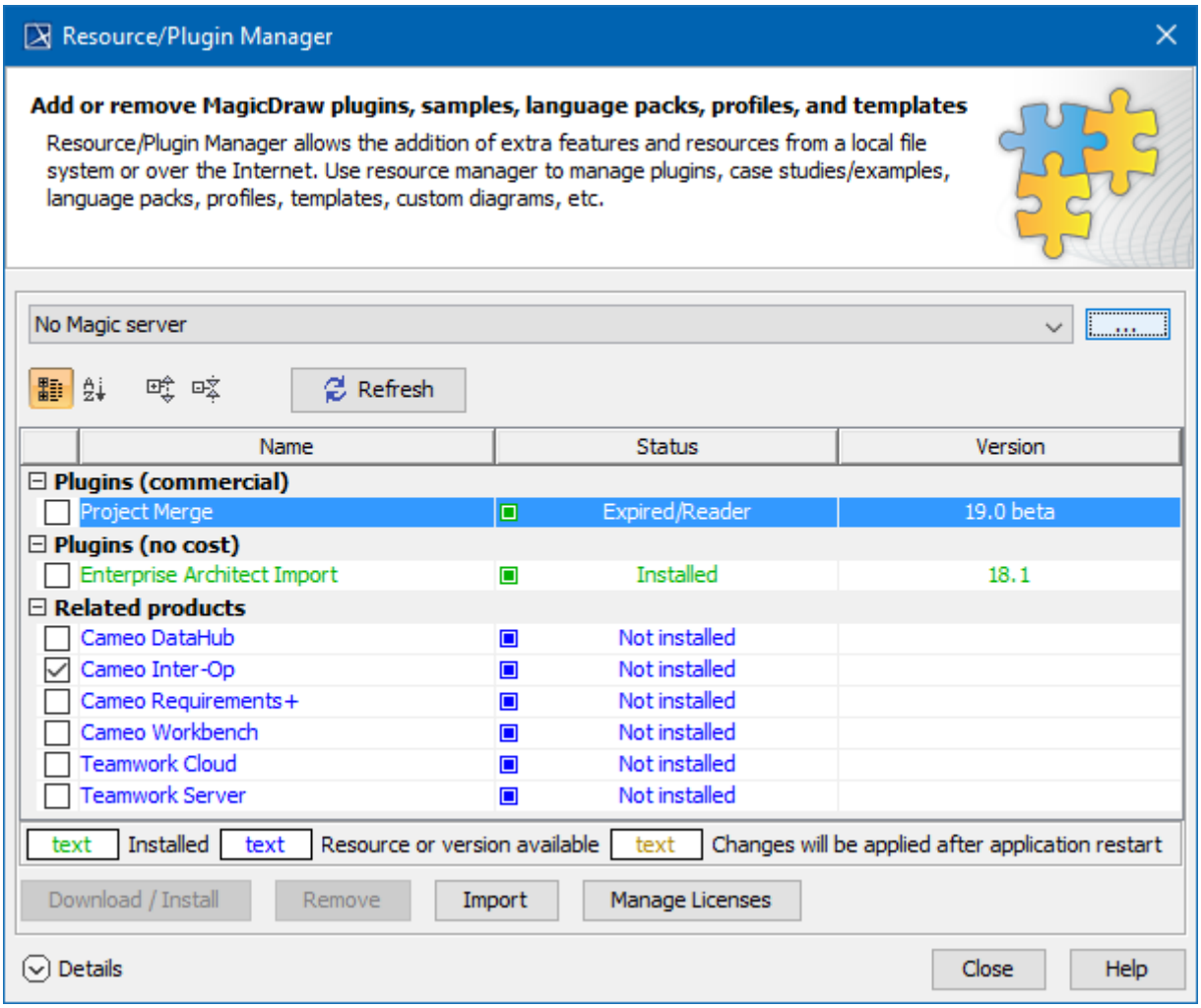
- Distribute needed resources internally (including company's developed resources)
- Check dependency of a resource.

The **Resource/Plugin Manager** helps you manage the resources. You can add the resources to your modeling tool from the following locations:

- Local file system
- Network share
- Web server

To open the Resource/Plugin Manager dialog

- From the **Help** menu, select **Resource/Plugin Manager**.



The Resource/Plugin manager dialog.

To see the resource details

1. In the **Resource/Plugin Manager** dialog, select the necessary plugin or other resource.
2. Click **Details**. The dialog expands with the resource description at the bottom.

Distributing resources

You can bundle your resources into one .rdzip file and then distribute that file on a local file system, network share, or web server.

To build the resources file, you need the Development Tools plugin to be installed. The **Development Tools** plugin can either be installed from the No Magic Server in the **Resource/Plugin Manager**, or can be found in the no-cost resource distribution file.

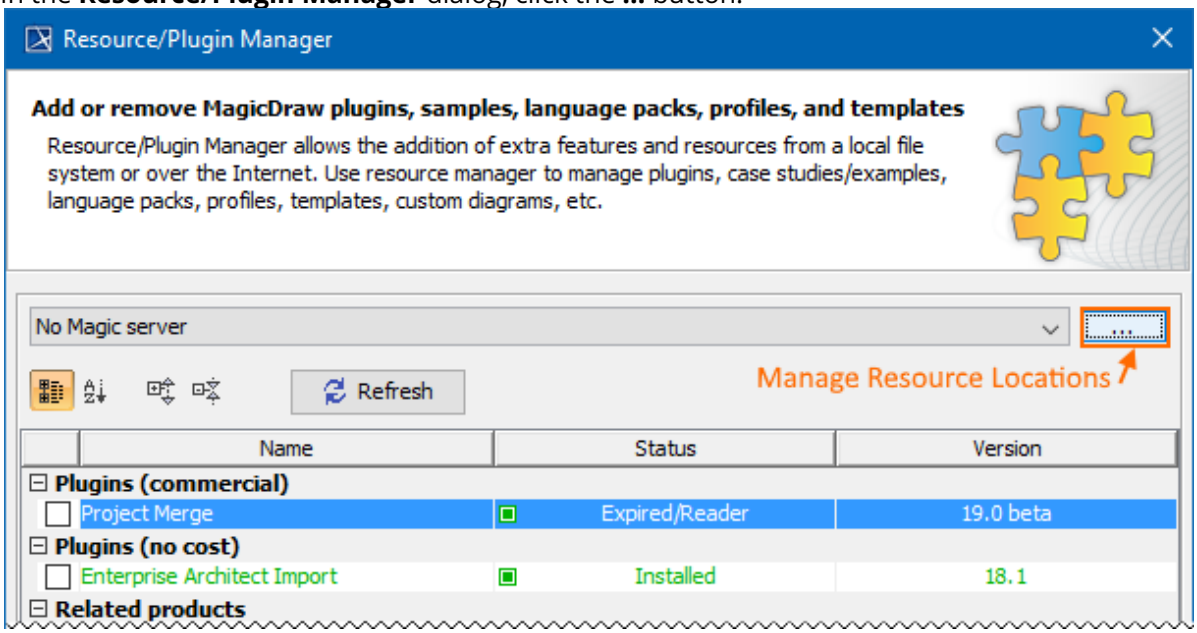
> For more information about building the resource distribution file, read Building a resource distribution file(see page 2196).

Manage Resource Locations

The **Manage Resource Locations** dialog is used for defining the locations from which the resources will be added to the **Resource/Plugin Manager**.

To open the **Manage Resource Locations** dialog

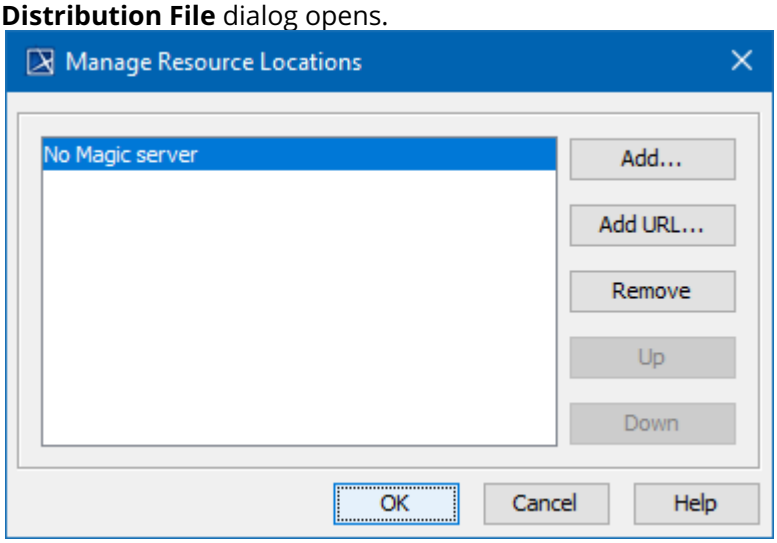
- In the **Resource/Plugin Manager** dialog, click the **...** button.



In the **Manage Resource Locations** dialog, you will be able to see the defined resource locations. You can manage this locations list by adding, removing, or reordering the resource locations.

To add location of resources

1. In the **Manage Resource Locations** dialog, click **Add** button. The **Select Resource Location or Distribution File** dialog opens.



2. In the open dialog, select one of the following:
   - A location from the local file system or a network share, where the resources and their descriptor file are located.
   - Resource distribution file (*.rdzip*)

The resources from the Resource Distribution File will be listed in **Resource/Plugin Manager** dialog.

> The resource distribution file (*.rdzip*) contains the resources and their descriptor file. If you are using the web server for the resource distribution, make sure that the resource distribution file (*.rdzip*) is extracted into the intended server location.

To specify the server where the resources and their descriptor file are located

1. In the **Manage Resource Locations** dialog, click **Add URL**. The **Resource Server URL** dialog opens.
2. In the open dialog, type or paste the server address.
3. Click **OK**.

The resources from the selected location will be listed in the **Resource/Plugin Manager**.

> If you need to add the No Magic server, enter <*nomagic.resource.server*> in the **Resource Server URL** dialog.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- The No Magic server is not available for the Magic Software Engineer, Magic Systems of Systems Architect, and Magic Cyber Systems Engineer modeling tools.
- The resource distribution file (*.rdzip*) is added to a specific folder (*<install.root>/data/resourcemanager/distribution*>) for the Magic Software Engineer, Magic Systems of Systems Architect, and Magic Cyber Systems Engineer modeling tools. This location is set by default for all the new modeling tools.

Related pages

- Distributing Resources(see page 2185)
- Distributing Profiles(see page 2156)
- Building a resource distribution file(see page 2196)
- License management[250]

Creating Setters and Getters

Modeling Tools
This functionality is available in Standard, Professional, Architect and Enterprise editions only.

Setters and Getters are common operations that contain almost every Class(see page 666). With the help of our modeling tool, **set** and **get** operations for Class attributes can be generated automatically.

To create a setter or a getter

There are two ways of creating a setter or a getter:

1. From the shortcut menu of the selected Class, select **Tools**, and then select **Create Setters/Getters**.
    a. The **Select Attributes/Association Ends** dialog opens.
2. Add a tagged value "getter/setter for attribute=attribute_name" to the selected Class.

The names of created operations (**setters**) are combined according to the following format:

public void set + <attribute name> (<attribute type> <attribute name>)

Example
If you have an attribute called x of type int, then the generated **setter** will look this way:
- *public void setx (int x);*

250 https://docs.nomagic.com/display/MD190SP4/License+Manager+dialog

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

The names of created operations (**getters**) are combined according to the following format:

public <attribute type> get + <attribute name> ( )

> **Example**
> If you have an attribute called x of type int, then the generated **getter** will look this way:
> - *public int getx ();*

Related Pages

- Class(see page 666)
- Tools(see page 1083)

Implementing or Overriding Operations

> **Modeling Tools**
> This functionality is available in Standard, Professional, Architect and Enterprise editions only.

When you inherit Classes from the base Class that has abstract functions, you have to redefine them in the inherited Classes. The implement/override operations tool will help you generate operations that are defined as abstract in the base Class.

The **Implement/Override Operations** command can be invoked in two cases:

- When one classifier inherits operations from the base classifier (Generalization relationship(see page 2110)).
- When some classifiers implement an Interface (Realization relationship(see page 1215)).

To start the Implement/Override Operations tool

1. From the shortcut menu(see page 1155) of the selected Class, select **Tools**.
2. Select **Implement/Override Operations.**
3. The Select Operations to Implement/Override dialog box opens.

Box	Function
All	Contains names of all operations defined within the selected class.
Selected	Contains the selected operations.

Box	Function
>	Moves the selected operation from the **All** list to the **Selected** list.
<	Moves selected operation from the **Selected** list to the **All** list.
>>	Moves all operations from the **All** list to the **Selected** list.
<<	Moves all operations from the **Selected** to the **All** list.
OK	Generates operations that are in the **Selected** list.
Cancel	Exits the dialog box without any changes.
Help	Displays the MagicDraw Help.

Shortcut Key
Double-click the item name and it will be moved to the opposite list.

Related Pages

- Tools(see page 1083)
- Class(see page 666)
- Class diagram(see page 297)
- Generalization relationship(see page 2110)
- Realization relationship(see page 1215)

Automatic Instantiation Wizard

This tool allows you to create instances of various entities with just a few clicks. It is useful for working with complex models, and for assembling large systems from parts.

While creating instances automatically, you can:

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Select properties for which the instances will be created
- Change types of instantiated properties
- Assign default values for the instantiated properties

For properties having the multiplicity more than one, you can create as many instances of the same type as you need, since parallel parts can be added while creating instances. Previously, this feature had been supported by the **SysML plugin only**.

The **Automatic Instantiation wizard** analyzes the structure of the selected element and collects all available information about that element from the model (attributes, properties, and so on). All this information is presented in the first step of the wizard.

The **Automatic Instantiation wizard** evaluates collected information and provides a suggestion of all possible instances to be created.

If you are satisfied with default values provided in the **Automatic Instantiation wizard,** you can finish the wizard in the first step without making any changes. The instances will be created with the default values and in your working package. But if you want to change default values, create several different configurations, see the **Working with Automatic Instantiation Wizard** and the **Case study:** sections on how to do that.

The **Automatic Instantiation Wizard** consists of three steps:

- Step #1. Parts selection
- Step #2. Package selection
- Step #3. Diagram creation (optional)

To open the **Automatic Instantiation** wizard

---

1. Select the elements for which you want to create instances.
2. From the shortcut menu, select **Tools** > **Create Instance**.
   The **Automatic Instantiation** wizard opens.

Related pages

- Instance Specification[251]
- Constraint Link(see page 691)
- Value Specification(see page 899)
- Working with Automatic Instantiation Wizard(see page 1157)
- Automatic Instantiation Wizard Case Study(see page 1161)
- Automatic Instantiation wizard in SysML Plugin[252]

Working with Automatic Instantiation Wizard

Following all steps of the **Automatic Instantiation** wizard, you can create several different sets of instances. You can add each set in a different package and even display on the newly created diagrams.

---

251 https://docs.nomagic.com/display/MD190SP4/Instance+Specification
252 https://docs.nomagic.com/display/SYSMLP190SP4/Automatic+Instantiation+wizard

After finishing the wizard, a very large number of instance specifications can be created. That depends on the size of your system. That is why we strongly recommend to select a new package wherein all created instance specifications will be stored.

Step #1. Parts selection

In this step, you can see all the available instance specification values (parts) of your selected element. At this point, you can select which parts or properties to instantiate, create or select values for parts and properties, as well as add or remove parallel parts.

Parts are displayed if the multiplicity is greater than 1. The number of parts displayed depends on the multiplicity.

- If the multiplicity is a particular value, for example 2, then you will see 2 parts displayed in the **Automatic Instantiation** wizard. Also in this case, you will not be able to add parallel parts or remove existing parallel parts.
- If the multiplicity is defined as an interval (for example, 0..*, 1..10, 1..*) or *, only one part will be created by default. If you need more parts, you can create them by pressing **Insert** or selecting **Add parallel part** from the selected property's shortcut menu.
- If the multiplicity is 1, parts will not be created.

The parts and properties are displayed with the suggested types and values. In this step of the **wizard**, you can change them. You can also change the type for the part only in that case, if there is an available sub element for that part in your model (see the Case study).

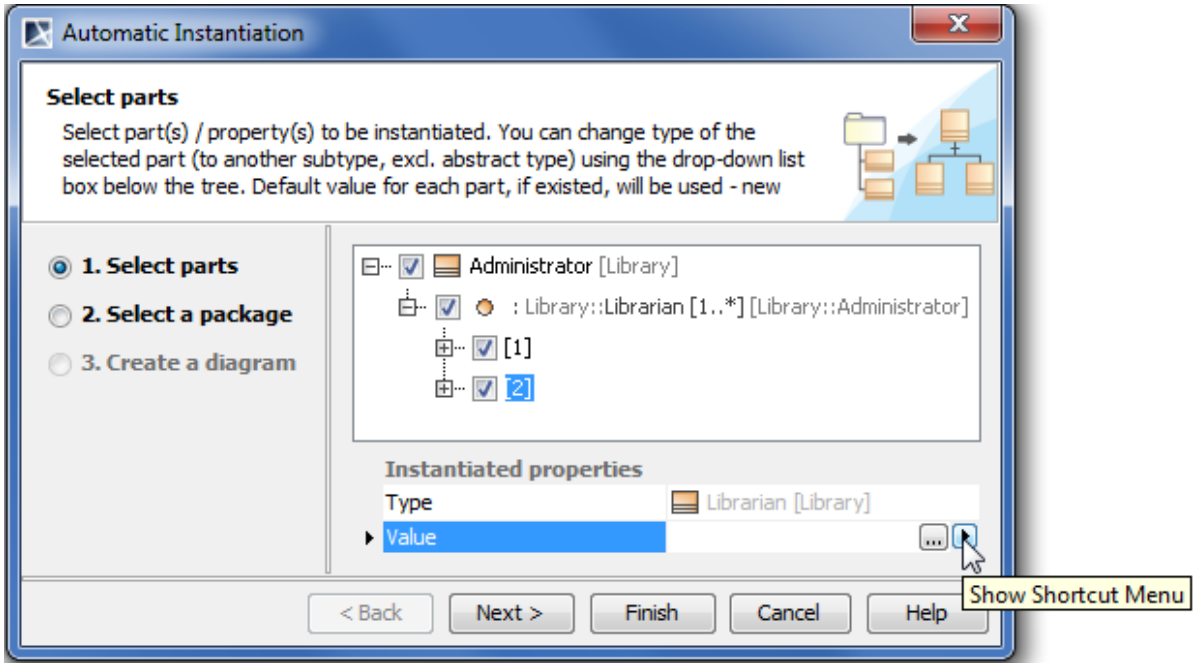
To select a value for a part

1. Select a part for which you want to add a value.
2. In **Instantiated properties**, click a **Value** specification cell and then click the ... button.
3. The **Select Instance Specification** dialog opens.
4. In the opened dialog, select the instance specification you want to add for the selected part.
5. Click the **OK** button.

To create a value for a part

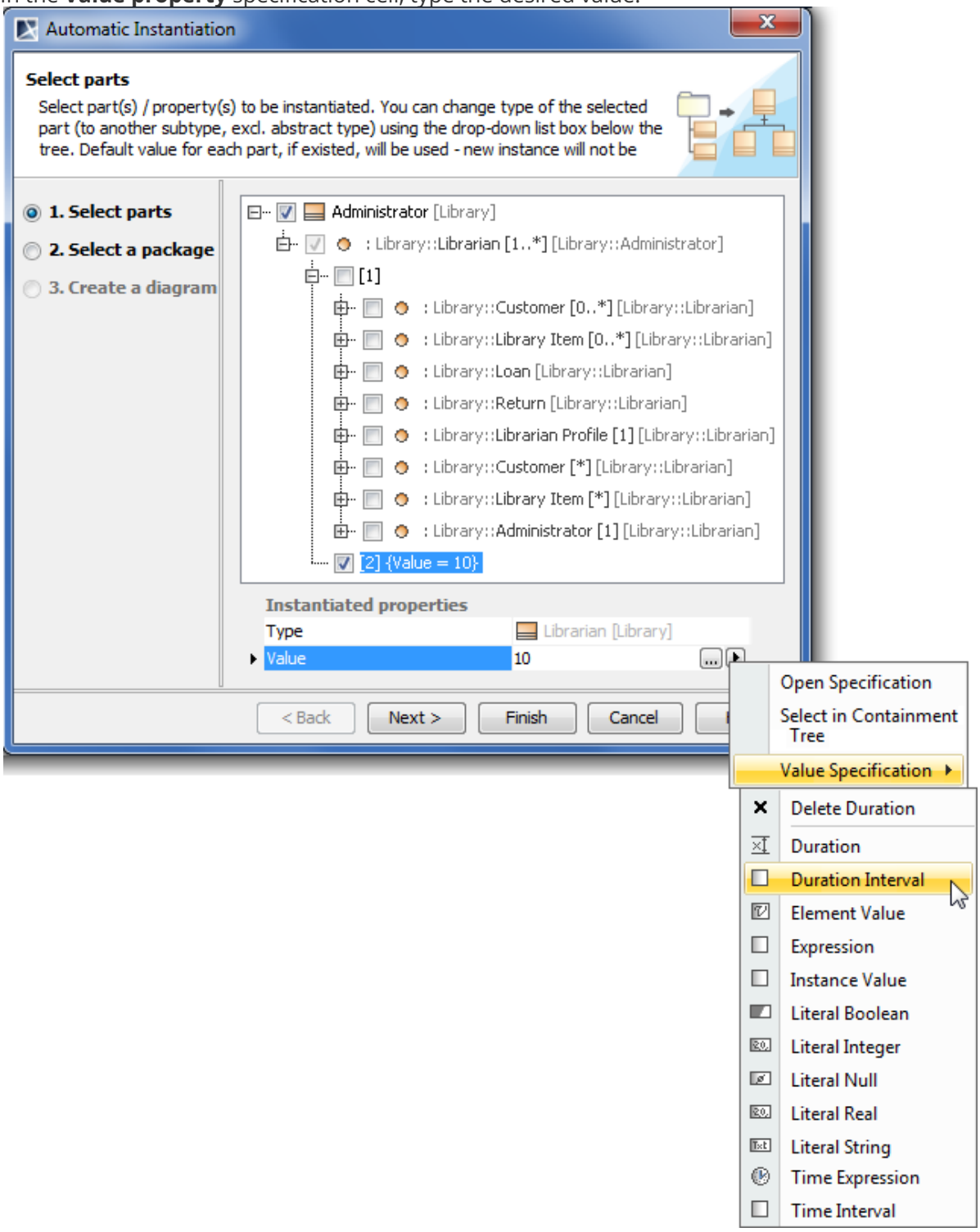
1. Select a part for which you want to create a value.
2. In **Instantiated properties**, click the **Value** specification cell and then click the **Show Shortcut Menu** button.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



3. In the opened shortcut menu, point to the **Value Specification** and then select one of the available value types.

4. In the **Value property** specification cell, type the desired value.



Step #2. Package Selection

In this step, you can select the existing package and either create a new one by clicking the **Create** button, or copy the selected package with all its content by clicking the **Clone** button.

After this step, if you do not wish to display your created instance specifications, click the **Finish** button.

Step #3. Diagram Creation (Optional)

In this step, you can choose whether you want to show your created instance specifications in a new diagram or not. If you do not wish to create a new diagram, you can simply skip this step.

To create a diagram using **Automatic Instantiation** Wizard

1. If you choose to create a new diagram for displaying instance specification(s), check off **Create a new diagram**.
2. If you wish to create and display links between your instance specification(s), check off **Create link between instances**.
3. Fill in the **Type diagram name** field with the appropriate name. By default the diagram name is Instance of the <element name>. For example, "Instance of the Administrator."
4. From the **Select diagram type** drop-down list, you can select one of the diagram types.
5. The final step is to select, create new, or clone an owner for your new diagram.
6. After clicking the **Finish** button, a new diagram with instance specifications is created and displayed in your project.

On this Page

- Step #1. Parts selection(see page 1158)
- Step #2. Package Selection(see page 1160)
- Step #3. Diagram Creation (Optional)(see page 1161)

Related pages

- Automatic Instantiation Wizard Case Study(see page 1161)
- Automatic Instantiation wizard in SysML Plugin[253]

Automatic Instantiation Wizard Case Study

Consider this example:

Let's say we have three classes named **Administrator**, **Librarian**, and **Personnel**. These classes are connected in the following way:

253 https://docs.nomagic.com/display/SYSMLP190SP4/Automatic+Instantiation+wizard

The multiplicity in the association end near the **Librarian** is 4. In this case, the **Automatic Instantiation wizard** suggests four parts.

In the following figure, these values are displayed as **[1]**, **[2]**, **[3]**, and **[4]**. Each of them have a set of properties inherited from the type of the parts. In this case, we cannot add more parts because the multiplicity points to the particular number of available parts. The type of the part can be changed, if there is a sub element in the model.

In this case, class **Librarian** has a sub-class **Personnel**, so the type of **part [1]** can be changed to **Personnel** as you can see in the following figure.

Let's select parts and properties for instantiation:

1. Select a new package, wherein our created instance specifications will be stored.
2. Create a new package **Instances of the Administrator**.
3. Create a new diagram, wherein our created instances will be displayed.
4. After selecting the diagram type (in this case, we will select to create a class diagram) and owner, let's click the **Finish** button.
5. See the result in the following figure.

Pattern Wizard

> **Edition Availability**
> This functionality is available in Standard, Professional, and Enterprise editions only.

In our modeling tool, you can find various GOF, Java, Junit, CORBA IDL, and XML Schema Design Patterns. Additionally, you can create new patterns and edit existing ones using Java code or JPython scripts.

The **Pattern Wizard** helps to create a class with some predefined properties.

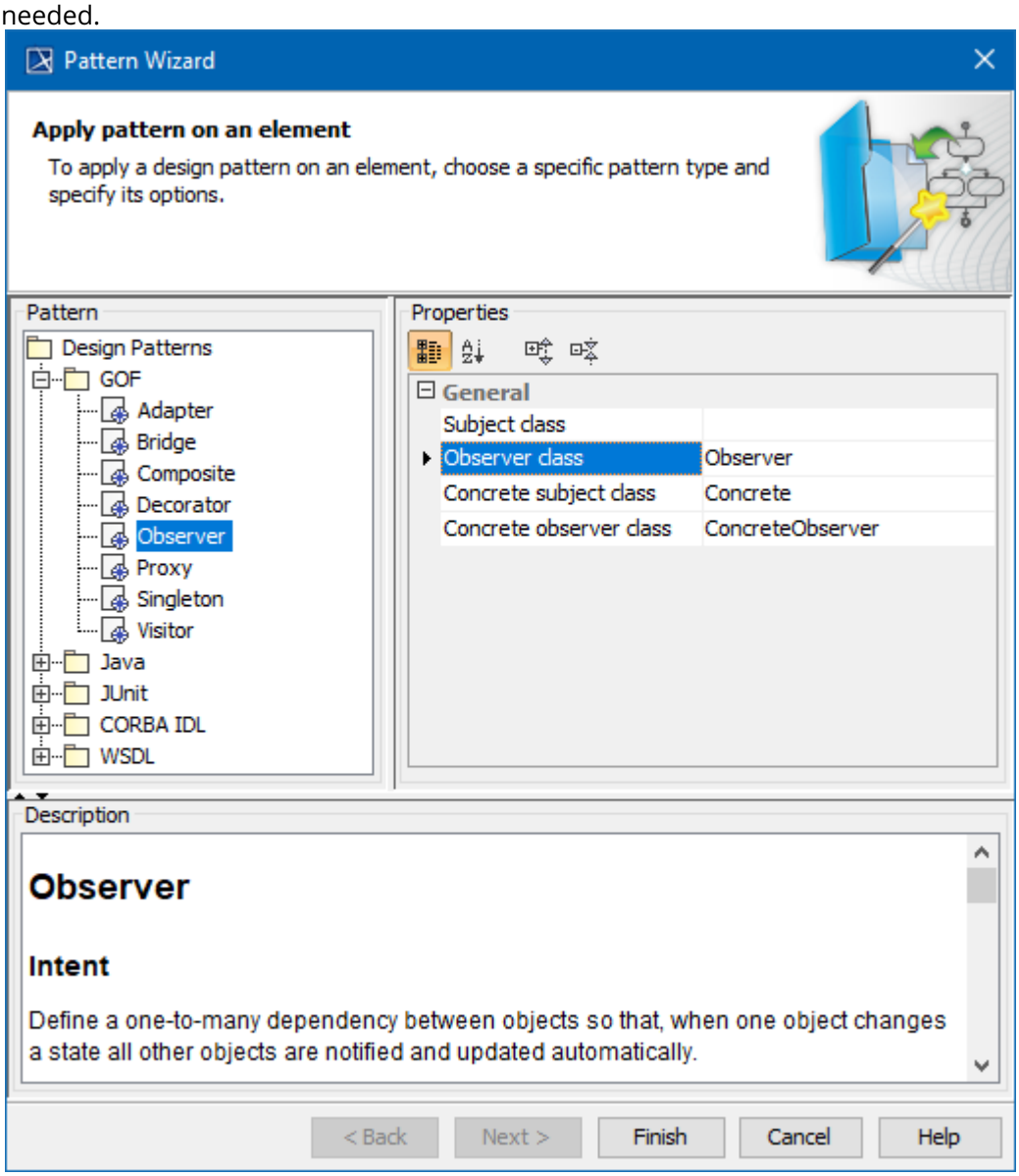
To apply a pattern to a class

In your project, if you have a Class diagram(see page 297) created, do the following:

1. Select a Class(see page 666) and do one of the following:
   - From its shortcut menu(see page 2197), select **Tools** > **Apply Pattern**.
   - On the main menu, click **Tools** > **Apply Pattern**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

2. The **Pattern Wizard** opens. In the wizard, select a desired pattern and specify properties if needed.



3. After you are done choosing your pattern, click **Finish**.

You can find a description of each pattern at the bottom of the dialog in the **Description** area.

Related pages

- Class(see page 666)
- Class diagram(see page 297)

Related pages:

- Class(see page 666)
- Class diagram(see page 297)

Spelling Checker

Spelling Checker enables you to:

- Check spelling as you type(see page 1164). A shortcut menu provides spelling options. Right-click the word underlined in red to enter the shortcut menu and display the Spelling commands. You can also enter words in a customized dictionary.
- Check the spelling of a whole project or of a selected part.(see page 1167) You can list all the spelling errors found in a project and easily correct them.
- Set Spelling Checker options(see page 1169). You can set spelling checker options, such as skipping numbers or uppercase words, in the **Environment Options** dialog(see page 65).
- Add a spell checking dictionaries(see page 1172). All "Open Office" supportive spelling languages can be added to the existing ones.

Related pages

- Spell Checking as you type(see page 1164)
- Spell checking for the entire project or a selected scope(see page 1167)
- Setting the spell checking options(see page 1169)
- Defining properties of the customized element to be spell checked(see page 1172)

Spell Checking as you type

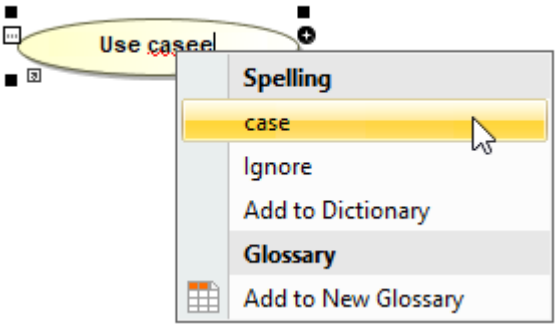
Spell Checker checks if a word is correct as you type. If the word is incorrect, it is underlined with a red winding line. Right-clicking the underlined word brings up a shortcut menu menu with suggested edits and gives you the option to add the word to the dictionary or custom glossary.

Spell checking as you type is performed in the following locations in a modeling tool:

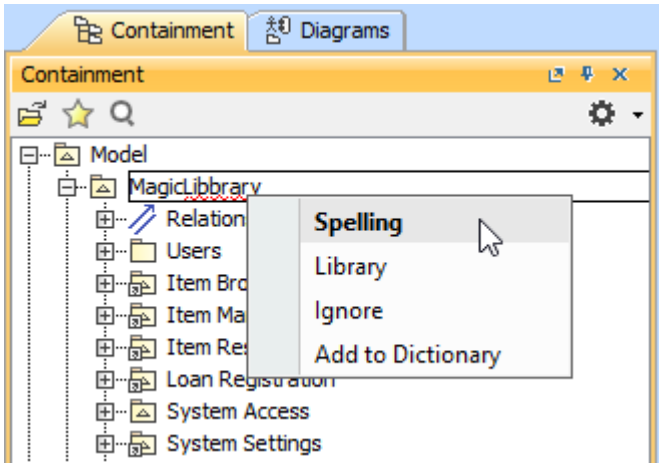
- symbol names on a diagram pane,
- for properties in the Specification windows(see page 913),
- in the Documentation tab(see page 54),
- in the Containment tree(see page 32),
- In various locations on log messages, names, typing and comment boxes.

Right-click the underlined word to invoke the shortcut menu in the following locations:

- Diagram pane

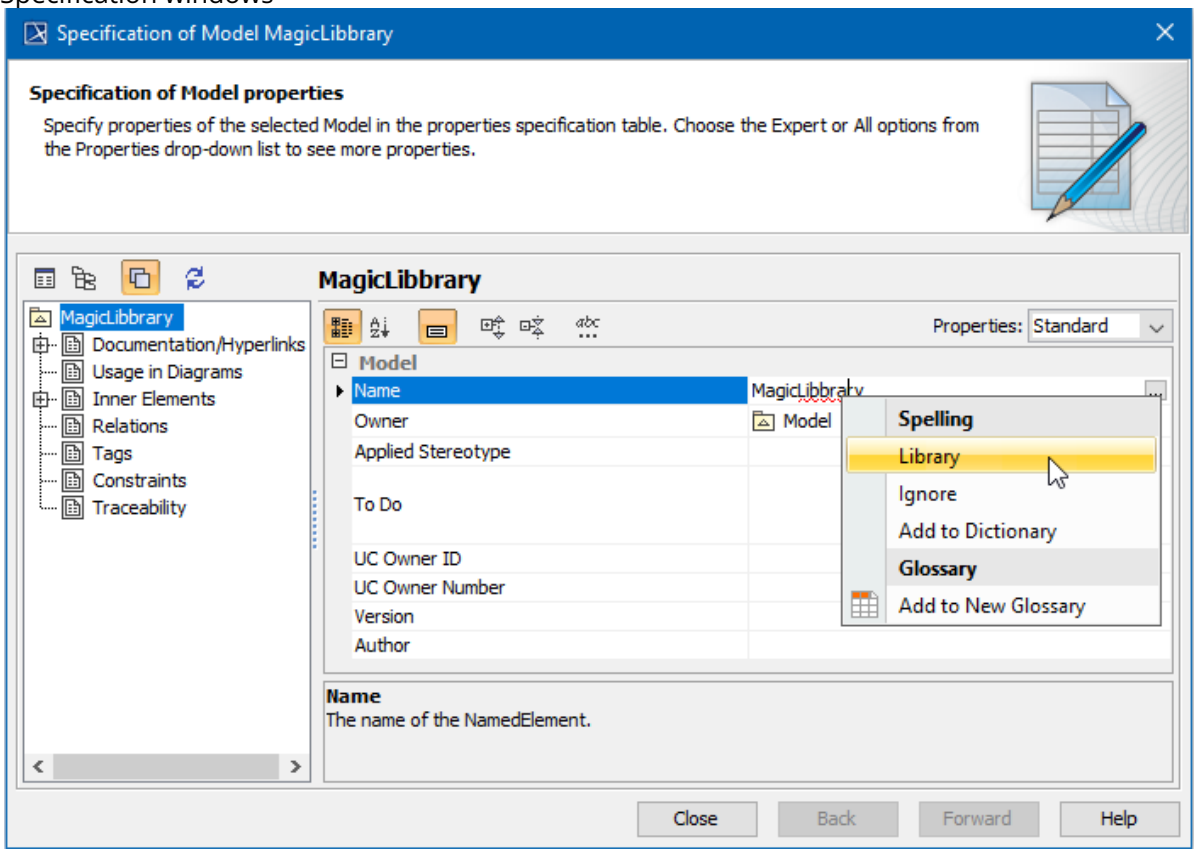


- Containment tree



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Specification windows



Correct the spelling error

Do one of the following:

- An incorrect word can be changed by typing or by selecting provided suggestion that is always correct syntactically, but not always correct semantically.
- The **Ignore** command. If you select **Ignore**, the incorrectly spelled word is treated as correct for this time but will be considered to be incorrectly spelled in other cases.
- The **Add to dictionary** command. If you select **Add to dictionary**, the incorrect word will be added to a custom dictionary. Next time it will not be treated as incorrect.
- The **Add to New Glossary** command. Select **Add to New Glossary** to create a new glossary(see page 452) of terms.

> **Note**
> Incorrectly spelled words are underlined only in the edit mode of a particular component. If you turn edit mode off, the underline disappears and vice versa.

Related Pages:

- Spell Checking as you type(see page 1164)
- Spell checking for the entire project or a selected scope(see page 1167)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

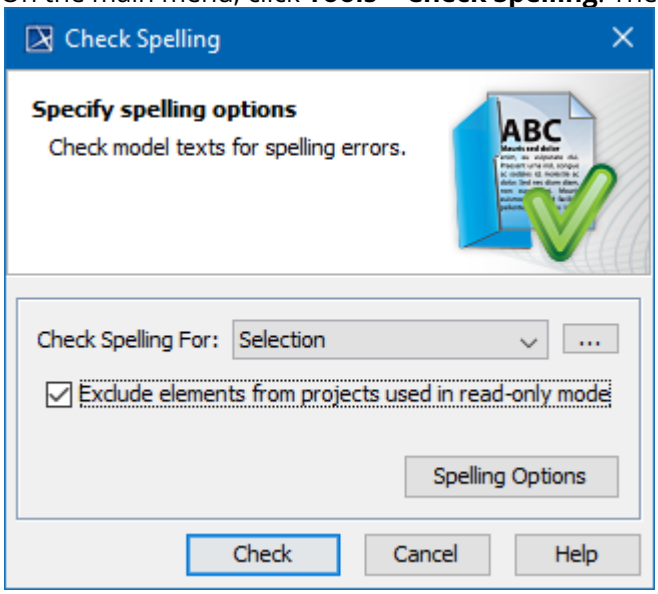
- Setting the spell checking options(see page 1169)
- Defining properties of the customized element to be spell checked(see page 1172)

Spell checking for the entire project or a selected scope

On this page

To check spelling for the entire project or a selected scope

1. On the main menu, click **Tools** > **Check Spelling**. The **Check Spelling** dialog opens.

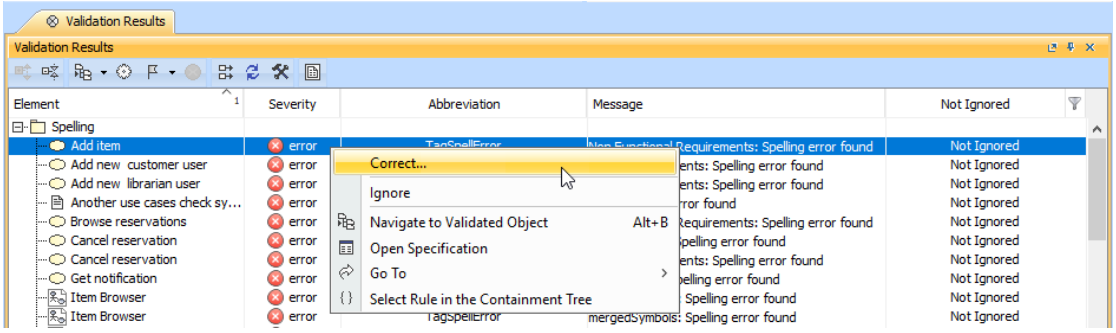


2. In the **Check Spelling For** drop-down list, select the scope. For **Selection**, click ... to define a project scope to check. It may be a package, a diagram, or an element.
3. Click the **Spelling Options** button to specify spelling options in the **Environment Options** dialog(see page 65).
4. Click the **Check** button. Results are displayed in the **Validation Results** panel(see page 1316).

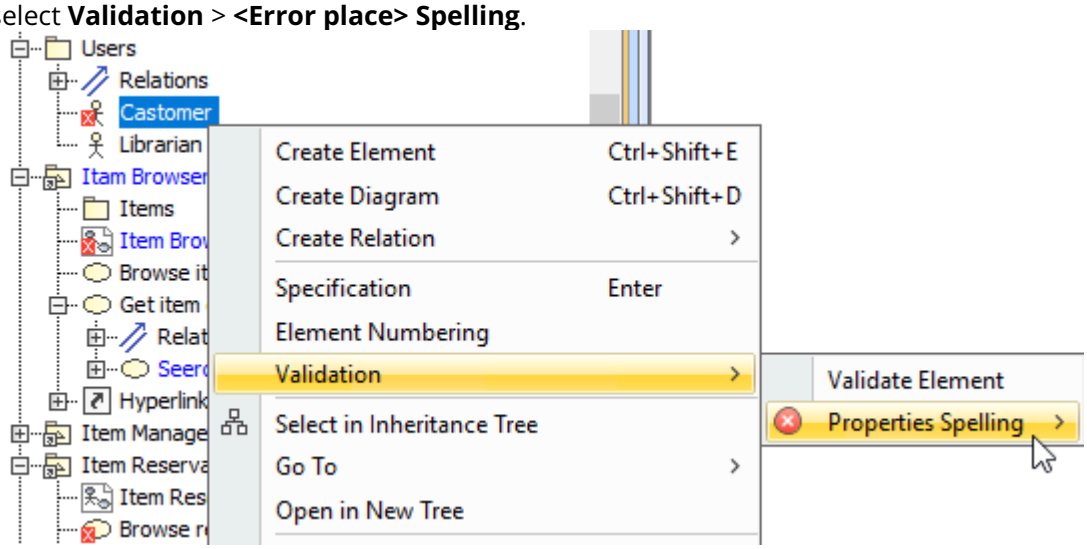
To correct an error

1. Do one of the following:
   - In the **Validation Results** panel, select the error and right-click it to open the shortcut menu.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

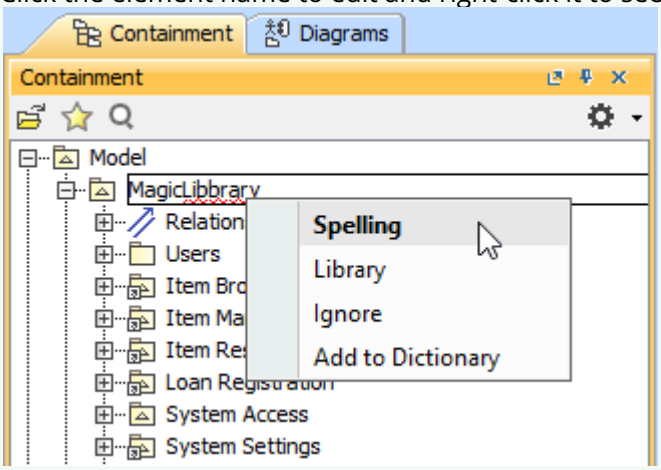


- In the Containment tree, select the element with an error (marked with a red circle and white cross inside) and right-click it to open the shortcut menu. In the shortcut menu, select **Validation** > **<Error place> Spelling**.



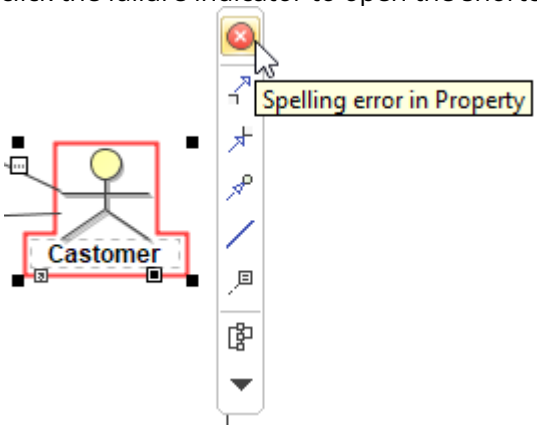
Correct the error

Click the element name to edit and right-click it to see a spelling solution.

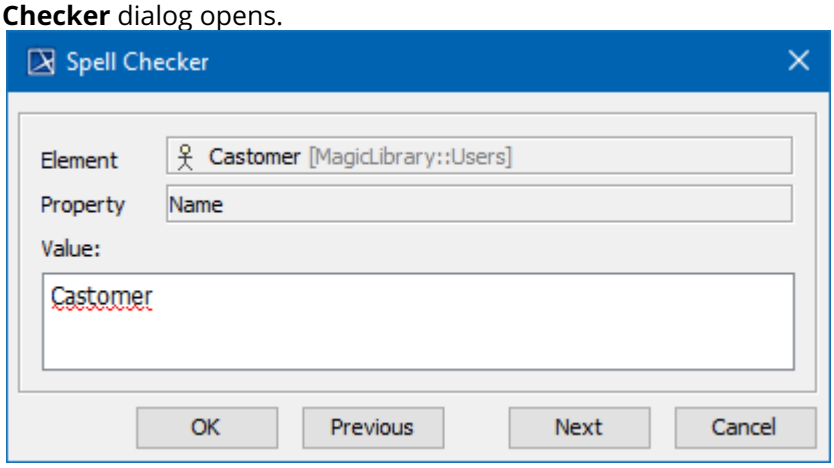


Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- On the diagram pane, select the element with an error (highlighted with a red border) and click the failure indicator to open the shortcut menu.



2. In the shortcut menu of an element with an error, select the **Correct** command. The **Spell Checker** dialog opens.



3. In the **Value** area, all incorrectly spelled words are underlined. Correct the words.
4. Click **Next** to go to the next spelling error (or **Previous** to return to the previous spelling error) found during the validation.
5. To close the dialog and save changes, click **OK**. To close the dialog without saving changes, click **Cancel**.

Related Pages:

- Spelling Checker(see page 1164)
  - Spell Checking as you type(see page 1164)
  - Spell checking for the entire project or a selected scope(see page 1167)
  - Setting the spell checking options(see page 1169)
  - Defining properties of the customized element to be spell checked(see page 1172)

Setting the spell checking options

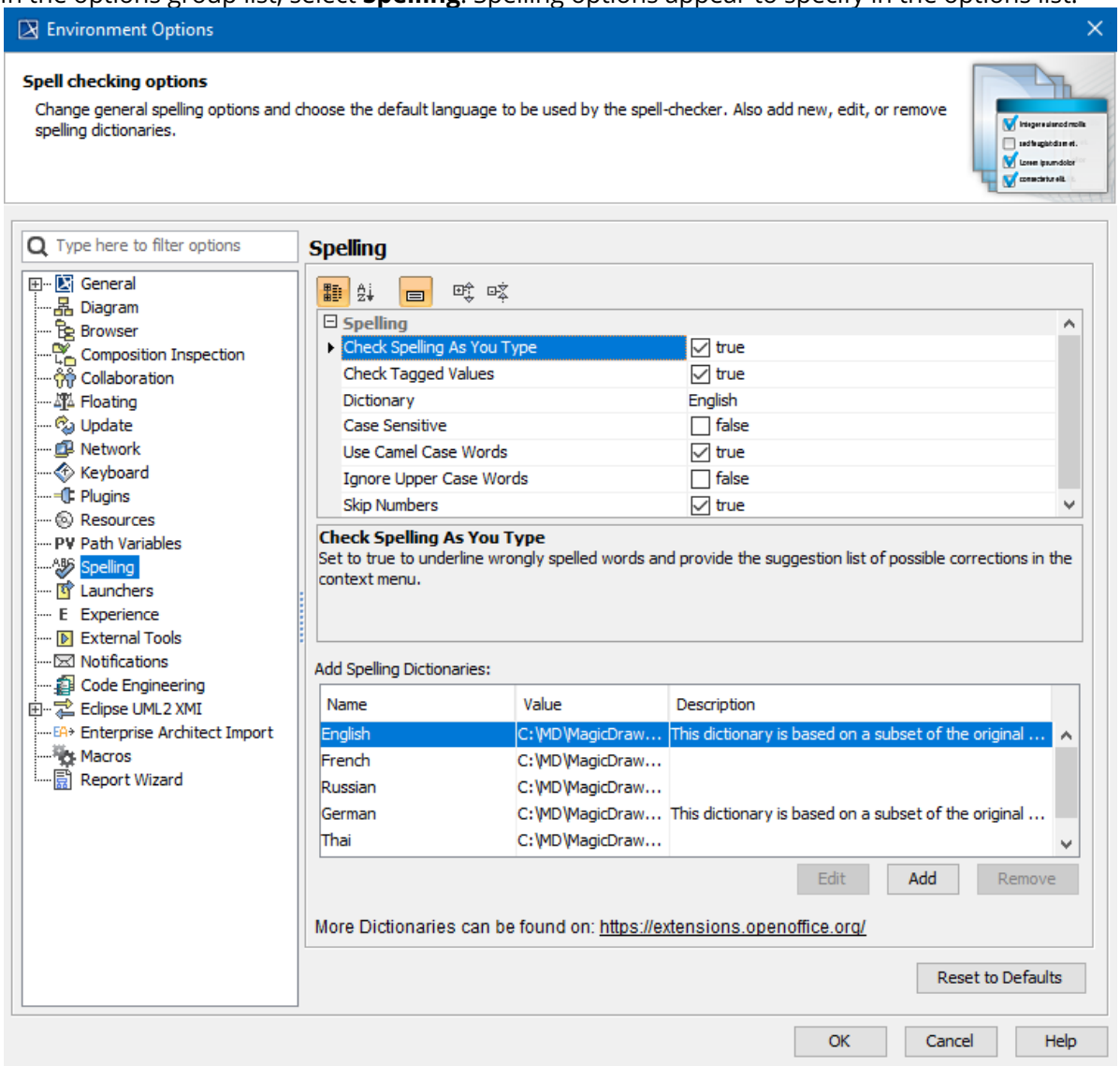
In the Environment Options(see page 65) dialog, you can change general spelling options and the default language to be checked in the modeling tool, as well as manage dictionaries to be used in the modeling tool.

To change spell checker options

1. On the main menu, click **Options** > **Environment**. The **Environment Options** dialog box opens.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

2. In the options group list, select **Spelling**. Spelling options appear to specify in the options list.



Spell checking options are described in the following table:

Property Name	Function
Check Spelling as you type	Underlines incorrectly spelled words and provides a list of suggested corrections in a shortcut menu. For more information about spell check while typing, see Spell Checking as you type(see page 1164).
Check Tagged Values	Checks all tagged values of a string type.

Property Name	Function
Dictionary	Select language for spelling. All "Open Office" supportive spelling languages can be added to the existing ones. For more information about spelling dictionaries, see Spell checking dictionaries(see page 1172).
Case Sensitive	If *true*, words will differ in meaning based on the differing use of uppercase and lowercase letters.
Use Camel Case Words	If *true*, compound words or phrases in which the words are joined without spaces and are capitalized within the compound, e.g. *BackColor* will be spelled as separate words.
Ignore Upper Case Words	If *true*, words with uppercase words only are not to be spell checked.
Skip Numbers	If *true*, numbers are not spell checked.

Spell checking dictionaries

All Apache OpenOffice supportive spelling languages are available.

To import spelling dictionaries

1. Click the **Add** button in the **Environment Options** dialog box > the **Spelling branch** > the **Add Spelling Dictionaries** group. The **Dictionary dialog** box opens.
2. Type the name of a new spelling dictionary in the **Name** text box.
3. Click the ... button and select the OpenOffice zip file location.
4. Type the description of the new spelling dictionary in the **Description** text box.

Figure 314 -- The Dictionary dialog box

Dictionary Link
More dictionaries can be found at http://wiki.services.openoffice.org/wiki/Dictionaries

Related Pages:

- Spelling Checker(see page 1164)
  - Spell Checking as you type(see page 1164)
  - Spell checking for the entire project or a selected scope(see page 1167)
  - Setting the spell checking options(see page 1169)
  - Defining properties of the customized element to be spell checked(see page 1172)

Defining properties of the customized element to be spell checked

You can check both DSL customization classes and their properties. You can choose which properties of the customized element(see page 2073) (Class (see page 666)with a «Customization» stereotype(see page 856)) to be spell checked.

You accomplish this by defining a customization class tag (see page 865)*checkSpelling* value, found in the properties tag group. By creating a value for this tag, you can choose String properties to check spelling for. By default, there are no properties marked as checkable.

Related pages

- Editing tagged value(see page 865)

- Spell Checking as you type(see page 1164)
- Spell checking for the entire project or a selected scope(see page 1167)
- Setting the spell checking options(see page 1169)
- Defining properties of the customized element to be spell checked(see page 1172)

Integration with external Evaluators

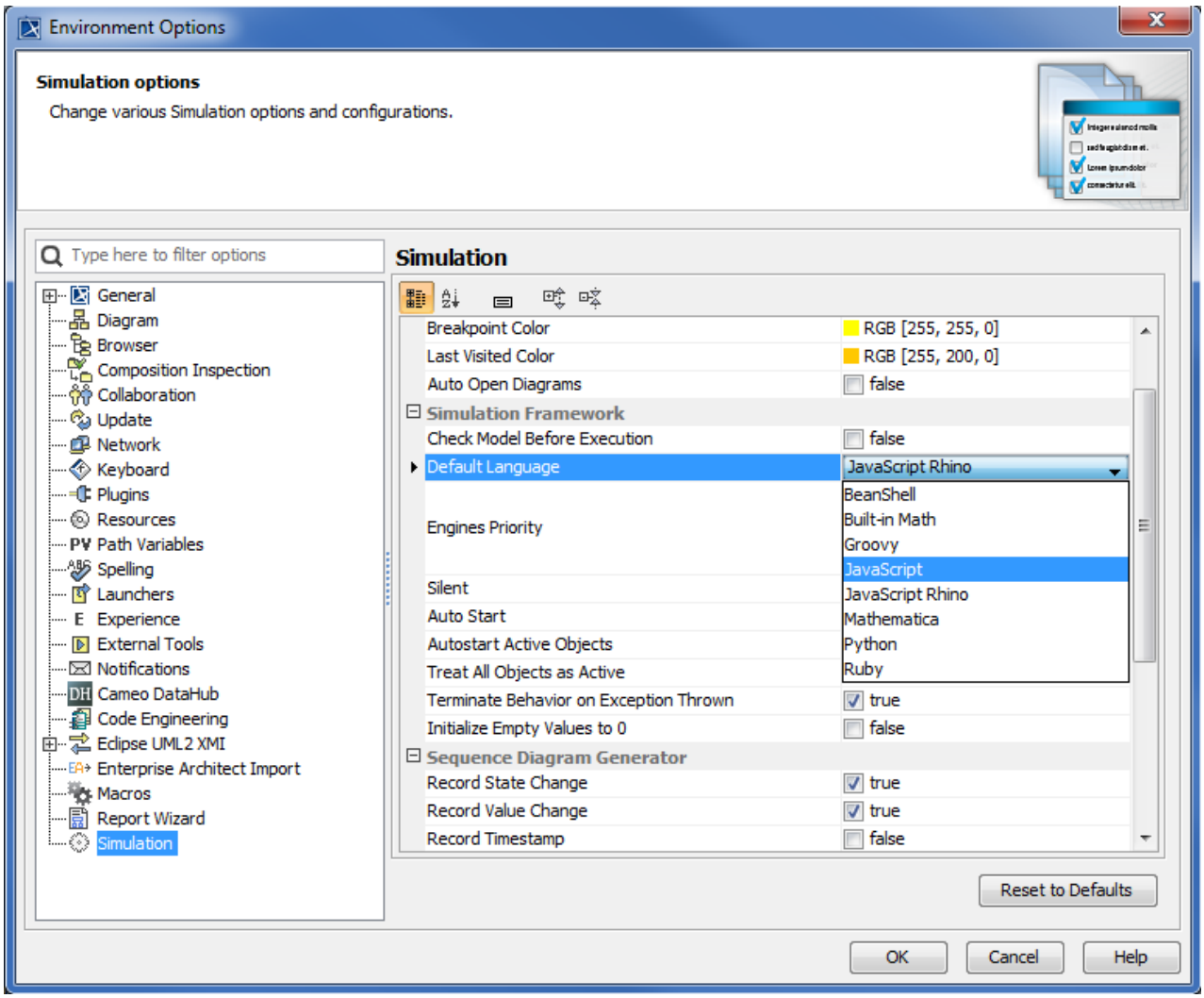
Modelling tool allows you to use an external Evaluator to evaluate an opaque expression in your model. Therefore, you can use any language supported by the external Evaluator in the body of the opaque expression.

- Integration with MATLAB[254] (**R2012a or later**)
- Integration with MapleTM[255] (**17 or later**)
- Integration with Mathematica[256] (**9 or later**)
- Integration with OpenModelica[257] (**1.9.6 or later**)

You must specify the name of the language in the opaque expression. If you do not specify the language of the opaque expression, will use the Default Language specified in **Simulation Framework** in the **Environment Options** dialog as follows.

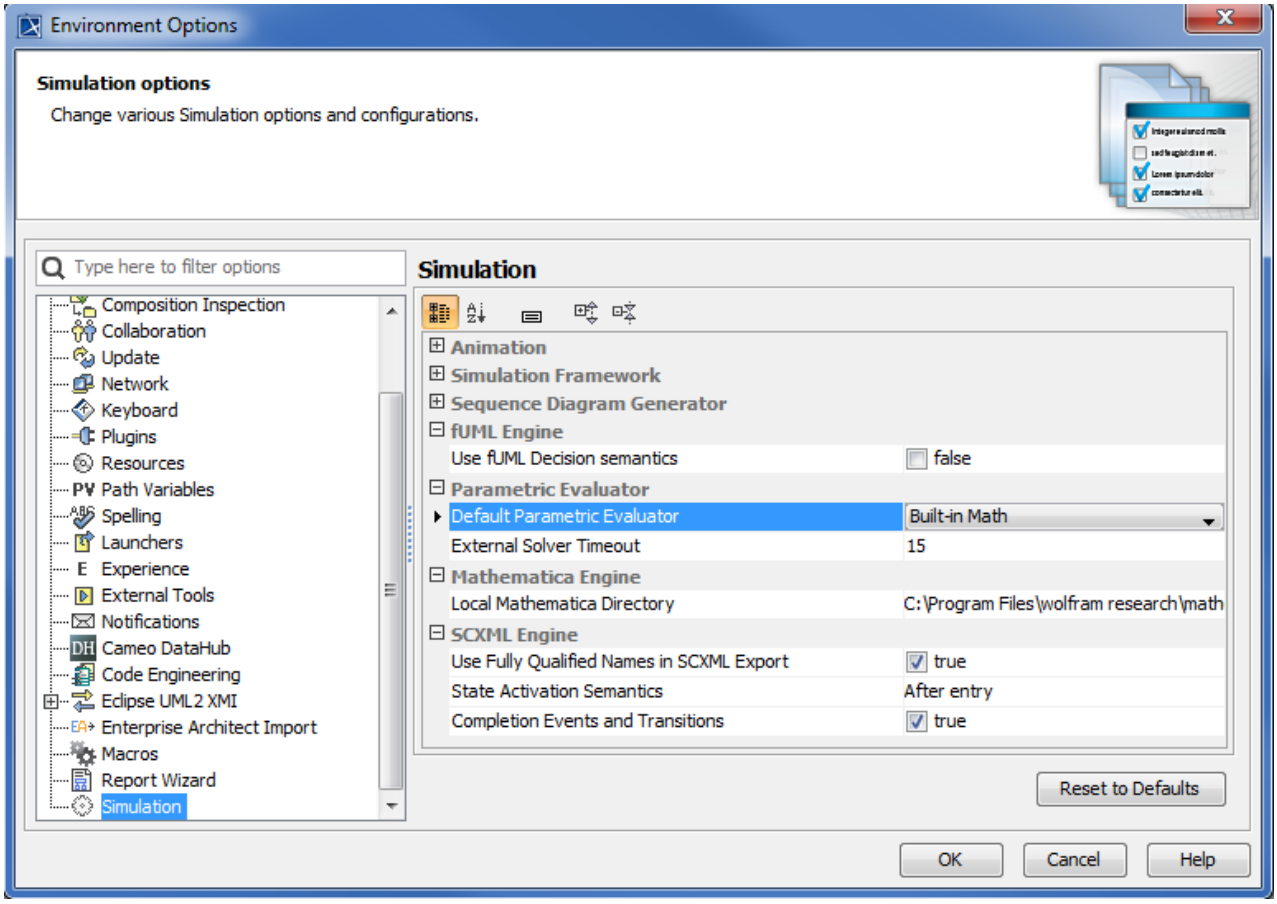
254 https://docs.nomagic.com/display/CST190SP4/Integration+with+MATLAB
255 https://docs.nomagic.com/display/CST190SP4/Integration+with+MapleTM
256 https://docs.nomagic.com/display/CST190SP4/Integration+with+Mathematica
257 https://docs.nomagic.com/display/CST190SP4/Integration+with+OpenModelica

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The Default Language property in the Environment Options dialog.

If the language of expressions of constraints of a SysML Constraint Block is not specified, modeling tool will use the Evaluator, specified in **Default Parametric Evaluator** in the **Environment Options** dialog, to solve the expressions as follows.



The Default Parametric Evaluator property in the Environment Options dialog.

Related pages

- Specifying the language for the expression[258]
- Value binding[259]
- Evaluating expressions[260]
- Evaluation with causality[261]
- Dynamic constraint[262]
- Manual value updates using the Parametric Evaluator[263]
- Communicating with evaluators through simulation console[264]
- Exchanging values between Cameo Simulation Toolkit and Parametric Evaluator[265]
- Built-in Math[266]

258 https://docs.nomagic.com/display/CST190SP4/Specifying+the+language+for+the+expression
259 https://docs.nomagic.com/display/CST190SP4/Value+binding
260 https://docs.nomagic.com/display/CST190SP4/Evaluating+expressions
261 https://docs.nomagic.com/display/CST190SP4/Evaluation+with+causality
262 https://docs.nomagic.com/display/CST190SP4/Dynamic+constraint
263 https://docs.nomagic.com/display/CST190SP4/Manual+value+updates+using+the+Parametric+Evaluator
264 https://docs.nomagic.com/display/CST190SP4/Communicating+with+evaluators+through+simulation+console
265https://docs.nomagic.com/display/CST190SP4/
Exchanging+values+between+Cameo+Simulation+Toolkit+and+the+Parametric+Evaluator
266 https://docs.nomagic.com/display/CST190SP4/Built-in+Math

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Trade study with Cameo Simulation Toolkit[267]
- Sample project[268]

Integration with MATLAB

You can use MATLAB® to evaluate expressions written in MATLAB syntax in Cameo Simulation Toolkit. You must install MATLAB® first and set up your system to call and use it in Cameo Simulation Toolkit.

> **Note**
> For Mac users, Cameo Simulation Toolkit version **18.5 SP2 and later** can integrate with MATLAB® **2016b** seamlessly without disabling System Integrity Protection (SIP) unless it is necessary to disable SIP as a new feature of the latest OS X El Capitan for successful integration.

> **Warning**
> MATLAB® 2014a on Mac OS is not capable of evaluating and returning values via Simulation Console, which causes MagicDraw to freeze. To solve this problem, please consider upgrading MATLAB® 2014a to 2014b or later.

To disable OS X El Capitan's System Integrity Protection (SIP)

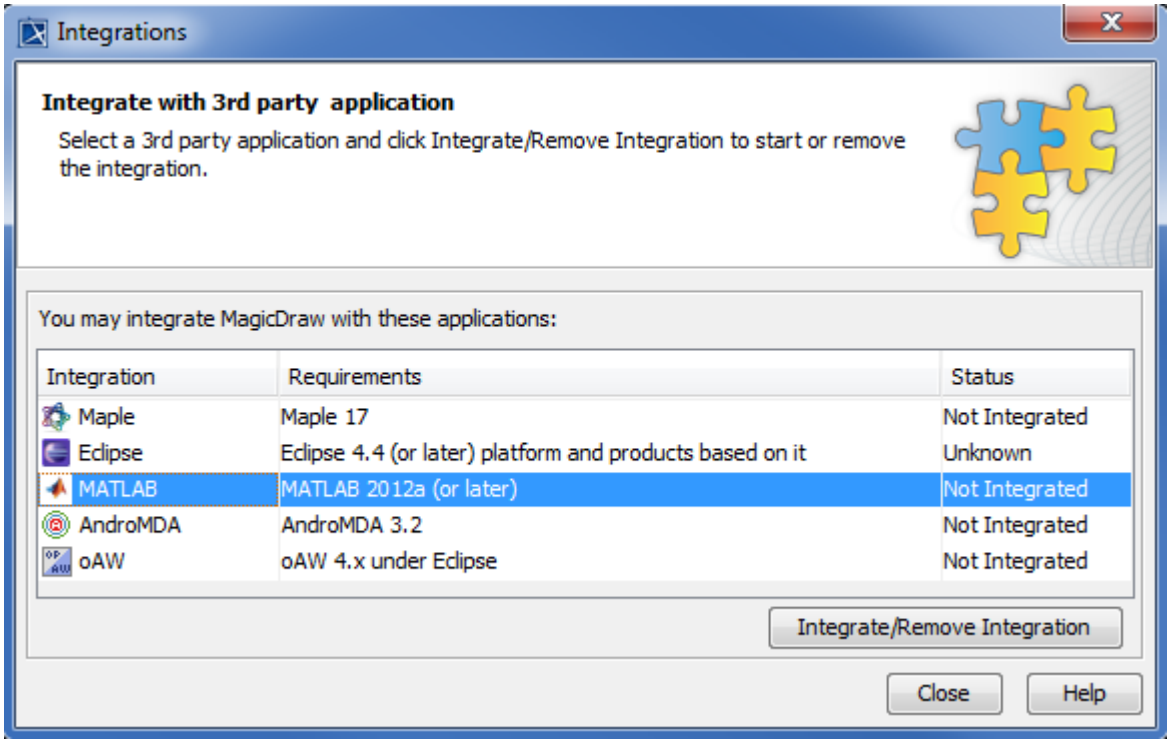
1. Restart your Mac.
2. As soon as the screen turns black, hold down the **command**+**R** keys to access the Recovery Partition.
3. Release the keys once the Apple logo appears on your screen.
4. Click the **Utilities** menu and select **Terminal**.
5. Type `csrutil disable` into the Terminal window and press **Return**.
6. The Terminal should display a message that SIP was disabled.
7. Restart your Mac for the changes to take effect.
8. Run MagicDraw/Cameo System Modeler and integrate it with MATLAB®.

To integrate MagicDraw or Cameo Systems Modeler with MATLAB® (on Microsoft Windows or Mac OS X)

1. From the main menu, click **Tools** > **Integrations**. The **Integrations** dialog opens.

267 https://docs.nomagic.com/display/CST190SP4/Trade+study+with+Cameo+Simulation+Toolkit
268 https://docs.nomagic.com/display/CST190SP4/Sample+project

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



2. Select **MATLAB®** and click **Integrate/Remove Integration**. The MATLAB® directory selection dialog opens.
3. Specify the directory where you installed MATLAB® and click **OK**.

Note

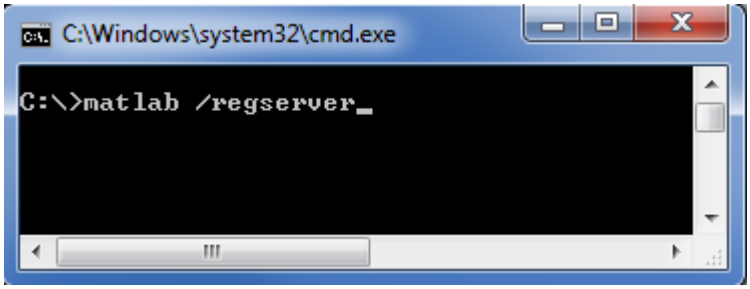
- If there are problems with integrating MATLAB® on Windows, run MagicDraw as an Administrator, and then try to integrate again.
- When integrating with MATLAB® for the first time or changing the MATLAB® version, restart your machine.
- If the system has been integrated with MATLAB® previously, you do not have to restart the system since system variables have already been updated. Only restart MagicDraw.

You can also integrate MagicDraw or Cameo Systems Modeler with MATLAB® manually using the following steps.

To use MATLAB® on a 32-bit or a 64-bit version of Microsoft Windows

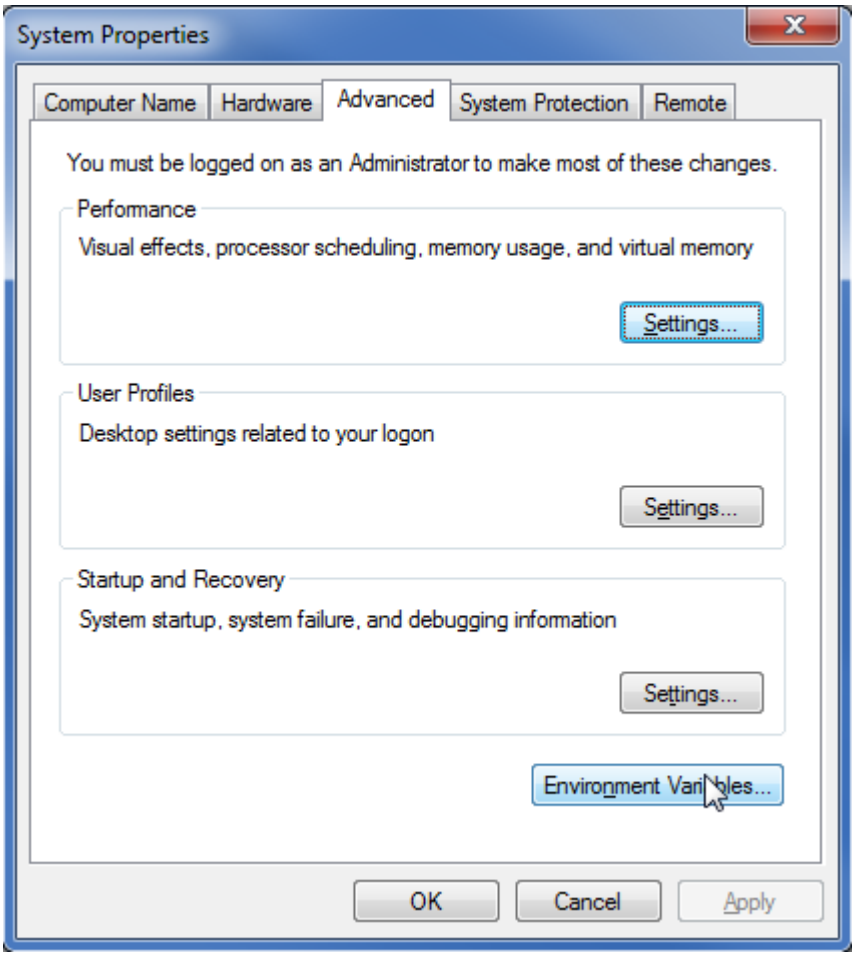
1. Install MATLAB®.
2. Press **Windows** + **R** to open the **Run** dialog.
3. Type *cmd* in the open combo box and click **OK** to open the command prompt window.
4. Type "matlab /regserver" and press **Enter** to register the MATLAB® components to Windows. The MATLAB® command prompt opens and is ready to use.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

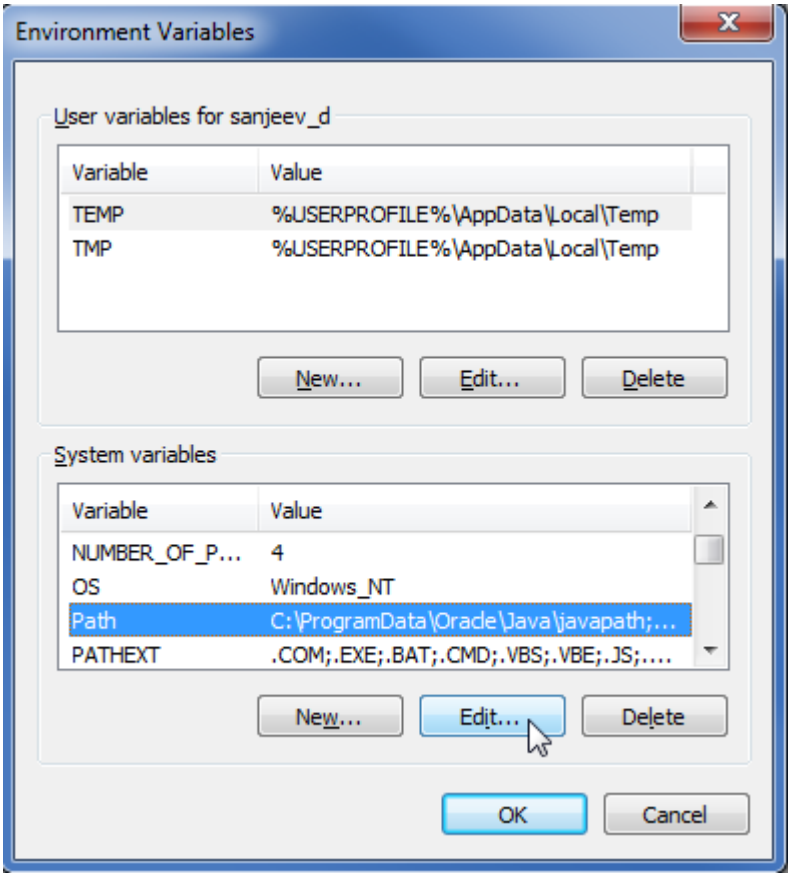


5. Add the path of the MATLAB® bin and bin/win32 (or bin/win64 for Microsoft Windows 64-bit) folders to the **Path** environment variable using the following steps

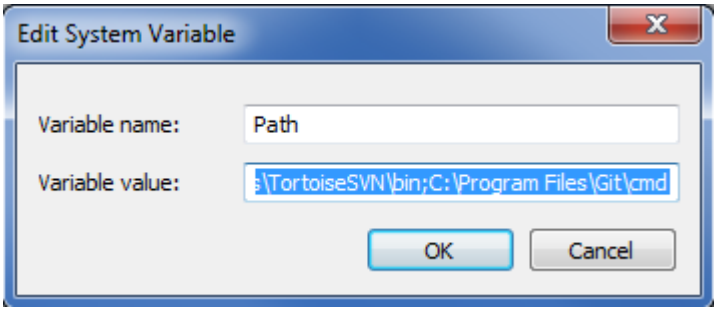
   5.1 Double-click **System** in **Control Panel** to open the **System Properties** dialog. Click the **Advanced** tab.



   5.2 Click **Environment Variables**. The **Environment Variables** dialog opens.



5.3 From the **System variables** list, select **Path** and click **Edit**. The **Edit System Variable** dialog opens.



5.4 In the **Variable value** box, enter the path to the MATLAB® bin and bin/win32 folders (or bin/win64 for Microsoft Windows 64-bit version), for example, *;C:\Program Files\MATLAB\R2010b\bin;C:\Program Files\MATLAB\R2010b\bin\win32;*.

5.5 In the **Variable value** box, enter the path to the MATLAB® runtime/win32 (or runtime/win64 for Microsoft Windows 64-bit version), for example, *;C:\Program Files\MATLAB\R2010b\runtime\win32;*.

Note

You can skip Step 5.5 in the above procedure if the MATLAB® runtime directory does not exist, since this directory only existed in earlier versions of MATLAB® and not in the newer versions. The MATLAB® runtime is not required in these cases.

5.6 Click **OK**.

6. Restart Windows.

To use MATLAB® on Mac OS 10.6 (Snow Leopard)

---

1. Install MATLAB®.
2. Type the following commands in the terminal to show all files in Finder
   - $ defaults write com.apple.finder AppleShowAllFiles TRUE
   - $ killall Finder

3. Add the **DYLD_LIBRARY_PATH** variable to Mac OS:
   3.1 Create an empty text file in the */etc* folder and name it: *launchd.conf*.
   3.2 Open it with a text editor, for example, TextEdit, and type the following text (no space)

   ```
   setenv DYLD_LIBRARY_PATH /Applications/MATLAB_R2010b.app/bin/maci64:
   /Applications/MATLAB_R2010b.app/runtime/maci64
   ```

   3.3 Save the text file as *launchd.conf* to the desktop.
   3.4 Move the *launchd.conf* file to the */etc* folder.

4. Create a link to the MATLAB® executable file in /usr/bin if it does not yet exist.
5. Call the following commands in the terminal
   - $ cd /usr/bin
   - $ ln -s /Applications/MATLAB_R2010b.app/bin/matlab matlab

6. Type the following commands in the terminal to reset Finder
   - $ defaults write com.apple.finder AppleShowAllFiles FALSE
   - $ killall Finder

7. Restart Mac OS.

Note

You can also use MATLAB® if you are using MagicDraw 18.0 on either Mac OS X 10.10 Yosemite or Mac OS X 10.11 El Capitan, by using the following steps.

1. Disable the SIP (if your Mac is OS X El Capitan) by following the instruction to disable OS X El Capitan's System Integrity Protection (SIP)[269].
2. Install MATLAB®.
3. Create the file *com.nomagic.magicdraw.simulation.mathengine.plist*.
   3.1 Create a text file and type the following text.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC"-//Apple//DTD PLIST 1.0//EN" "http://
www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
<key>Label</key>
<string>com.nomagic.magicdraw.simulation.mathengine.plist</string>
<key>ProgramArguments</key>
<array>
<string>sh</string>
<string>-c</string>
<string>
launchctl setenv DYLD_LIBRARY_PATH
/Applications/MATLAB_R2012a.app/bin/maci64:/Applications/
MATLAB_R2012a.app/runtime/maci64
</string></array><key>RunAtLoad</key><true/></dict></plist>
```

   3.2 Change the */Applications/MATLAB_R2012a.app* to your MATLAB directory.
   3.3 Save the text file. (If you are using TextEdit, change the file to plain text by clicking **Format** > **Make Plain Text**).
   3.4 Rename it as *com.nomagic.magicdraw.simulation.mathengine.plist*.

   (Note: If you already have the file *com.nomagic.magicdraw.simulation.mathengine.plist* in */Library/LaunchAgents*, add *:/Applications/MATLAB_R2012a.app/bin/maci64:/Applications/MATLAB_R2012a.app/runtime/maci64* to your *DYLD_LIBRARY_PATH* in your *com.nomagic.magicdraw.simulation.mathengine.plist*. For example, *launchctl setenv DYLD_LIBRARY_PATH <Other_Path>:/Applications/MATLAB_R2012a.app/bin/maci64:/Applications/MATLAB_R2012a.app/runtime/maci64*.)

4. Create the file *com.nomagic.magicdraw.simulation.mathengine.matlab.plist*.
   4.1 Create a text file and type the following text.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://
www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
<key>Label</key>
<string>com.nomagic.magicdraw.simulation.mathengine.matlab.plist</
string>
<key>ProgramArguments</key>
<array>
```

269 https://docs.nomagic.com/display/CST190SP4/Integration+with+MATLAB

```
<string>sh</string>
<string>-c</string>
<string>
launchctl setenv MD_MATLAB_MATHENGINE
/Applications/MATLAB_R2012a.app/bin/maci64:/Applications/
MATLAB_R2012a.app/runtime/maci64
</string></array><key>RunAtLoad</key><true/></dict></plist>
```

4.2 Change "*/Applications/MATLAB_R2012a.app*" to your MATLAB directory.
4.3 Save the text file. (If you are using TextEdit, change the file to plain text by clicking **Format** > **Make Plain Text**).
4.4 Rename it as *com.nomagic.magicdraw.simulation.mathengine.matlab.plist*.

5. Move the files to */Library/LaunchAgents/* by using the Terminal.
   5.1 Run the Terminal.
   5.2 Go to the plist files directory.
   - $ cd [your plist file directory]

5.3 Move the plist file to */Library/LaunchAgents/* using the following command
- $ sudo mv com.nomagic.magicdraw.simulation.mathengine.plist /Library/LaunchAgents/
- $ sudo mv com.nomagic.magicdraw.simulation.mathengine.matlab.plist /Library/LaunchAgents/

6. Create a link to the MATLAB® executable file in */usr/bin* if it does not exist, by using the following command in the Terminal
- $ cd /usr/bin
- $ sudo ln -s /Applications/MATLAB_R2012a.app/bin/matlab matlab

> **Note**
> You need to change the */Applications/MATLAB_R2012a.app* in the command line to your MATLAB directory.

7. Restart Mac OS.

To use MATLAB® on 32-bit and 64-bit (tested with Ubuntu) versions of Linux

---

1. Install MATLAB® (Assume that your MATLAB installation directory is */home/username/MATHWORKS_R2011A*).
2. Make sure that C Shell has already been installed on your Linux. To install C Shell on Ubuntu, type the following command in the terminal
   - ~$ sudo apt-get install csh

3. Create a link to the MATLAB® executable file in /usr/bin if it does not exist, and type the following commands in the terminal
   - ~$ sudo -i
   - ~$ cd /usr/bin
   - ~$ ln -s /home/username/MATHWORKS_R2011A/bin/matlab matlab

4. Use a text editor to open the **magicdraw** file in the **bin** folder in the MagicDraw installation directory. Type the following text under the line that contains **cd"$APP_HOME"** to add the MATLAB® **bin** folder to **LD_LIBRARY_PATH** of Java and save the **magicdraw** file.

- on Linux 32-bit, type: *export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/username/MATHWORKS_R2011A/bin/glnx86:/home/username/MATHWORKS_R2011A/sys/os/glnx86*
- on Linux 64-bit, type: *export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/username/MATHWORKS_R2011A/bin/glnxa64:/home/username/MATHWORKS_R2011A/sys/os/glnxa64*

```
                JAVA_LAUNCHER=`which java`
                JAVA_BIN_DIR=`absdirname "$JAVA_LAUNCHER"`
                JAVA_HOME=`absdirname "$JAVA_BIN_DIR"`
        fi
fi

JAVA_LAUNCHER="$JAVA_HOME"/bin/java

if [ -x "$JAVA_LAUNCHER" ]; then
        if [ -n "$MAIN_CLASS" ]; then
                MAIN_CLASS=`echo -DMAIN_CLASS=$MAIN_CLASS`
        fi
        BOOT_CLASSPATH=`echo -Xbootclasspath/p:$BOOT_CLASSPATH`
        cd "$APP_HOME"

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/username/MATHWORKS_R2011A/bin/glnxa64:/
home/username/MATHWORKS_R2011A/sys/os/glnxa64
#APP LAUNCHER
"$JAVA_LAUNCHER" "${JAVA_ARGS[@]}" "${MAC_ARGS[@]}" "$BOOT_CLASSPATH" "-
Dlauncher.properties.file=$PROP_FILE" "$MAIN_CLASS" "-cp" "$CLASSPATH"
com.nomagic.launcher.Launcher "${APP_ARGS[@]}" "$@"

else
        echo "ERROR!"
        echo "Java executable not found in" $JAVA_HOME/bin/java
        echo "Please add java to your PATH environment variable, or specify it in
file" $PROP_FILE
fi
```

LD_LIBRARY_PATH added to the MagicDraw file.

> **Information**
> If there are problems with integrating MATLAB® on Windows, run MagicDraw as an Administrator, and then try to integrate again.

Related page

- Integration with external Evaluators(see page 1173)

Integration with Maple

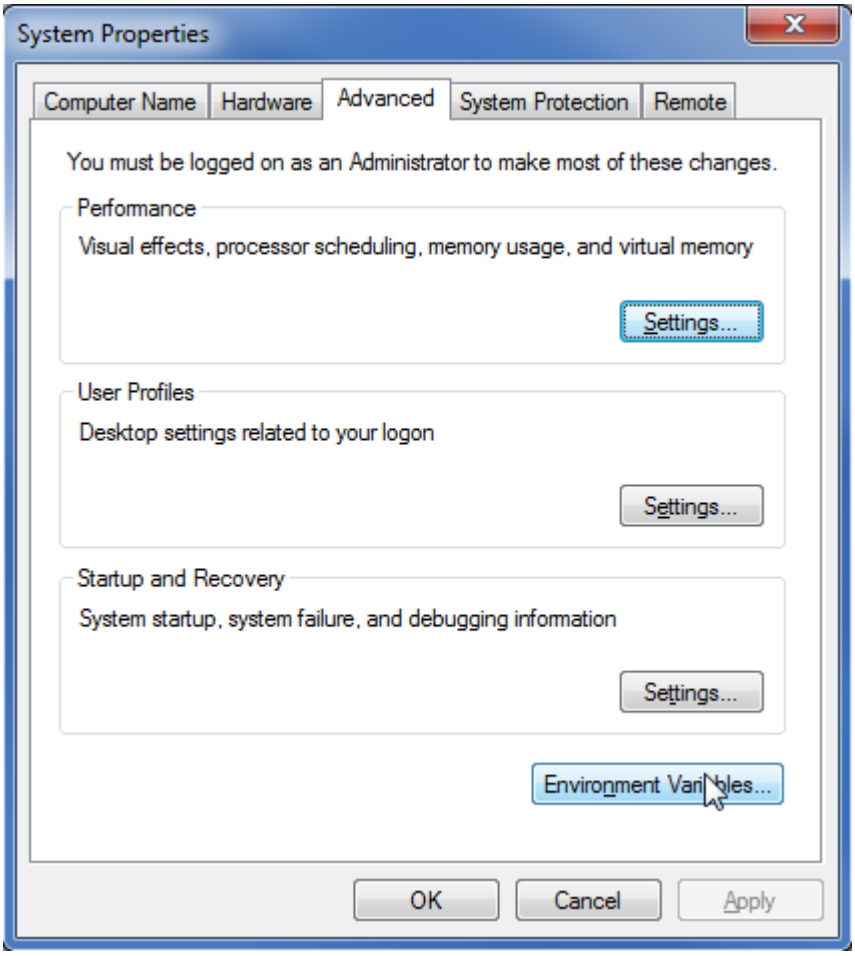
Cameo Simulation Toolkit supports Maple™, a mathematical computation engine, to analyze and solve mathematical expressions. Once you have installed Maple™, you can specify it as the language of opaque expressions.

First, you must install Maple™ on your local machine and set up your system in order to call Maple™ and use it in Cameo Simulation Toolkit.

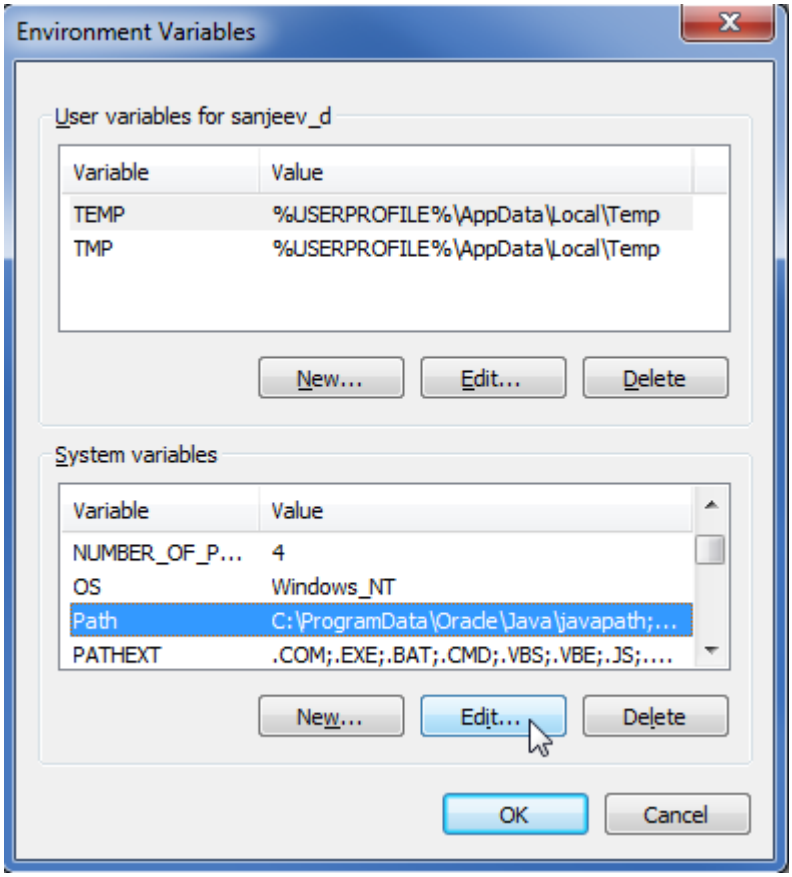
To use Maple™ on a 32-bit or a 64-bit version of Microsoft Windows

---

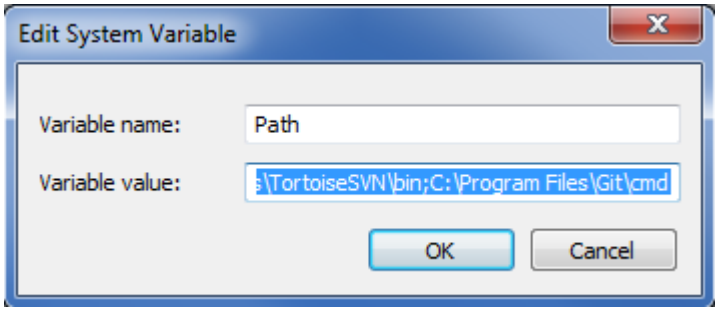
1. Install Maple™.
2. Add the path of the Maple™ bin folder to the path environment variable using the steps as follows
   2.1 Double-click **System** in the Control Panel to open the **System Properties** dialog.
   2.2 Click the **Advanced** tab.



   2.3 Click the **Environment Variables** button to open the **Environment Variables** dialog.



2.4 Select **Path** from the **System** variables group and click the **Edit** button to open the **Edit System Variable** dialog.
2.5 Enter the path to the Maple™ bin folder in the **Variable value** box. There are two methods to enter the path:if the **Variable value** box has a default value, you need to add the following path at the end of the text **;***C:\Program Files\Maple 17\bin.X86_64_WINDOWS* or (ii) if the **Variable value** box is empty, you can just add the following path *C:\Program Files\Maple 17\bin.X86_64_WINDOWS*.



2.6 Click **OK**.

3. Restart Windows.

To use Maple™ on Mac 10.7 or Linux

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

1. Install Maple™.
2. Open Maple™ and follow these steps:
   2.1 Type: kernelopts(mapledir) into command input to find the maple dir.
   2.2 Type: kernelopts(bindir) into command input to find the bin dir.

3. Open the /etc/launch.conf (if this file does not exist, create launch.conf with any text editor) and follow these steps:
   3.1 Type setenv DYLD_LIBRARY_PATH <BINDIR> where <BINDIR> is the bin dir from step 2.2
   3.2 Type setenv MAPLE <MAPLEDIR> where <MAPLEDIR> is the maple dir from step 2.1
   3.3 Save launch.conf.

4. Restart the computer.

Footnote

Maple™ is a registered trademark of Waterloo Maple, Inc.

Integration with Mathematica

Cameo Simulation Toolkit supports Mathematica®, a Mathematica®l computation engine, to analyze and solve Mathematica®l expressions. Once you have installed Mathematica® , you can specify it as the language of opaque expressions.

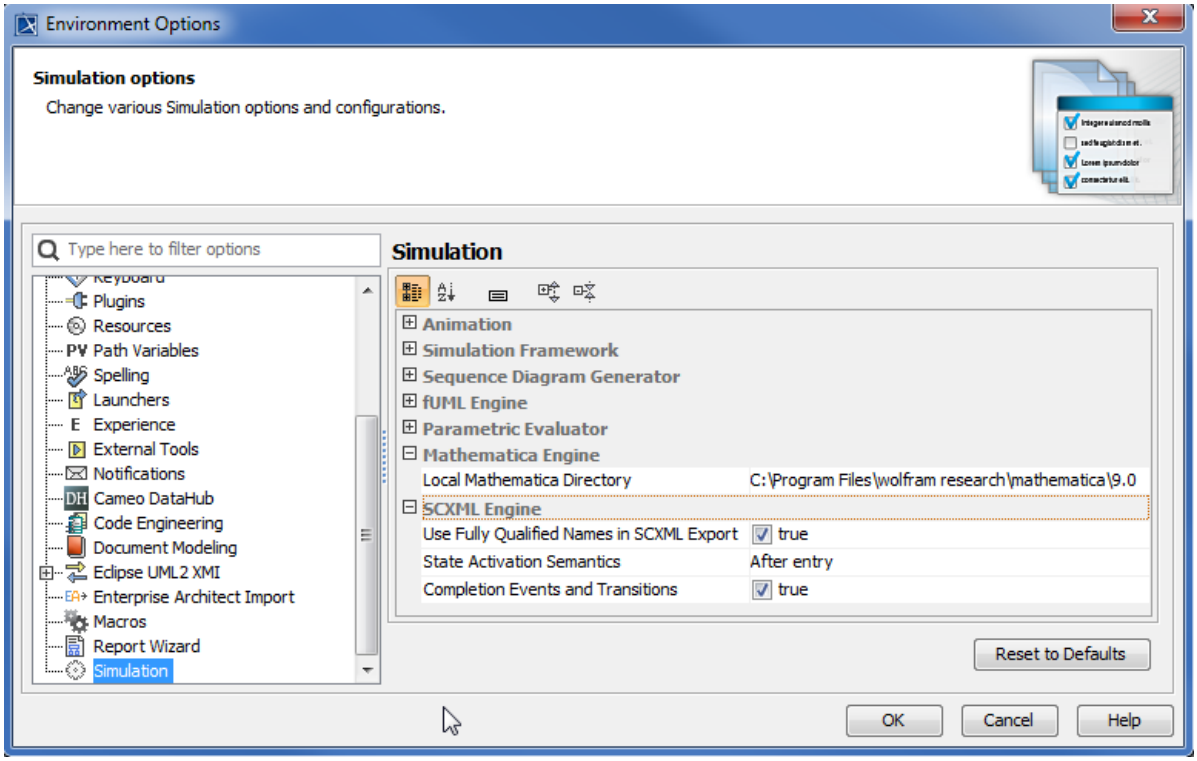
First, you must install Mathematica® on your local machine, and then set up your system to allow Cameo Simulation Toolkit to use the installed Mathematica®.

To use Mathematica® on a 32-bit or a 64-bit version of Microsoft Windows and Linux

---

1. Select **Options** > **Environment** on the main menu. The **Environment Options** dialog opens.
2. Select **Simulation** on the left-hand side pane.
3. Set the **Local Mathematica® Directory** field to the location of the directory to which your Mathematica®® was installed.
4. Restart MagicDraw.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



To use Mathematica® on Mac OSX

---

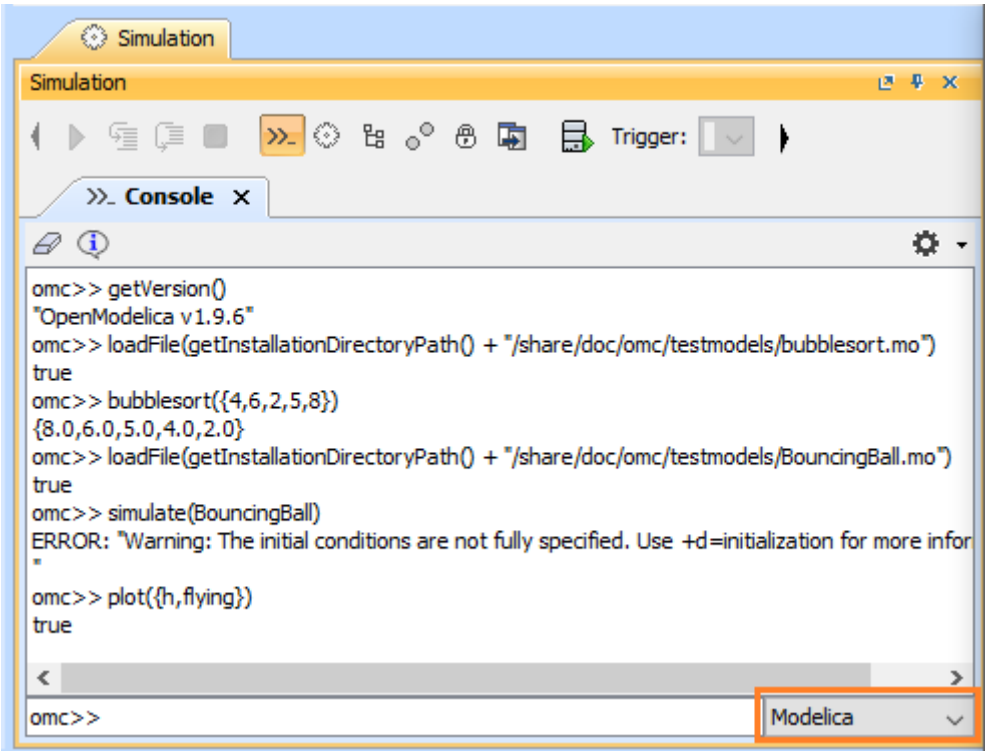
1. Install Mathematica® and Cameo Simulation Toolkit.
2. Restart MagicDraw twice. It is now ready for you to use.

Integration with OpenModelica

Cameo Simulation Toolkit supports OpenModelica, an open-source Modelica-based modeling and simulation environment for industrial and academic use, to analyze and solve mathematical expressions. Integrating Cameo Simulation Toolkit with OpenModelica allows you to specify Modelica as the language of opaque expressions. Download OpenModelica from https://www.openmodelica.org/, install it, and set up your system to allow Cameo Simulation Toolkit to integrate with it.

After successful integration, **Modelica** will be listed where a language selection is available (e.g. **Simulation Console** pane, **Constraint Block**, and **Environment Options** > **Simulation)**. Cameo Simulation Toolkit will be able to process a Modelica command from the **Simulation Console** pane (e.g. plot function).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



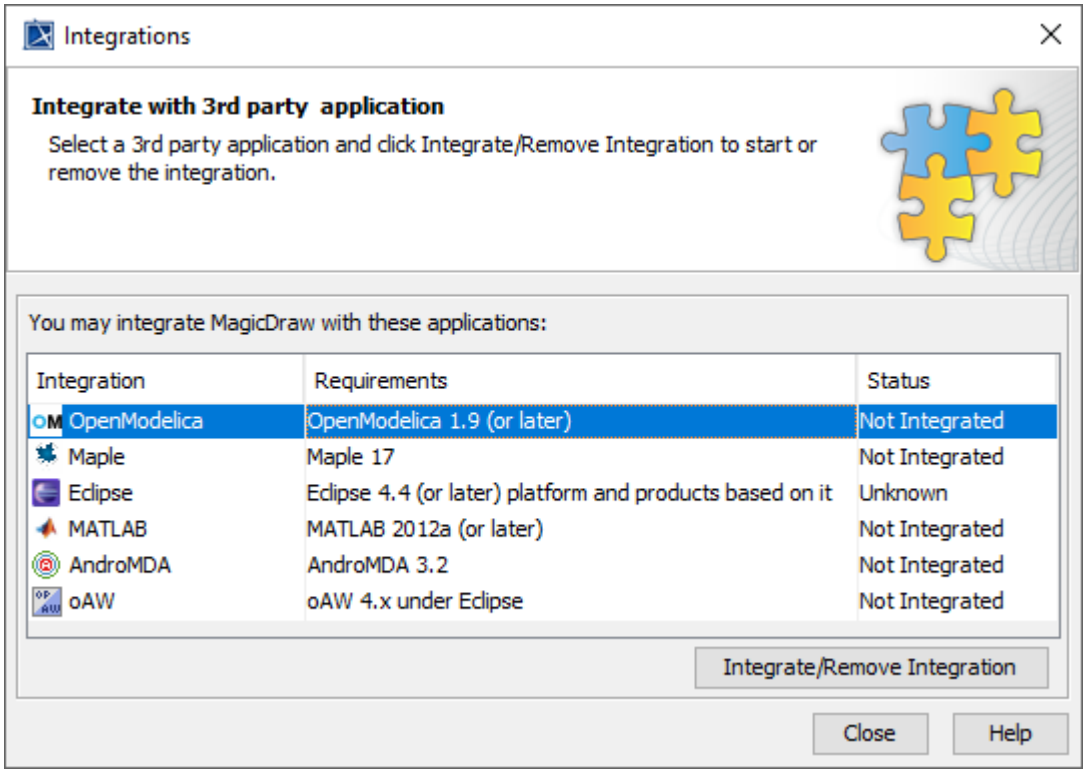
Selecting Modelica in the Simulation Console pane as the language of opaque expressions in Cameo Simulation Toolkit.

To integrate with OpenModelica

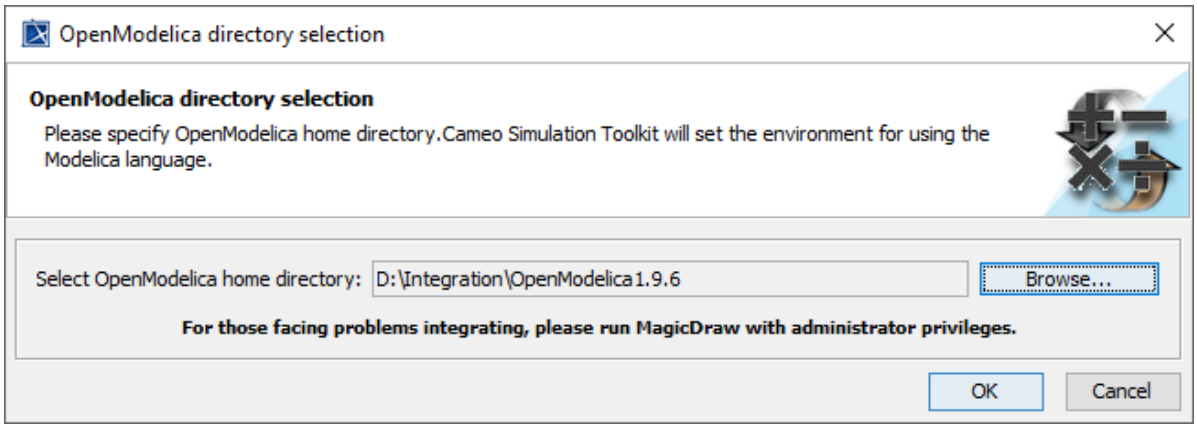
---

1. On the MagicDraw main menu, select **Tools** > **Integrations**. The **Integrations** window opens.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



2. Select **OpenModelica** and click Integrate/Remove Integration.
3. Select a home directory for OpenModelica from the **OpenModelica directory selection** dialog.



4. Click OK and restart the OS.

Project Statistics

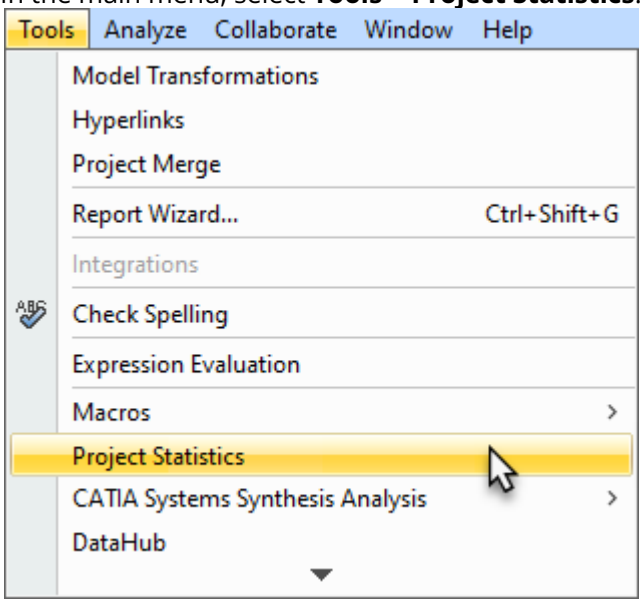
Detailed project statistics

The project statistics feature calculates detailed statistics of the whole project and exports results to a separate file. The exported statistical report will help us to solve performance problems you may be experiencing. Project statistics is calculated both in the main and used projects.

Confidential Information
The exported file contains the names of the main project, used projects, and used elements. If they are confidential, you should select **Export Detailed Statistics (Confidential)** to keep their names hidden.

To export detailed statistics

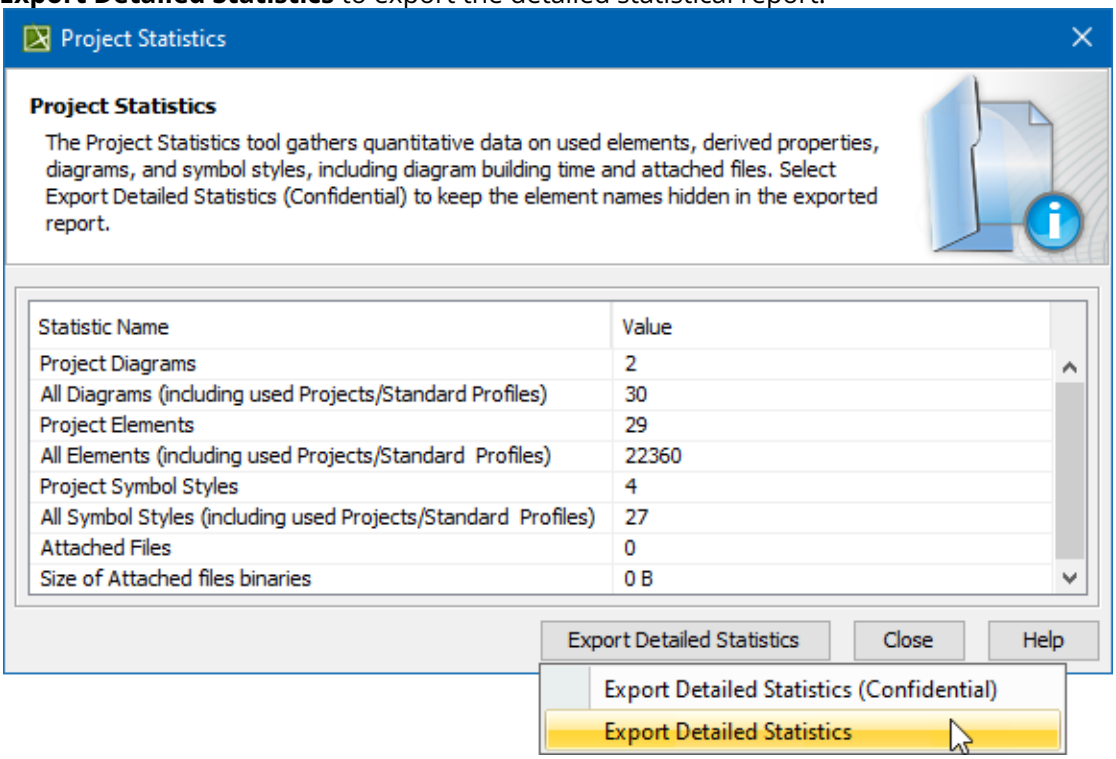
1. In the main menu, select **Tools** > **Project Statistics**.



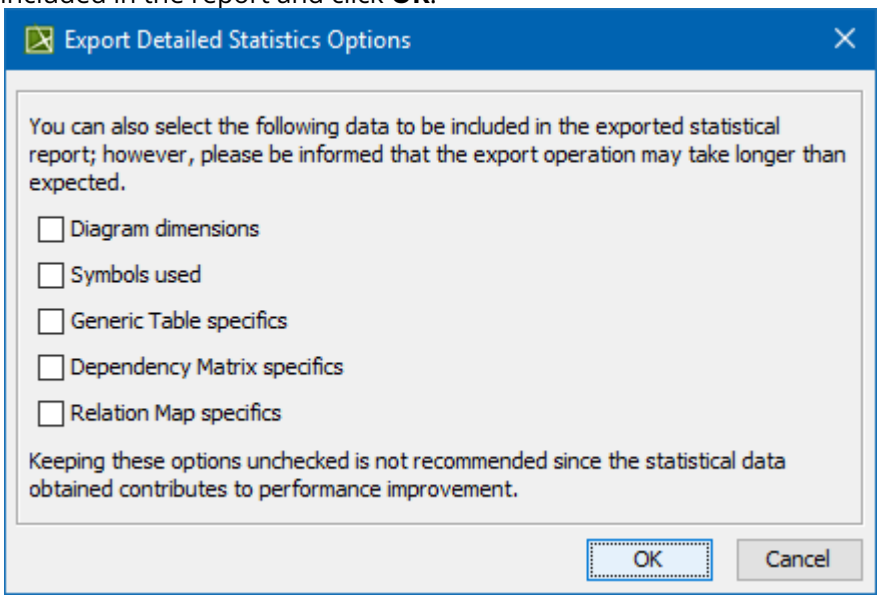
2. In the **Project Statistics** dialog, click the **Export Detailed Statistics** button and choose one of the following:
   - **Export Detailed Statistics (Confidential)** to export the statistical report with hidden element and project names.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- **Export Detailed Statistics** to export the detailed statistical report.



3. In the **Export Detailed Statistics Options** dialog, select the types of data to be additionally included in the report and click **OK**.



4. When the statistics file is generated, click **Close**.

The generated detailed project statistics file includes the following quantitative data:

Diagrams

- Diagram dimensions
- Diagrams count by type

Elements

- Elements count by UML type
- Elements count by DSL type
- Derived properties count
- Elements used for CDH mapping count
- Elements used for transformation mapping count
- Longest flat list count (longest flat list of elements under one element's root, e.g., under a package or a classifier)
- Smart packages count
- Number of stereotype applications by stereotype name and stereotype tagged values by defining feature name and tagged value size
- Number of strings in the model by size and metaProperty
- Number of multi-references by size and metaProperty
- Number of profiling elements (stereotype instance, slots, and value specifications)

Project

- Active/passive validation options
- Number of active validation rules by implementation language
- Number of code engineering options by language
- Symbol styles count
- Symbol styles properties count

Hardware & system

- Hardware (RAM, OS, disks size and type) and Java info (heap)
- Restricted dir size
- Local cache size on disk

Teamwork Cloud projects

- TWC project local cache size on disk
- TWC project local cache size in a binary blob
- TWC project history depth
- Locked elements count

Performance

- Time to build diagrams

Model analysis

The links below describe Magic Draw tool features for performing analysis of your model and displaying analysis views and results.

Analyzing usages and dependencies

On this page

- Used By command(see page 1193)
- Depends On command(see page 1193)
- Searching for usages and dependencies(see page 1194)

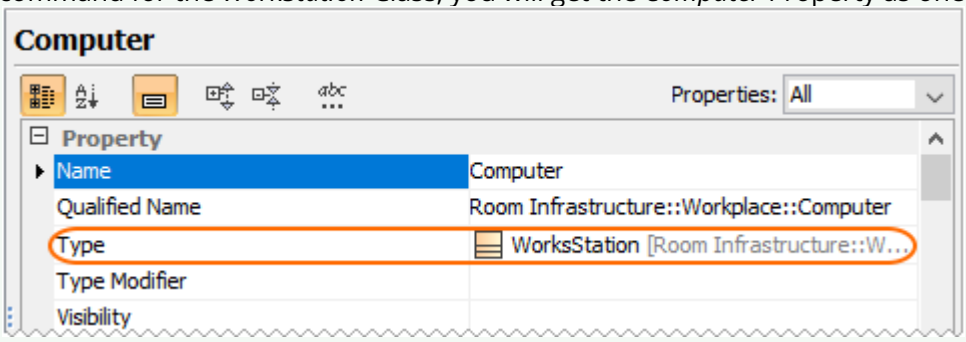
No Magic modeling tools provide a way to track and analyze element usages and dependencies in models, understand the relationships between used and dependent elements, and find the diagrams where these elements are represented. You can do all of this using the commands **Used By** and **Depends On**, as described below.

Used By command

The **Used By** command allows you to find all the elements that use a selected element (e.g., as property, tag etc.). This also includes usage in diagrams.

Example

In the figure below, the *Computer* Property has its type set to the *WorkStation* Class. In other words, the *Computer* Property uses the *WorkStation* Class as its type. If you execute the **Used By** command for the *WorkStation* Class, you will get the *Computer* Property as one of the results.



The Specification window showing that the *WorkStation* Class is used by the *Computer* Property.

Note that containing (owning) other elements is not considered usage. For example, when a Package contains a Class, this does not mean that the Package uses the Class. The Package is only a container of the Class.

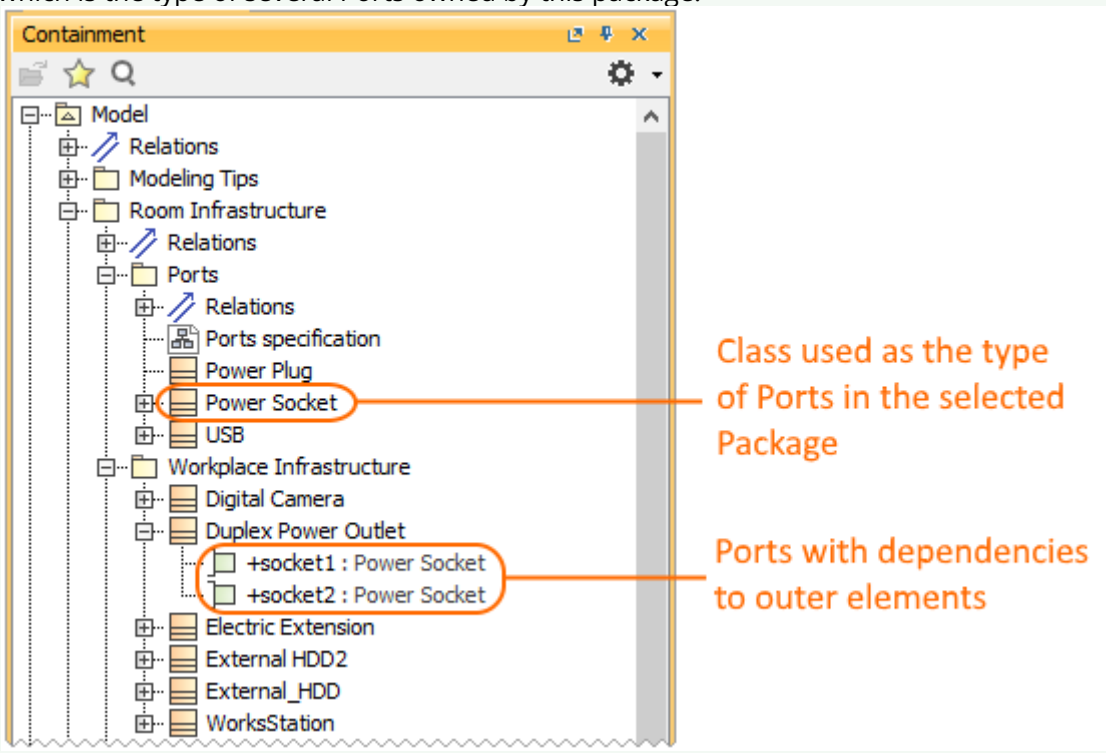
Depends On command

Use the **Depends On** command to find all elements used by a selected element (e.g., properties, tags etc.).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example

Let's say you have a model with the content tree shown in the figure below. Execute the **Depends On** command for the *Workplace Infrastructure* Package to see how its content depends on the outer elements (not owned by the Package) of the model. One of the search results shows that the *Workplace Infrastructure* Package depends on the *Power Socket* Class, which is the type of several Ports owned by this package.



The content tree of a sample model with the *Workplace Infrastructure* Package having dependencies to the outer elements of the model.

Note that in this particular situation you must select the **Search recursively** check box in the **Usages/Dependencies Search Options** dialog. This allows you to search for dependencies of the Package content rather than the Package itself. Also, select the **Ignore inner interrelations** checkbox to eliminate the dependencies between the elements contained by the package from the search results.

Searching for usages and dependencies

You can find usages and dependencies of a particular model element by opening its shortcut menu from the model browser or a diagram.

To search for usages/dependencies of a model element

1. In the model browser or a diagram right-click the element, the usages or dependencies of which you want to find.
2. In the shortcut menu of the element click **Related Elements**, and select **Used By** or **Depends On** (depending on what you want to find).

3. In the **Usages/Dependencies Search Options** dialog(see page 1195), select the search options you want to use.
4. Click **OK** to start the search.

The search results are listed in the usages/dependencies search results window(see page 1199) at the bottom of the modeling tool. The search results window also provides additional information, such as element type and how a particular element is used by/in the element for which you perform the search.

Related pages

- Model analysis(see page 1192)
  - Analyzing usages and dependencies(see page 1192)
  - Traceability(see page 1201)
  - Specifying criteria for querying model(see page 1227)
  - Element usage in diagrams(see page 1278)
  - Identifying Package Dependencies(see page 1280)
  - Comparing projects(see page 1291)
  - Validation(see page 1306)
  - Model Visualizer(see page 1346)
  - Suspect Links(see page 1354)

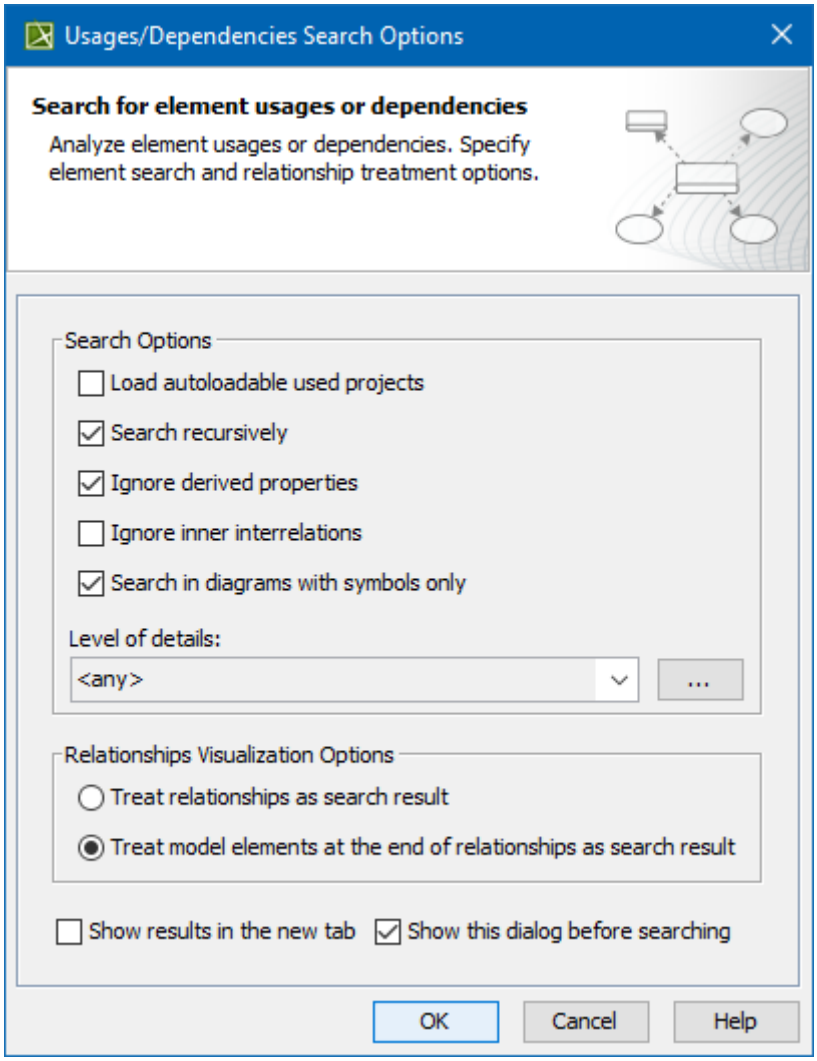
Usages and Dependencies Search Options dialog

Use the **Usages/Dependencies Search Options** dialog to find model elements that either use the selected element, or are used by it. You can search using different sets of criteria and analyze the search results to understand the relationships between model elements.

To open the **Usages/Dependencies Search Options** dialog

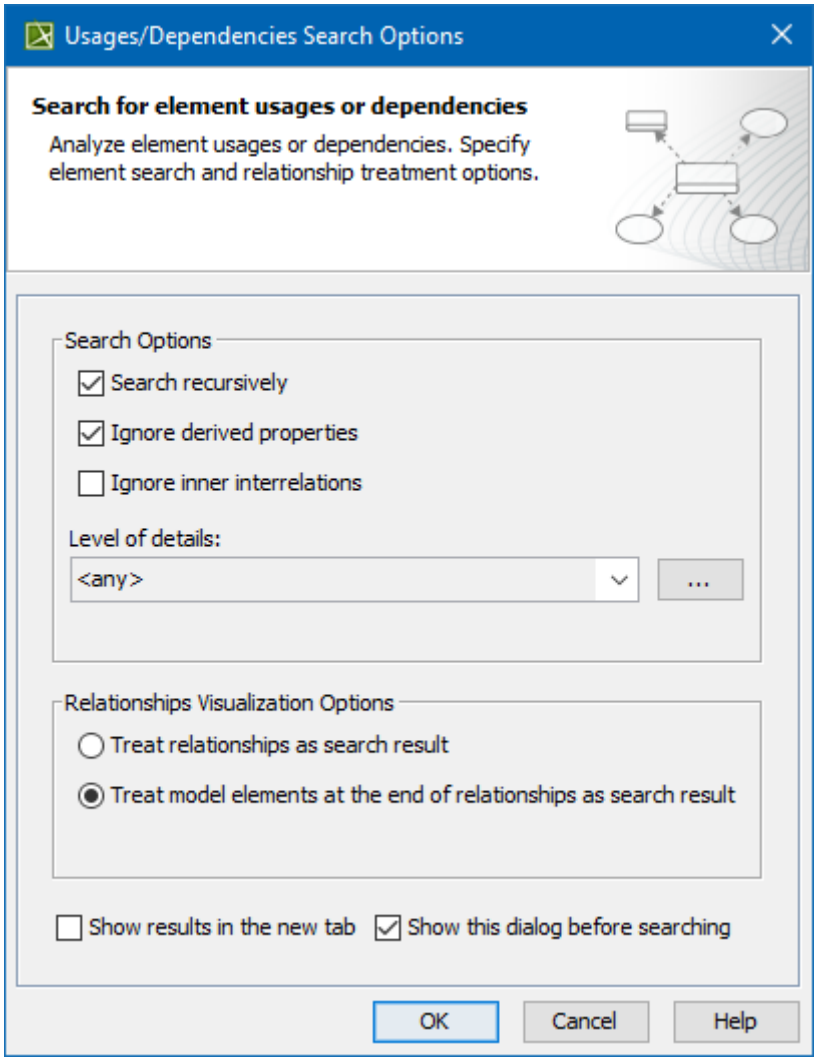
- In the model browser or a diagram, right-click the element whose usages or dependencies you want to find out, and select **Related Elements** > **Used by**. The dialog allows specifying the search options to find the elements that use the selected elements in the model.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



- In the model browser or a diagram, right-click the element whose usages or dependencies you want to find out, and select **Related Elements** > **Depends On**. The dialog allows specifying the search options to find model elements that are used by the selected elements.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The Usages/Dependencies Search Options dialog with the default search criteria selected.

The following table describes the search options of the **Usages/Dependencies Search Options** dialog.

Dialog element	Description
Load autoloadable used projects	Select this check box to include unloaded used projects in the usages search scope. Before starting the search, you can select the unloaded projects you want to load. This check box is available only if you select the **Used By** command.
Search recursively	Select this check box to search for usages/dependencies for the selected element and all the elements it owns.

Dialog element	Description
Ignore derived properties	Select this check box to exclude the derived properties(see page 1224) of the selected element from the usages/dependencies search results. It is recommended to select this check box, as including derived properties in the search scope may cause performance downgrade.
Ignore inter interrelatio ns	Select this check box to exclude the usages/dependencies between the elements owned by the selected element from the usages/dependencies search results. This option allows you to analyze how the selected element and its owned elements are related to the outer elements (no t owned by the selected element) of the model. **Usage information** This option is only effective if you select the **Search recursively** check box as well.
Search in diagrams with symbols only	Select this check box to check the element usage in diagrams containing symbols only.
Level of details	Select one of the predefined values (**Classifiers** or **Packages**), or click ... and select desired element types to define the level of details. If you select element types manually, the usages/dependencies search results will only display the elements of these types.
Treat relationship s as search result	Select this check box to find a complete list of used/dependent relationships together with other results.
Treat model elements at the end of relationship s as search result	Select this check box to exclude relationships from search results and only find the usages/ dependencies connected with these relationships.
Show results in the new tab	Select this check box to display the results of each search in a new tab.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

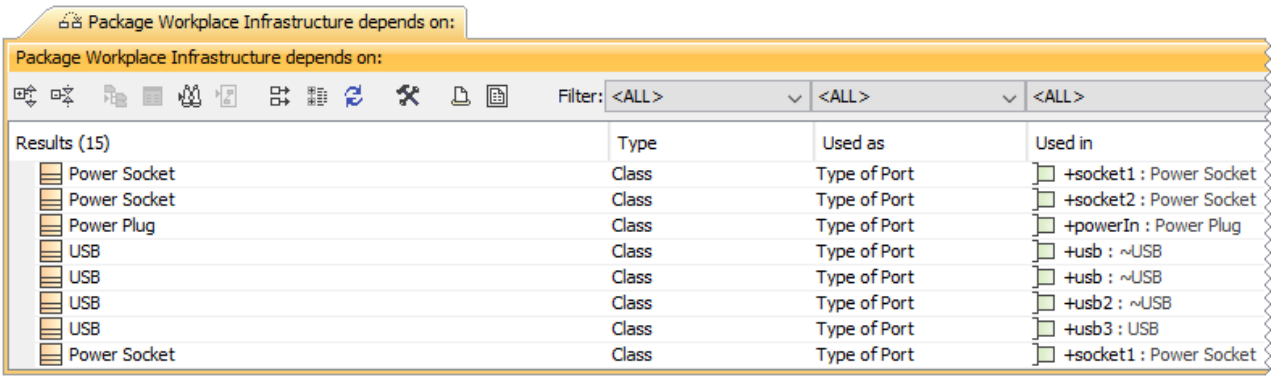
Dialog element	Description
Show this dialog before searching	Select this check box to open the **Usages/Dependencies Search Options** dialog before a new search is started. If you leave the check box empty, the dialog will not open, and the next search will use the search options specified during your last search. To open the **Usages/Dependencies Search Options** dialog, click on the toolbar of the usages/dependencies search results window.

Related pages

- Analyzing usages and dependencies(see page 1192)
- Usages and dependencies search results window(see page 1199)

Usages and dependencies search results window

The usages/dependencies search results window lists the usages/dependencies of the element for which you perform the search. In the usages/dependencies window you can analyze, group and filter search results, find element locations, and more.



The dependencies search results window.

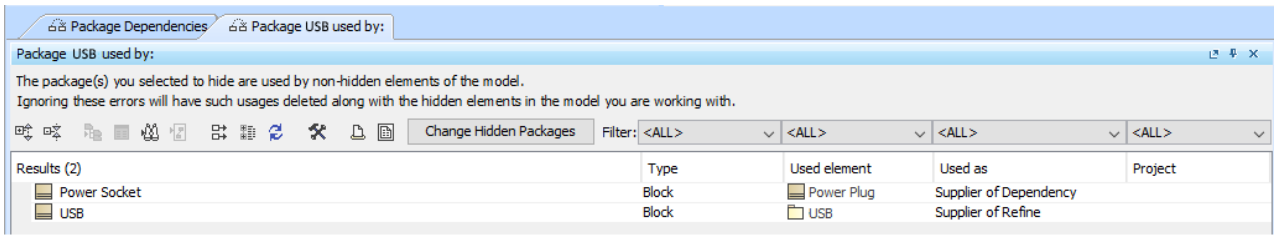
The following table describes how you can use the commands of the usages/dependencies search results dialog toolbar to manage usages/dependencies search results.

Toolbar element	Description
Expand All	Expands all search results listed in groups. To expand the content of a single group, click the plus sign next to the group name.

Toolbar element	Description
Collapse All	Collapses all search results listed in groups. To collapse the content of a single group, click the minus sign next to the group name.
Select in Containment Tree	Selects the desired element in the containment tree. To enable the button, select one of the search results in the usages/dependencies search result window.
Open Specification	Opens the Specification window of the desired element. To enable the button, se lect one of the search results in the usages/dependencies search result window.
Add to Search Results	Moves all usages/dependencies search results to the **Search Results** tab of the model browser. In the **Search Results** tab, all results are organized in two groups: **Found in loaded diagrams** and **Found in model**.
Open all diagrams that contain current Usages/ Dependencies	Opens all diagrams that use at least one the usages/dependencies search results. The button is inactive when there are no search results used in diagrams.
Display the full path of the elements	Displays full paths to the elements next to their names in the usages/ dependencies search results window.
Group by children	Groups all usages/dependencies search results by children.
Refresh	Refreshes usages/dependencies search results if elements or symbols are deleted from the model or new usages/dependencies are created.
Usages/Dependencies Search Options	Opens the Usages/Dependencies Search Options dialog.
Print	Prints the usages/dependencies search results table. Make sure that you expand the search results listed in groups if you want them to be shown on the printed page.
Export	Exports usages/dependencies search results to a .html or .csv file.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Toolbar element	Description
Filter	The filter boxes above each column of the usages/dependencies search results table allow you to filter search results by element type, used/dependent element name and usage type. Click a filter box to select the desired filter option.



The usage search results window

The following table describes usages search results window.

Column	Description
Results	Displays results with found usage.
Type	Displays element type in which usage was found.
Used element	Displays used elements.
Used as	Displays usage type.
Project	Displays the name of the used project file that owns the model element, which the package/model element uses.

Related pages

- Analyzing usages and dependencies(see page 1192)
- Usages and Dependencies Search Options dialog(see page 1195)

Traceability

Compatibility
The traceability solution is available in Architect and Enterprise editions.

Our modeling tools support the traceability feature that allows you to track, visualize, navigate, and analyze the elements involved in the traceability relations. The traceability relations are used to relate the elements representing the same concept in different UML models at different levels of abstraction or from different viewpoints. Different levels of abstraction may contain, include, or even correspond to different stages of system development process (starting from requirements analysis and finishing with implementation). The higher level of abstraction (e.g., requirements analysis), contains models with specifying elements, and the lower level of abstraction (e.g., implementation) includes models with realizing ones.

The traceability relations help to determine how your requirements or the other model artifacts are satisfied. As they may change, you can use traceability relations to monitor the impact of these changes.

Multiple types of custom and extended UML relationships (e.g., realization, abstraction), tags (e.g., **Alternative Flow of Events Diagrams**), and properties (e.g., **Owned Behavior**) are used to represent traceability relations between the specifying and realizing elements for showing traceability from requirements analysis to implementation and deployment models.

Relation Map and Dependency Matrix

You can visualize the traceability relations of your project in order to analyze them using the modeling tool's features, such as Relation Map(see page 364) (for the analysis of traces among multiple levels of abstraction) and Dependency Matrix(see page 334) (for the analysis of traces between any two levels of abstraction).

Element's Specification window, Properties panel, Go To, and Notes

You may track and navigate the elements, that are directly or indirectly related to a specific element through the traceability properties that will be represented in a special property group of the element's Specification window(see page 913) and Properties panel(see page 1207), or on the element's shortcut menu under the **Go To** menu. The traceability properties can also be visualized on a diagram using the standard modeling tool mechanism for displaying property values in notes.

Traceability Report

A Traceability Report is particularly useful when there is a need to visualize and verify that requirement analysis, design, and implementation model elements are all covered in higher or lower levels of abstraction, for example, all use cases should be covered with design classes realizing them.

Model Sample
The model used in all the examples in this section is the **Traceability** sample model that comes with our modeling tool. To open this sample do one of the following:

- On the Welcome screen, select **Samples > Product Features > Traceability**.
- Go to the folder *<MagicDraw installation directory>\samples\product features* and open the *Traceability.mdzip* file.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related pages

- Creating traceability relations(see page 1203)
- Traceability relations representation(see page 1204)
- Navigating between different levels of abstraction(see page 1211)
- Analyzing traceability relations(see page 1212)
- Predefined traceability rules(see page 1214)
- Custom traceability rules(see page 1224)

Creating traceability relations

Multiple relationships can be used for traceability representation, depending on the element type and method used.

The traceability relations can be represented by the following types of element dependencies:

- UML relationships (such as abstraction, realization, derivation)
- UML properties (such as **Owned Behavior**)
- UML tags (such as **Alternative Flow of Events Diagrams**)

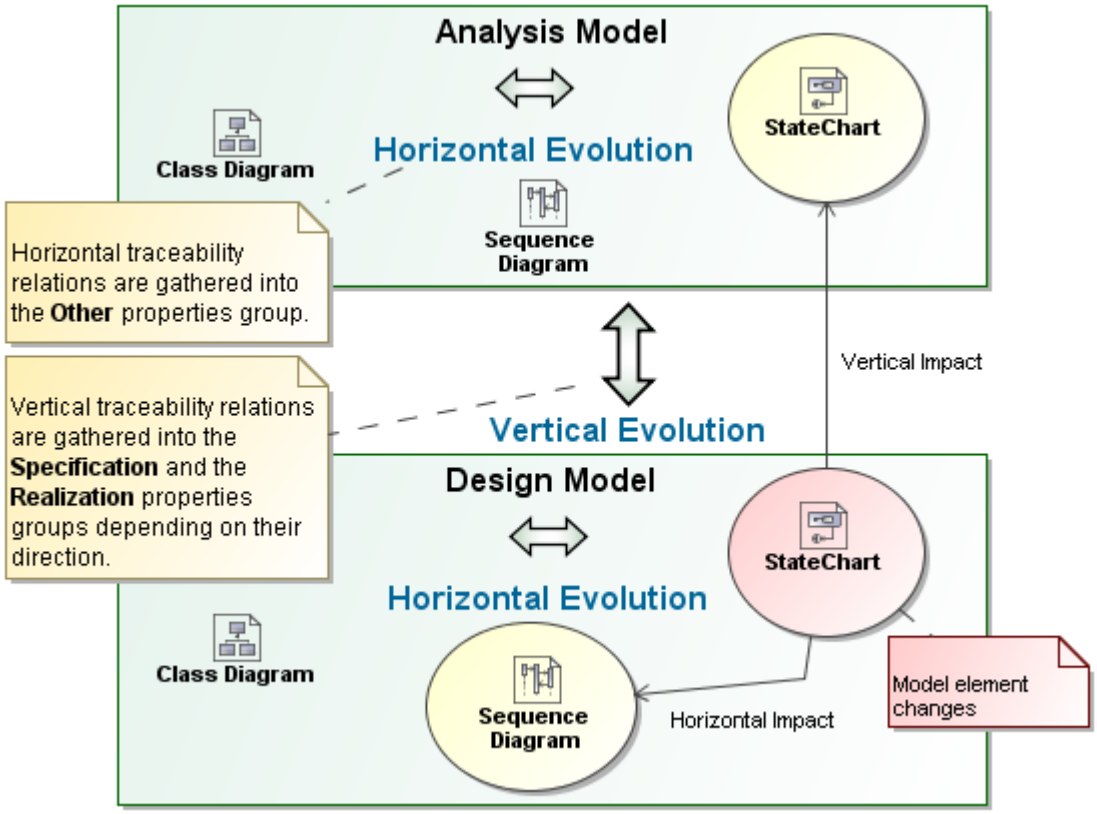
The traceability relations can be single level and multilevel.

- When elements are related directly by using a custom UML relationship, property, or tag, the traceability relation is considered to be a single level relation.
- When elements are related indirectly by usually using multiple types of the above mentioned dependencies, the traceability relation is considered to be a multilevel relation.

The traceability relations can be specifying, realizing, or other. This depends on which direction a relation is analyzed and which element is considered as a basis.

- A relationship between a specific element and an element that is the realization of this specific element, from the point of view of the specific element, is considered to be realizing traceability (for forward traceability(see page 1215)).
- A relationship between a specific element and an element that is the specification of this specific element, from the point of view of the specific element, is considered to be specifying traceability (for backward traceability(see page 1217)).
- All other traceability relations(see page 1219).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Horizontal and vertical traceability representation through the realizing, specifying, and other traceability relations

Related Pages

- Traceability(see page 1201)
  - Creating traceability relations(see page 1203)
  - Traceability relations representation(see page 1204)
  - Navigating between different levels of abstraction(see page 1211)
  - Analyzing traceability relations(see page 1212)
  - Predefined traceability rules(see page 1214)
  - Custom traceability rules(see page 1224)

Traceability relations representation

The traceability relations are represented through the so-called traceability properties that have been predefined for each element type according to the traceability method supported in the modeling tool.

A single traceability property shows an element or a set of elements that are related to a particular element through the relationships which are specified by some traceability rule. The predefined traceability rules are stored in the Traceability customization profile used by the MagicDraw Profile.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

The properties for the traceability relations are grouped into the realization, specification, and other groups, owning both direct (single level) and indirect (multiple levels) traceability relations. The indirect traceability relations are represented by the so-called **All** properties, for example, **All Specifying Elements**, **All Realizing Elements**, and **All Specific Classifiers**.

The relations for traceability creation(see page 1203), visualization(see page 1204) and navigation(see page 1211) is also possible without using traceability properties. However, the customizable model driven traceability properties, which represent the traceability relations in a single place, can greatly help in traceability information visualization and access.

> **Tip**
> You can customize the predefined traceability properties according to your needs or create your own traceability properties and specify your own rules. You can also customize the grouping of the traceability properties according to your needs. For more information, please, refer to the sections Extending metamodel with derived properties(see page 2146) and Creating your own property groups and subgroups(see page 2126).

Element's traceability properties are represented in the following places:

- Specification window(see page 1205)
- Properties panel(see page 1207)
- Note on a diagram(see page 1208)
- Go To submenu(see page 1209)
- Traceability Report(see page 1210)

Related pages

- Traceability(see page 1201)
  - Creating traceability relations(see page 1203)
  - Traceability relations representation(see page 1204)
  - Navigating between different levels of abstraction(see page 1211)
  - Analyzing traceability relations(see page 1212)
  - Predefined traceability rules(see page 1214)
  - Custom traceability rules(see page 1224)

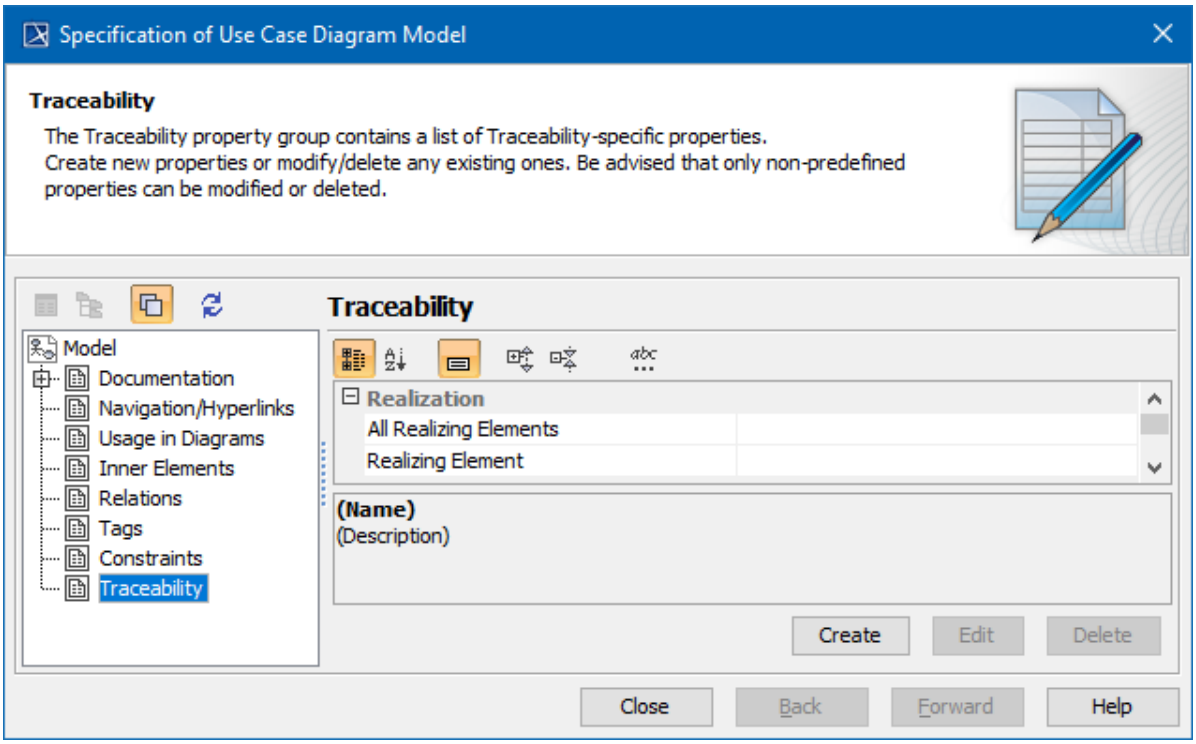
Traceability properties in Specification window

The Traceability property group in element's Specification window is one of the places, wherein the element's traceability properties, showing its realizing and/or more specific elements, are represented.

To view an element's traceability properties in its Specification window

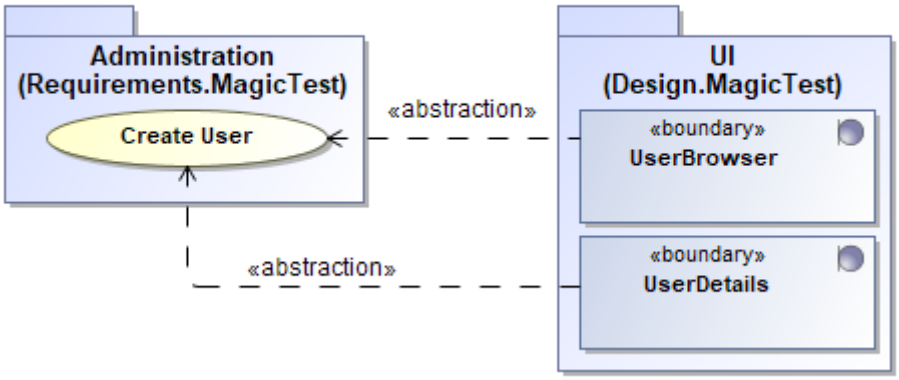
1. Select an element and open the element's Specification window by right clicking on the element.
2. Click the Traceability property group.

3. Now you can view element's traceability properties.



Traceability property group in the Specification window

The following picture gives an example of the traceability relations between the elements from different abstraction levels of the same project.



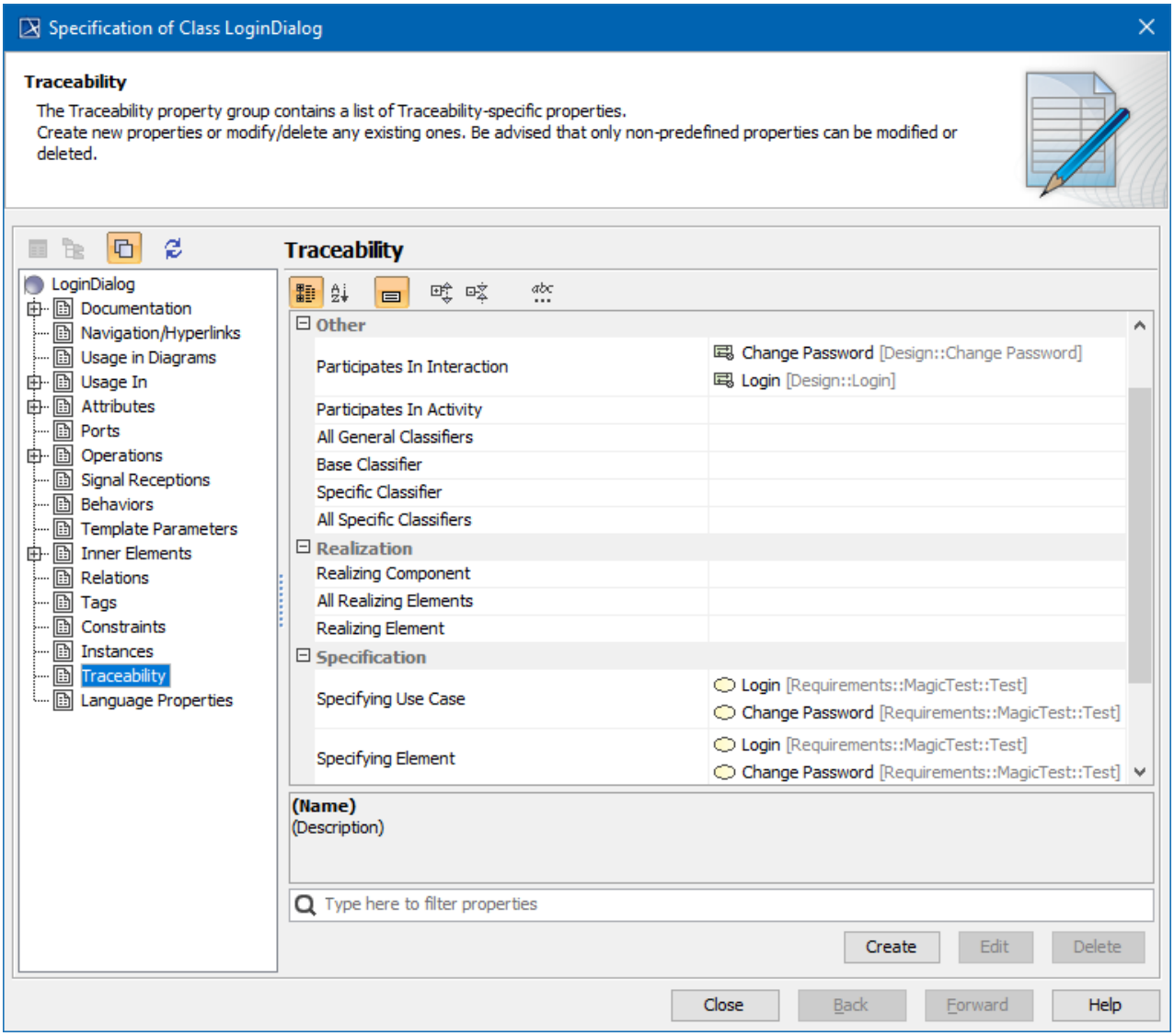
An example of single level traceability relations

As you see, the "Create User" use case from the "Requirements" package is covered by two design classes, and these classes are accordingly specified by this particular use case.

Therefore, the traceability properties such as **Realizing Class and Realizing Element** represent both design classes as realizing elements of this use case. Accordingly, the traceability properties such as **Sp ecifying Use Case** of each design class represent the "Create User" use case as a more specific element.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Note that the same related element can be represented through different traceability properties.



Realizing and Specifying traceability properties in Specification window

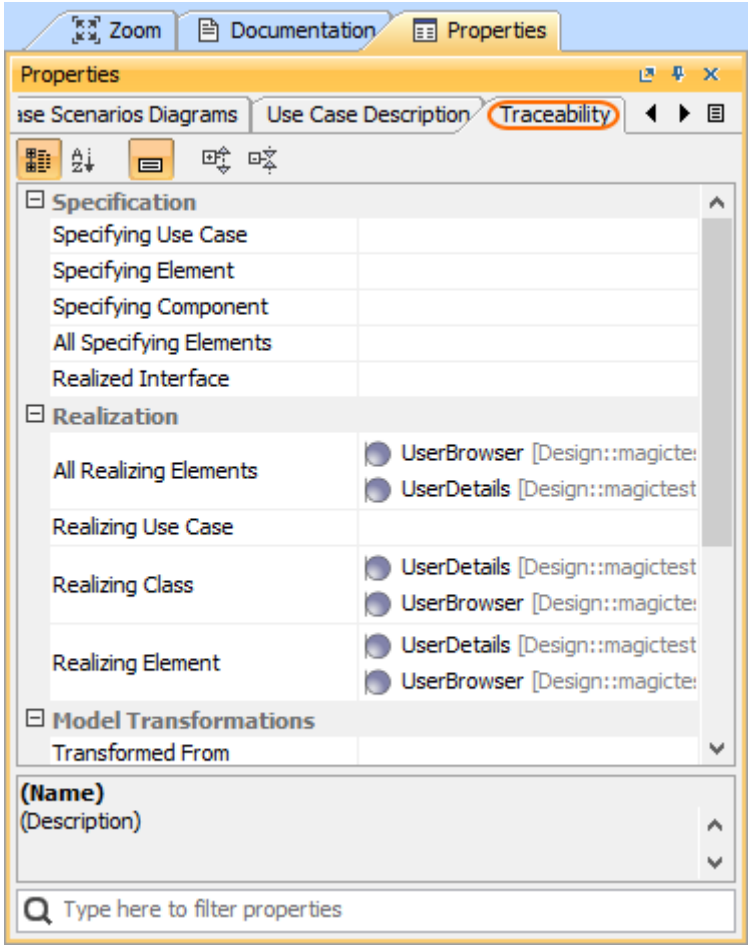
Traceability properties in Properties panel

The **Traceability** tab in the element's **Properties** panel (at the bottom of the Browser window) is one of the places, wherein the element's traceability properties, showing its realizing and/or more specific elements, are represented.

To view an element's traceability properties in its **Properties** panel

1. Select the element in the Containment tree or its shape on the diagram.
2. In the Model Browser, click on the Properties panel > Traceability tab. Now you can view element's traceability properties.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Traceability tab in the Properties panel

Traceability properties in notes

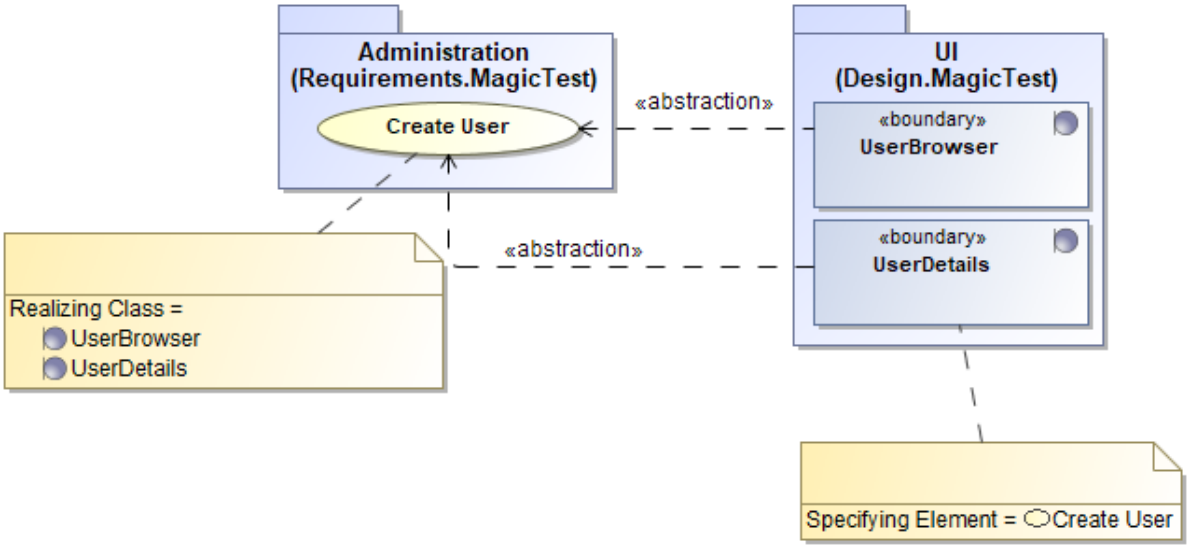
The traceability properties can be visualized on a diagram using the standard MagicDraw mechanism displaying the property values in notes.

To visualize an element's traceability with other elements in the note on a diagram

---

1. Create a note for the element, whose traceability properties you want to visualize.
2. From the note's shortcut menu, select Edit Compartments > Element Properties .
3. In the list on the left-hand side, click the traceability property of you choice (e.g., Realizing Class or Specifying Use Case) and then click the > button.
4. Click OK. Now you can view the element's traceability with other elements.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Traceability properties in the notes on a diagram

Traceability properties in Go To submenu

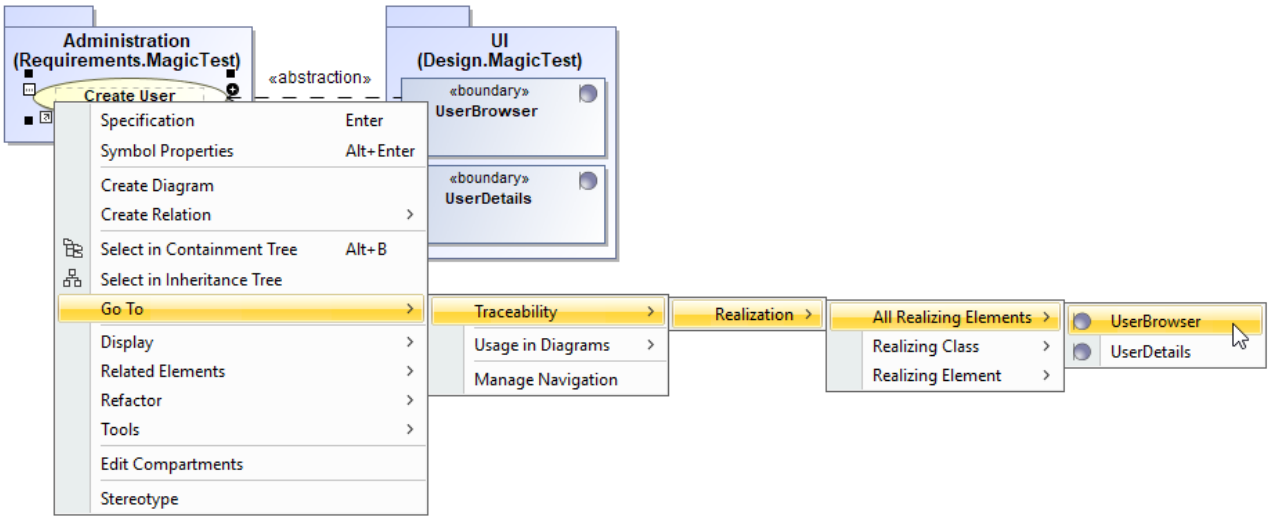
The submenu of the Go To menu on the element's shortcut menu allows you to easily find and navigate to the related elements through the traceability relations.

To select an element from the higher/lower level of abstraction in the Containment tree

---

1. Select the element in the Containment tree or its shape on the diagram.
2. From the element's shortcut menu, select Go To > Traceability > Specification/ Realization/ Other, choose a property, and then select an element.
3. The element will be selected in the Containment tree.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Traceability properties in the Go To submenu

Traceability properties in Traceability Report

The Traceability Report feature supports the coverage analysis and publishes elements that are related to the selected elements through the traceability relations. You can generate a report either for the whole project or for a selected part of the project. This feature is the output of the coverage analysis.

Coverage analysis provides the visibility of each element's related artifacts, indicated as realizing (lower level of abstraction) and/ or specifying (higher level of abstraction) ones.

The main objective of the Traceability Report is to visualize and verify that Analysis, Design, and Implementation model elements are all covered.

The Traceability Report provides the ability to:

- Find the areas of uncovered parts.
- According to the report information, create additional artifacts to increase coverage.
- Measure the coverage quantitatively.
- Identify the redundant artifacts.

To generate a traceability report of the selected scope

---

1. On the **Tools** menu, click **Report Wizard .**
2. In the **Select Template** area, select **Traceability > Traceability** and then click **Next** .
3. Click **Next** .
4. In the **Select Element Scope** area, define the scope of the report by using the buttons placed between two lists, and then click **Next**.
5. In the **Output Options** area, define the appropriate options.
6. Click **Generate**. Your traceability report will be generated (generation time depends on the selected scope).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

6. Use Case Realizing Classes

Shows the classes representing the use case realization in the Design model.

Use Case	Class
Create User [Administration (Requirements.MagicTest)]	UserBrowser [UI (Design.MagicTest)] UserDetails [UI (Design.MagicTest)]
Coverage	100 %

A fragment of Traceability Report

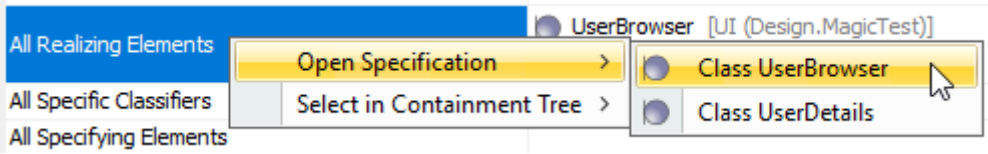
Navigating between different levels of abstraction

During the development process and in order to understand the system fast, navigating between elements from different levels of abstraction is necessary. Navigating from one element to another is easy using the modeling tool's GUI capabilities of the traceability feature, if there is at least one traceability relation between them.

To navigate between different levels of abstraction, use the following features:	
Element's **Specification** window	Open the element's **Specification** window and select the **Traceability** property group, and then right-click on the property. Choose **Open Specification** and select an element (if there is more than one related element). The Specification window of this element will be opened.
Element's **Properties** panel	Open the element's **Properties** panel, select the **Traceability** tab, and then right-click on the property. Choose **Open Specification** and select an element (if there is more than one related element). The Specification window of this element will be opened.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To navigate between different levels of abstraction, use the following features:	
Go To submenu on element's shortcut menu	From the element's shortcut menu, select **Go To > Traceability > Specification/ Realization/Other** . Choose a property and then select an element. The element will be selected in the Containment tree.



Selecting an element to open specification

Related pages

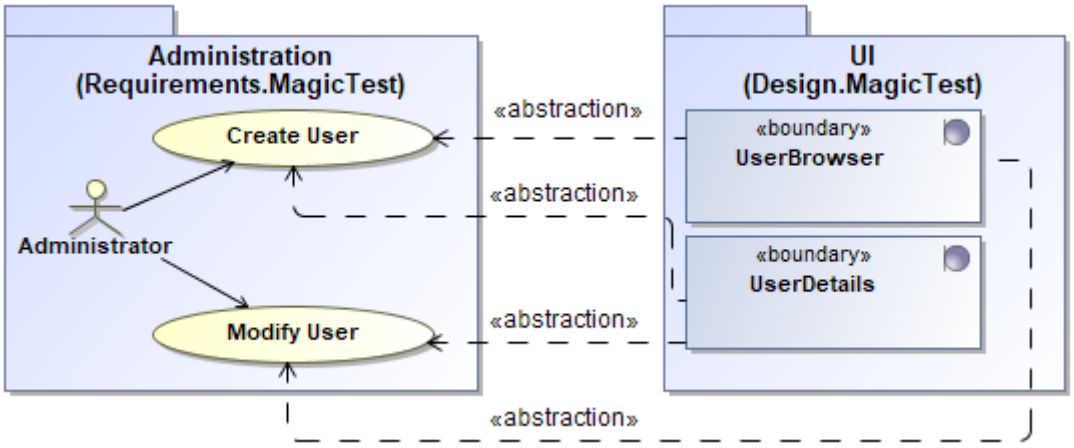
- Traceability(see page 1201)
  - Creating traceability relations(see page 1203)
  - Traceability relations representation(see page 1204)
  - Navigating between different levels of abstraction(see page 1211)
  - Analyzing traceability relations(see page 1212)
  - Predefined traceability rules(see page 1214)
  - Custom traceability rules(see page 1224)

Analyzing traceability relations

You can perform the impact and/or gap analysis in your project using the Dependency Matrix feature, which is a powerful way for representing traceability relations between multiple elements from different packages, levels of abstraction, views, or other relations that cannot be represented on diagrams, for example, relations through UML tags. You can create your own dependency matrices or even custom dependency matrix types for visualization of various traceability relations. You only need to define a relevant traceability property as **dependency criteria** for this.

The following figure depicts an example of traceability relations between several model elements.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Example of traceability relations

These relations can be represented by the following traceability properties:

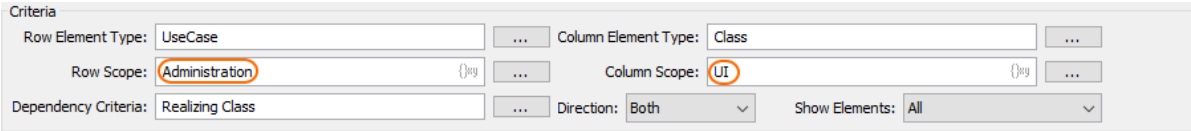
- Realizing relations (use cases classes):
  - Realizing Class
  - Realizing Element
  - All Realizing Elements
- Specifying relations (classes use cases):
  - Specifying Use Case
  - Specifying Element
  - All Specifying Elements

You can create a dependency matrix to visualize these traceability relations. The following steps will show you how to create a dependency matrix for the Realizing Class predefined property.

To create a dependency matrix for the Realizing Class predefined property:

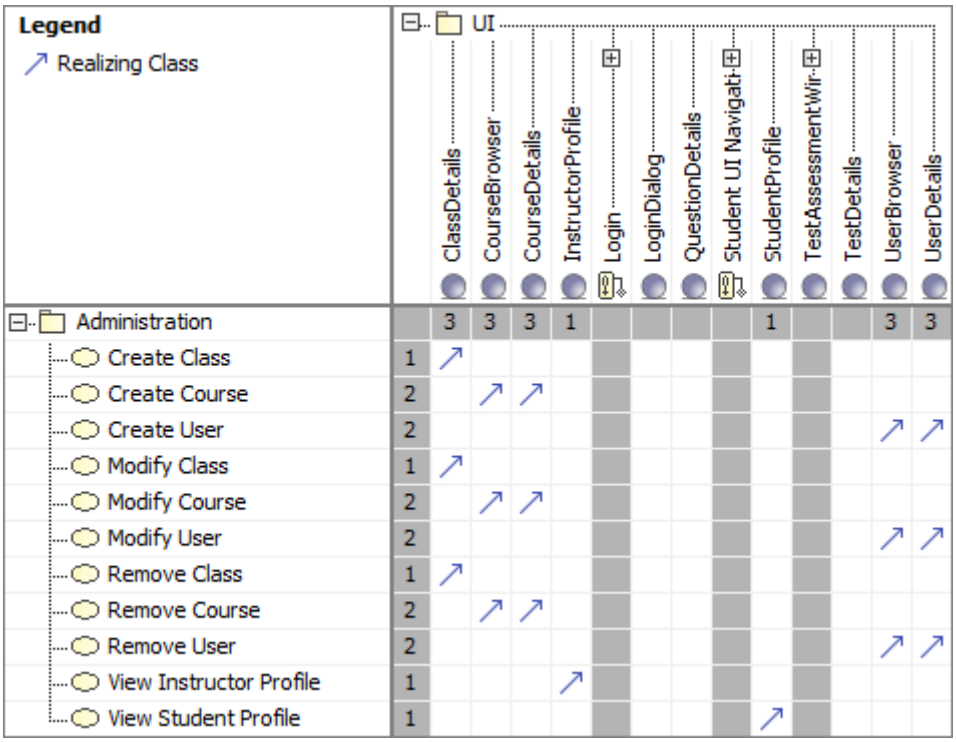
---

1. Create a Dependency Matrix diagram.
2. Define row and column type element types as follows:
   a. For the Row Element Type, select **UseCase**
   b. For the Column Element Type, select **Class**
3. Define row and column scopes as follows:
   a. For the Row Scope, click ... **> Requirements > MagicTest** and select **Administration**.
   b. For the column scope, click ... **> Design > MagicTest** and select **UI**.
4. Define the **Realizing Class** property as Dependency Criteria.
5. Rebuild the matrix (click in the diagram toolbar).



Defining criteria for visualization of traceability relationships that can be represented by Realizing Class predefined property

The following figure depicts the created Dependency Matrix showing the traceability relationships between use cases and design classes they realize:



Visualization of Realizing Class traceability relations on Dependency Matrix

Related Pages

- Traceability(see page 1201)
  - Creating traceability relations(see page 1203)
  - Traceability relations representation(see page 1204)
  - Navigating between different levels of abstraction(see page 1211)
  - Analyzing traceability relations(see page 1212)
  - Predefined traceability rules(see page 1214)
  - Custom traceability rules(see page 1224)

Predefined traceability rules

Besides allowing you to create your own traceability rules, our modeling tool allows you to use a set of predefined element properties for traceability purposes. The traceability solution also includes a report template used for coverage analysis. You can customize this template or create your own templates for producing your own traceability analysis reports.

The main objective of the predefined traceability rules is to allow the use of the traceability engine instantly (out- of-the-box) for a typical project. The predefined rules allow for analyzing and verifying links between elements residing in Analysis, Design, and Implementation layers of your models. Elements in these layers are extended with a set of properties, pointing to elements from the higher abstraction layer (backward traceability) and the lower abstraction layer (forward traceability).

Overall, there are three groups of predefined traceability properties, which will be detailed in subsequent sections:

- Forward traceability - realization(see page 1215)
- Backward Traceability – Specification(see page 1217)
- Other traceability relations (see page 1219)

Related pages

- Traceability(see page 1201)
  - Creating traceability relations(see page 1203)
  - Traceability relations representation(see page 1204)
  - Navigating between different levels of abstraction(see page 1211)
  - Analyzing traceability relations(see page 1212)
  - Custom traceability rules(see page 1224)

Forward traceability - realization

The forward traceability ensures that all specified artifacts are covered by elements from the lower abstraction level.

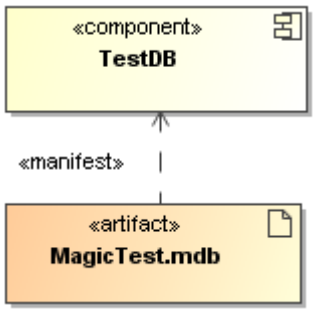
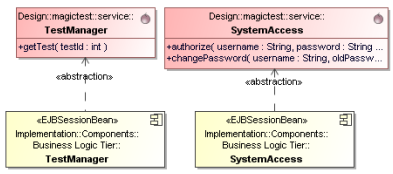
Property Name	Description	Applied for	Reference through...	Value elements type	Example
Manifested By Artifacts	The property shows artifacts that physically render the current component.	Component	Relationships: Manifestation	Artifact	Example

Property Name	Description	Applied for	Reference through...	Value elements type	Example
Realizing Component	The property shows the components representing the class realization in the Implementation model.	Class	Relationships: Abstraction	Component	Example
Realizing Classifier	The property shows classifiers that realize components through component realization.	Component	Relationships: Component Realization, Realization	Classifier	Example
Realizing Class	The property shows the classes representing the use case realization in the Design model.	Use Class	Relationships: Abstraction	Class	Example
Realizing Use Case	The property shows the realizing use cases of the current use case in the lower level of abstraction thus demonstrating how the use case is implemented. For example, the Requirements Use Case realizes the Business Use Case.	Use Case	Relationships: Abstraction	Use Case	Example
Realizing Classifier	The property shows the classifiers conforming to the contract specified by the interface and related with this interface through the interface realization relationship.	Interface	Relationships: Interface Realization	Classifier	Example

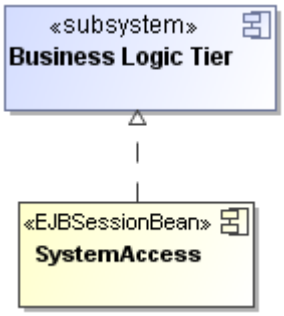
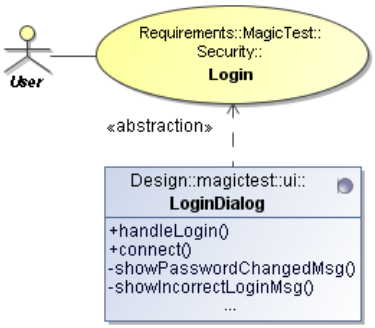
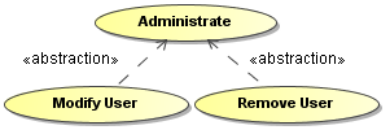
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Property Name	Description	Applied for	Reference through...	Value elements type	Example
Realizing Element, All Realizing Elements.	The **Realizing Element** property gathers realizing elements from the lower abstraction level. The **All Realizing Elements** property transitively gathers realizing elements from all lower abstraction levels.	Element	Relationships: Abstraction, Component Realization, Interface Realization.	Element	Example

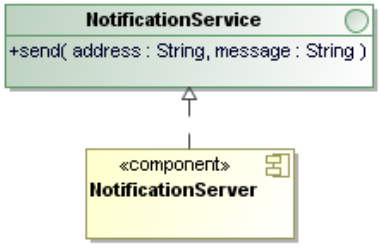
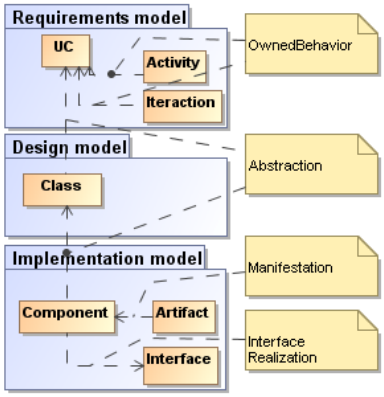
Backward traceability – specification

Property Name	Description	Applied For:	Reference Through:	Value elements type	Example
Manifested in Artifacts	The property shows which components are manifested in an artifact.	Artifact	Relationships: Manifestation	Component	Example 
Specifying Class	The property shows the classes representing the component specification from the Design model.	Component	Relationships: Abstraction	Class	Example 

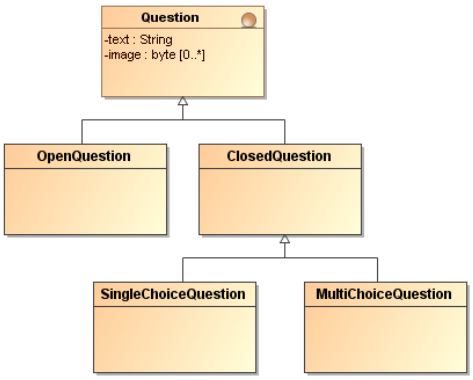
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Property Name	Description	Applied For:	Reference Through:	Value elements type	Example
Specifying Component	The property shows the components that are realized by classifiers through component realization.	Classifier	Relationships: Component Realization, Realization	Component	Example 
Specifying Use Case	The property shows the use cases (from the Requirements model) representing the class specification.	Class	Relationships: Abstraction	Use Case	Example 
Specifying Use Case	The property shows the use cases that specify the given use case in the higher level of abstraction. For example, the Business Use Case specifies the Requirements Use Case.	Use Case	Relationships: Abstraction	Use Case	Example 

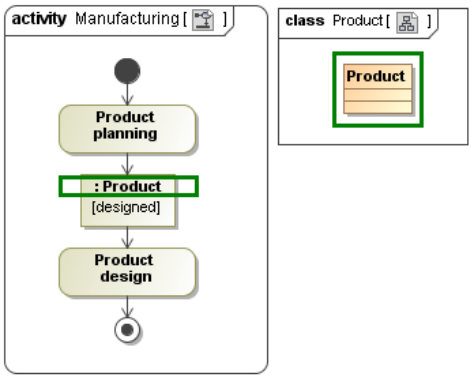
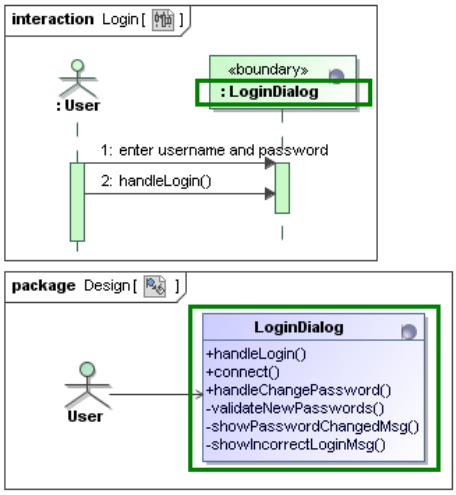
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Property Name	Description	Applied For:	Reference Through:	Value elements type	Example
Realized Interface	The property shows the interfaces specifying the contract, in which the related classifier conforms to.	Classifier	Relationships: Interface Realization	Interface	Example 
Specifying Element, All Specifying Elements	The **Specifying Element** property gathers specifying elements from the upper abstraction level. The **All Specifying Elements** property transitively gathers specifying elements from all upper abstraction levels.	Element	Relationships: Abstraction, Component Realization, Interface Realization.	Element	Example 

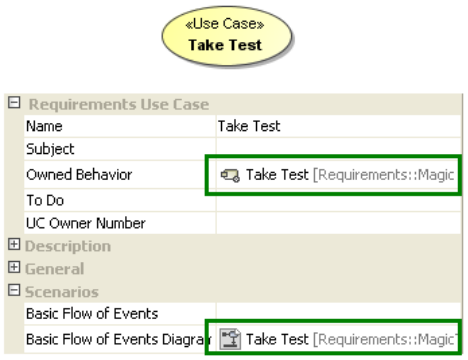
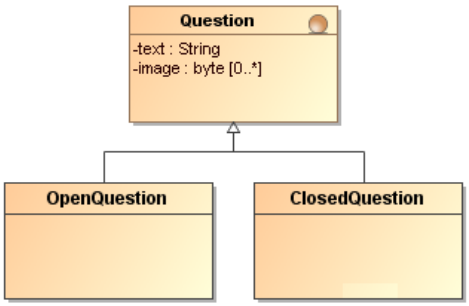
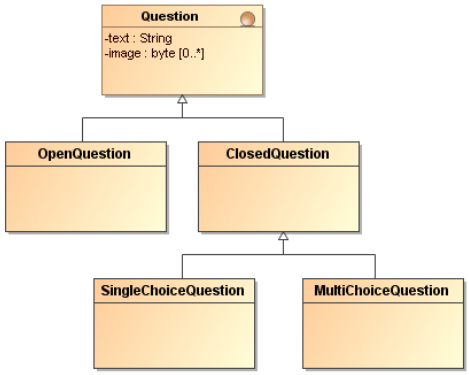
Other traceability relations

Property Name	Description	Applied For:	Reference Through:	Value elements type	Example
Base Classifier, All General Classifiers	The **Base Classifier** property gathers classifiers from which the current element directly inherits. The **All General Classifiers** property gathers classifiers from which the current element directly or indirectly inherits.	Classifier	Relationships: Generalization.	Classifier	Example 

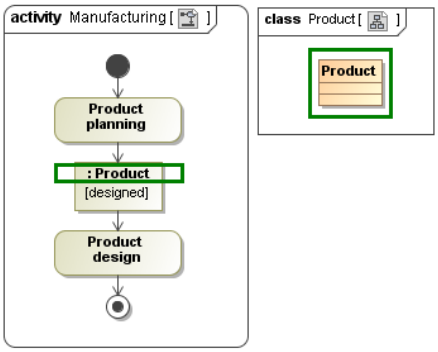
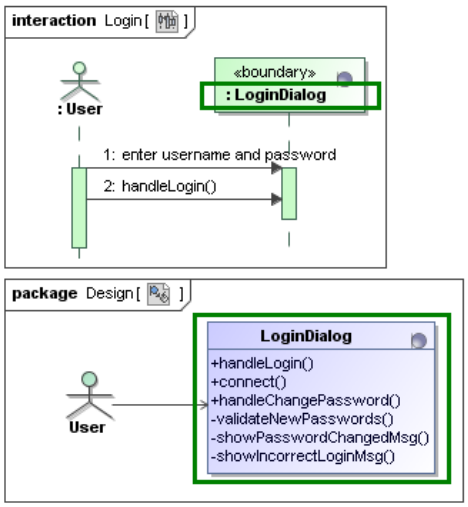
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Property Name	Description	Applied For:	Reference Through:	Value elements type	Example
Participates in Activity	The property shows either the activities wherein the classifier is used as an object type, or the activities that are owned by the classifier.	Classifier	Property Chain	Activity	Example 
Participates in Interaction	The property shows either the interactions wherein the classifier is used as an object type, or the interactions that are owned by the classifier.	Classifier	Property Chain	Interaction	Example 

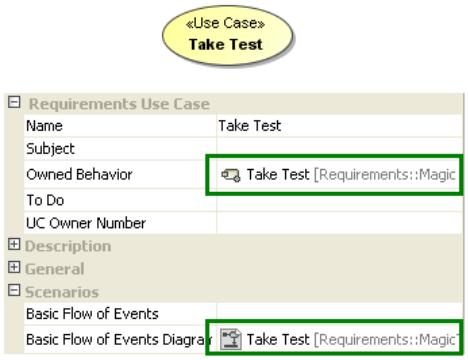
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Property Name	Description	Applied For:	Reference Through:	Value elements type	Example
Describing Behavior	The property gathers behavior diagrams (state, activity, sequence, communication, and interaction) that describe the use case.	Use Case	Properties: ownedBehavior Tags: Exceptional Flow of Events Diagrams, Basic Flow of Events Diagrams, Alternative Flow of Events Diagrams.	State, Activity, Sequence, Communication, and Interaction Diagrams	Example 
Specific Classifier	The property shows directly inherited classifiers.	Classifier	Relationships: Generalizations.	Classifier	Example 
All Specific Classifiers	The property gathers directly or indirectly inherited classifiers.	Classifier	Relationships: Generalization.	Classifier	Example 

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Property Name	Description	Applied For:	Reference Through:	Value elements type	Example
Used Classifier	The property gathers what classifiers are used in the activity as object types, or the classifier that owns the activity.	Activity	Property Chain	Classifier	Example 
Used Classifier	The property gathers what classifiers are used in the activity as object types, or the classifier that owns the activity.	InterAction	Property Chain	Classi	Example 

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Property Name	Description	Applied For:	Reference Through:	Value elements type	Example
Described Use Case	The property shows the described use cases by behavior or behavior diagram.	State, Activity, Sequence, Communication, and Interaction Diagrams	Properties: ownedBehavior Tags: Exceptional Flow of Events Diagrams, Basic Flow of Events Diagrams, Alternative Flow of Events Diagrams.	Use Case	Example 

Custom traceability rules

On this page

- Creating a custom traceability rule(see page 1224)
- Editing a custom traceablity rule(see page 1225)
- Displaying a custom traceability rule(see page 1226)
- Removing a custom traceability rule(see page 1226)

You can create custom traceability rules if predefined traceability rules are not sufficient to trace and analyze relationships between elements in your model. In a model, custom traceability rules are considered as derived properties(see page 2146), because their values are produced or calculated from other properties. Usually, the values of custom traceability rules cannot be changed directly, but you can edit(see page 1225) and display(see page 1226) these rules the same way as any other properties.

Creating a custom traceability rule

Custom traceability rules can be quickly created from the Specification window of a particular element. This way you create a new derived property for all elements of the selected element type.

To create a custom traceability rule

1. In the model browser or a diagram, double-click the element for which you want to create a custom traceability rule to open its Specification window.

2. In the property group list on the left side of the Specification window, select the **Traceability** group.
3. Click the Create button.
4. In the open **Expression** dialog, specify the name and expression(see page 1227) of a new traceability rule.
5. Click **OK** and close the Specification window.If you want to create a derived property with a result other than an element, you need to change the derived property type manually.

The customization of the newly created traceability rule is saved in the *Derived Properties* Package with the «derivedPropertiesSuite» stereotype. This folder is normally hidden in the containment tree of your model. You can find it using the **Quick Find** functionality(see page 223). This may be useful when you want to change traceability rule properties.

Editing a custom traceablity rule

You can modify a custom traceability rule from the Specification window of a particular element or directly from a table if the traceability rule is displayed in it.

> **Editing traceability rule values**
> You can edit the value of a custom traceability rule directly only when its operation type is Simple Navigation(see page 1231) and you specified only one criterion.

To edit a custom traceability rule

1. In the model browser or a diagram, double-click the element with the traceability rule you want to edit to open its Specification window.
2. In the property group list on the left side of the Specification window, select the **Traceability** gro up.
3. In the property specification area on the right side of the window, select the traceability rule you want to edit.
4. Click the **Edit** button.
5. In the **Expression** dialog, make desired changes and click **OK**.
6. Close the Specification window.

If a custom traceability rule is displayed in a table, you can edit it directly from that table.

To edit a custom traceability rule in a table

1. In a table, right-click the header of the desired custom traceability rule column and select **Edit Derived Property**.
2. In the **Expression** dialog, make desired changes and click **OK**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Displaying a custom traceability rule

If you want to visualize custom traceability rules, you can display them in the compartments of element shapes or in tables.

To display a custom traceability rule on an element shape

1. Do one of the following:
   - In a diagram, right-click the desired element shape and select **Edit Compartments**.
   - In a diagram, select the desired element shape and click the **Compartments** smart manipulator, click on the down arrow to expand the menu, and select **Edit Compartments**.
2. In the **Compartment Edit** dialog, select the **Element Properties** tab.
3. In the **Hidden** list on the left side of the dialog, select the traceability rules you want to display.
4. Click > to add the selected traceability rules to the **Selected** list on the right side of the dialog. If you want to display all properties from the **Hidden** list, click >>.
5. Click OK.

To display a custom traceability rule in a table

1. In the toolbar of an active table, click the **Show Columns** button and from the menu select the **Select Columns** commnad.
2. In the **Select Columns** dialog, set the traceability rules you want to display as *true*.
3. Click **OK**.

Removing a custom traceability rule

You can remove a Custom traceability rule from the Specification window of a particular element or directly from a table if the traceability rule is displayed in it.

To remove a custom traceability rule

1. In the model browser or a diagram, double-click the element with the traceability rule you want to remove to open its Specification window.
2. In the property group list on the left side of the Specification window, select the **Traceability** gro up.
3. In the property specification area on the right side of the window, select the traceability rule you want to remove.
4. Click the **Delete** button and close the Specification window.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To remove a custom traceability rule in a table

- In a table, right-click the header of the unwanted custom traceability rule column and select **Remove Derived Property**.

Related pages

- Traceability(see page 1201)
  - Creating traceability relations(see page 1203)
  - Traceability relations representation(see page 1204)
  - Navigating between different levels of abstraction(see page 1211)
  - Analyzing traceability relations(see page 1212)
  - Predefined traceability rules(see page 1214)

Specifying criteria for querying model

Specifying criteria is necessary to

- Associate row and column elements in a dependency matrix.
- Display relations between elements in a relation map.
- Gather the contents of a smart package.
- Calculate/gather the value of a derived property.

A structured expression can also be the body of the executable opaque behavior, metric definition or validation rule, when the StructuredExpression language is selected for specifying that body.

Use one of the following operation types to specify the criterion:

- Using Simple Navigation(see page 1231)
- Using Metachain Navigation(see page 1235) (indirect relations through chains of properties)
- Find(see page 1239)
- Filter(see page 1250)
- Union(see page 1252)
- Exclude(see page 1253)
- Using Implied Relations(see page 1242)
- Type Test(see page 1251)
- Property Test(see page 1247)
- Operations from the model(see page 1255)
- Various scripts(see page 1267)

You can also use custom operation types. In addition, a criterion can be an expression combining several operations by passing the results of one operation call to another operation.

If multiple criteria (operations) are specified, the result will be calculated according to all these criteria (operations).

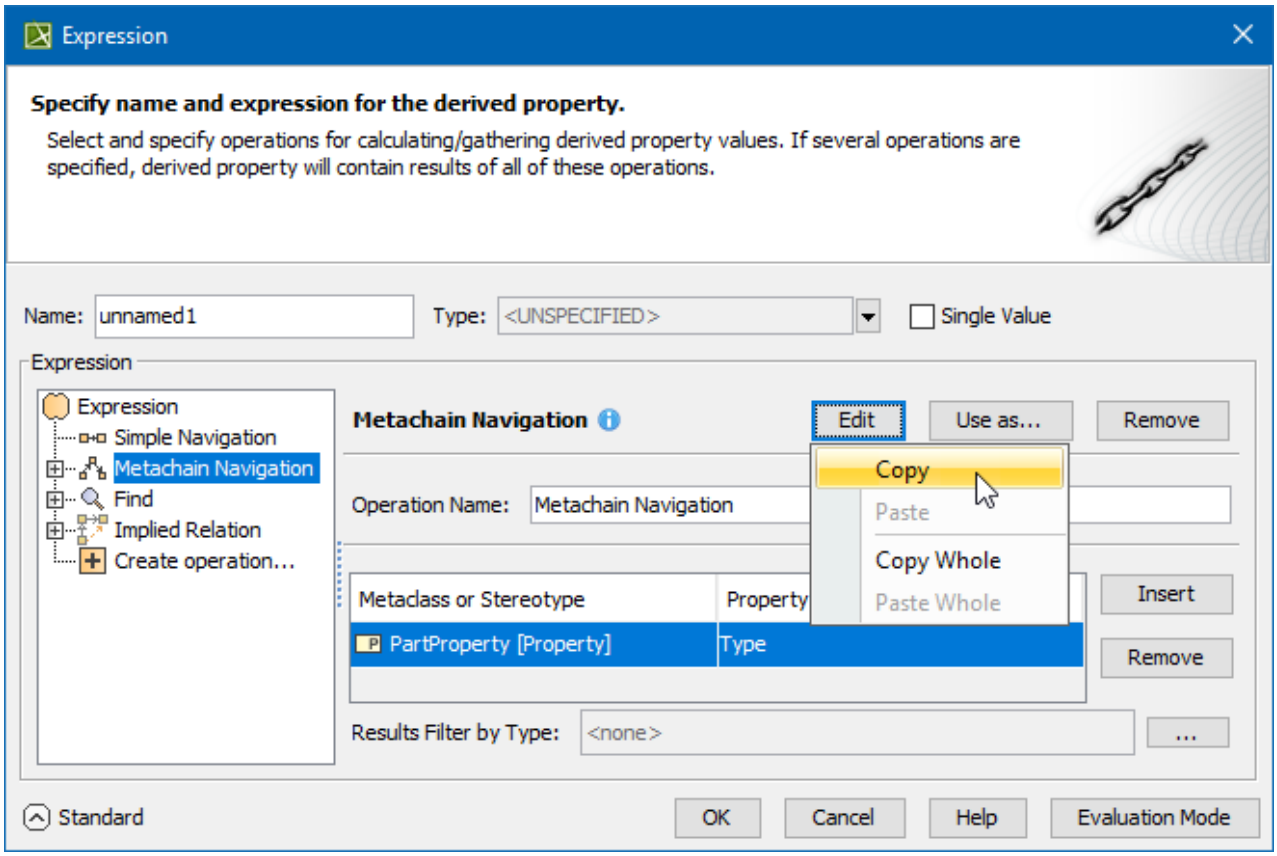
You can copy and paste the selected structured expressions in the **Expression** dialog. Select all or part of the structured expression in one dialog and paste it into the same or another dialog.

You can also copy the structured expression:

- from one smart package and paste it into another;

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- from the matrices and paste them into the relation maps and vice versa;
- and use it in the smart package.



Copying from the Expression dialog

Concepts

Read the following description to better understand the material.

Concept	Description
Contextual element	The element that begins the calculation of the result according to the specified criteria. In a dependency matrix, the context is a row or column element. In a relation map, this is every node of the structure. In a table, the context is the table scope value. A smart package is the context itself. The context of a derived property is the target of the customization element that owns this derived property.

Related pages

Getting started with specifying criteria

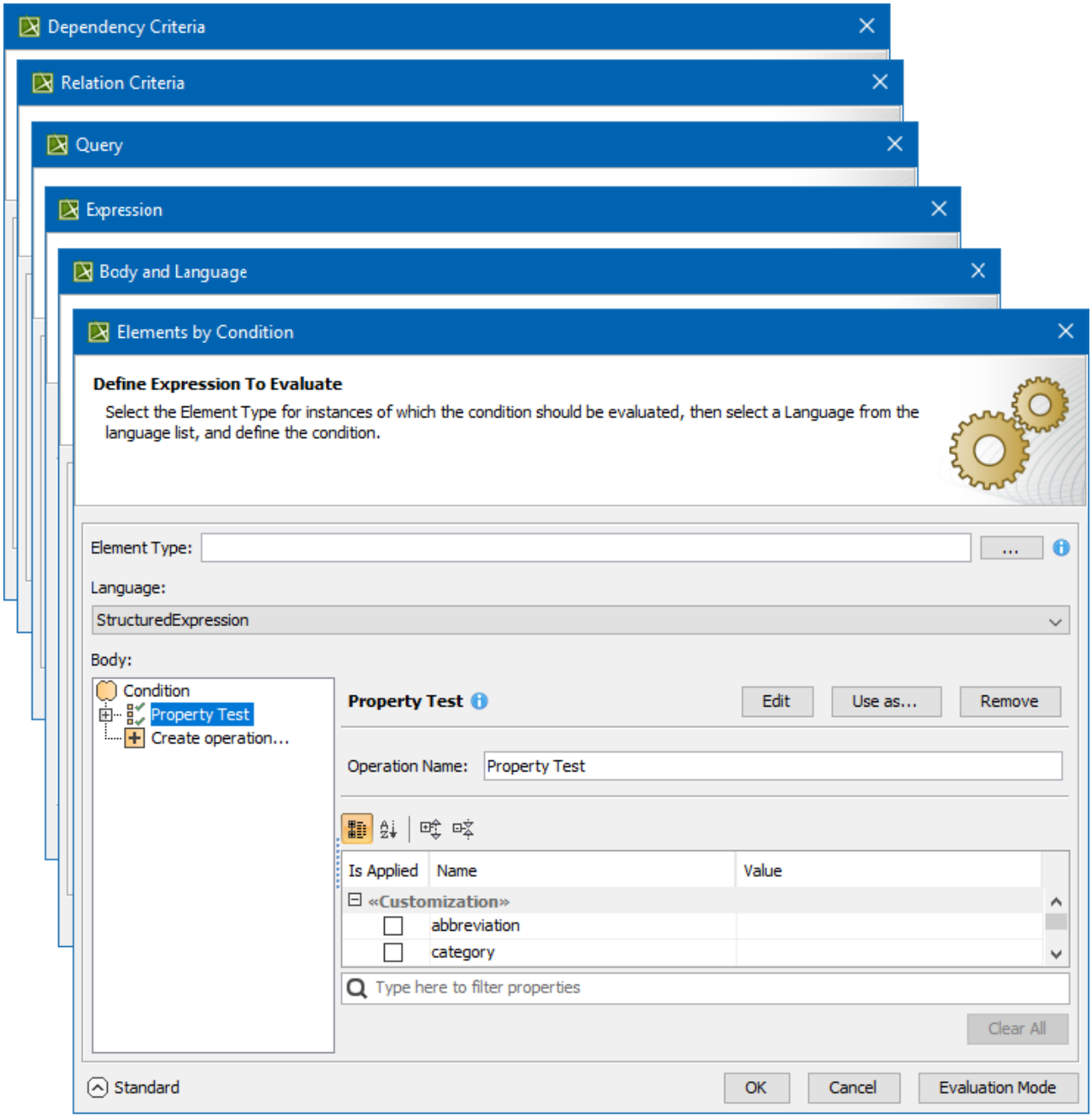
The dialog for specifying criteria can be opened from several different places.

To open the dialog for specifying criteria, do one of the following

---

- In the **Criteria** area of a dependency matrix(see page 334), click the ... button next to the **Dependency Criteria** box. The **Dependency Criteria** dialog opens.
- In the **Criteria** area of a relation map(see page 364), click the ... button next to the **Relation Criteria** box. The **Relation Critera** dialog opens.
- Open the Specification window(see page 913) of a smart package(see page 824), under the **Content** category on the right, click the cell of the **Query** property value, and then click the ... button. The **Query** dialog opens.
- Open the Specification window of a selected element, click Traceability(see page 1201) on the left, and then click **Create** on the right. The **Expression** dialog opens.
- On the toolbar of a generic table, click the **Columns** button, and then select New Derived Property(see page 1224). The **Expression** dialog opens.
- Open the Specification window of an opaque behavior(see page 1265), click the cell of the **Body and Language** property value, and then click the ... button. When the **Body and Language** dialog opens, select **StructuredExpression** from the **Language** drop-down list.
- Open the Specification window of a Legend Item(see page 534), click the specification cell of the **Elements by Condition** property, and then click ... . The **Elements by Condition** dialog opens.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

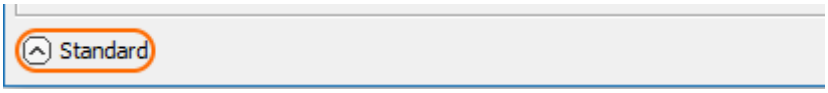


The layout of the dialog is the same in all the described cases. On the left side of the dialog, you can select an operation type that best suits for specifying the needed criterion. Selecting an operation type opens the criteria specification panel on the right side of the dialog.

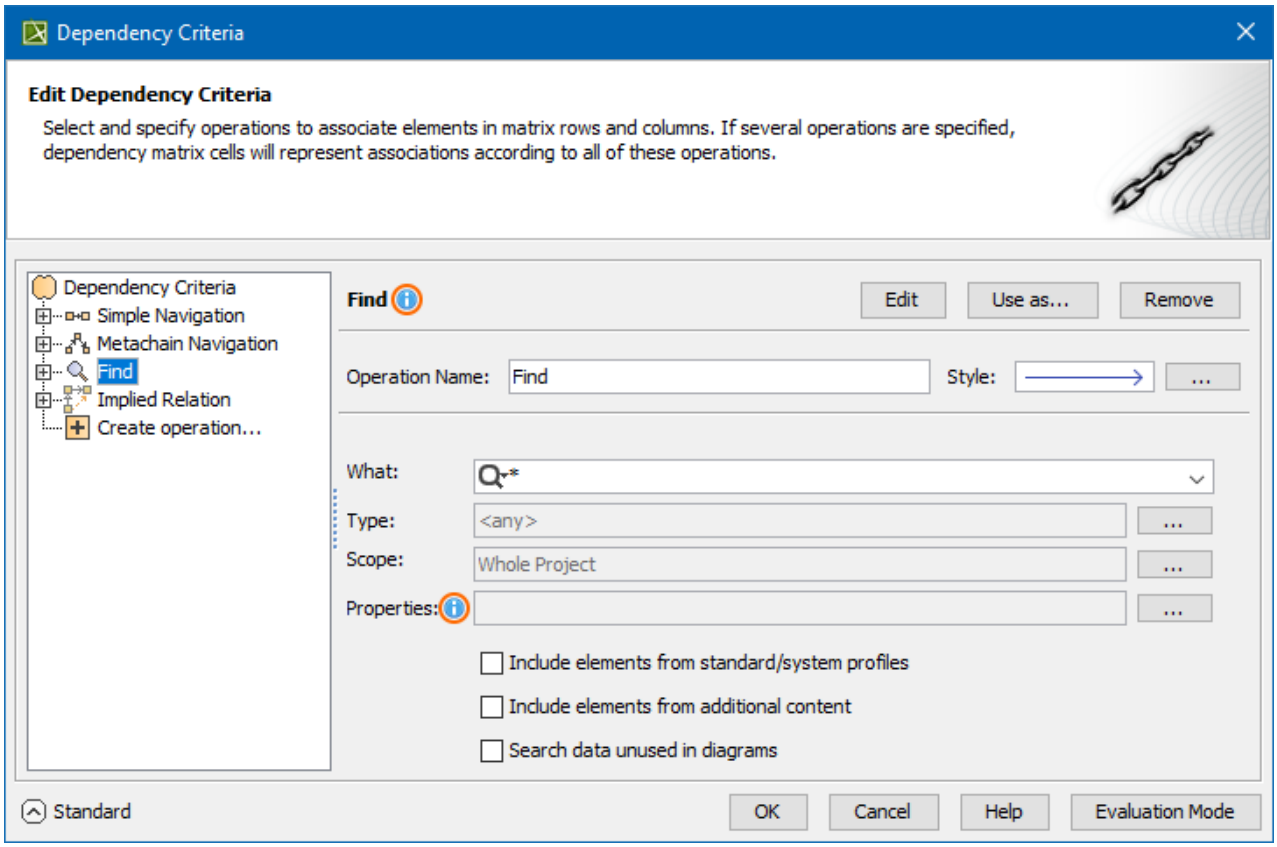
The dialog has two modes: Standard and Expert. *The Expert mode offers more options for specifying criteria and allows for creating combinations of several operations*.

To switch between Standard and Expert modes for specifying criteria

- Click **Expert** at the bottom of the dialog. The dialog is switched to the Expert mode. Click **Standard** to get back to the Standard mode.



Also, be advised that before starting to specify criteria it is useful to read the tips and warnings (if there are any), which appear after the pointer is hovered over the button or the button respectively.



Related pages

- Specifying criteria for querying model(see page 1227)
  - Using Simple Navigation(see page 1231)
  - Using Metachain Navigation(see page 1235)
  - Using Find operation(see page 1239)
  - Using Implied Relations(see page 1242)
  - Creating new operations(see page 1249)
    - Calling operations from the model(see page 1255)
    - Creating script operations(see page 1267)

Using Simple Navigation

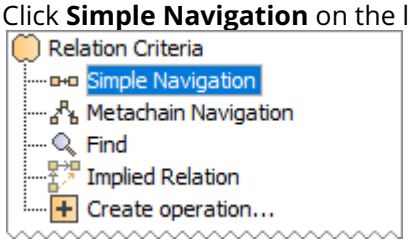
On this page

- Manipulations in Expert mode(see page 1234)

Use the simple navigation operation type, if you need to specify a direct relation between the elements through model relationships, properties, or tags.

To specify a simple navigation as criteria

1. Click **Simple Navigation** on the left side of the dialog.



2. Select a relationship, property, or tag and specify other options described in the following table.

Column	Mode	Description
Relation Criterion	Standard/ Expert	Name of a relationship, property, or tag. The whole list is by default filtered and shows only these relationships, properties, and tags that are available for the Contextual element(see page 1227) type. To see the full list, click to clear the check box below the list.
Is Applied	Standard/ Expert	Select the check box to define the appropriate relationship, property, or tag as criteria for the calculation. To clear all the selected criteria, click the **Clear All** button.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Column	Mode	Description

Direction	Standard/
	Expert
		Select from the drop-down list the direction of the appropriate relationship, property, or tag.
		A source is the Contextual element(see page 1227).
		A target is the result of the calculation.
		
		If the direction is defined for relationships, then
		
		• The **Source To Target** direction means that only the outgoing relations, which are pointing from
		the source element to the target element, will be treated as a result of this criterion.
		
		• The **Target To Source** direction means that only the incoming relations, which are pointing from the target element to the source element, will be treated
		as a result of this criterion.
		
		• If the **Both** direction is chosen, the both above described cases will be treated as a valid result.
		
		If the direction is defined for properties or tags, then
		• The **Source To Target** direction means that only the properties, which exist in the source element, will be treated as a result of this criterion.
		• The **Target To Source** direction means that only the properties, which exist in the target element, will be treated as a result of this criterion.
		• If the **Both** direction is chosen, the both above described cases will be treated as a valid result.

Include Subtypes	Expert	Select the check box to include subtypes of the selected relationship.
Filter by Property Value	Expert	Click the button and in the open dialog select one or more properties of the selected relationship and specify their values for more specific filter.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

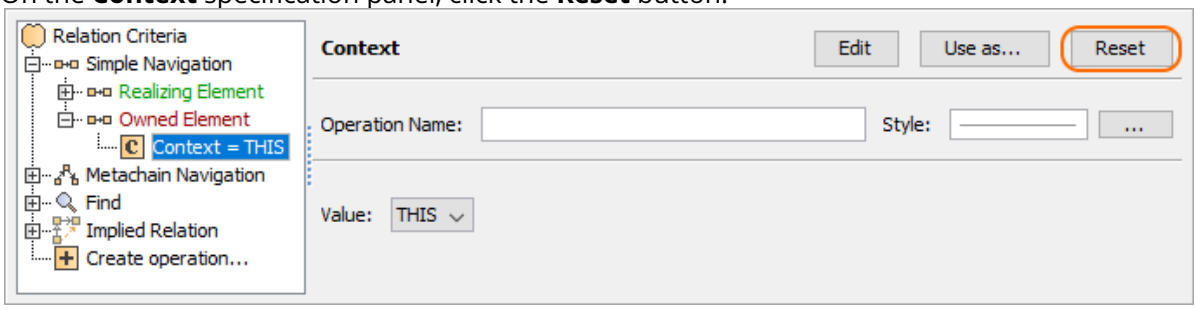
Column	Mode	Description
Result Type	Expert	Click the button and in the open dialog select one or more result element types. The column is available only in the Query dialog and the Expression dialog. You can define element types(see page 335) for representation on a dependency matrix and. Moreover, you can define the element types(see page 366) for representation on a relation map.
Color	Standard/ Expert	Click the button and in the open dialog select the color to represent the appropriate relationship, property, or tag. The column is available only in the **Relation Criterion** dialog.

Manipulations in Expert mode

The Expert mode allows for changing the Contextual element(see page 1227) of a simple navigation criterion as well as creating a new operation, where the result of the simple navigation operation is used as an argument.

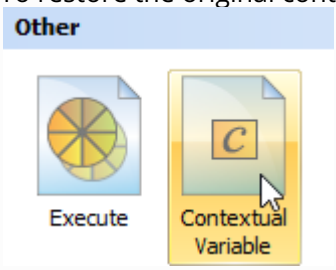
To change the Contextual element(see page 1227) of a simple navigation criterion

1. On the left side of the dialog, expand the simple navigation criterion and select the **Context** parameter.
2. On the **Context** specification panel, click the **Reset** button.



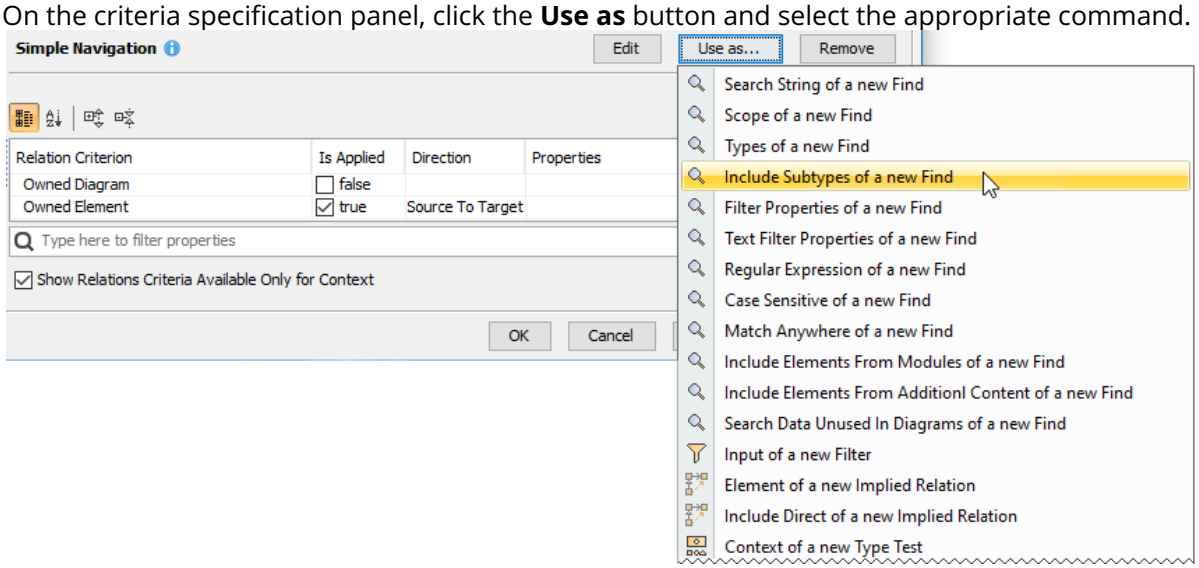
   The value of the **Context** parameter becomes null.
3. Select a new context from the list of operations and element types.

To restore the original context, expand the **Other** group and click **Contextual Variable**.



To use the expression of a simple navigation operation as a parameter of another operation

1. On the left side of the dialog, click **Simple Navigation**.
2. On the criteria specification panel, click the **Use as** button and select the appropriate command.



3. Specify other parameters of the new operation.

Related pages

- Specifying criteria for querying model(see page 1227)
  - Getting started with specifying criteria(see page 1228)
  - Using Metachain Navigation(see page 1235)
  - Using Find operation(see page 1239)
  - Using Implied Relations(see page 1242)
  - Creating new operations(see page 1249)
    - Calling operations from the model(see page 1255)
    - Creating script operations(see page 1267)

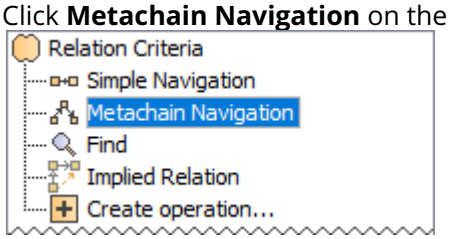
Using Metachain Navigation

Metachain operation is one of the operation types used to specify criteria for querying models. Use the **Metachain Navigation** operation type when specifying indirect (multi-level) relations between elements through the chains of properties.

> **Note**
> Metachains allow you to search for indirectly related elements, although they do not support loops and recursive relations.

How to specify a Metachain Navigation as criterion

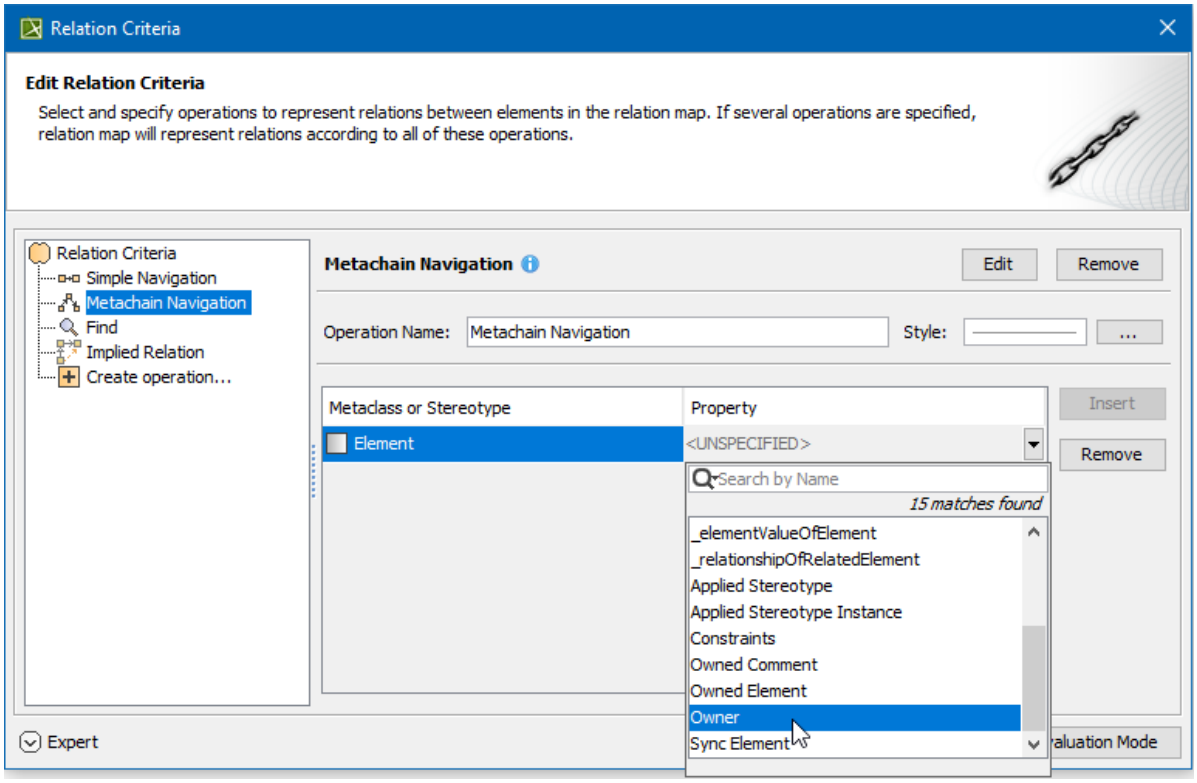
1. Click **Metachain Navigation** on the left side of the dialog box.



2. Change the name of the operation, if needed.
3. Click ... to set a style that represents the specified metachain navigation symbol.
4. Click the **Insert** button to create a new step of the metachain. A new row is added to the list in which you need to specify a metaclass or stereotype and a property for a metachain navigation as shown in the following table:

Column	Description
Metaclass or Stereotype	Either a metaclass or a stereotype of a metachain step (see Figure 2). The first step, when created, already has a metaclass or a stereotype selected by default. This value is the context element type. The default metaclass or stereotype of the first step can be changed. The second step and all further steps are more specific: the values suggested for selection are limited according to the values selected in the first step, since you will not be able to add a new step until the current step is not specified.
Property	A property of the selected metaclass or stereotype.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



5. Repeat the previous step to create as many steps of the metachain as you need for specifying the criteria.

> **Note**
> To remove a step from the metachain, click the **Remove** button.

> **Information**
> You can add a new metachain operation(see page 1249).

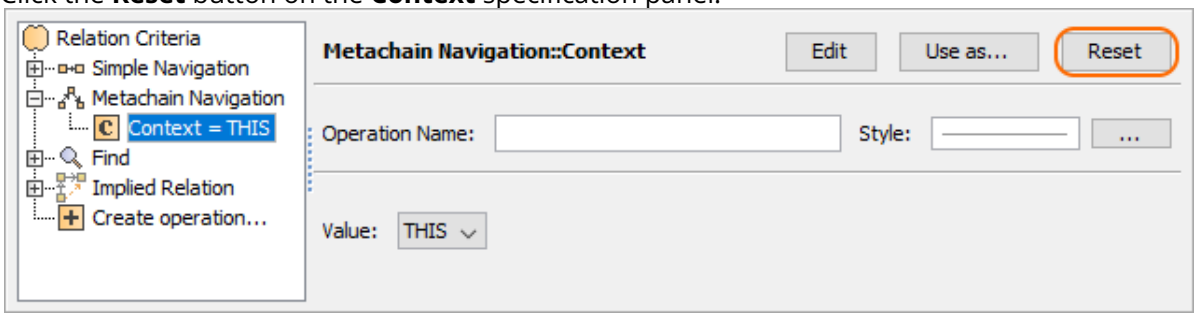
Manipulations in Expert Mode

The **Expert** mode allows you to change the Contextual element(see page 1227) of a **Metachain Navigation** operation, and to create a new operation in which the result of the **Metachain Navigation** operation is used as an argument.

How to change the Contextual element of a Metachain Navigation criterion

1. On the left side of the dialog box, expand the **Metachain Navigation** criterion and select the **Context** parameter.

2. Click the **Reset** button on the **Context** specification panel.

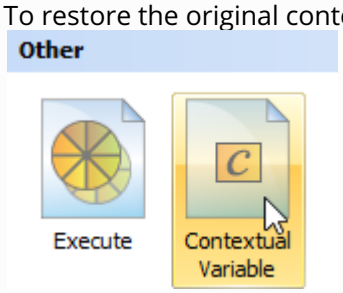


The value of the **Context** parameter becomes null.

3. Select a new context from the list of operations and element types.

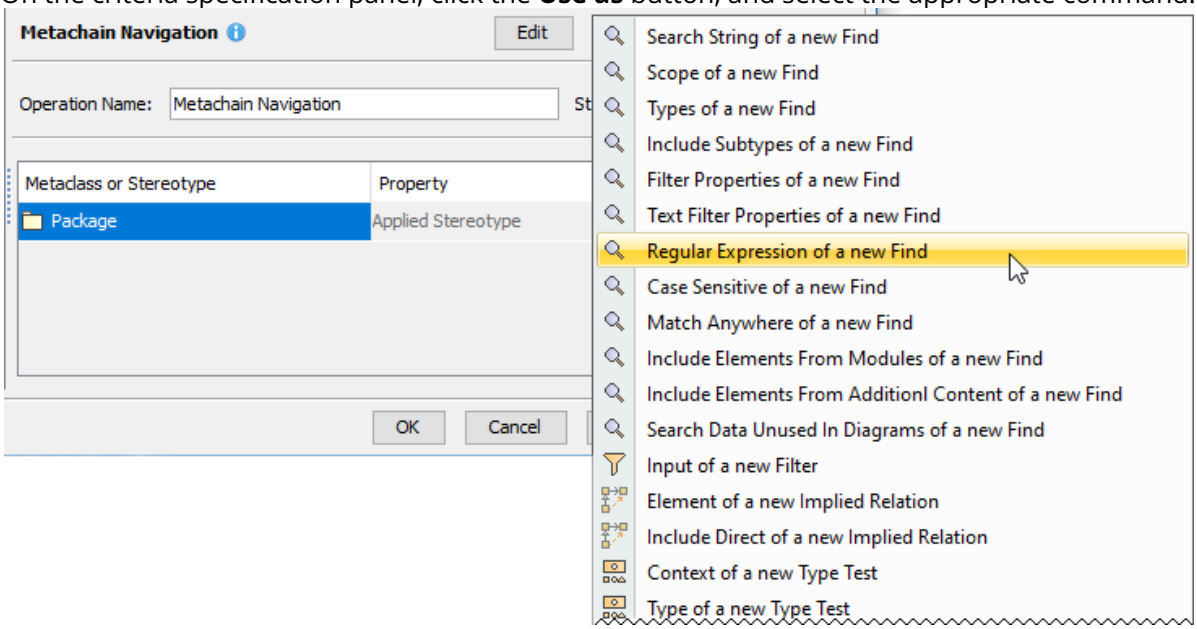
Information

To restore the original context, expand the **Other** group, and click **Contextual Variable**.



How to use the expression of a Metachain Navigation operation as a parameter of another operation

1. On the left side of the dialog box, Click **Metachain Navigation**.
2. On the criteria specification panel, click the **Use as** button, and select the appropriate command.



3. Specify other parameters of the new operation.

Related pages

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Specifying criteria for querying model(see page 1227)
  - Getting started with specifying criteria(see page 1228)
  - Using Simple Navigation(see page 1231)
  - Using Find operation(see page 1239)
  - Using Implied Relations(see page 1242)
  - Creating new operations(see page 1249)
    - Calling operations from the model(see page 1255)
    - Creating script operations(see page 1267)

Using Find operation

On this page

- Specifying a Find operation as criteria(see page 1239)
- Manipulations in the Expert mode(see page 1241)

MagicDraw utilizes a Find operation that allows you to automatically search for model elements, represent relations between these elements, or gather their derived property(see page 1224) values each time you update your model. The Find operation is customizable, so you can change the default values of all the properties for the selected operation at any any time. These properties need to be customized in the Expert mode(see page 1240). The Find operation can be accessed and used:

- In a Relation Map(see page 364) to represent relations between elements.
- In a Dependency Matrix(see page 334) to associate elements in rows and columns of a matrix.
- From the Specification window(see page 913) of a Smart Package(see page 824) to query a model and gather specific elements.
- From the Specification window of the selected element to calculate/gather the values of derived properties.
- In a Generic Table(see page 441) to calculate/gather the values of derived properties.
- From the Specification window of an opaque behavior(see page 1265) to specify the body of a StructuredExpression.

> **Tip**
> If you would like to simply find the desired model element, and don't need to automatically query your model every time you update it, use the **Find** dialog(see page 217) instead.

To use a Find operation, you need to specify it as criteria. The next section describes how to complete this task.

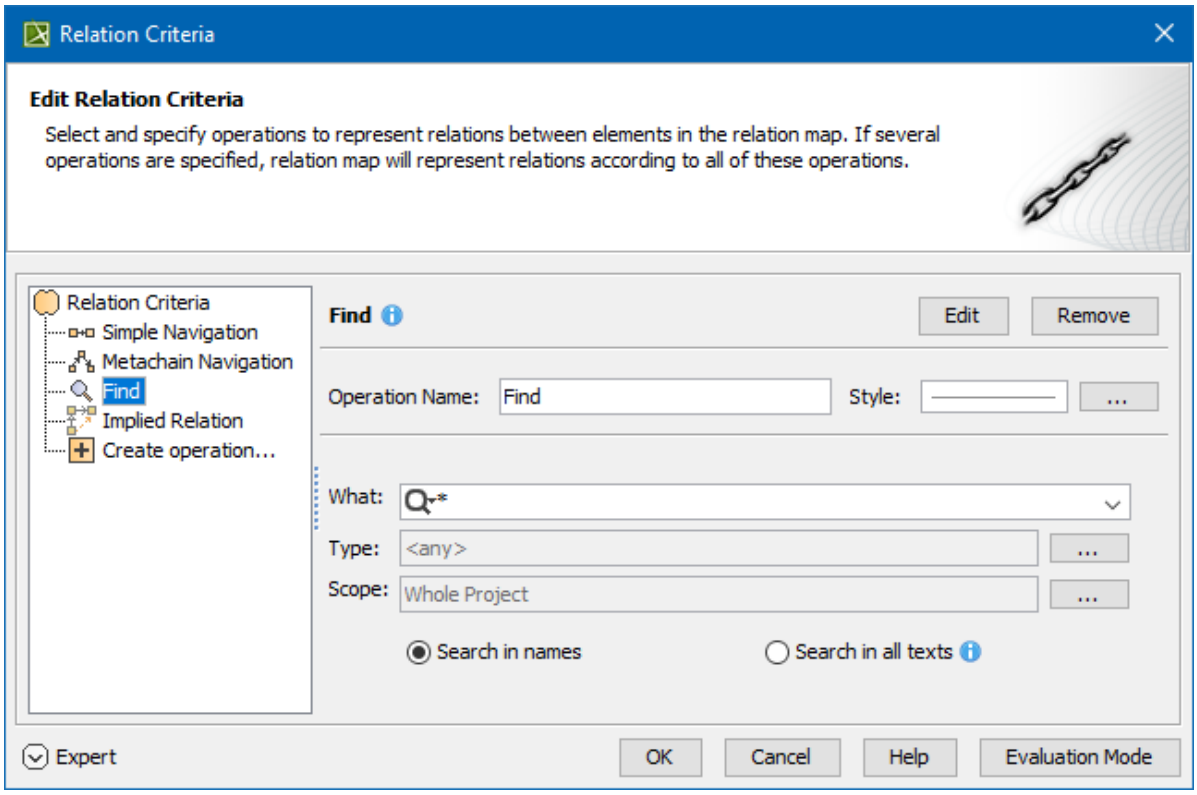
Specifying a Find operation as criteria

Specifying a Find operation as criteria allows you to automatically query your model in order to find and render specific model elements, relations between these elements, or gather their derived properties depending on where the operation is used (as mentioned above).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To specify a Find operation as criteria

1. Open the dialog for specifying criteria(see page 1228).
2. Select **Find** on the left side of the open dialog.



3. Change the **Operation Name**, if needed.
4. Select the **Style** to represent the specified Find criteria. Note that the style selection is available only in the **Relation Criteria** and **Dependency Criteria** dialogs.
5. Specify the Find operation properties(see page 217), and click **OK**.

Useful information
Selecting the **Search in all texts** check box allows you to search for text in Documentation and TODO properties.

Completing the steps described above specifies a Find operation as criteria. If specifying the Find operation properties available in the Standard mode is not sufficient, and you want to customize the advanced properties and/or options of the selected operation, you can do this in the Expert mode.

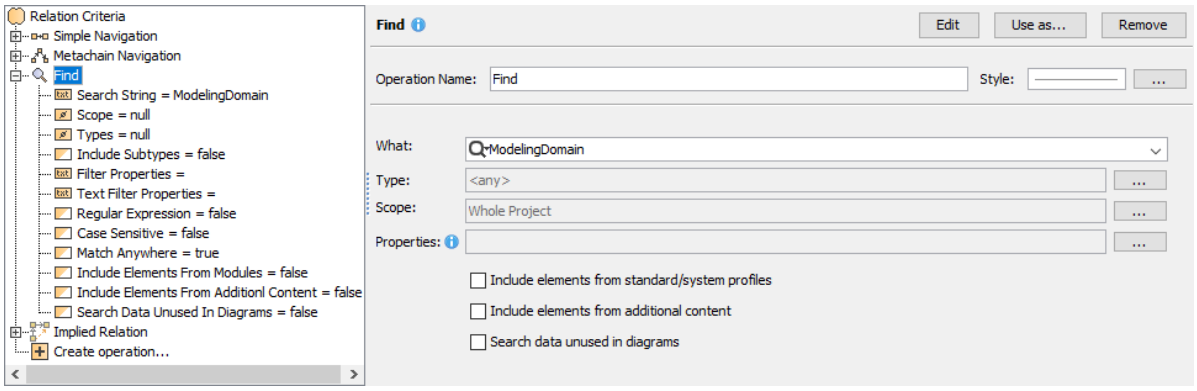
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Manipulations in the Expert mode

The Expert mode allows you to search for model elements by the property or tag value and the applied stereotype name. This mode gives you the option to change the contextual element(see page 1227) of a Find operation or create a new operation, and use the selected Find operation as its argument(see page 1241).

To specify the Find operation properties available in the Expert mode

1. Open the dialog for specifying criteria(see page 1228).
2. On the lower left corner of the open dialog, click Expert to enable the Expert mode.
3. On the left side of the dialog, expand the Find operation.
4. Select the property you want to specify, and, in the property specification panel on the right side of the dialog, specify the value of the selected property.



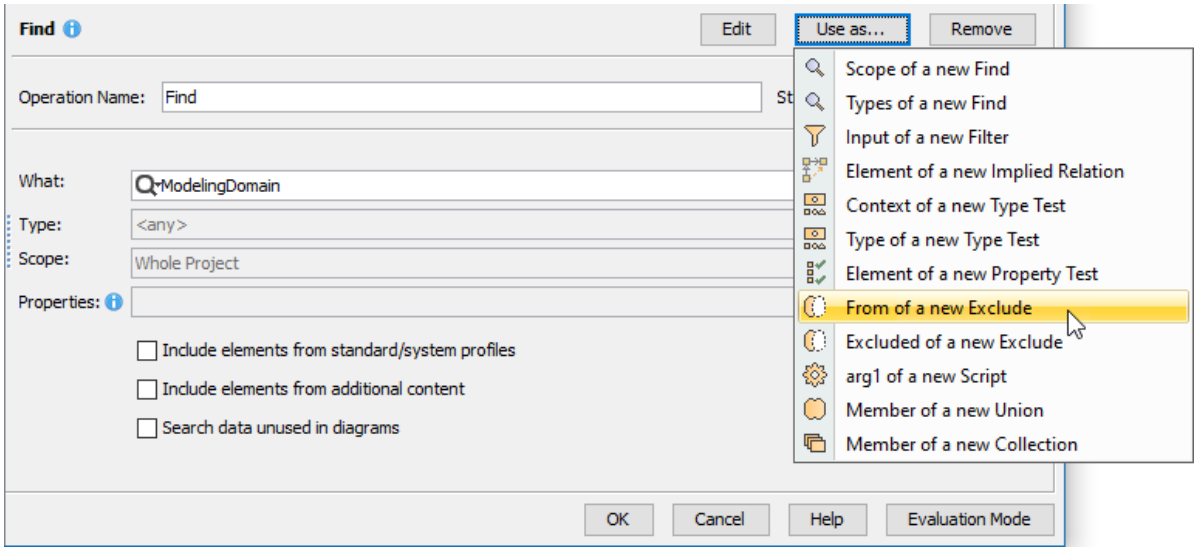
5. Click **OK**.

The selected Find operation is now customized to query a model according to your specified properties. As mentioned before, the Expert mode also gives you the option to use a specific Find operation as an argument of another operation. In other words, you can create a new Find, Filter, Implied Relation, Exclude, Script, Union or Collection operation, and use the results of a specific Find operation as its property.

To use the expression of a Find operation as a property of another operation

1. Open the dialog for specifying criteria(see page 1228).
2. On the lower left corner of the open dialog, click Expert to enable the Expert mode.
3. Select **Find** on the left side of the dialog.
4. In the criteria specification panel on the right side of the dialog, click the **Use as** button, and select the desired command that corresponds to the operation you want to create as shown below. A new operation is created.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



5. Specify other parameters of the created operation, and click **OK**.

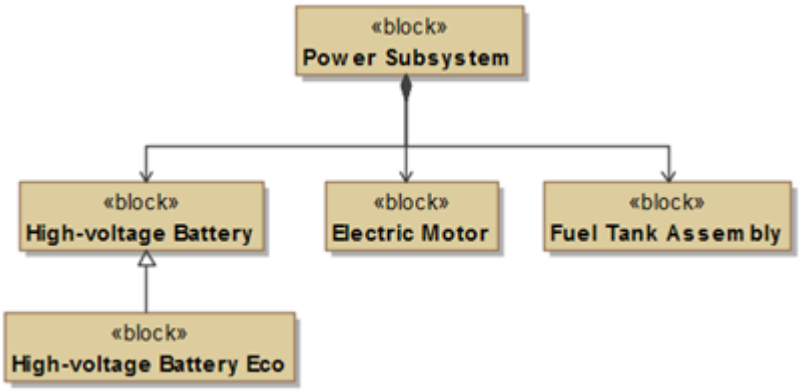
You now have a new operation that has been created with the selected Find operation as one of its properties.

Related pages

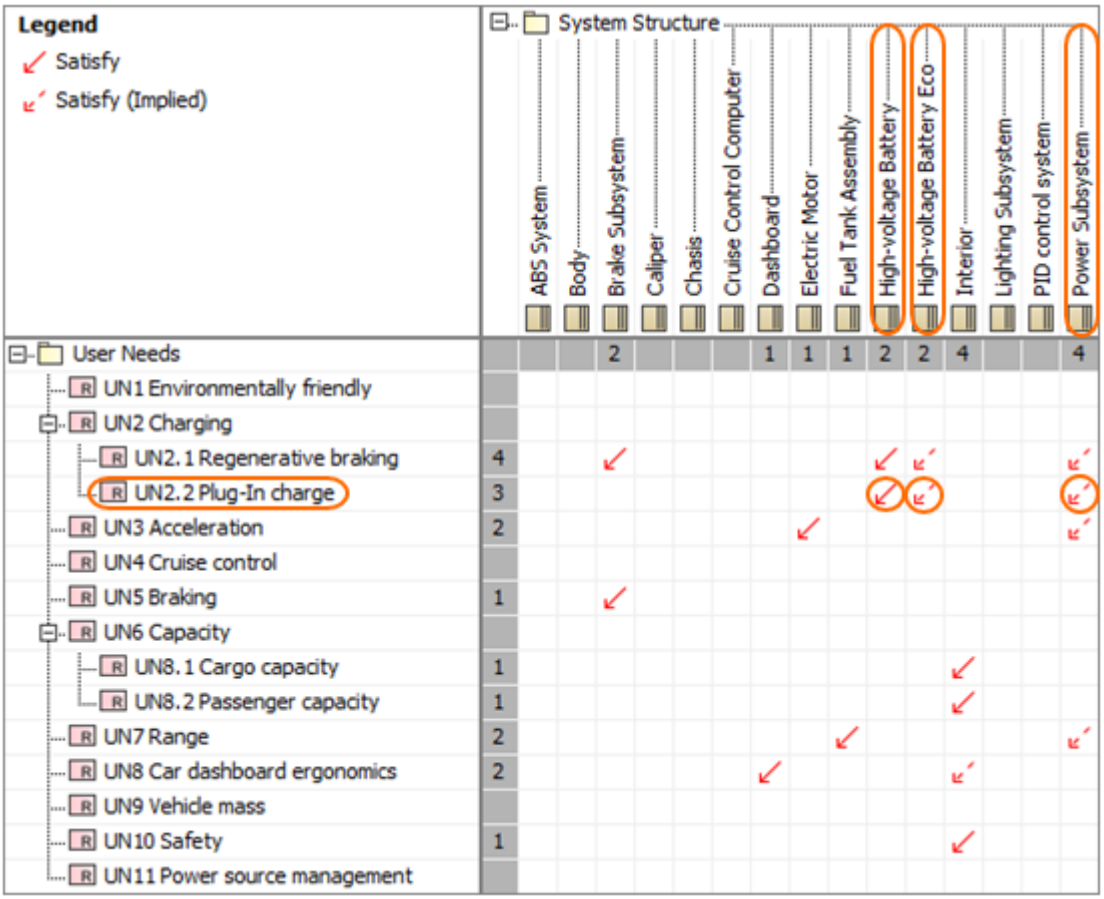
- Specifying criteria for querying model(see page 1227)
  - Getting started with specifying criteria(see page 1228)
  - Using Simple Navigation(see page 1231)
  - Using Metachain Navigation(see page 1235)
  - Using Implied Relations(see page 1242)
  - Creating new operations(see page 1249)
    - Calling operations from the model(see page 1255)
    - Creating script operations(see page 1267)

Using Implied Relations

Use the implied relation operation type if you need to specify indirect relations between the elements when going through a definable hierarchy. Implied relation operation collects indirectly associated elements according to the specified **Source**, **Criteria** and **Target** parameters.



The figure above displays composition and inheritance structure between blocks. If a composite block depends on a particular element, the whole block also depends on this element. The same situation exists with inheritance between blocks. Inherited elements indirectly depend on the elements to which the parent element has relations. These indirect relations are referred to as implied relations, and can be displayed in a dependency matrix, as shown in the figure below.



As shown in the figure, the *High-voltage Battery* block satisfies the *Plug-in charge* requirement. According to the composition and generalization, the *Power SubSystem* and *High-voltage Battery Eco* blocks should

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

satisfy this requirement as well. In the matrix, these relationships are shown as implied, and marked with red, dashed arrows.

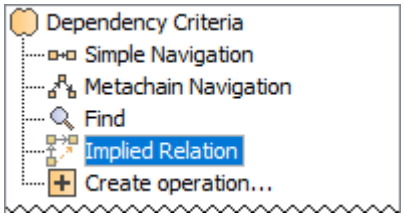
Implied relations can be created in:

- Matrices
- Relation maps
- Tables
- Derived properties
- Smart packages

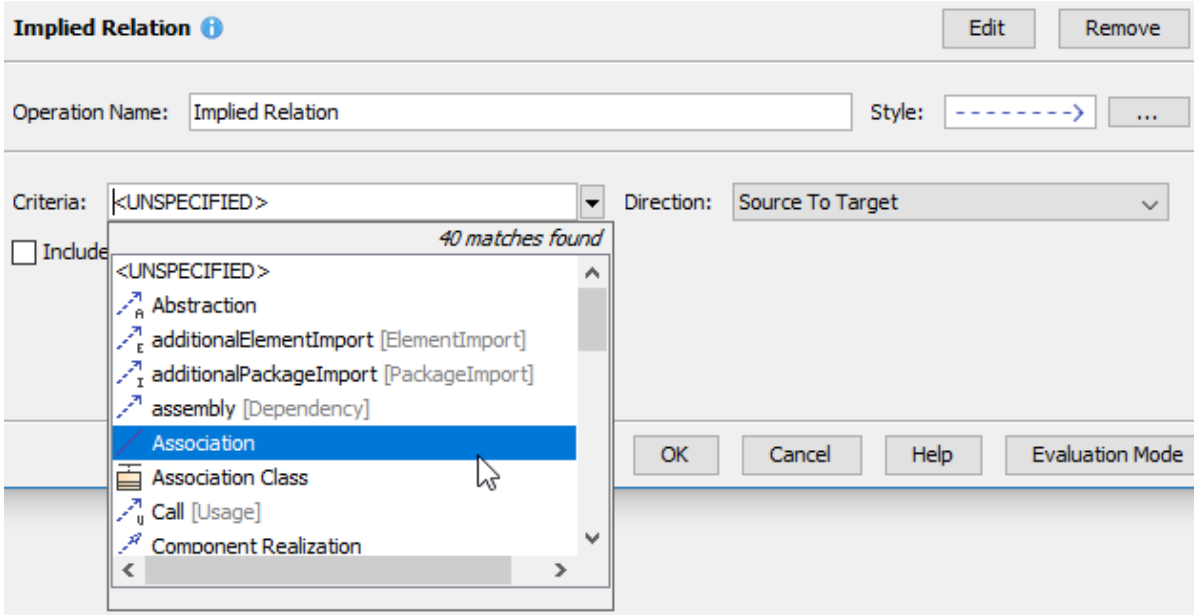
To specify an Implied Relation operation as a criterion

---

1. Open the dialog for specifying criteria(see page 1228).
2. In the criteria list on the left side of the dialog, select the **Implied Relation** operation. The **Implied Relation** criteria specification panel opens.



3. In the **Criteria** drop-down list, select the desired relation.



4. In the **Direction** drop-down list, select the direction of the implied relation:

   - The **Source To Target** direction means that only the outgoing relations, which are pointing from the source element to the target element, are treated as the result of this criterion.

- The **Target To Source** direction means that only the incoming relations, which are pointing from the target element to the source element, are treated as the result of this criterion.
- If the **Both** direction is chosen, both cases described above are treated as a valid result.

> A source is a Contextual element(see page 1227), and a target is the result of the calculation.

5. If you want direct relations to be displayed together with (and the same way as) implied relations, select the **Include Direct Relations** check box. However, be aware that after selecting this check box, there is no way to visually distinguish direct relations from implied relations.
6. If the implied relation is created in a matrix or relation map, optionally modify the style of the relation by clicking ... next to the **Style** box to open the **Style** dialog, then select preferred style properties and click **OK**.
7. Click **OK**.

Criteria can be other relations than those listed in the **Criteria** drop-down list. In the **Expert** mode, any operation or even a combination of operations can be set as a criterion, e.g., metachain or find operation. In the **Standard** mode, an implied relation operation goes through a predefined hierarchy (a composition and generalization) to collect indirectly associated elements, whereas in the **Expert** mode, you can define your own hierarchy.

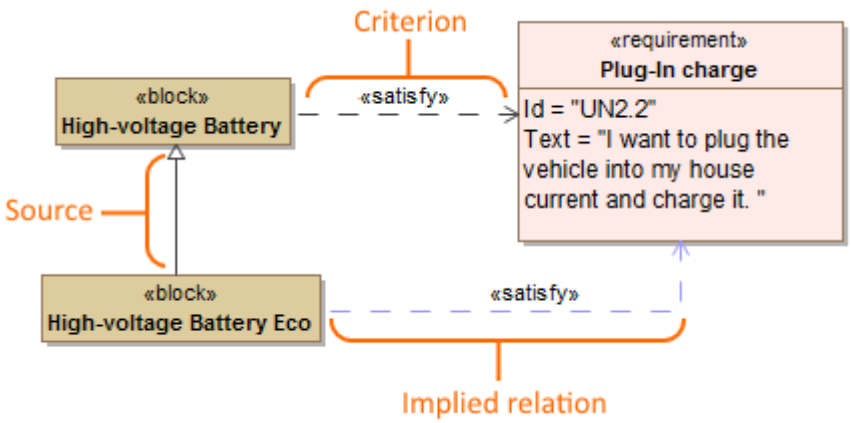
Manipulations in the Expert mode

The **Expert** mode allows you to change the **Criteria**, **Source,** and **Target** parameters of an implied relation operation. Specifying the **Source** and **Target** parameters lets you define your own hierarchy. You also have more flexibility in setting criteria with the ability to choose from a larger scope of operations, or even combine a few of them.

Implied relation operation goes through the **Source** parameter value to recursively collect elements, and uses the **Criteria** parameter value to navigate from the **Source** parameter results to implied elements. In addition, the **Target** parameter value recursively collects implied elements starting from the results of the **Criteria** parameter. The results of this operation are the results of the **Criteria** and **Target** parameters.

The parameters that can be specified in the **Expert** mode:

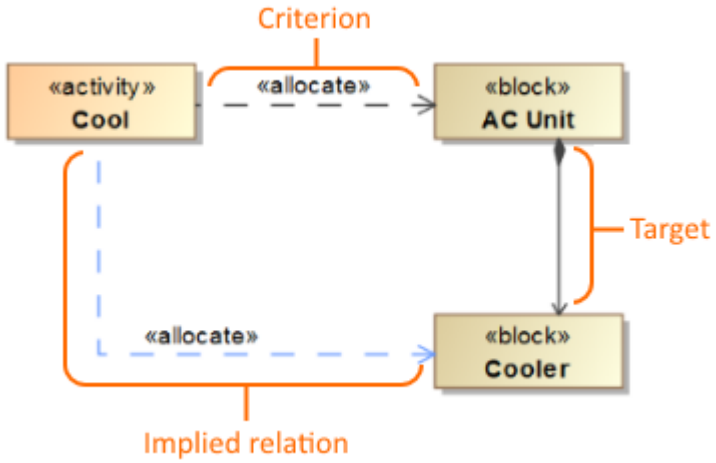
- **Source**: specifies the operation that recursively collects elements starting from the Contextual element.



In the example above, the value of the **Source** parameter is a generalization (direction – **Source To Target**). This means that starting from the *High-voltage Battery Eco* block, all generic classifiers

are collected. In this case, the *High-voltage Battery* block is the result of the **Source** parameter. By default, the values of the **Source** parameter are simple navigation operations: a generalization (direction – **Source To Target**) and part (direction - **Source To Target**). Essentially, when a generalization is selected as the value of the **Source** parameter (direction - **Source To Target**), all general classifiers are recursively collected. And, when the value of the **Source** parameter is a part (direction - **Source To Target**), all composed part types (blocks) are recursively collected.

- **Criteria**: specifies the operation that navigates to the elements from the results of the **Source** pa rameter. In other words, the value of the **Criteria** parameter is the relation for which implied elements are calculated. In the example above, the value of the **Criteria** parameter is a satisfy (direction – **Source To Target**). It means that starting from the result of the **Source** parameter - the *High-voltage Battery* block, the operation navigates through satisfy relation to collect implied elements. In this case, an implied element is the *Plug-In charge* requirement. Note that, by default, the direction of the criteria is **Source To Target**, and that sub-types are also included.
- **Target**: specifies the operation that recursively collects elements starting from the result of the **C riteria** parameter.



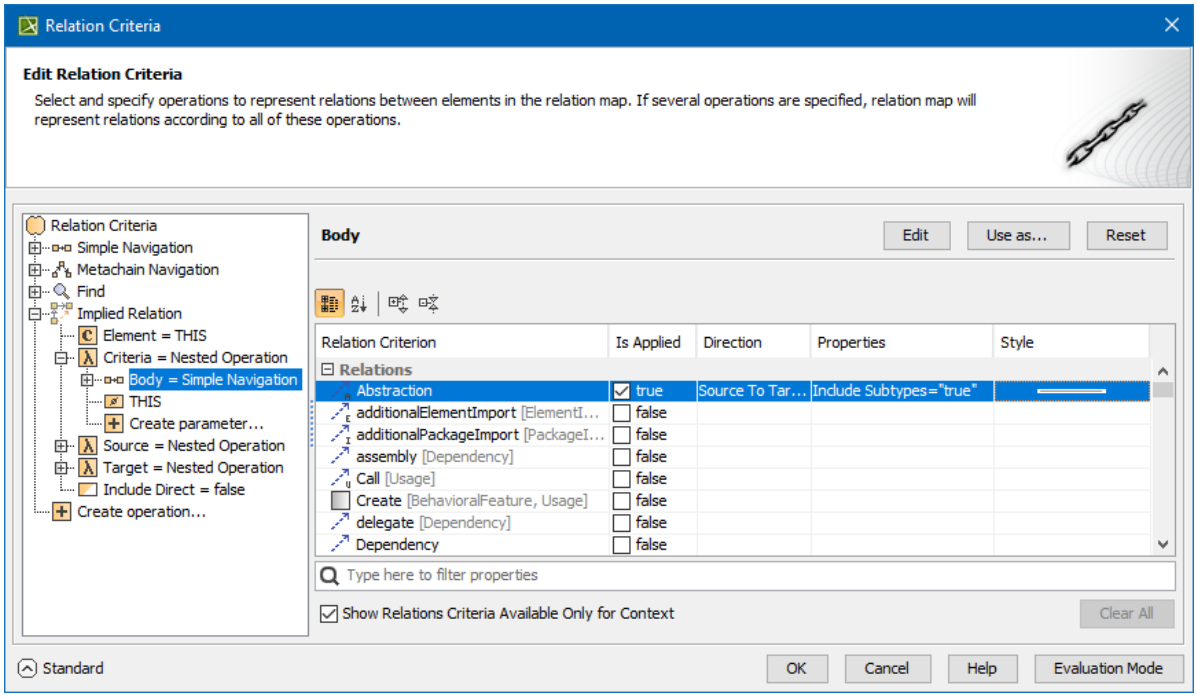
In the example above, the value of the **Target** parameter is a generalization (direction – **Target To Source**). This means that starting from the *AC Unit* block, all composed blocks are collected. In this case, the *Cooler* block is the result of the **Target** parameter. By default, the values of the **Target** parameter are simple navigation operations - a generalization (direction - **Target To Source**) and part (direction - **Target To Source**).

To specify implied relation parameters in the **Expert** mode

---

1. Once you select implied relation as a criterion, in the lower left corner of the dialog for specifying criteria, click **Expert** to enable the **Expert** mode.
2. In the criteria list on the left side of the dialog, expand the implied relation operation, and select the parameter you want to specify.
3. In the open pane for specifying the parameter, select the value you want to use.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



4. When you finish specifying required parameters, click **OK**.

> ⓘ To set the value of the selected parameter as **null**, click the Reset button.

Related pages

- Specifying criteria for querying model(see page 1227)
  - Getting started with specifying criteria(see page 1228)
  - Using Simple Navigation(see page 1231)
  - Using Metachain Navigation(see page 1235)
  - Using Find operation(see page 1239)
  - Creating new operations(see page 1249)
    - Calling operations from the model(see page 1255)
    - Creating script operations(see page 1267)

Using Property Test operation

On this page

- Manipulations in the Expert mode(see page 1248)

A Property Test operation checks if model elements contain the selected properties with the specified values. If the property values of a model element match the specified property values, the operation returns *true*, and if they do not match - *false.*

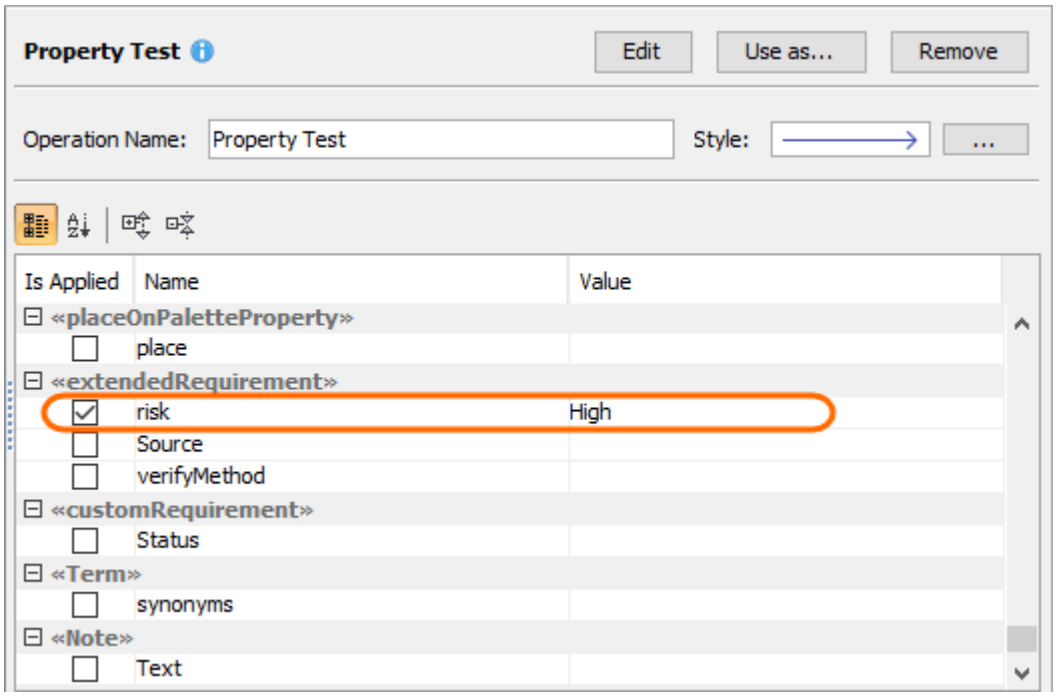
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To specify a Property Test operation

1. Open a dialog for querying model(see page 1228).
2. From the operation list on the left side of the dialog, select the **Property Test** operation. If Property Test operation node is not displayed, select **Create operation**, and then select **Property Test** in the **Operations** area on the right side of the dialog. The **Property Test** panel appears.

   > **Important**
   > To create a new Property Test operation, you must enable the Expert mode.

3. From the property list, select the desired properties and specify their values, as shown below.



4. Click **OK** and close the dialog for querying model if it is still open.

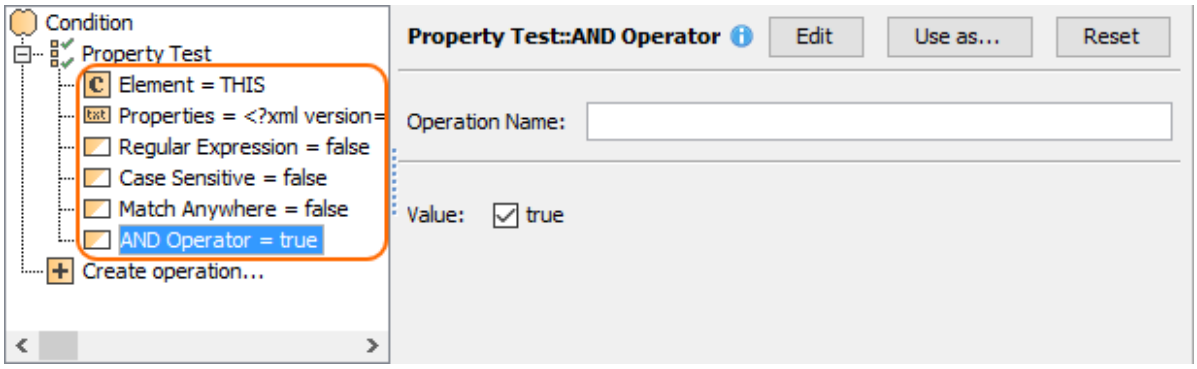
After specifying a Property Test operation, it checks if model elements contain the property values that you specified in the step 3. For example, if you select the **risk** property and set it to **High**, as highlighted in the figure above, the Property Test operation returns *true* for all model elements with the **risk** property set to **High**. Advanced operation properties can be specified in the Expert mode(see page 1247).

Manipulations in the Expert mode

The Expert mode allows you to specify advanced Property Test operation properties.

To specify a Property Test operation in the Expert mode

1. Open the dialog for querying model(see page 1228).
2. Click Expert to enable the Expert mode if it is not already enabled.
3. In the operation list on the left side of the dialog, expand the **Property Test** node.
4. Select the desired property from the list under the **Property Test** node highlighted in the figure below, then specify its value in the specification panel on the right side of the dialog.



5. Click **OK** and close the dialog for querying model if it is still open.

> **Property information**
> If **AND Operator** is set to *true* (default value), the operation returns *true* only for the elements that contain all of the selected properties with specified values. If **AND Operator** is set to *false*, the operation returns *true* for the elements that contain at least one of the selected properties with the specified value.

Related pages

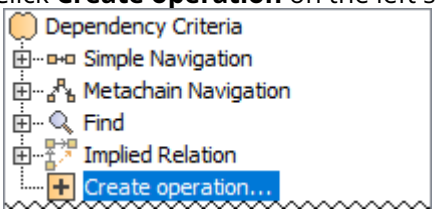
- Specifying criteria for querying model(see page 1227)
  - Getting started with specifying criteria(see page 1228)
  - Using Simple Navigation(see page 1231)
  - Using Metachain Navigation(see page 1235)
  - Using Find operation(see page 1239)
  - Using Implied Relations(see page 1242)
  - Using Property Test operation(see page 1247)
  - Creating new operations(see page 1249)

Creating new operations

On this page

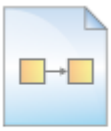
To create a new operation

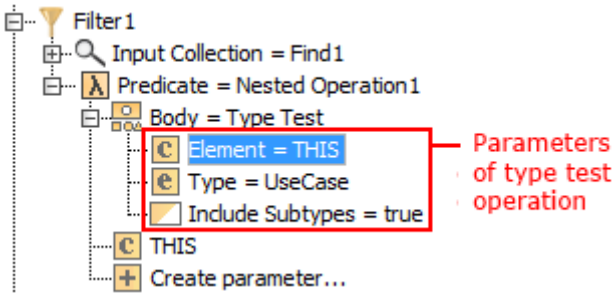
1. Click **Create operation** on the left side of the dialog.

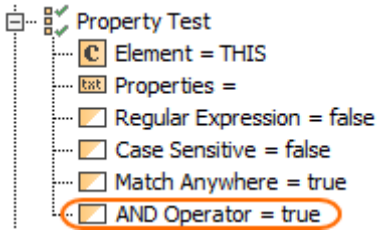
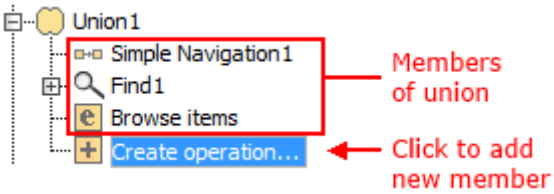


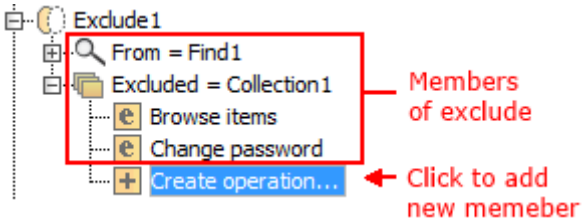
2. Click the appropriate icon on the right side of the dialog to select a new operation type (see the descriptions in following table).

> ⚠ If the Standard mode is on, switch to the Expert mode to make more operation types available.

Icon	Mode	Description
Operations group		
 Simple Navigation	Expert	Click to create a new simple navigation operation(see page 1231). This icon is available only when specifying members of a union or exclude operation, input collection for a filter operation, scope and type parameters for the find operation.
Metachain Navigation	Standard / Expert	Click to create a new metachain navigation operation(see page 1235).
Find	Standard / Expert	Click to create a new find operation(see page 1239).
Filter	Expert	Click to create a new filter operation and then specify both the input collection that you need to filter and the filter criteria.

Icon	Mode	Description
Type Test	Expert	Click to specify a new type test operation as a condition for selected filter operation. In other cases, the icon is not available. The operation tests, whether the type of the element matches the given type or stereotype. If the types matches, it returns *true*, and if they not – *false*. You can also use this operation to check, if the element is an instance of the given classifier. For this operation, you must specify the following parameters: • Element – a model element, which type you need to test. • Type – a type, stereotype, or classifier for testing the element. • Include Subtypes – true, if the inherited types, stereotypes, or classifiers of the selected Type parameter value should be included in the test; false, if not. 

Icon	Mode	Description
Property Test	Expert	Click to create a new Property Test operation. This operation tests model elements to find out if they contain the specified property values. If the property values of a model element match with the ones you specified, the operation returns *true*, otherwise it returns *false.* To specify a Property Test operation, select the desired properties and specify their values in the Property Test specification area on the right side of the dialog. Additionally expand the operation node in the operation list on the left side of the dialog and, if needed, change the **AND Operator** property value. If **AND Operator** is set to *true* (by default), the operation returns *true* only for the elements that contain all of the specified property values. If **AND Operator** is set to *false*, the operation returns *true* for the elements that contain at least one of the specified property values. 
Nested Operation	Expert	Click to specify a new Nested Operation. Nested Operation is an operation, the body of which is a structured expression. It is inline equivalent to defining an executable operation in the model as an Opaque Behavior with the StructuredExpression language and then using it. Nested Operation can be used as an argument to higher order operation calls, such as Filter. In simple operation calls it is not available.
Union	Expert	Click to create a new union operation and then specify as many members for it as you need. 

Icon	Mode	Description
Exclude	Expert	Click to create a new exclude operation and then specify both **From** and **Excluded** members for it. 
Operation from Model	Expert	Click to add a new operation from the model(see page 1255). You can select either one of the built-in operations, which are stored in standard/system profiles of your project, or a custom operation.
Script	Expert	Click to create a new script operation(see page 1267), which language can be one of the following: • BeanShell • Groovy • Jython • JRuby • JavaScript • OCL 2.0 It can also be a reference to a java class (Binary).
Values group		
Element	Expert	Click to select an element from the model.
String	Expert	Click to create a String value.

Icon	Mode	Description
Boolean	Expert	Click to create a Boolean value.
Integer	Expert	Click to create a Integer value.
Null	Expert	Click to create a Null value.
Collection	Expert	Click to create a collection of values.
Other group		
Execute	Expert	Click to add an operation executing a specified expression. Execute operation takes the supplied expression fragment and grafts it into the current expression tree for executing. Expression fragment is an XML string that can be dynamically fetched from any source, such as another tag. The unique value of this operation is that the specified expression can be calculated dynamically instead of being fixed. As a result, the calculation to be executed can depend on the Contextual element(see page 1227).
Contextual Variable	Expert	Click to add an expression retrieving the contextual variable specified by name. Most common case is accessing the THIS variable, which stores the current Contextual element(see page 1227). This is usually the starting point – argument for other operations – such as Simple Navigation or Metachain Navigation.

Related pages

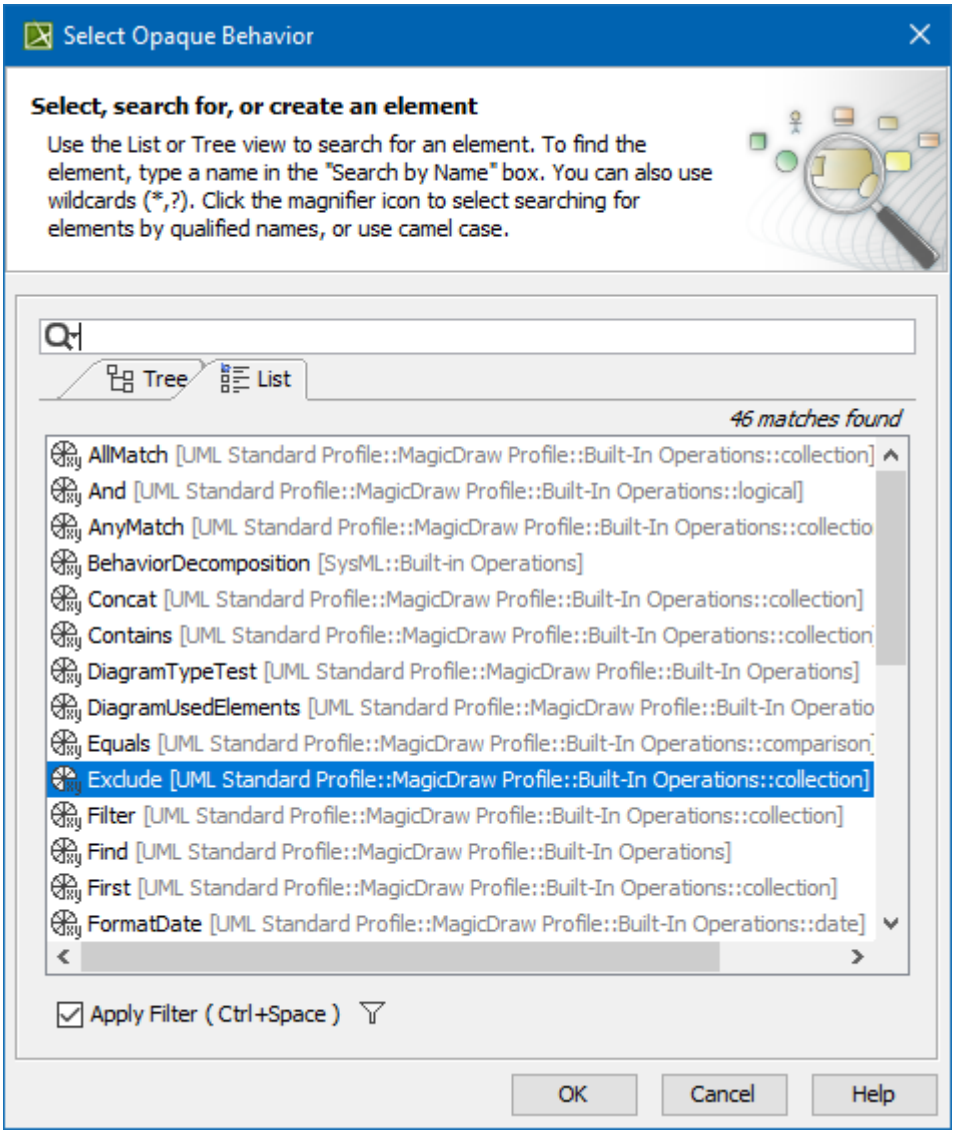
- Specifying criteria for querying model(see page 1227)
- Getting started with specifying criteria(see page 1228)
- Using Simple Navigation(see page 1231)
- Using Metachain Navigation(see page 1235)
- Using Find operation(see page 1239)
- Using Implied Relations(see page 1242)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Calling operations from the model

On this page

In a structured expression, you can call an operation defined in the model. An operation from the model can be either built-in or custom. The built-in operations, such as Union, Find, or Filter, are stored as executable opaque behaviors in the *Built-In Operations* package, which is owned by *UML Standard Profile::MagicDraw Profile*.



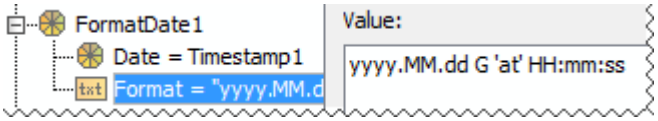
Be sure the **Exclude Used Projects** command on the Filter Options button menu is unchecked!

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

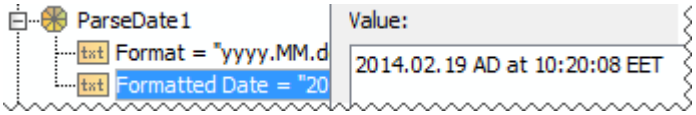
Related pages

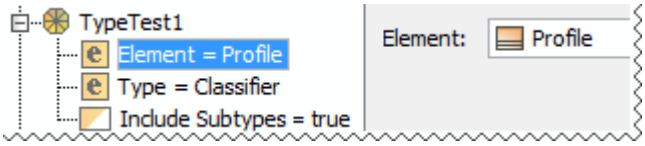
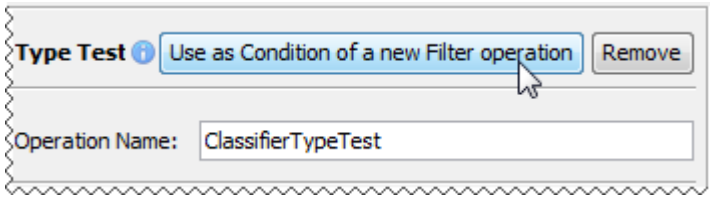
- Creating new operations(see page 1249)
  - Creating script operations(see page 1267)

Built-in operations

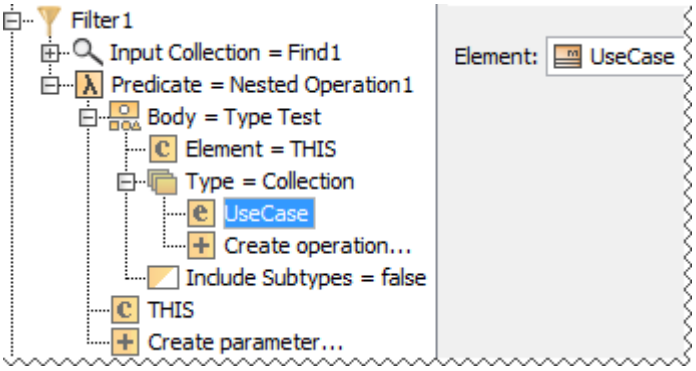
Operation name	Description	Example
Date		
Timestamp	The operation returns current system time in milliseconds as a string type value. It takes no parameters.	Result = “1392798008000”
Format Date	The operation converts a date and time given in milliseconds to a human readable format. The operation takes two parameters: • **Date** – a date and time in milliseconds that should be converted to a human readable format. It must be a string type value and can be the result of a Timestamp operation. • **Format** – a date and time format for the conversion. It must be a string type value. For the date and time formats, refer to the SimpleDateFormat page.  The result of this operation is a string type value.	Date = “1392798008000” Format = “yyyy.MM.dd G 'at' HH:mm:ss z” Result = “2014.02.19 AD at 10:20:08 EET”

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

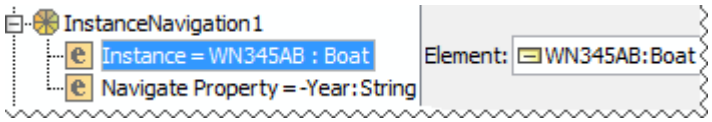
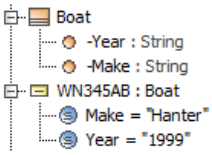
ParseDate	The operation converts a date and time in a human readable format to milliseconds. In other words, the operation reverses the result of the FormatDate operation (it returns the value that can be the Date parameter for a FormatDate operation). The operation takes two parameters: • **Format** – a current format of the date and time that should be converted to milliseconds. It must be a string type value. • **Formatted Date** – a date and time that should be converted to milliseconds. It must be a string type value.  The result of this operation is a string type value.	Format = “yyyy.MM.dd G 'at' HH:mm:ss z” Formatted Date = “2014.02.19 AD at 10:20:08 EET” Result = “1392798008000”
Collection		

TypeTest	The operation tests, whether the type of an element matches a given type or stereotype. If the types matches, it returns *true*, and if they not – *false*. You can also use this operation to check, if an element is an instance of a given classifier. The operation takes three parameters: • **Element** – a model element, whose type you need to test. • **Type** – a type, stereotype, or classifier for testing the element. • **Include Subtypes** – *true*, if the inherited types, stereotypes, or classifiers of the selected Type parameter value should be included in the test; *false*, if not.  The TypeTest operation can be used in the Filter operation as the condition. For this, click the **Use as Condition of a new Filter operation** button on the Type Test operation specification panel.  	Element = class Profile Type = Class Include Subtypes = false Result = true The element type matches the given type. Element = class Profile Type = Classifier Include Subtypes = true Result = true Though the element type does not match the given type, it is its subtype, and in this case subtypes are included in the test. Element = class Profile Type = Classifier Include Subtypes = false Result = false Thought the element type is a subtype of the given type, in this case subtypes are not included in the test.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Filter	The operation analyses given collection of elements (one by one) and returns only those elements that meet the specified filter criteria. The operation takes two parameters: • **Input Collection** – an arbitrary collection of elements of arbitrary types. • **Predicate** – an operation determining filter criteria. Keep in mind that *this operation must have exactly one input parameter and return a result of boolean type!* Otherwise, the Filter operation fails and returns no value. Hence, if you need to use the script operation with many parameters as the filter criteria, you must wrap it inside the nested operation (the TypeTest operation or any other built-in operation is wrapped automatically).  The result of this operation is collection of elements as well.	Input Collection = use case *Log in* use case *Log out* actor *User* use case *Change password* Predicate = TypeTest: Element = Input Collection Type = UseCase Include Subtypes = *false* Result Collection = use case *Log in* use case *Log out* use case *Change password*

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Instance Navigation	The operation finds the needed slot of the given instance and returns one or more values of that slot. The operation takes two parameters: • **Instance** – an instance specification (instance), whose slot value you need to access. • **Navigate Property** – a property, whose slot you need to find.  The result type of this operation matches the type of the property specified as the Navigate Property parameter.	Instance = *WN345AB* Navigate Property = *Year* Result = “1999” 
First	The operation finds the first element in a sequence from a specified scope. The operation consist of: • **Input** - objects list.	
Contains	The operation verifies specified object in given collection and returns Boolean value *true* if sets contains object, otherwise operation returns value *false.* The operation consist of: • **Input** - operation will verify this collections. • **Obj** - sets will be verified according to this object.	Input =
IsEmpty	The operation determine whether collection is empty, if it is empty, operation returns Boolean value *true*, otherwise operation returns value *false.* The operation consist of: • **Input** - specified object (for example array, set, list).	
Size	The operation returns the number of elements in collection. The operation consist of: • **Input** - an object or collection (for example array, set, list, string).	

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Intersect	The operation returns the data common to both collections, with no repetitions and in sorted order. The operation consist of: • **Collection1** and **Collection2** - collections which will be intersected.	C1 = [7 0 5], C2 = [7 1 5], returns C = [5 7];
Concat	The operation joins two lists. The **List1** and **List2** are joined end-to-end. Keeps the order, allows duplicates. The operation consist of: • **List1** and **List2** - lists which will be concatenated.	
Get	The operation returns the element at the specified position in collection. The operation consist of: • **Input** - specified collection. • **Index** - index of the element to return (integer type).	
Map	The operation maps objects from collection to collection of other objects and returns a collection of results in the same order. The operation consist of: • **Input** - collections with objects to map. • **MapOperation** - given function, this function will be applied to each input element.	Input = [1 2 3 4 5], applies operation map with function (Input + 1) returns = [2 3 4 5 6]
MapFlat	The operation maps objects from collection to collection of other objects. Flattens mapping result if it is a collection. The operation consist of: • **Input** - collections with objects to map. • **MapOperation** - given function, this function will be applied to each input element.	Input = { {1,2}, {3,4}, {5,6} } applies operation MapFlat returns = {1,2,3,4,5,6}

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Reduce	The operation reduce collection using given operation. The operation consist of: • **Input** - collection that will be reduced. • **ReduceOperation** - given function, this function determines how collection will be reduced.	
Zip	The operation zips two collections using given zip operation. The operation consist of: • **Collection1** and **Collection2** - to these collections, zip operation will be applied. • **ZipOperation** - given function, according to this function, collections will be zipped.	
Min	The operation returns minimum value from given collection. The operation consist of: • **Input** - collection which will be tested and minimum value determined.	
Max	The operation returns maximum value from given collection. The operation consist of: • **Input** - collection which will be tested and maximum value determined.	
AllMatch	The operation checks if all collection elements match given predicate and returns boolean value. The operation consist of: • **Input** - specified collection. • **Predicate** - a predicate is applied to each element of collection.	

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

AnyMatch	The operation checks if all collection elements match given predicate and returns boolean value. The operation consist of: • **Input** - specified collection. • **Predicate** - a predicate is applied to each element of collection.	
	Logical	
And	The logical conjunction operation returns Boolean value *true* if all parameters are *true*, in all other cases - operation returns *false* value. The operation by default takes three parameters: • **A** and **B** - two logical values. • **Result** - a Boolean value which shows results of logical conjunction	
Or	The logical disjunction operation returns Boolean value *true* if either or both parameters is *true*, otherwise operation returns *false* value. The operation takes these parameters: • **A** and **B** - two logical values.	
Xor	The logical exclusive disjunction operation returns Boolean value *true* if both parameters differ, otherwise operation returns *false* value. The operation takes these parameters: • **A** and **B** - two logical values.	
Not	The logical negation operation returns Boolean value *true* if parameter have value *false*, and *false* when parameter have value *true*. The operation takes one parameter: • **A** - logical value.	
	Comparison	

IfThenElse	The operation returns one or other object depending on condition. The operation consist of: • **Condition** - defines the *operation* that determines which value to assign. • **Then** - *expression* defines the value to assign if *condition* is *true.* • **Else** - *expression* defines the value to assign if *condition* is *false*.	
LessThan	The operation returns *true* if the left parameter is less than the right parameter (**A < B**). The operation consist of: • **A** and **B** - parameters which will be compared.	
LessThanOrEquals	The operation returns *true* if the left side parameter is less than or equal to the right parameter (**A <= B**). The operation consist of: • **A** and **B** - parameters which will be compared.	
Greater Than	The operation returns *true* if the left parameter is greater than the right parameter (**A > B**). The operation consist of: • **A** and **B** - parameters which will be compared	
Greater ThanOr Equals	The operation returns *true* if the left parameter is greater than or equal to the right parameter (**A >= B**). The operation consist of: • **A** and **B** - parameters which will be compared	
Equals	The operation converts parameters if they are not the same type, then compares them. If parameters are equal, operation returns Boolean value *true,* otherwise operation returns *false.* The operation consist of: • **A** and **B** - parameters which will be compared	

NotEquals	The operation converts parameters if they are not the same type, then compares them. If parameters are not equal, operation returns Boolean value true, otherwise operation returns *false.* The operation consist of: • **A** and **B** - parameters which will be compared	
	String	
StringConcat	The operation joins two parameters with string values. A and B string values are joined end-to-end. The operation consist of: • **A** and **B** - two string values.	A = Hello B = world! Return = Hello world!
StringContains	The operation returns Boolean value true, if specified string value is find in specified scope, otherwise. The operation consist of: • **A** - according to this string, operation will verify **B** • **B** - string value.	A = o B = world! Return = *True*
	Other	
DiagramTypeTest	The operation returns Boolean value true if project uses specified diagram, otherwise returns value false. The operation consist of: • **Diagram** - all diagrams in the specified scope. • **Type** - write diagram name exactly how it is named in modeling tool.	Diagram = Use Case Diagram Class Diagram Free Form Diagram Type = Class Diagram Return = *True*

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

On this page

- How many parameters can an opaque behavior have?(see page 1266)
- How to create a parameter for the opaque behavior?(see page 1266)
- How to access the arguments and other values from script body of the opaque behavior?(see page 1267)
- How many statements can a script have?(see page 0)
- What MagicDraw functionality can a script use?(see page 0)

Language of an executable opaque behavior can be OCL 2.0, binary, BeanShell, Groovy, JRuby, JavaScript, Jython, or StructuredExpression. Also, it must have the proper number of parameters with proper types.

How many parameters can an opaque behavior have?
The number of parameters an opaque behavior can have depends on the selected language of the opaque behavior body:

- An OCL 2.0 expression must have a single parameter.
- A binary expression must declare the exact number of parameters has the Java class, to which the expression body of the opaque behavior references.
- Other script expressions, such as JavaScript or Groovy, can have as many parameters as you need.
- Structured expression can have as many parameters as you need.

How to create a parameter for the opaque behavior?

To create a parameter for the opaque behavior

---

1. In the Specification window(see page 913) of the opaque behavior, **Parameters** property group, click the **Create** button. The Specification window(see page 913) of the new parameters opens.
2. Enter a name of the new parameter.
3. Specify the multiplicity of the parameter: either select a value from the drop-down list or type a new one.

   In this case, only the upper bound of the multiplicity is important. For example, the multiplicity [0..1] has the same meaning as the multiplicity [1].

4. Specify the type of the parameter: either select a value from the drop-down list or create a new one by typing directly in the cell.

   Available types:
   - Built-in UML primitives (Integer, Real, String, and so forth)
   - UML element types
   - Java classes (java.io.File, java.util.Properties, and so forth)

5. If the multiplicity upper bound is 1, you may skip this step. Otherwise, specify the following:
   - Whether the arguments are unique.
   - Whether the arguments are ordered.

   If you do not see the Is Unique and Is Ordered properties, select All from the Properties drop-down list in the upper right of the Specification window(see page 913).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

6. Close the Specification window(see page 913).

How to access the arguments and other values from script body of the opaque behavior?

> ⚠ The following instructions applies to BeanShell, Groovy, JRuby, JavaScript, and Jython scripts only.

To access an argument from a script body, you should refer to the corresponding parameter name.

The script body can access the following values:

- Arguments passed to an opaque behavior as parameters.
- Globally defined values:
  - project (current project)
  - application

Related Pages

- Model Elements(see page 628)
- Specification Window(see page 913)

Creating script operations

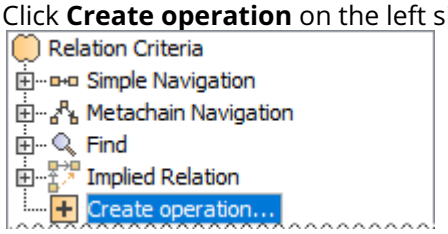
Use the script operation, when you need to specify an inline expression in OCL 2.0, as Java binary, or in any of MagicDraw supported script languages. MagicDraw supports any script language that has a JSR-223-compliant engine.

Adding a script operation directly into a structured expression is the same as defining an operation in the model and then calling it from the structured expression(see page 1255).

To create a script operation

---

1. Click **Create operation** on the left side of the dialog.



2. Click the **Script** icon on the right side of the dialog.

   > ⚠ If you do not see the **Script** icon, switch to the Expert mode.

3. If needed, change the operation name.
4. Click ... to define the **Style** for your script operation.
5. From the **Language** drop-down list, select a language.

> ⓘ If the engine of the language you need is not built into MagicDraw by default, you can add it(see page 2737) to the application yourself. Make sure the Automaton plugin is enabled!

6. Type in the **Body** box. Use the syntax of the selected language.
7. In the **Listeners Configuration** box, type a reference to the java class, which specifies when the result of the script operation must be recalculated.

> ⓘ The listeners configuration is not necessary, but using it can improve the performance. You can create your own listener(see page 2300) using MagicDraw API.

8. Click the **...** button next to the **Results Filter By Type** box and select one or more element types to filter the result of the script operation.

   The script operation is created. Now you can create a new one, where the result of the previously created operation is used as an argument.

Related pages

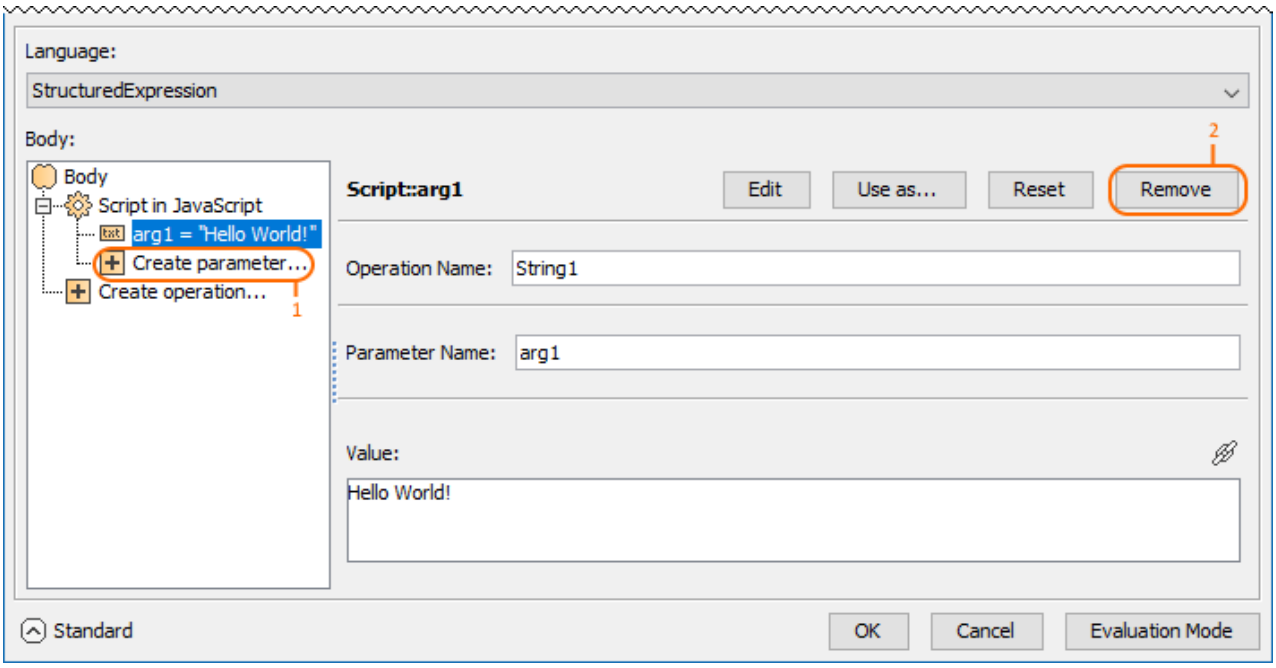
- Creating new operations(see page 1249)
  - Calling operations from the model(see page 1255)

Specifying parameters

The number of parameters a script operation can have depends on the script operation language:

- An OCL expression can have a single parameter, which by default specifies the Contextual element(see page 1227). Though the parameter cannot be removed, you can easily reset the Contextual element(see page 1227), if necessary.
- A binary expression inherits parameters from the Java class to which the expression body references. You cannot create additional parameters.
- Other script operations, such as JavaScript or Groovy, can have as many parameters as you need (see the following figure).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

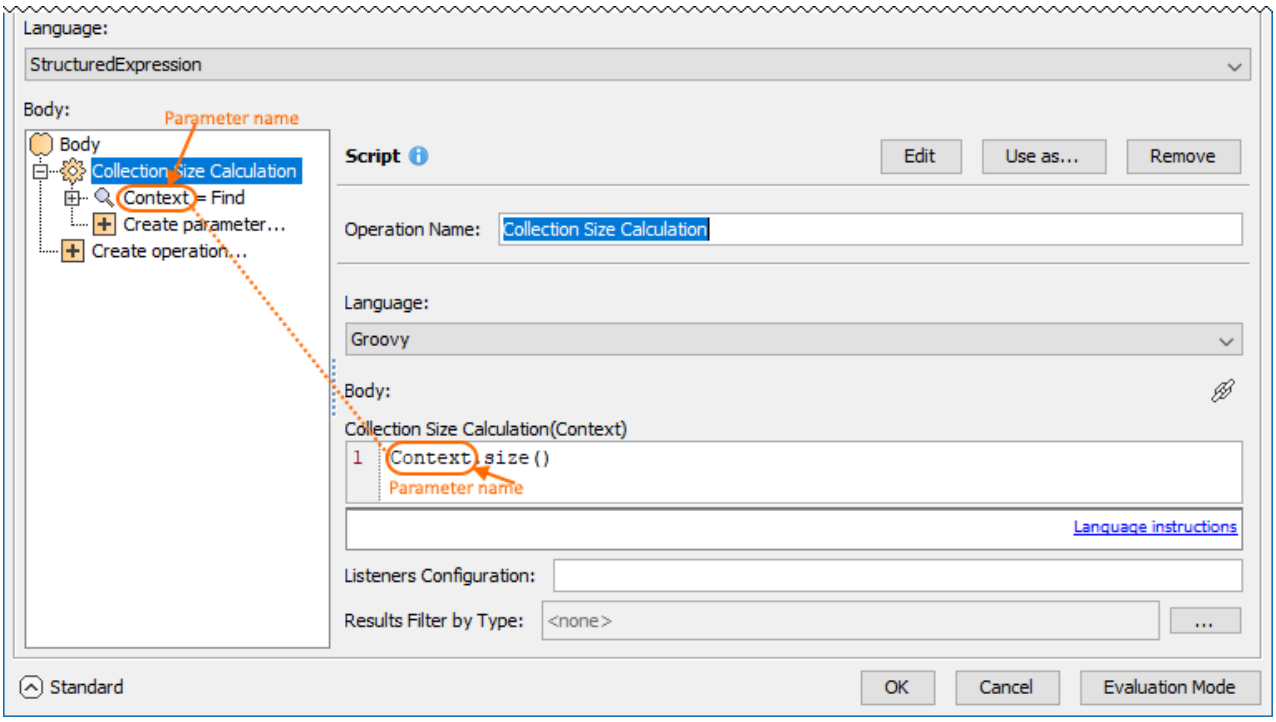


Writing scripts

> The following instructions applies to BeanShell, Groovy, JRuby, JavaScript, and Jython scripts only.

How to access the arguments and other values from the script body?
To access an argument from the script body, you should refer to the corresponding parameter name.

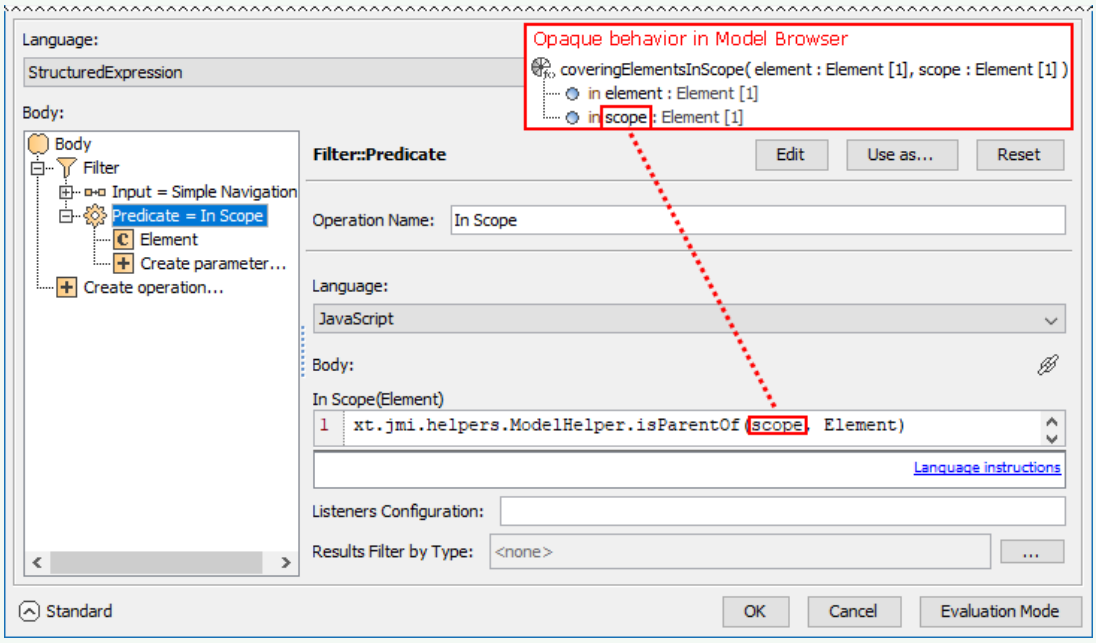
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



A script body can access the following values:

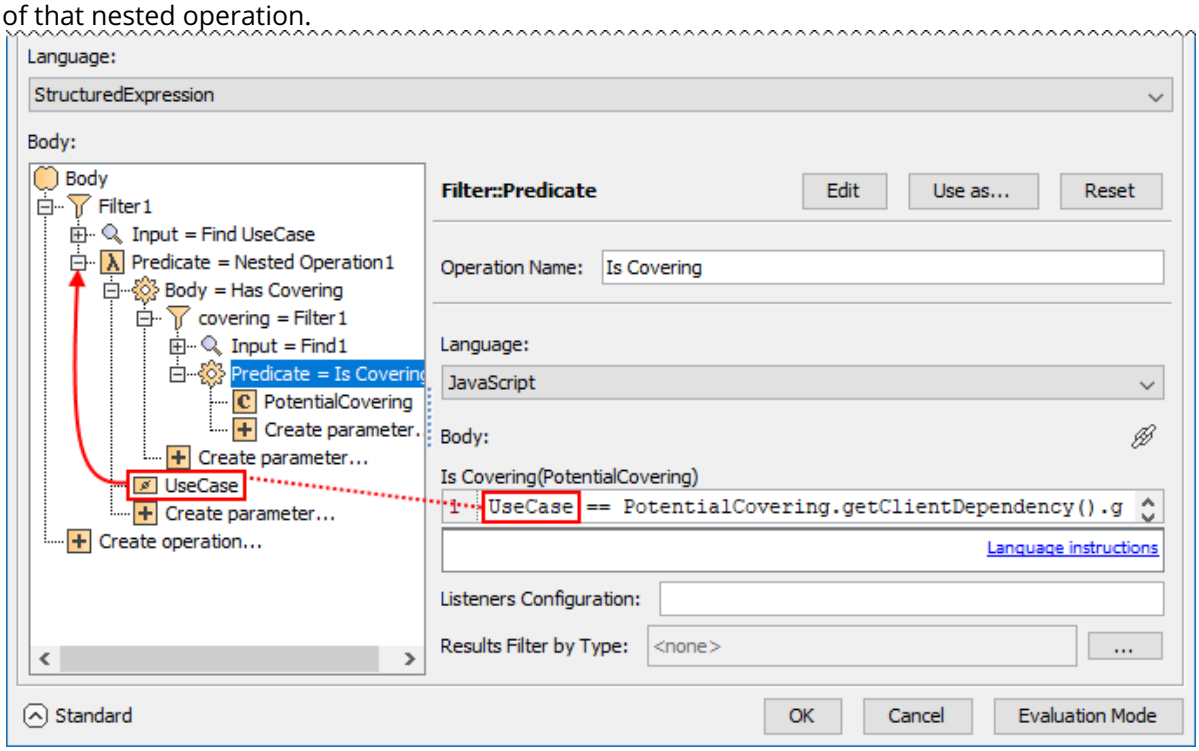
- Arguments passed to this script as parameters, such as the Context parameter in the preceding figure.
- Values passed to the structured expression, inside of which the script operation is defined.

If you have an opaque behavior with a couple of parameters, you can refer to them from a script operation, which is inside of the structured expression - the body of this opaque behavior.



If you have a structured expression specifying a derived property, validation rule, relation criterion, etc., use *THIS* to refer to the Contextual element(see page 1227).

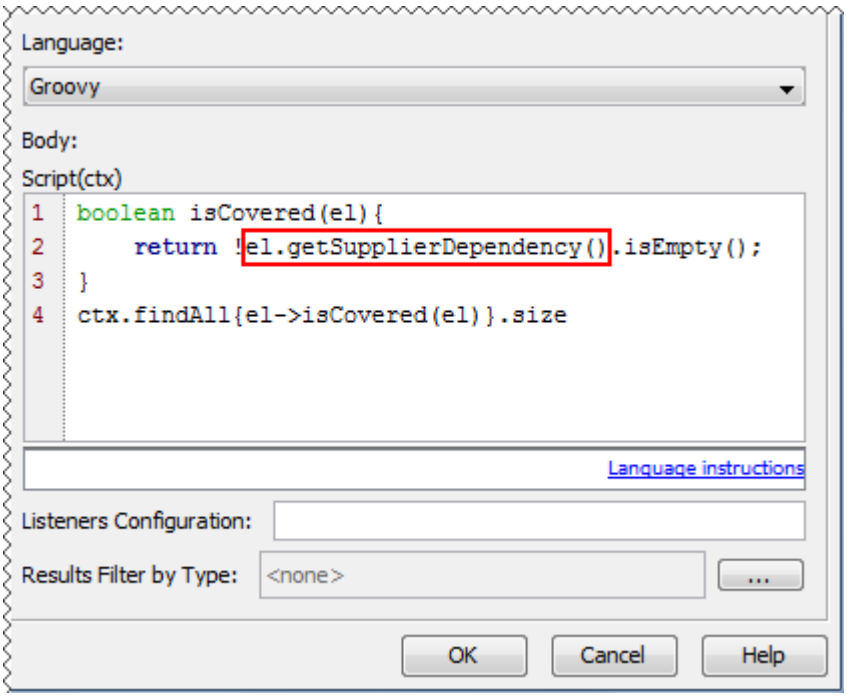
- Arguments passed to a nested operation as parameters, if the script operation is defined inside of that nested operation.



- Globally defined values:
  - project (current project)
  - application

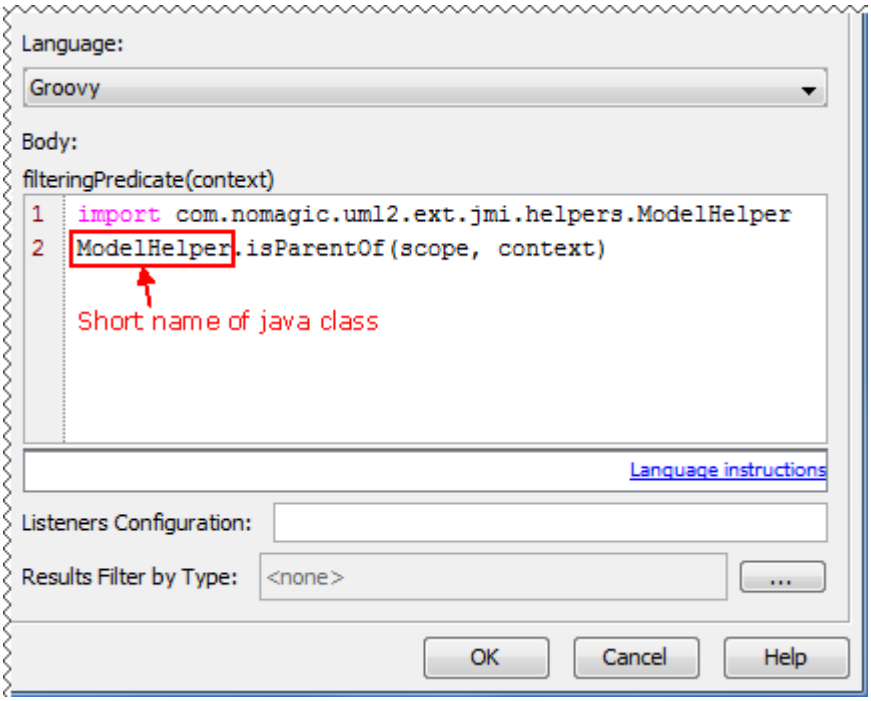
What MagicDraw functionality can a script use?
The script can call MagicDraw Open API.

For the list of available MagicDraw Open API methods(see page 2156), refer to javadoc.zip, which can be found in <MagicDraw installation directory>\openapi\docs.



More complex model access operations are available in *ModelHelper* and *StereotypesHelper*.

Use import statements to shorten java class names as shown in the following figure.

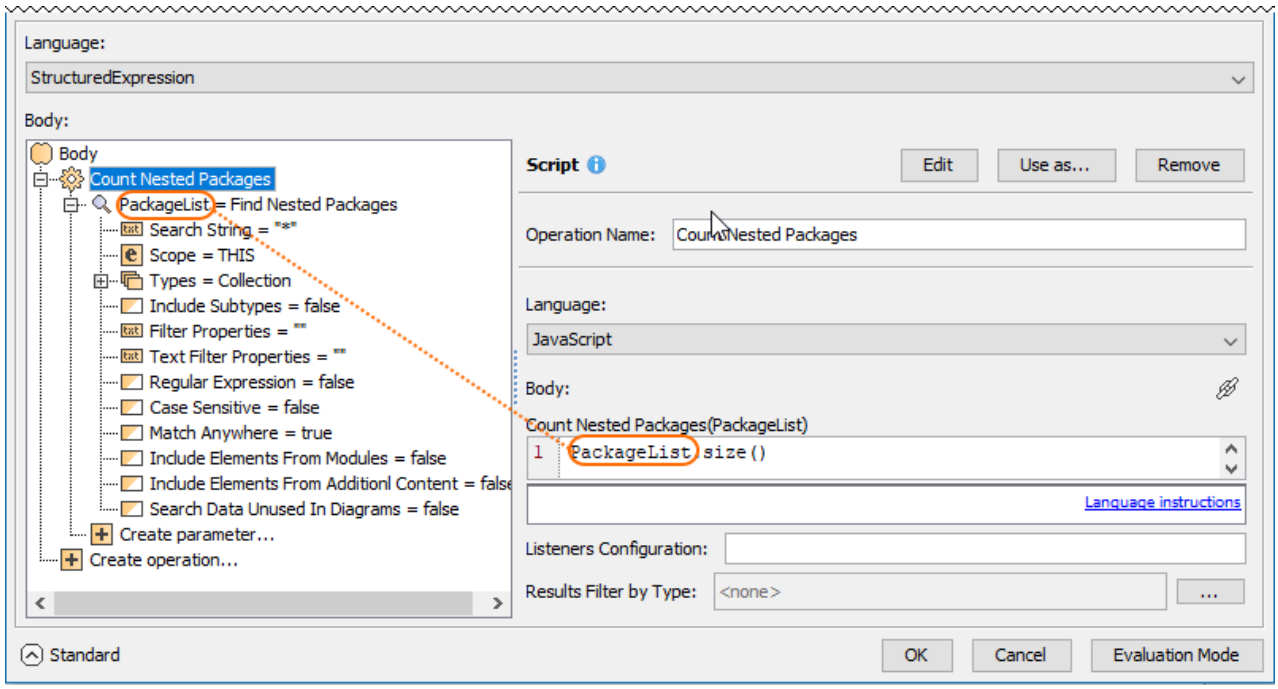


How to simplify scripts?
To simplify the script, move complex sub calculations out of the script. Use other operations of the structured expression to specify these sub calculations and then pass the results to the script through parameters.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

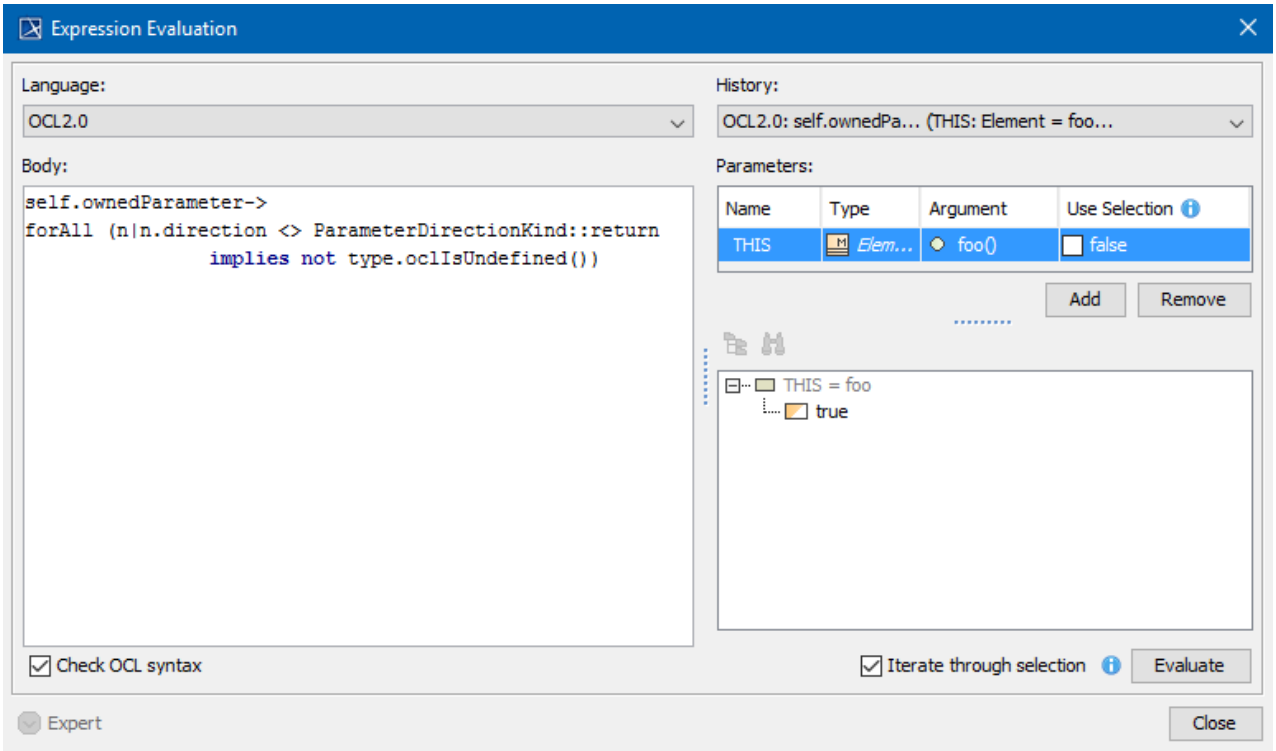
Let's say we need a script operation, which counts the number of nested packages. You can use MagicDraw Open API to navigate through the model and find all the nested packages, but it would be complicated.
To simplify the script:

1. Use the Find operation to list these nested packages.
2. Use the script operation for nothing else than counting the number of the list items. Pass the result of the Find operation to this script through the PackageList parameter as the following figure shows.



Expression Evaluation

The expression evaluation capability in the expression editor allows you to run/execute an expression on the actual testing model while editing.



Expression Evaluation dialog showing the expression body on the left and values of parameters on the right.

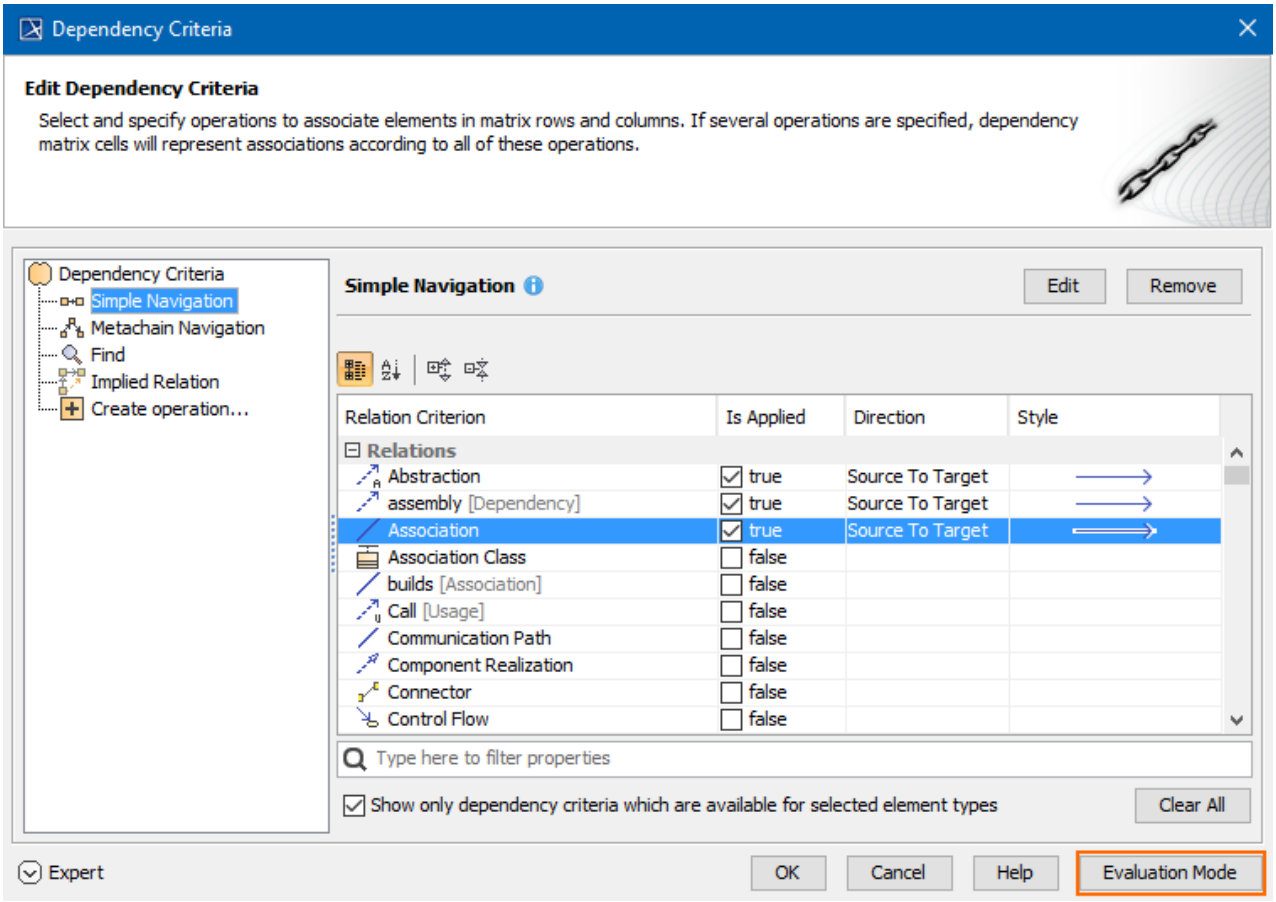
To open the **Expression Evaluation** dialog

- In the modeling tool main menu, click **Tools,** and select **Expression Evaluation**.

While evaluating your expressions, choose:

- **Use Selection**. Set to True, if you want to use the selected elements as the evaluation criteria. Elements can be selected either in the Model Browser or in the diagram, but not both.
- **Iterate through selection**. Select the check box to evaluate model elements one by one.

The Expression Evaluation is also available in the dialogs, wherein you can define expressions, e.g., **Exp ression** editor, **Dependency Criteria**, **Body and Language** dialogs. If you want to start using the expression evaluation in one of these dialogs, click the **Evaluation Mode** button.



Dependency Criteria dialog showing the location of the Evaluation Mode button on the right.

Related pages

- Specifying criteria for querying model(see page 1227)
- Object Constraint Language(see page 1147)
- Smart Package(see page 824)

Case Studies for Querying the Model

The most frequent scenarios for querying the model can be found here. Each case contains detailed instructions for common scenarios found when querying the model.

- Case 1. Coloring Ports by Type(see page 1276)
- Case 2. Finding Element Usages(see page 1276)
- Case 3. Requirements from the Owning Package of a Smart Package(see page 1276)
- Case 4. Properties that Satisfy System Requirements(see page 1277)
- Case 5. Validating Elements Not Used in Diagram(see page 1277)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Case 1. Coloring Ports by Type

To color ports by their type

---

1. Create a Legend and a legend item.
2. Open the Specification window of a legend Item.
3. Click the specification cell of the **Adornment** property, and then click ...
.
4. In the **Adornment Properties** dialog, specify the desired adornment properties.
5. Select the specification cell of the **Elements by Condition** property, and click ...
.
6. Click **Expert** at the bottom of the dialog. The dialog is switched to the Expert mode.
7. In the **Structured Expression** dialog on the left, click **Create operation**.
8. Select the **Metachain Navigation** operation type.
9. Click the **Insert** button to create the first step of the **Metachain**.
10. Select **Port** under **Metaclass or Stereotype** and under **Property**, select **Type**.
11. In the **Structured Expression** dialog on the left, click **Create operation**.
12. Select the **Operation from Model** operation type.
13. Select the **Contains** opaque behavior from the model.

    ⚠ The **Apply Filter** check box must be unselected!

14. On the left side of the dialog, expand the **Contains** operation and drag-and-drop the previously created **Metachain Navigation** on the **Input** parameter.
15. On the left side of the dialog, click the **Obj** parameter of the **Contains** operation.
16. Select **Element** from the **Values** category.
17. Click ... and specify the port type to verify.

Case 2. Finding Element Usages

To create a derived property to find blocks used as types of a part property

---

1. Open a Block's specification window and click the **Traceability** node (in **All** mode).
2. Click the **Create** button to introduce a derived property.
3. On the left side of the dialog, select **Metachain Navigation.**
4. Click the **Insert** button to create the first step of the metachain.
5. Under **Metaclass or Stereotype**, select **Block** and under **Property**, select **_typedElementOfType**.
6. Click the **Insert** button to create the second step of the metachain.
7. Under **Metaclass or Stereotype**, select **Part Property** and under **Property**, select **Type**.

Case 3. Requirements from the Owning Package of a Smart Package

To create a Smart Package that collects requirements from its owning package

---

1. Create a **Smart Package**.
2. Open its specification window and click to edit the **Query** property.
3. On the left side of the dialog, click **Simple Navigation**.
4. Select the **Owner** property to apply.
5. Click **Expert** at the bottom of the dialog. The dialog is switched to the Expert mode.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

6. On the criteria specification panel, click the **Use as...** button and select the **Scope of a new Find** command.
7. Click to specify the **Type** property of **Find**.
8. In the open dialog, choose **Requirement**.

Case 4. Properties that Satisfy System Requirements

To create a Generic Table showing only the properties that satisfy system requirements

---

1. Create a **Generic Table**.
2. In the **Criteria** area, click the button next to the **Element Type** box, set the **Property** as an element type, and select the **Include Subtypes** check box. Click **OK**.
3. In the **Criteria** area, next to the **Scope** box, click the icon and select **Custom.**
4. Click **Expert** at the bottom of the dialog. The dialog is switched to the Expert mode.
5. On the left side of the dialog, click **Find**.
6. Click the button next to the **Type** box, set the **Requirement** as a type, then select the **Include Subtypes** check box. Click **OK**.
7. On the left side of the dialog, expand the **Metachain Navigation** and drag-and-drop the previously created **Find** on the **Context** parameter.
8. Click the **Metachain Navigation.** Click the **Inser**t button to create the first step of the metachain.
9. Under **Metaclass or Stereotype**, select **Requirement** and under **Property**, select **Supplier Dependency**.
10. Click the **Insert** button to create the second step of the metachain.
11. Under **Metaclass or Stereotype**, select **Satisfy** and under **Property**, select **Client**.

Case 5. Validating Elements Not Used in Diagram

To validate elements that are not used in a diagram

---

1. Create a Package and apply the «activeValidationSuite» stereotype.
2. Under the created Package, create a Constraint and apply the «validationRule» stereotype.
3. Specify the **Severity**, **Error Message**, and **Abbreviation** for the created validation rule.
4. Specify the **constrained element** - metaclasses and stereotypes whose instances are validated if not used on the diagram.
5. Click to specify the **Specification** property.
6. From the **Language** drop-down list, select a **StructuredExpression**.
7. Click **Expert** at the bottom of the dialog. The dialog is switched to the Expert mode.
8. In the **Body** area of the **Specification** dialog, click **Create Operation**.
9. Click the **Operation from Model** operation type on the right side of the dialog.
10. Select the **Not** opaque behavior from the model.

    > The **Apply Filter** check box must be unselected!

11. On the left side of the dialog, select the **A** parameter of the **Not** operation and press **Delete**.
12. On the right side of the dialog, click the **Operation from Model** operation type.
13. Select the **IsEmpty** opaque behavior from the model.

    > The **Apply Filter** check box must be unselected!

14. On the right side of the dialog, click the **Operation from Model** operation type.
15. Select the **UsageInDiagrams** opaque behavior from the model.

> The **Apply Filter** check box must be unselected!

16. On the left side of the dialog, click the **Element** parameter of the **UsageInDiagrams** operation and press **Delete**.
17. Select the **Contextual Variable** from the **Other** category.

Element usage in diagrams

An element symbol is a visual representation of a model element in a diagram. There are two types of symbols: shapes and paths. In a diagram, all types of relationships are represented as paths. Model elements that are not relationships are represented as shapes.

No Magic modeling tools allow you to see element usages in diagrams by listing all diagrams that display the symbols of a particular model element. You can access the diagrams in which an element is used from the Specification window(see page 913) of that element, the element's shortcut menu, or the main menu(see page 25).

> **Additional information**
> For more information about element usages and relations between model elements, see Analyzing usages and dependencies(see page 1192).

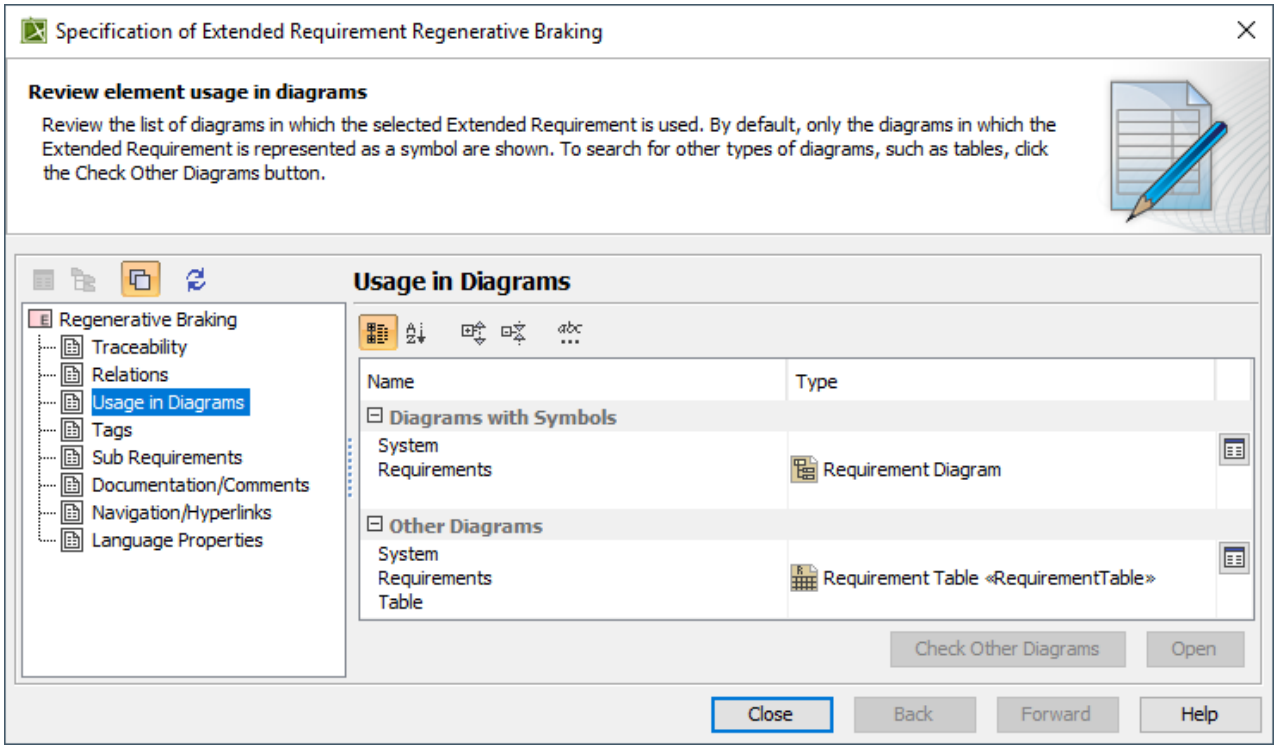
To search for element usage in diagrams from the Specification window

---

1. Double-click the desired element to open its Specification window.
2. On the left side of the Specification window, in the property group list, select the **Usage in Diagrams** node.
   The area on the right side of the window shows all diagrams in which the element is represented as a symbol. To list other types of diagrams (e.g., Generic Table) the selected element is used in, click the **Check Other Diagrams** button.
3. Depending on what you want to do, complete one of the following actions:
   - To open one of the listed diagrams, select the desired diagram and click the **Open** button or double-click the desired diagram. Alternatively, you can right-click the diagram and select **Open** from the drop-down menu
   - To open the Specification window of one of the listed diagrams, click near the desired diagram in the list.

If you open a diagram (described in the first option of step 3), the symbol of the related element is selected in that diagram. If the diagram includes several symbols of the same element, all of those symbols are selected in the diagram.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The Specification window of a model element displaying all the diagrams it is used in.

To search for element usage in diagrams from the element's shortcut menu

1. Right-click either the desired element in the model browser or its symbol in a diagram.
2. From the element's shortcut menu, select **Go To** > **Usage in Diagrams**.
3. Select either **Diagrams with Symbols** or **Other Diagrams**.
4. Depending on what you want to do, complete one of the following actions:
    - To open one of the listed diagrams, select the desired diagram.
    - To open all listed diagrams, click **Open All Diagrams**.

Depending on the action you performed in step 3, one or all of the diagrams in which the related element is used are opened with the selected element symbol(s).

To search for element usage in diagrams from the main menu

1. Select either the desired element in the model browser or its symbol in a diagram.
2. In the main menu, select **Analyze** > **Go To** > **Usage in Diagrams**.
3. Select either **Diagrams with Symbols** or **Other Diagrams**.
4. Depending on what you want to do, complete one of the following actions:
    - To open one of the listed diagrams, select the desired diagram.
    - To open all listed diagrams, click **Open All Diagrams**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

The selected diagram or all of the diagrams in which the related element is used are opened with the element symbol(s) selected.

Related pages

- Model analysis(see page 1192)
  - Analyzing usages and dependencies(see page 1192)
  - Traceability(see page 1201)
  - Specifying criteria for querying model(see page 1227)
  - Element usage in diagrams(see page 1278)
  - Identifying Package Dependencies(see page 1280)
  - Comparing projects(see page 1291)
  - Validation(see page 1306)
  - Model Visualizer(see page 1346)
  - Suspect Links(see page 1354)

Identifying Package Dependencies

In a model, packages(see page 800) can be associated with other packages or model elements in a project scope or outside the project scope. In other words, packages can have relations with external elements in used projects, shared libraries, profiles, etc. An element depends on the used project when at least one of its metaproperties reference elements from at least one share of that used project. In this case, the element depends on that used project.

To avoid any violations on editing a package, especially on removing or exporting a package to another project, it is always good to know all of the package dependencies. Dependency analysis allows users to detect cyclic dependencies that impede reusing the package in the current or other projects because it can cause an infinite recursion. This analysis manages the impact of changes to other related elements.

In the modeling tool, the Dependency Checker allows you to analyze package dependencies in the whole project scope or analyze dependencies between the selected package/model and shared packages. The dependency severity level status shows how a package depends on a particular element. There are three severity levels:

- **Info**. The lowest severity.
- **Warning**. Conflicts that require your attention are displayed.
- **Error**. The highest severity.

> **Severity levels**
> While checking dependencies, conflicts of the selected severity (and higher) are displayed in the **Package Dependencies** panel(see page 1282).

In the **Project Options** dialog(see page 214), you can define the default values for checking dependencies. You can specify the following:

- The **Check for Cyclic Dependencies among Used Projects** option. Click to select *true* if you want the cyclic dependencies to be checked while exporting or sharing the packages.
- The **Dependency Checker Severity Level** option. Specify the severity level. The dependencies of the specified severity (and higher) will be displayed.
- The **Ignore Standard/System Profiles** option. Click to select *true* if you want to exclude the standard/system profiles from checking for cyclic dependencies in used projects.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Each time you start the dependency checking, you can change the default settings.

In the following procedures, learn how to work with the Dependency Checker:

- How to set default values for the Dependency Checker(see page 1281)
- How to analyze package dependencies of the whole project or between the selected package/model and shared packages(see page 1281)
- How to analyze package dependencies while exporting packages(see page 1281)
- How to analyze package dependencies while sharing packages(see page 1282)

To set default values for the Dependency Checker

---

1. On the main menu, click **Options** > **Project**. The **Project Options** dialog opens.
2. Expand the **General** option in the option group list.
3. Click the **Dependency Checker** option. The list of options opens.
4. In the Dependency Checker options list, set the desired values and click **OK** when you are done.

> **Description of an option**
> Each option has a short description. Click the option to see the description in the description area of the dialog.

To analyze package dependencies of the whole project or between the selected package/model and shared packages

---

1. Choose one of the following:
   - To analyze package dependencies of the whole project, on the main menu, click **Analyze** > **Dependency Checker**.
   - To analyze dependencies between the selected package/model and shared packages, right-click the package or model in the Model Browser or on the diagram pane, point to **Tools** and then click **Dependency Checker**.
2. The **Dependency Checker** dialog opens. Select the **Minimal Severity** from the list, check ignore or not standard and system profiles while checking and click **OK** to execute the dependency checker. If some dependencies are found, they are displayed in the **Package Dependencies** panel(see page 1282).

To identify package dependencies while exporting packages

---

1. From the package/model shortcut menu, choose **Project Usages** > **Export Packages to New Project**. The **Export Packages to New Project** dialog appears.
2. Select the packages you want to export. For more information, see Exporting packages to new projects(see page 237).
3. Click **OK**.
4. In the question dialog, click **Yes**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Notes

- Dependencies between the selected package/model and shared packages will be analyzed and shown in the opened **Package Dependency** panel(see page 1282).
- If you want to discover *cyclic dependencies*, select the **Check for cyclic dependencies among used projects** check box. This enables for the dependencies which have **Error** and **Warning** severity levels to be displayed.

To identify package dependencies while sharing packages

1. From the package/model shortcut menu, choose **Project Usages** > **Share Packages**. The **Shared Packages** dialog appears.
2. Select the packages you want to share. For more information, see Sharing project data(see page 239).
3. Click **OK**.
4. In the question dialog, click **Yes**.

Notes

- Dependencies between the selected package/model and shared packages will be analyzed and shown in the opened **Package Dependency** panel(see page 1282).
- If you want to discover *cyclic dependencies*, select the **Check for cyclic dependencies among used projects** check box. This enables for the dependencies which have **Error** and **Warning** severity levels to be displayed.

Cyclic dependency

If there is a chain of dependencies such that A → M(1), M(1) → M(2), M(2) → M(3), ..., M(X) → M(A), where:

- A is an element from the used project M(A)
- M(1..X) are other used projects
- A → M(x) is element A dependency on used project M(x)
- M(y) → M(x) is a dependency of at least one element in used project M(y) on used project M(x),

then this chain is called a cyclic dependency and every atomic dependency in this chain is considered as part of cyclic dependency.

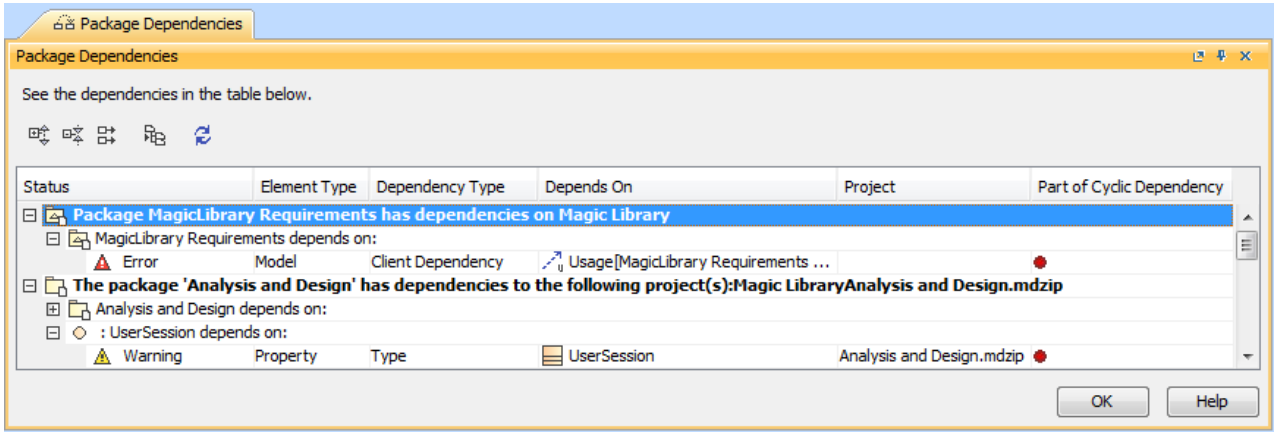
Related pages

Package Dependencies panel

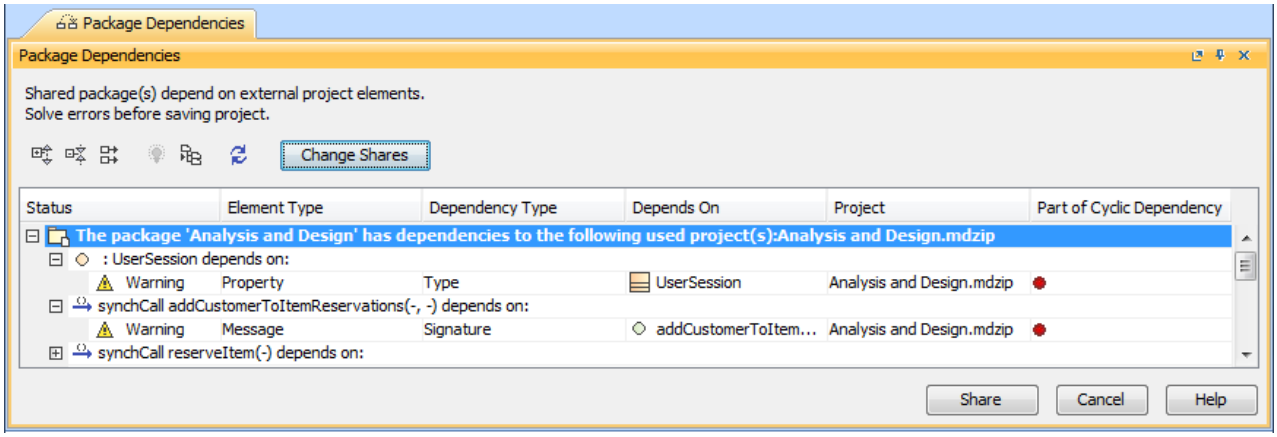
Before exporting or sharing a package as a independent used project, you need to identify and analyze package dependencies first. The **Package Dependencies** panel will help you with that. The **Package Dependencies** panel is displayed when you select to share(see page 239) or export(see page 237) packages.

Before hiding a package, you can identify and analyze package usages and usages of all elements in the package first. The **Package Usage** window will help you with that. The **Package Usage** panel is displayed when you choose to check usages of elements in the selected packages for hiding(see page

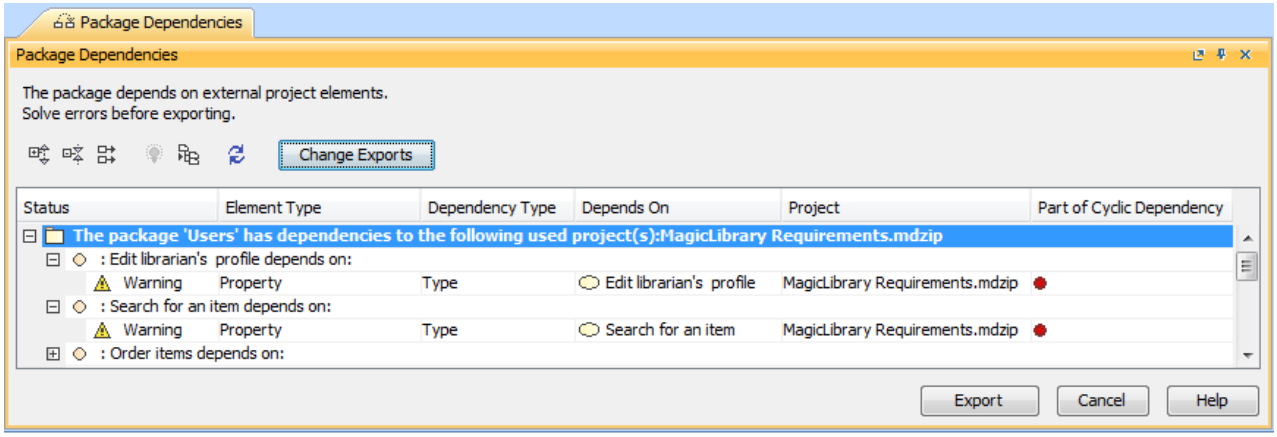
1457).



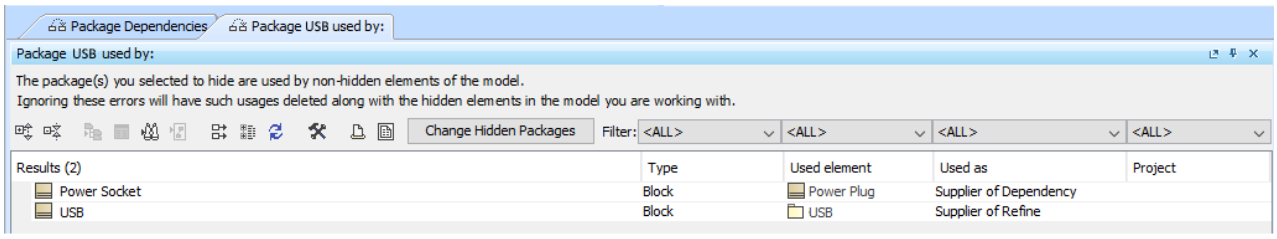
Package Dependencies panel



Package Dependencies panel when checking dependencies on package sharing



Package Dependencies panel when checking dependencies on package exporting



Package Usage search window when checking usages on package hiding

The **Package Dependencies** panel has a table which shows the list of dependencies and buttons for managing data displayed in this table.

Button	Description
Expand All Tree Branches	Expands all nodes in the package dependencies tree.
Collapse All Tree Branches	Collapses all nodes in the package dependencies tree.
Display the Full Path of the elements	Displays the element full path next to the element name.
Solve	The button is enabled, when a dependency with **Error** status is selected in the table and a solution for the problem can be found. Clicking the button opens the dialog for choosing the solution for a specific dependency problem. This button is only visible when exporting or sharing a package.
Select in Containment Tree	Shows the selected element in the Browser. The button is available, when a dependency is selected in the table.
Refresh	Performs dependency analysis and refreshes the dependency table with the new analysis results.
Change Shares	Opens the **Shared Packages** dialog wherein you can select another packages to be shared. This button is visible only when sharing a package.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Button	Description
Change Exports	Opens the **Export Package to New Project** dialog wherein you can select another packages to be exported. This button is visible only when exporting a package.
Share	Closes the **Package Dependencies** panel and makes the package shared. The button is available, when the **Check Dependencies on Package Export/Sharing** environment option is set to *Allow dependencies* (to open the **Environment Options** dialog, select **Options** > **Environment**). This button is visible only when sharing a package.
Export	Closes the **Package Dependencies** panel and opens the **Save as/ Commit Settings** dialog for saving/committing the package as separate project. The button is available, when the **Check Dependencies on Package Export/Sharing** environment option is set to *Allow dependencies* (to open the **Environment Options** dialog, select **Options** > **Environment**). This button is visible only when exporting a package.

Column	Description
Status	Shows severity of the element dependency problem. The status can be **Error**, **Warning**, or **Info**. Dependencies that have **Error** status: • used project depends on the project Dependencies that have **Warning** status: • cyclic used project depends on other used projects Dependencies that have **Info** status: • project element depends on elements from shared packages (shared packages can belong both to the project and external project)
Element Type	Displays element type.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Column	Description
Dependency Type	Displays dependency type.
Depends On	Displays the model element, on which the package/model element is dependent.
Project	Displays the name of the used project file that owns the model element, on which the package/model element depends.
Part of Cycling Dependency	Indicates that the element is a part of the *cycling dependency**.

*** Cyclic dependency**
If there is a chain of dependencies such that A → M(1), M(1) → M(2), M(2) → M(3), ..., M(X) → M(A), where:

- A is an element from the used project M(A)
- M(1..X) are other used projects
- A → M(x) is element A dependency on used project M(x)
- M(y) → M(x) is a dependency of at least one element in used project M(y) on used project M(x),

then this chain is called a cyclic dependency and every atomic dependency in this chain is considered as part of cyclic dependency.

Related pages

- Identifying Package Dependencies(see page 1280)
- Analyzing dependencies among elements(see page 1286)
- Unresolved dependencies(see page 1290)

Analyzing dependencies among elements

Before exporting or sharing a package as an independent used project, you need to identify and solve package dependencies first. More specifically, you have to make sure that the package does not depend on any external elements (except other used projects). Furthermore, cyclic dependencies between several used projects are not allowed.

There are three types of dependencies:

- Package dependencies by relationship(see page 1286)
- Dependencies by reference(see page 1288)
- Diagram dependencies(see page 1289)

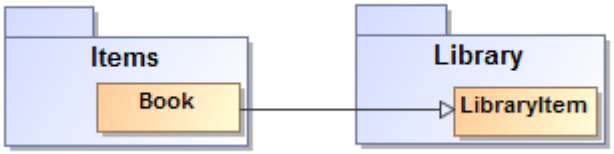
Package dependencies by relationship

The used project depends on the external elements which have relationships with the model elements from the used project packages.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

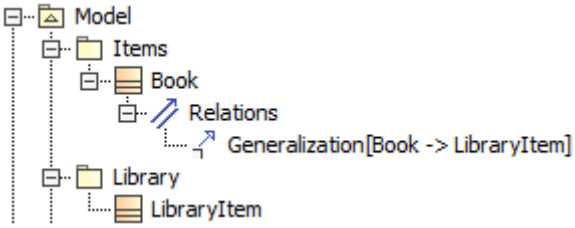
The used project depends on external elements

If a used project's element has a relationship with an external element and that relationship is contained in the used project package, an error message appears when exporting the used package.

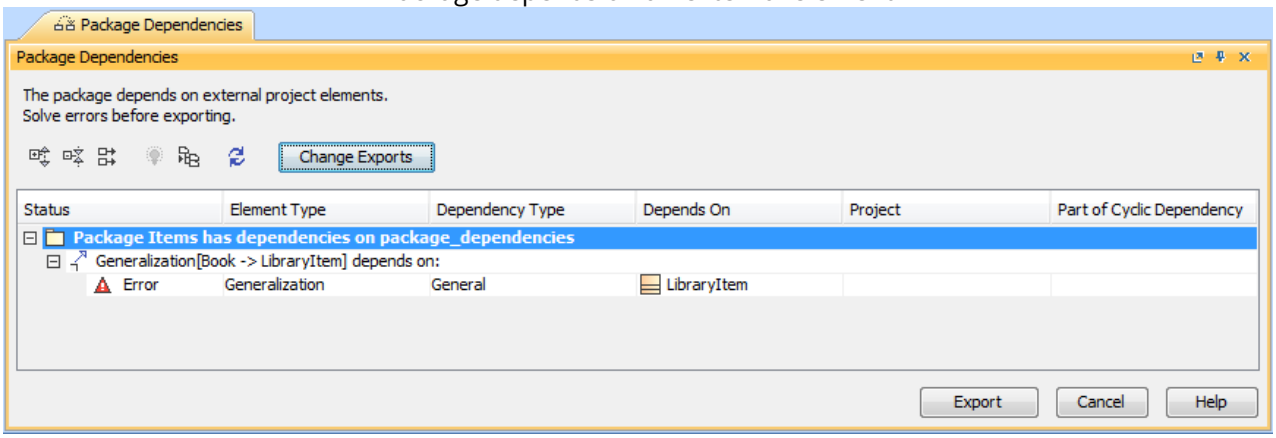


Example of the used project dependency on an external element

Such dependencies on external elements are displayed in the Model Browser:

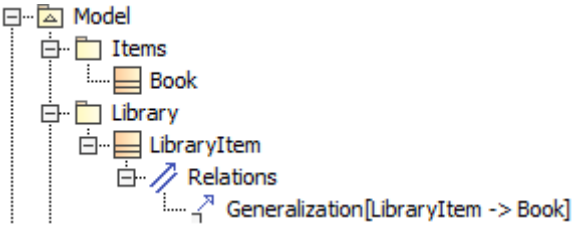


Package depends on an external element



Error in the Package Dependencies panel

In this situation, a modeling tool can suggest moving the relationship into the parent package of this external element. For example, the package *Items* is a parent of a class *Book*, so the relationship can be moved from the *Items* into the package *Library*:



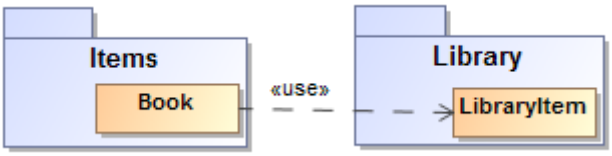
Resolved package dependency on an external element

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Some movements can be achieved by clicking **Solve** in the **Package Dependencies** panel(see page 1282).

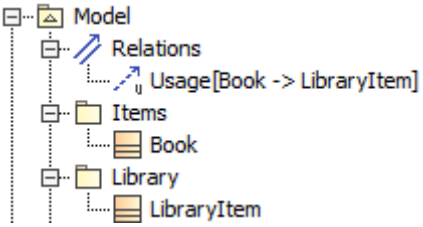
The used project depends on an external element, but can be exported (with warning)

Even though the used project element has a relationship with an external element, that relationship is contained in an external package:



Example of a "legal" used project dependency on an external element

In this case, the dependency on an external element is displayed in the Model Browser:



"Legal" used project dependency on an external element in the Model Browser

The package can be exported as the used project because the relationship is contained in an external package.

The used project does not depend on an external element

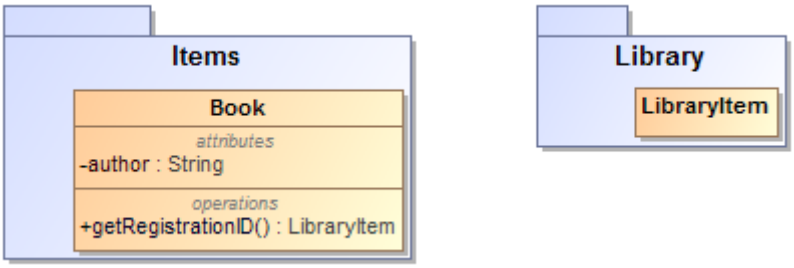
If the used project element has a relationship with an external element it is irrelevant in the context of UML (for instance, the external model uses the used project, but not vice versa) and that relationship is contained in an external model, the package can be exported into an independent used project:

Example of a relationship when the used project does not depend on an external element

Dependencies by reference

The used project depends on the external elements which have references to the model elements from the used project packages.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



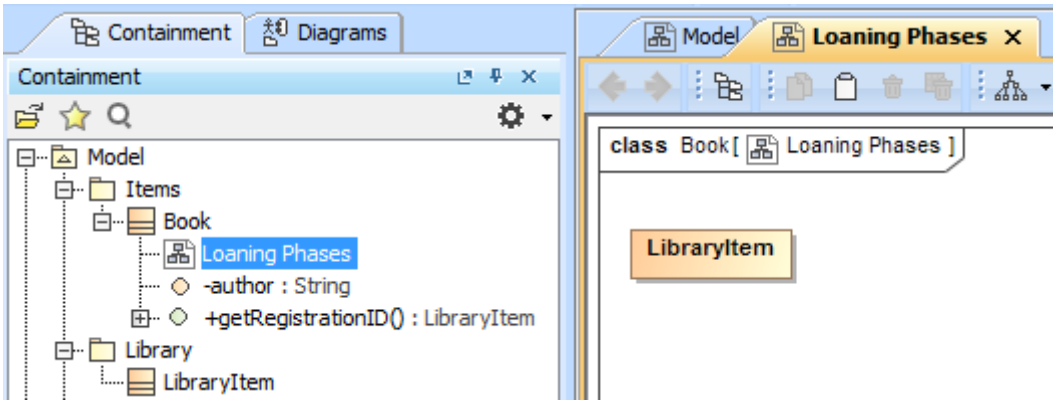
Example of a dependency by reference

In this case, the package *Items* cannot be exported to an independent project because this package has reference to the *Library* package.

Diagram dependencies

The diagram depends on all model elements displayed within it.

If the diagram is contained in the package *Items* and depends on external elements, this package cannot be exported to a separate project.



Example of the relationship when a diagram depends on an external element

For more information about the package dependencies on external elements, see The used project depends on external elements(see page 1287).

In this case, if the diagram is not important to the used project, it can be moved from the used project package into any external package by dragging-and-dropping it within the Model Browser:

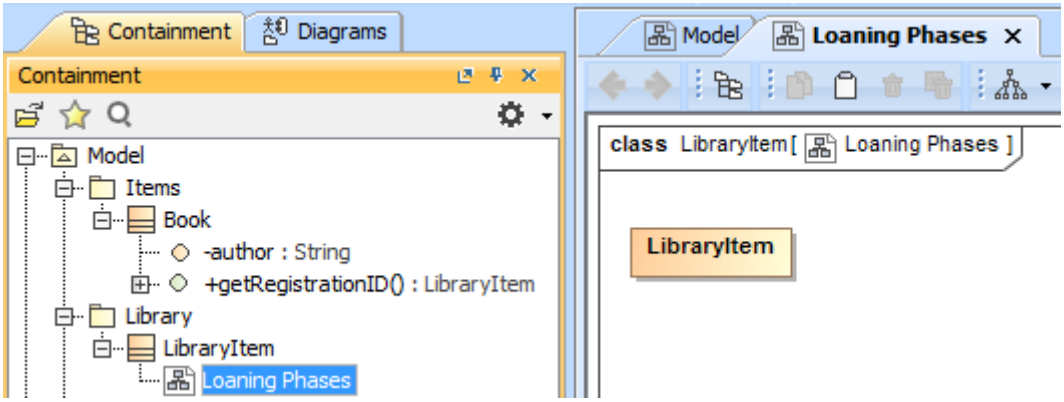


Diagram is moved from the package *Items* to the package *Library*

The package *Items* can now be exported into an independent project.

Teamwork Server or Teamwork Cloud is an ideal solution for a collaborative work on the same project. For more information about using a modeling tool in the collaborative environment, see Collaborative modeling(see page 1360).

Related pages

- Identifying Package Dependencies(see page 1280)
- Package Dependencies panel(see page 1282)
- Unresolved dependencies(see page 1290)

Unresolved dependencies

Before exporting or sharing a package as an independent used project, you need to identify and solve package dependencies first. More specifically, you have to make sure that the package does not depend on any external elements (except other used projects) in a project.

The same situation occurs when you edit the used project inside the current project (the used project permissions should be set to read-write) and introduce dependencies from the used project back to the project where it is being used. In this case, you will be asked to solve these dependencies when saving the used project.

However, the modeling tool allows for continuing without solving these dependencies right away. The elements, which were referenced, but are missing in the used project will be shown as recovered elements (see Maintaining Decomposed Model Integrity(see page 254)).

You can choose how the dependency checking will be carried on in your projects by changing the **Check Dependencies on Package Export/Sharing** environment option (from the **Options** menu, select **Environment** > **General** > **Save/Load**). You can choose from one of the following:

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- **Do not allow dependencies**. If this option is selected, then unsolved dependencies should not be allowed while exporting or sharing packages. The **Export** (or **Share**) button is not available until the dependencies are solved.
- **Allow dependencies**. If this option is selected, then the dependencies are identified, but it is not required to solve them and the **Export** (or **Share**) button is available.
- **Do not check**. If this option is selected, then the dependencies will not be checked at all. In this case, the selected package will be exported or shared as is. Be aware, that this might cause an increasing number of the recovered elements in your project (see Maintaining Decomposed Model Integrity(see page 254)).

Related pages

- Identifying Package Dependencies(see page 1280)
- Package Dependencies panel(see page 1282)
- Analyzing dependencies among elements(see page 1286)

Comparing projects

The Compare Projects functionality allows you to compare:

- Two server project versions
- Open server project with a local or server project version
- Open local project with a server project version

Various different properties of the elements, that share the same ID in different projects, are compared.

> This functionality is available in Architect and Enterprise editions only.

Basic concepts

For a better understanding of the further material, learn the following concepts.

Concept	Description
Base	The original project version without changes.
Compared	A project version that contains modifications added to original project version.

Related pages

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Understanding change types

Change is a difference, found between the base and compared project versions.

Read the following definitions to get familiar with different change types.

Addition change
If an element has been added to a compared project version, an addition change occurs.

Deletion change
If an element has been removed from a compared project version, a deletion change occurs.

Modification change
If an element property in a compared project version has been modified, a modification change occurs.

> If the **IsAbstract** property value of a class in the base had the default value *false* and the same property value in a compared project version has been changed to *true*, a modification change occurs.

There are three types of modification changes:

- **Addition modification change** that occurs when a value is added to a property.
- **Deletion modification change** that occurs when a value is removed from a property.
- **Replacement modification change** that occurs when one value is replaced with another. This type of modification change occurs only for properties that have multiplicity less or equal to 1.

Movement change

If an element owner has been changed in a compared project version, a movement change occurs.

> Let's say package A contains some class in the base and package B contains the same class in a compared project version. This means that the class has been moved from package A to package B in the compared project version. This case is recognized as a movement change.

> Another case of the movement change is when an attribute or an operation that has been owned by class A in the base, becomes the attribute or an operation of class B in a compared project version.

Order change
If the order of elements has been changed in a compared project version, an order change occurs. Order changes can occur on elements such as attributes, operations, and other ordered elements. Even if a single element in a collection has changed its place, the order change is applied to the entire collection.

Since an element can have several ordered collections, several order changes can occur on a single element.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Let's say class A has attributes a, b, and c in the base. The attribute c has been moved up and placed above attribute a in a compared project version. This means that the order of attribute collection in class A has changed in the compared project version. This is a case of the order change.

Order changes can be skipped while comparing projects. For this you need to specify names of properties wherein order changes should not be detected.

To turn off the order changes detection in specific properties

1. From the **Options** menu select **Environment**. The **Environment Options** dialog opens.
2. Find the **Do Not Detect Order Changes for** option under the **Merge and Compare** category in the **General** options group.
3. In the option value cell, specify names of properties wherein order changes should not be detected while merging.

Property names must be written in camel case, for example, ownedAttribute, ownedElement, and so on.

Dependent change

In some cases, changes depend on other changes and are called dependent changes.

For better understanding of the concept of dependent changes, study the following examples.

Let's say a class attribute type has been changed to a type that had been created by another change. In consequence, the attribute type change depends on the change that has created the type.

Let's suppose there is an attribute type change in a compared project version. An old type has been deleted and a new type has been added to the compared project version. In this case, three changes occur:

- deletion change (for the old type)
- addition change (for the new type)
- modification change (for the property type)

The modification change depends on the addition change, and the deletion change depends on the modification change.

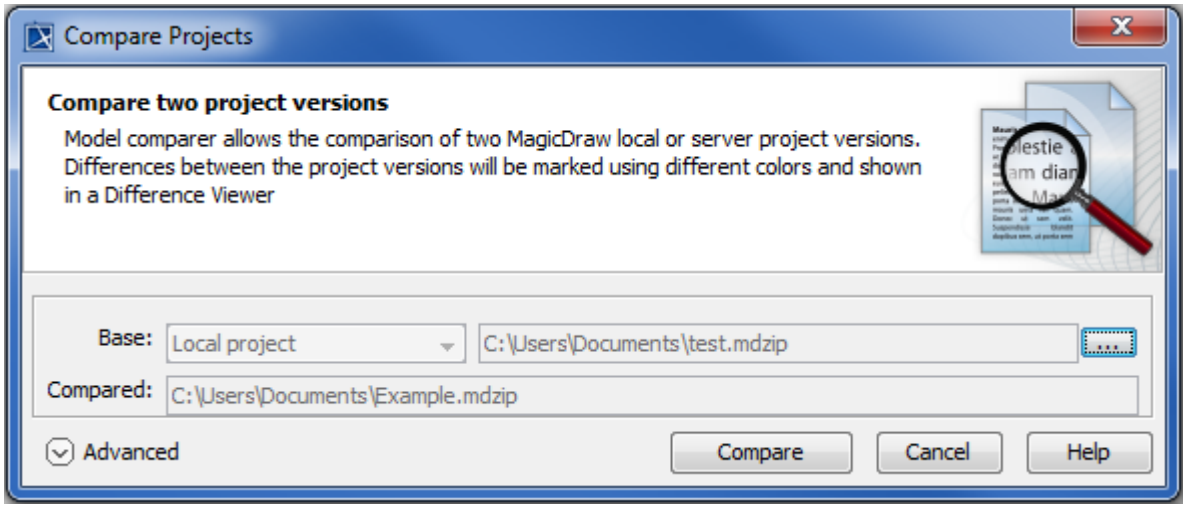
Starting Compare Projects

There are two methods to compare projects:

- To compare projects from the Analyze menu(see page 1294)
- To start comparing projects from the Collaborate menu(see page 1295)

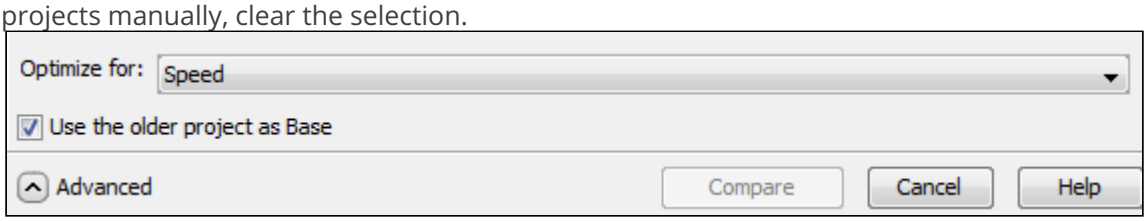
To compare projects from the Analyze menu

1. Open a project you want to compare.
2. From the **Analyze** menu, select **Compare Projects**. By default, the open project is selected in the **Compared** box.



3. From the **Base** box, select a project you want to compare your open project to. The following options are available:
   - Local project. Click ... and select a project from your file system you want to compare the current project to.
   - Server project (a Teamwork Server project or Teamwork Cloud server project). Click ... From the **Open Server Project** dialog, select a project and its version, trunk or branch you want to compare it with. If your compared project is also a server project, you can only choose a server project as a base.

4. Advanced **Compare projects** options:
   In the same dialog, click the **Advanced** button to see more compare options. The **Use the older project as Base** option is selected by default. If you keep this selection, the Base project will be selected automatically, according to the dates of the projects. In this case, all changes are marked as made in the Compared project. If you prefer to select both Base and Compared projects manually, clear the selection.



   In the **Optimize for** drop-down list, select
   - **Speed**, to compare the projects faster. This is the default option and it is sufficient for most projects. Keep in mind that this option requires more memory.
   - **Memory,** to compare bigger projects. Sometimes the memory required for the **Speed** compare mode exceeds the memory available for MagicDraw on your machine. In that case, switch to **Memory** mode, which conserves memory. However, this option slows the procedure down considerably.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

> You can also specify the **Optimize for** option in the **Environment Options** dialog. Find the option under the **Merge and Compare** category in the **General** options group.

5. Click **Compare**.

To start comparing projects from the Collaborate menu

---

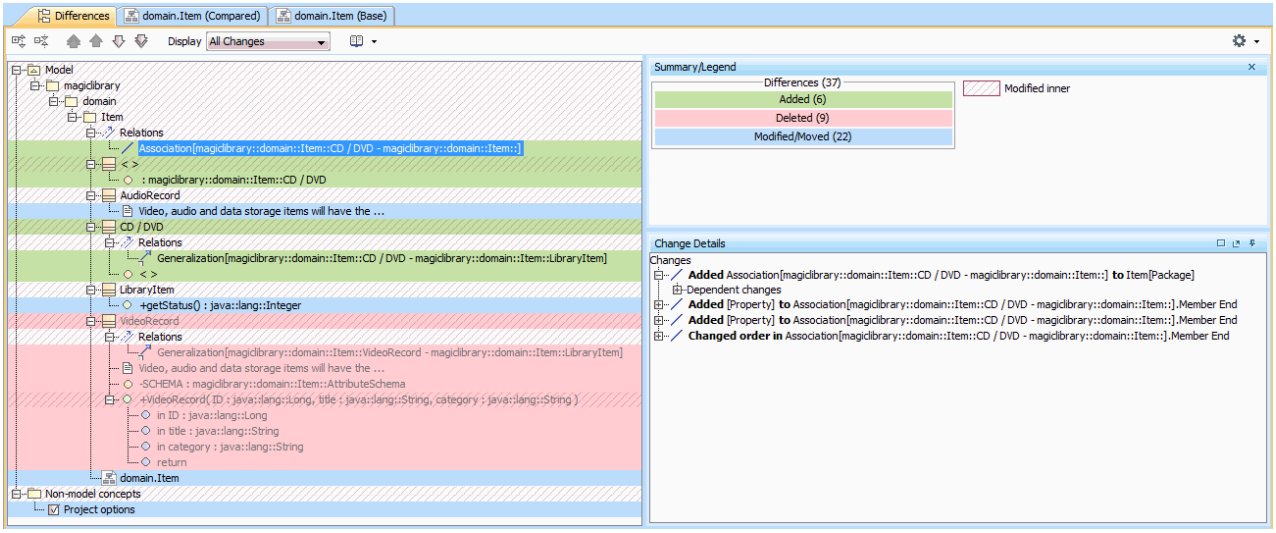
1. Log in to Teamwork Server or Teamwork Cloud server.

   > To compare project versions in the Teamwork Cloud environment, you are required to have a **Read Resources** permission[270] for every used project in compared versions.

2. On the main menu, click **Collaborate**, then select either:
   - **Projects** – the **Manage Projects** dialog opens
   - **Open Server Project** – the **Open Server Project** dialog opens
3. Select the project you want to compare, then click the **Project History** button in the toolbar above project selection list.
4. While holding down Ctrl (or ⌘), select the two versions of the server project you want to compare.
5. Click **Compare**.

Analyzing comparison results

Comparison results are displayed in the **Difference Viewer** window.



Read about analyzing comparison results in

- Getting comparison summary information and changes legend(see page 1296)
- Understanding content of the Differences tree(see page 1296)
- Inspecting element property changes in the differences Specification panel(see page 1101)
- Inspecting differences in the Change details panel(see page 1300)
- Displaying and navigating through differences(see page 1301)

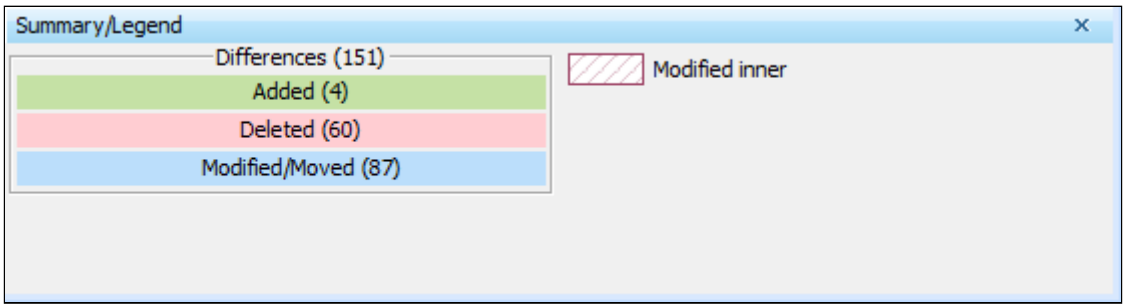
---

270 https://docs.nomagic.com/display/TWCloud190/Permissions

- Viewing and analyzing differences of modified diagrams (see page 1303) (see page 1303) (see page 1303)
- Generating a project comparison differences report(see page 1305)

Getting comparison summary information and changes legend

You can see how many differences have been detected between the base and compared project version on the **Summary and legend** panel that is located at the top right of the **Difference Viewer** window. The panel also helps to understand the decorations and highlighting of elements in both the **Differences** tree and the **Specification** panel.

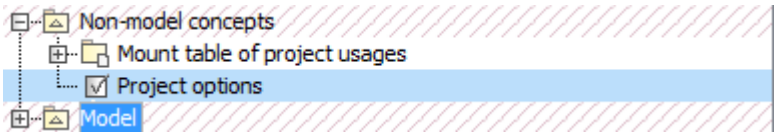


Understanding content of the Differences tree

Elements with all change types are displayed in the **Differences** tree that is located at the top left of the **Difference Viewer** window.

Used project representation

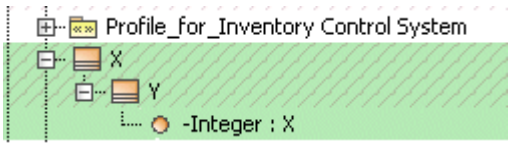
This node indicates that the compared project contains modified used projects or used projects with modified usages.



Recognizing addition changes

For the addition change description, refer to Addition change(see page 0).

Added elements are highlighted in green background.

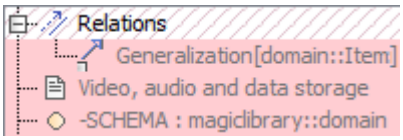


Recognizing deletion changes

For the deletion change description, refer to Deletion change(see page 0).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

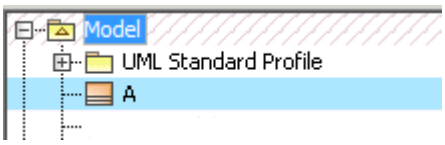
Deleted elements are highlighted in red background.



Recognizing modification changes

For the modification change description, refer to Modification change(see page 0).

Elements whose properties have been changed are highlighted in blue.



To see what properties have been modified, click the element and see changes in the **Specification** panel.

For more information about using the Specification panel, see Inspecting differences in the Change details panel(see page 1300).

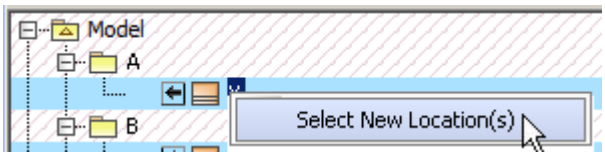
Recognizing movement changes

For the movement change description, refer to Movement change(see page 0).

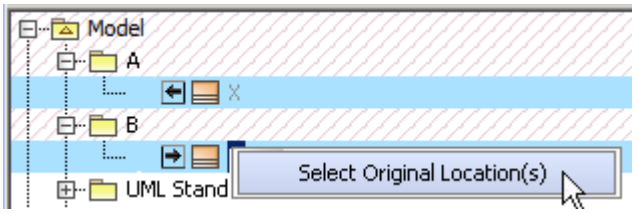
Elements with movement changes are highlighted in the same blue background that is used to highlight modification changes, since movement changes are some kind of modification changes. Icon of an element with movement changes is represented with arrow symbol on the left.

The following figure illustrates a movement change, when element X has been moved from package A to package B.

You can navigate from the element's original location to the new one (and vice versa) using the commands on the element shortcut menu.



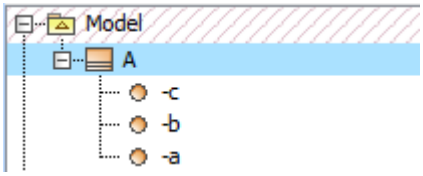
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



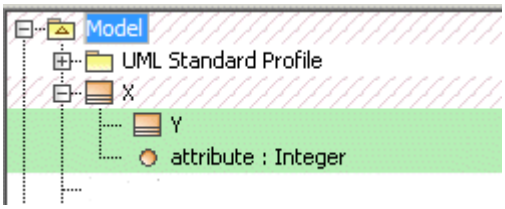
Recognizing order changes

For the order change description, refer to Order change(see page 0).

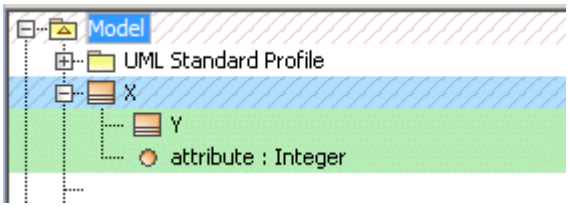
Elements with order changes are highlighted in the same blue background that is used to highlight modification changes, since order changes are some kind of modification changes.



Recognizing elements with changed owned elements
Elements whose owned elements (in any nesting level) have changes are highlighted using red diagonals.



Modified elements with changed owned elements are highlighted in blue with red diagonals.

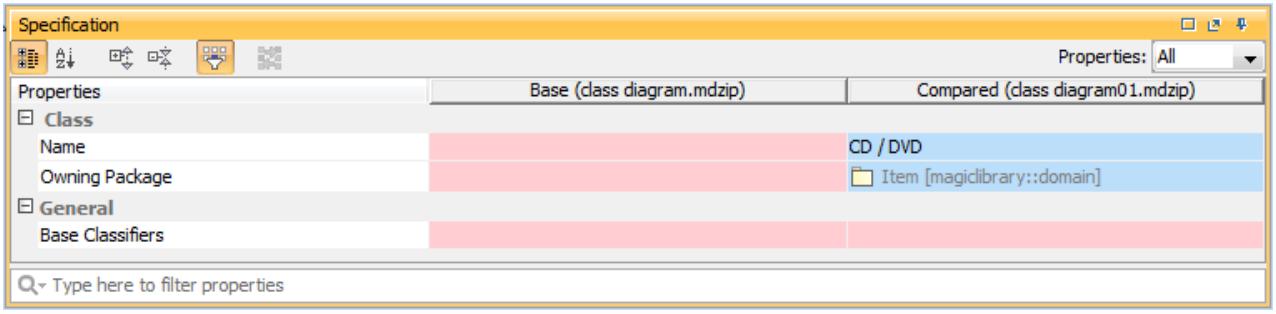


Inspecting element property changes in the differences Specification panel

You can see all the details of the element property changes in the **Specification** panel that is located at the bottom of the **Difference Viewer** window. All you need to do is selecting the appropriate element in the **Differences** tree.

As you can see in the following figure, the first column of the panel contains a list of element property names, and other columns display their value changes that have been detected in both base and compared projects.

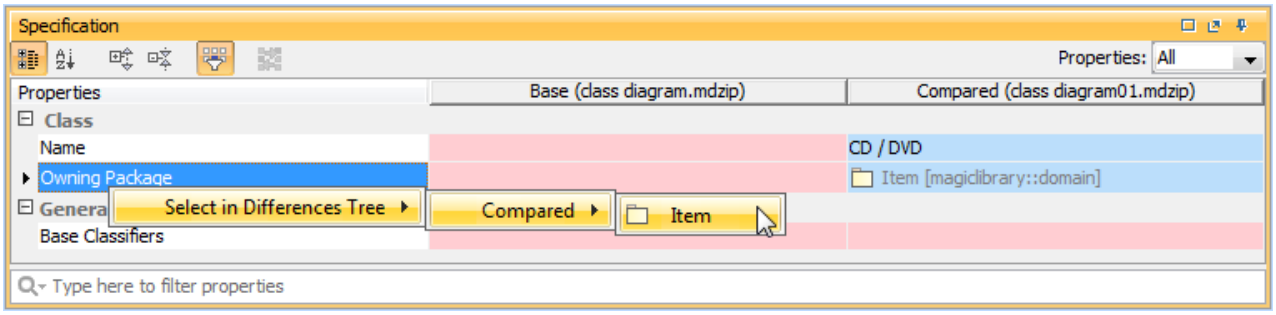
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



To navigate from the **Specification** panel to the **Differences** tree, do either

---

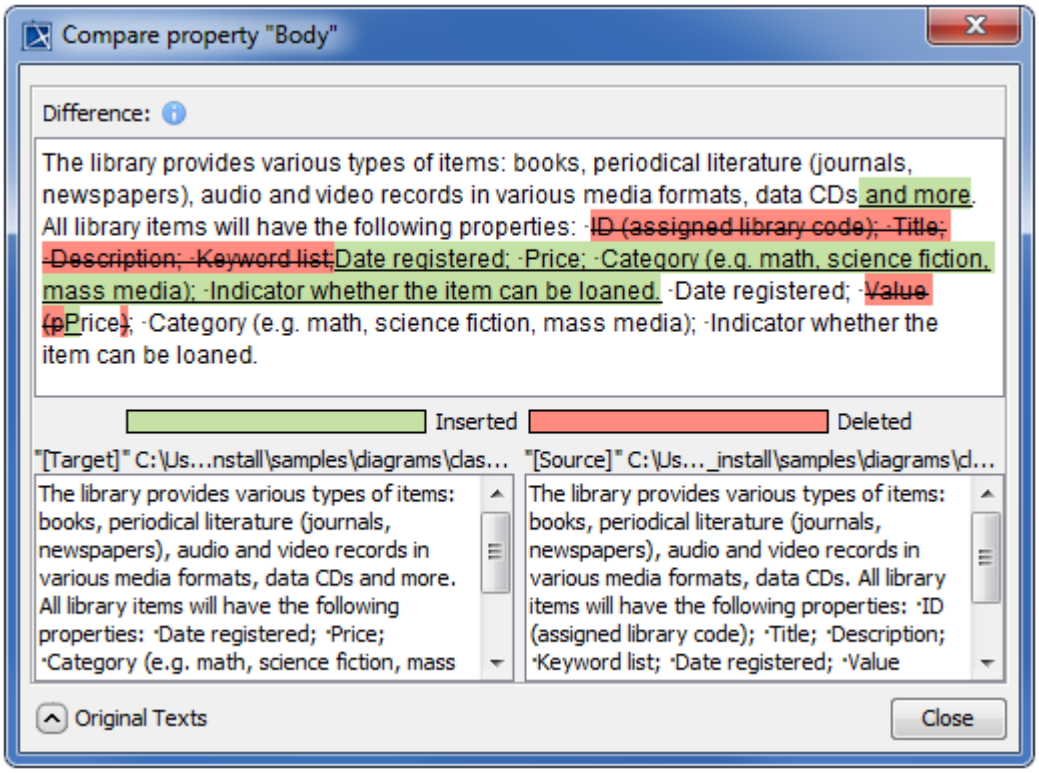
- In the **Properties** column, right-click the name of a property, which references other elements. Then on its shortcut menu, point to **Select in Differences** tree and choose an element to which you want to navigate.
- In the **Base** or **Compared** column, right-click the value of a property, which references other elements. Then on its shortcut menu choose an element to which you want to navigate.



Inspecting differences between textual values

You can see exact changes between textual values of element properties, such as element documentation, comments, pre- and post- conditions of a use case. Differences of compared texts can be displayed in a single dialog, where inserted and deleted parts of the text are appropriately highlighted. As of version 18.3, the differences between compared text written in HTML are now represented as rich text, enabling you to see all changes in formatting as well.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



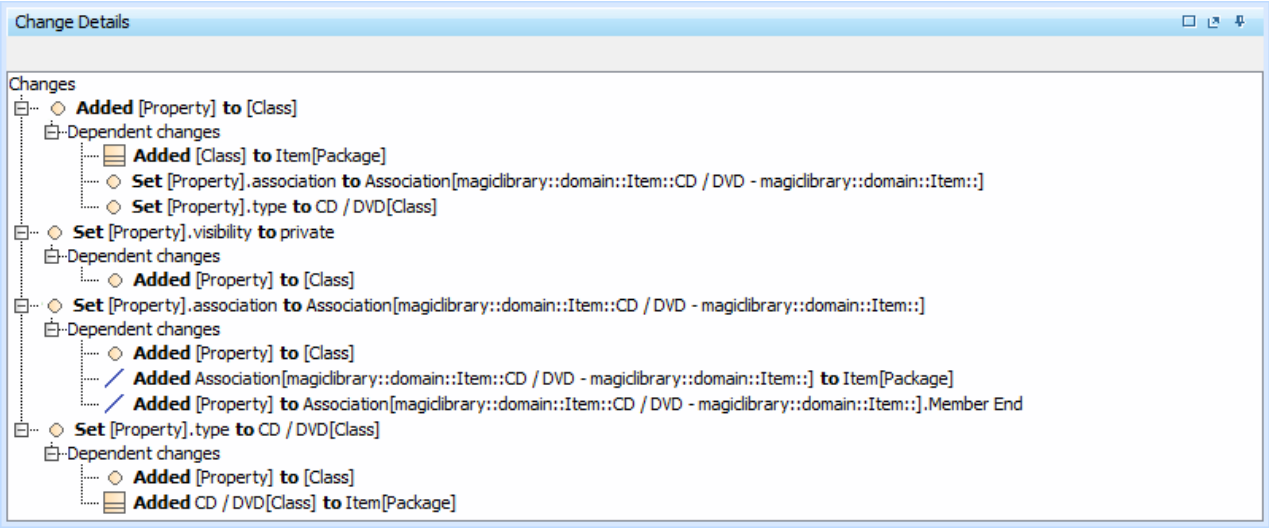
To see the differences of compared texts, do either

- Select a property with a modified text on the **Specification** panel and then click on the toolbar.
- Double-click the property on the **Specification** panel. The **Compare property** dialog opens.

Inspecting differences in the Change details panel

The **Change details** panel displays a tree structure reflecting the differences between the two versions of either the element selected in the **Differences** tree or the property selected in the **Specification** panel. If several elements or properties are selected, then all changes that have been detected in these elements or properties accordingly will be displayed in the **Change details** panel. The panel is located above the **Specification** panel on the **Difference Viewer** window.

The panel contains a tree structure that has 1 top-level branch – Changes.



In this panel, you can see dependent changes. In order to see them, expand the **Dependent changes** branch under an element with dependent changes.

For the dependent changes description, refer to Dependent changes(see page 0).

Displaying and navigating through differences

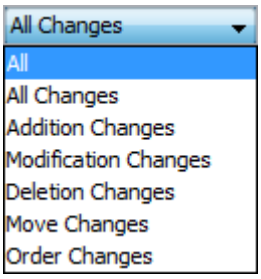
You can customize the content of the **Differences** tree as well as navigate through differences in both the **Difference** tree and the **Specification** panel using the toolbar that is located at the top of the **Difference Viewer** window.

The toolbar buttons for customizing the **Differences** tree and navigating through differences are described in the following tables.

Button / Shortcut keys	Tooltip text	Description
	Expand	Click to expand all nodes in the **Differences** tree.
	Collapse	Click to collapse all nodes in the **Differences** tree.
Alt+Home	Go To First Change	Click to select the first change. The button is not available, if the first change is already selected.

Alt+ up arrow	Go To Previous Change	Click to select the previous change.
Alt + down arrow	Go To Next Change	Click to select the next change.
Alt+End	Go To Last Change	Click to select the last change. The button is not available, if the last change is already selected.
	Report	Click to generate a project comparison differences report in Microsoft Word format.
>	Filter	Click to open the **Items Filter** dialog. Clear the check-box next to an element that you need to hide in the **Differences** tree.
>	Show Auxiliary Resources	Click to show or hide in the **Differences** tree the profiles and used projects with the «auxiliaryResources» stereotype applied (for example, UML Standard Profile).
>	Reset Windows Configuration	Click to place all tabs and windows to their default positions.

You can also filter the elements displayed in the **Differences** tree by change type. Use the **Display** drop-down list to specify a change type. Be advised that filtering elements by a particular change type displays both elements with the particular changes and elements with the particular owned element changes.



Viewing and analyzing differences of modified diagrams

The different diagram versions are displayed in separate **Difference Viewer** tabs in the **Difference Viewer** window next to the **Differences** tree (see the following figure).

To open diagram difference viewers of a modified diagram

- Double-click the modified diagram in the **Differences** tree.

One, two or three new tabs open in the **Difference Viewer** window. The base tab is opened by default. If a diagram is only available in compared, only this tab is opened in the difference viewer.

All the diagram difference viewers are used to display the following changes:

- One or more symbols were moved or modified on the diagram.
- A new symbol was added to the diagram.
- A symbol was deleted from the diagram.
- The number of inner model elements has changed.
- A model element has been modified.

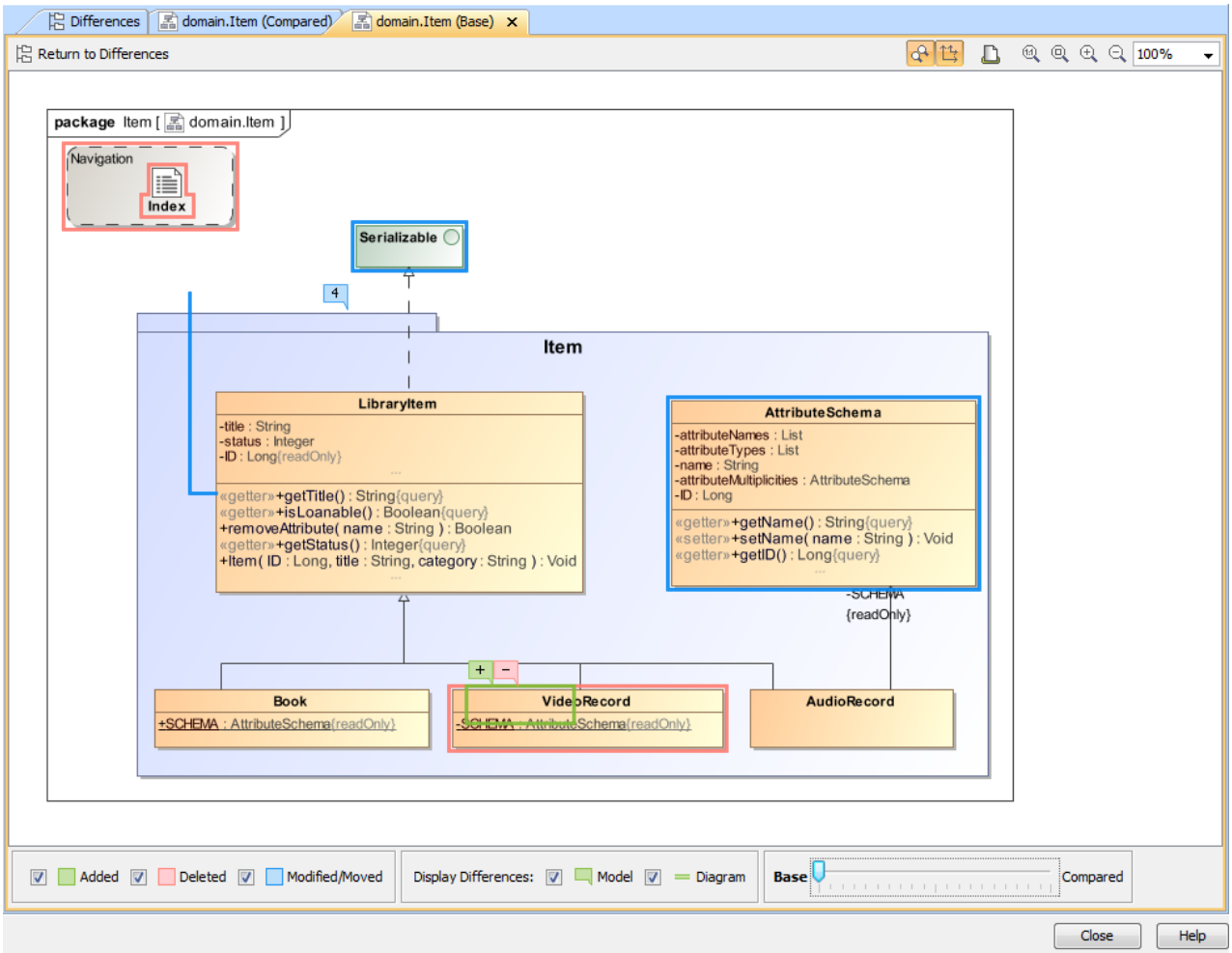
> Be advised that symbols with property value changes that actually have no effect on their appearance are highlighted in a diagram difference viewer as well. For example, changing the **Show Stereotypes** property value in the **Symbol Properties** dialog of a class makes the class symbol highlighted as changed in a diagram difference viewer.

As you can see in the following figure, all these changes are marked by color coded lines and flags allowing the quick distinguishing between different types of changes. Numbers on flags indicate the number of changes made to that element. Clicking a flag switches to the **Differences** tab and selects the flagged element in the **Differences** tree.

The colors of lines and flags correspond to the color system explained in the difference legend:

- Blue flags indicate the number of modification/movement changes made to the element, blue lines represent modified/moved relations, and blue borders around symbols mark modified/moved elements in the diagram.
- Green flags indicate the number of addition changes made to the element, green lines represent added relations, and green borders around symbols mark added elements in the diagram.
- Red flags indicate the number of deletion changes made to the element, red lines represent deleted relations, and red borders around symbols mark deleted elements in the diagram.

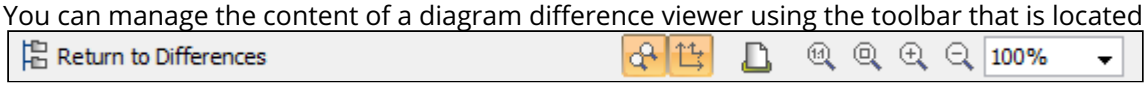
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



You can customize the types of changes you want to be displayed in the difference viewer by using filtering options available in the legend. Simply select or clear selection from the appropriate check-boxes and view the differences that are relevant to you.

If you would like to see the diagram version from other project version without switching between tabs, use the slider available at the bottom of the **Difference Viewer**. Moving the slider makes the current diagram symbols to fade, and the symbols from another diagram version start appearing. When the slider is set to the middle position, different symbols from both diagram.

You can manage the content of a diagram difference viewer using the toolbar that is located above it.



See the button descriptions in the following table.

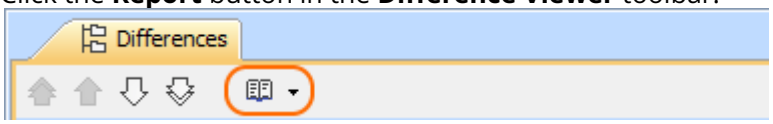
Button	Tooltip text	Description
	Return to Merged Result	Click to open the **Differences** tree with the appropriate diagram selected within.
	Synchronize Zooming	Click to turn on or off the zooming synchronization in all difference viewers of the same diagram.
	Synchronize Scrolling	Click to turn on or off scrolling synchronization in all difference viewers of the same diagram.
	Print Diagram	Click the button to print a diagram. The diagram will be printed with highlighted areas on it, if the highlighting of the diagram changes is turned on.
	Zooming	Use the buttons to control the diagram size.

Generating a project comparison differences report

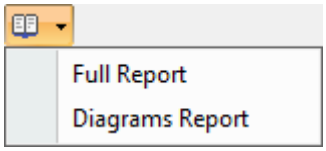
When you're done reviewing differences and wish to see them *on paper*, you can simply export all the diagrams with visible changes to a Microsoft Word document using one of the predefined templates or your custom report template.

To generate a differences report using a predefined template

1. Click the **Report** button in the **Difference Viewer** toolbar.



2. Choose the preferred template – **Full Report** for a more detailed overview, including the entire **Differences** tree with all changes, or **Diagrams Report**.



3. The **Save Report** dialog opens. Specify the location in your file system for your report to be stored, type the file name and click **Save**.

If you need to slightly modify the report, for example, to show less data or add your company's information, you can edit the generated Microsoft Word document as per your needs.

However, if you require more control over what's represented in reports, you can customize report templates and generate differences reports of any desired depth and scope.

Creating custom report templates

All the .docx files found in the Merge Reports directory are treated as report templates, therefore you can copy and edit any of the existing templates, then save it as your custom report template with a different name. You will see your saved template in the drop-down list of available templates when you click the **Report** button .

You can read more about creating and modifying report templates in the **Report Wizard Template Creation Tutorial**[271].

Validation

On this page

- Purpose(see page 1306)
- Validation suites(see page 1306)
- Validating your model(see page 1307)
- Invalid elements representation(see page 1307)

Purpose

Validation checks the accuracy, completeness, and correctness of a model, marks invalid elements in the model, and suggests solutions.

Validation suites

Models are validated against a set of validation rules. Each validation rule captures an imperative condition that must be checked against the model. Validation rules are grouped into meaningful groups called validation suites. There are predefined validation suites that you can customize. You can also create your own rules and group them into suites. Some validation rules are active; that is, they run all the time. Validation rules and suites can be exported into a separate project in order to reuse them. They are stored in the Teamwork Server or Teamwork Cloud server for exchange.

- Learn more about predefined validation suites >>(see page 1308)
- Learn more about SysML validation suites >>[272]
- Learn more about active validation suites >>(see page 1310)
- Learn more about SysML active validation suites >>[273]
- Learn more about Requirements validation suites >>[274]
- Learn how to create new validation suites >>(see page 1323)
- Learn how to create new validation rule >>(see page 1328)

271 http://www.nomagic.com/files/manuals/MagicDraw%20ReportWizard%20Template%20Creation%20Tutorial.pdf
272 https://docs.nomagic.com/display/SYSMLP190SP4/SysML+validation+suites
273 https://docs.nomagic.com/display/SYSMLP190SP4/SysML+active+validation+suites
274 https://docs.nomagic.com/display/CRMP190SP4/Requirements+validation+suites

- Learn more about global validation rules >>(see page 1339)

Validating your model

The validation can run on the entire model or on a selected part of it. All the elements in that scope are checked against relevant validation rules, and elements that violate these rules are reported in the **Validation Results** panel(see page 1316). They are also marked in the Model Browser(see page 29), and their symbols are highlighted. Diagrams, tables, matrices, and relation maps displaying these highlighted symbols have the validation results marker bar.

- Learn how to start the validation >>(see page 1314)
- Learn more about validation results marker bar >>(see page 1315)
- Learn more about Validation Results panel >>(see page 1316)
- Learn more about Validation dialog >>(see page 1321)
- Learn more about Pre-Commit Validation >>(see page 1341)

Invalid elements representation

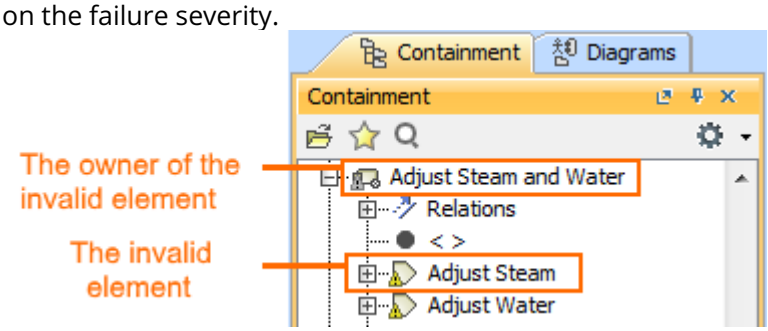
The invalid elements are marked as follows:

- - error or fatal error.
- - warning.
- - debug or info.

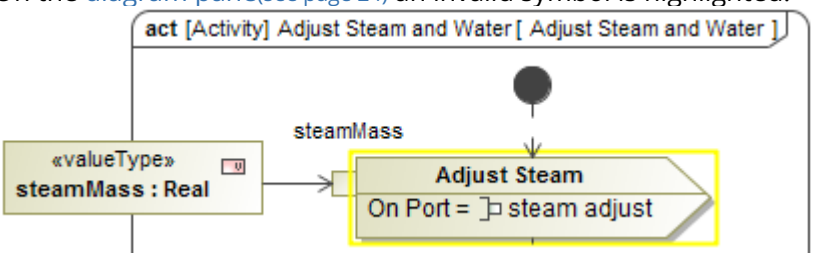
> Learn how to customize severity levels >>(see page 1332)

You can see invalid elements in the following areas:

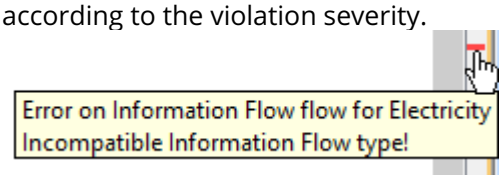
- In the Model Browser(see page 29) an invalid element is marked with a small symbol that depends on the failure severity.



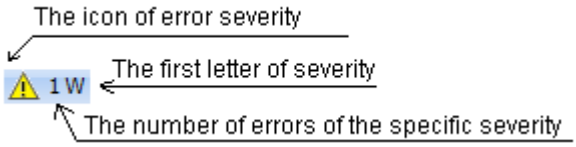
- On the diagram pane(see page 24) an invalid symbol is highlighted.



- In the validation results marker bar(see page 1315) every marker of an invalid symbol is colored according to the violation severity.



- On the status bar(see page 24) the failure indicator displays the severity icon, number of errors, and the first letter of the error severity (F - fatal error; E - error; W - warning; D - debug; I - info).



Solvers of invalid elements
You can analyze invalid elements and solve problems by selecting appropriate commands from the invalid element or symbol shortcut menu or in the **Validation Results** panel(see page 1316).

Related pages

- Predefined validation suites(see page 1308)
- Active validation suites(see page 1310)
- Starting the validation(see page 1314)
- Validation results marker bar(see page 1315)
- Validation Results panel(see page 1316)
- Validation dialog(see page 1321)
- Custom validation(see page 1323)
- Pre-commit validation(see page 1341)

Predefined validation suites

On this page

- Introduction(see page 1308)
- Limiting the scope of the constraints to be validated against(see page 1309)
- Customizing validation options(see page 1309)

Introduction

You can validate your model against a set of constraints called validation suites. You must perform the validation manually to check your model against the predefined validation suites. The predefined validation suites have the «validationSuite» stereotype applied.

Each Profile contains its own validation suites. You can create your own validation suites(see page 1323).

The following predefined validation suites appear in the **Validation** dialog(see page 1321) or **Validation Suites** dialog(see page 1325):

- **Composition Integrity** contains validation rules that check the structure of the project, such as recovered elements, references, versions of project usages, etc.
- **Project Integrity Correctness** contains validation rules that check cyclical dependencies between current project and used projects.
- **Spelling** contains validation rules that check the spelling of properties and tagged values.
- **Text References Validation** contains validation rules that check if the hyperlink text and the referenced element are coherent.
- **UML completeness constraints** contains validation rules that check if the model is complete, that there are no gaps, and the essential information fields in the elements have been filled in (e.g. checks that all the properties have type specified etc.).
- **UML correctness constraints** contains validation rules that check if the model is correct, that related information is consistent across the model (e.g. the multiplicity between elements coincides between their instances).

You can find predefined validation suites in the Containment tree(see page 32) when the **Show Auxiliary Resources** option is enabled, expand the *UML Standard Profile* Package> *Validation Profile* Package.
To show the auxiliary resources

- In the Model Browser(see page 29), click and enable the **Show Auxiliary Resources** option.

Learn how to start the validation >>(see page 1314)

Limiting the scope of the constraints to be validated against

If you select the **All Validation Suites** as the **Validation Suite** in the **Validation** dialog(see page 1321), your model is validated against all validation suites. To limit the scope of the constraints to be validated against, you can select a more specific validation suite, such as **Numbering Validation**. The model or its part is validated against the selected validation suite. The **Validation Results** panel(see page 1316) shows the results, listing all elements that do not conform to constraints in the selected validation suite. These elements are called “invalid” elements and are highlighted.

Customizing validation options

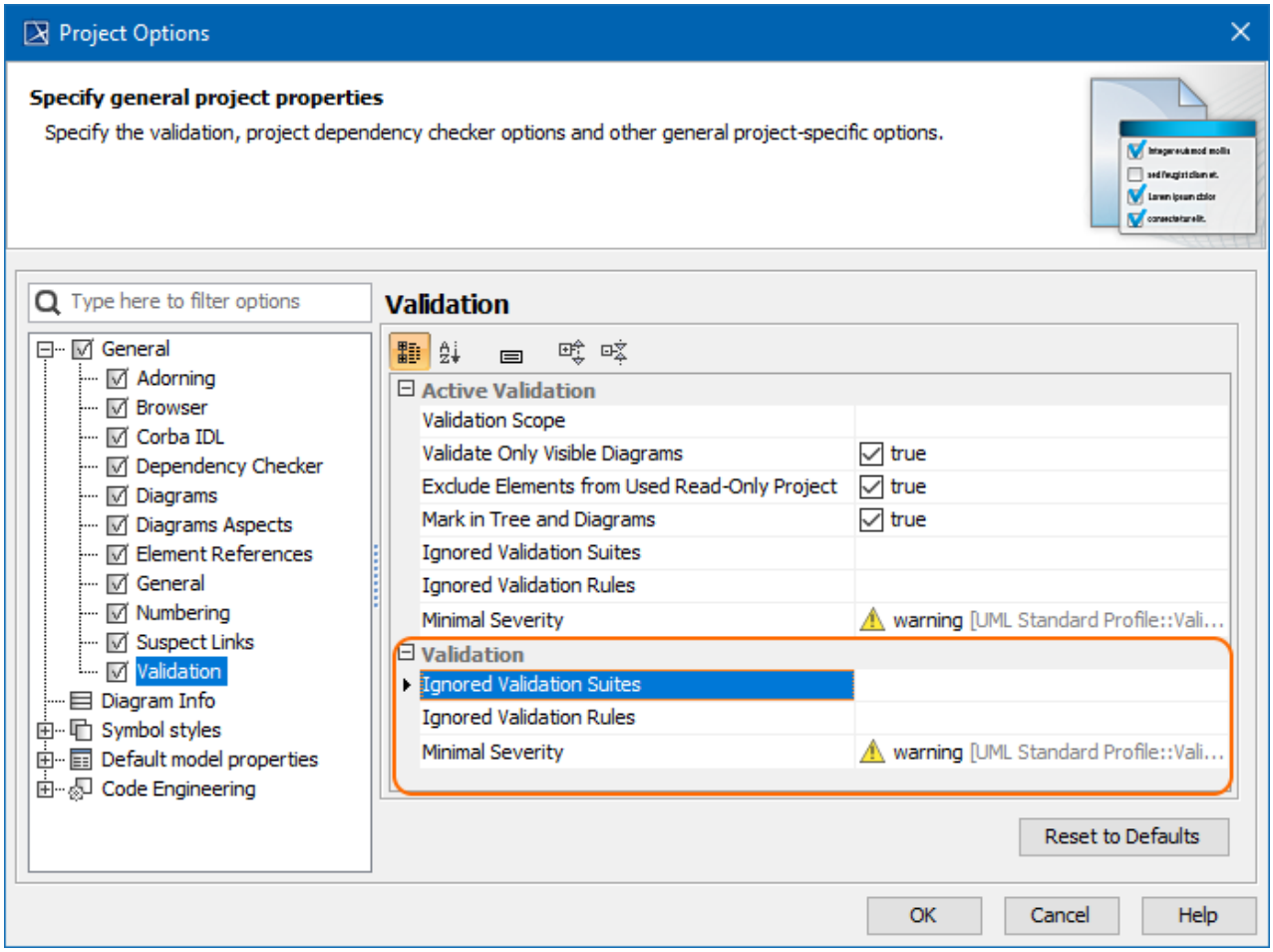
You can customize the Validation options in the **Project Options** dialog.

To open the Validation options

1. Open the **Project Options** dialog in one of the following ways:
   - On the main menu, select **Analyze** > **Validation** > **Validation Options**.

- On the main menu, select **Options** > **Project**. In the **Project Options**(see page 214) dialog, select **General** > **Validation**.

2. Specify the options you need in the **Validation** area. How to use the **Project Options** dialog >>(see page 214)
3. Click **OK**.



Validation options in the Project Options dialog.

Related pages

- MagicDraw active validation suites(see page 1310)
- SysML validation suites[275]
- SysML active validation suites[276]
- Requirements validation suites[277]

Active validation suites

275 https://docs.nomagic.com/display/SYSMLP190SP4/SysML+validation+suites
276 https://docs.nomagic.com/display/SYSMLP190SP4/SysML+active+validation+suites
277 https://docs.nomagic.com/display/CRMP190SP4/Requirements+validation+suites

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

On this page

- Introduction(see page 1311)
- Validating a project with active validation suites(see page 1312)
- Selecting active validation mode(see page 1312)
- Specifying the active validation period(see page 1312)
- Customizing active validation options(see page 1313)
- Turning off the parameters synchronization(see page 1314)

Introduction

Active validation suites enable you to immediately see if a model is correct and complete, instantly displaying any errors in the model and suggesting appropriate solutions. The active validation suites have the «activeValidationSuite» stereotype applied.

> Each profile contains its own active validation suites. You can create your own validation suites(see page 1323).

You can see the following active validation suites in the **Validation** dialog(see page 1321) or **Validation Suites** dialog(see page 1325):

- **Numbering Validation** contains constraints that check if the element number is unique.
- **Parameters Synchronization** contains constraints that keep your model valid and synchronized with your changes. You can turn off parameters synchronization >>(see page 1314)
- **Path Overlapping** contains constraints that check to ensure that paths do not overlap.
- **Pins Displaying** contains constraints that check to ensure that actions do not contain hidden Pins.
- **Relations Ownership** contains constraints that check if a relationship has the same owner as its end elements.
- **Shape Ownership** contains constraints that check the element and path owner: if the symbol owner on the diagram correctly reflects the element owner in the model and the element, and If the diagram on which the symbol is drawn belongs to the same owner.
- **Text References Validation (Auto)** contains constraints that check if the hyperlink text and the referenced element are coherent.
- **UML Correctness** contains constraints that check the Ports compatibility, Pin types compatibility, Slot and Tags multiplicity correctness, etc.
- **Diagram Merge** contains constraints that check if the diagrams or symbols are merged correctly.

> You can find active validation suites in the Containment tree(see page 32) when the **Show Auxiliary Resources** option is enabled. Expand the *UML Standard Profile* Package > *Validation Profile* Package > *Active Validation* package.
> To show the auxiliary resources
>
> - In the Model Browser(see page 29), click and enable the **Show Auxiliary Resources** option.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Validating a project with active validation suites

To manually validate a project with active validation suites, you can select the specific validation suite (mentioned above) as the **Validation Suite** in the **Validation** dialog.(see page 1321)

How to start the validation >>[278]

Selecting active validation mode

Active validation automatically runs in the background according to the selected active validation mode.

To select the active validation mode

1. Do one of the following:
   - On the main menu, select **Analyze** > **Validation** > **Active Validation**.
   - On the main menu, select **Options** > **Environment.** On the left of the **Environment Options** dialog, select **Active Validation** group. Select the **Active Validation** option.
2. Select one of the following modes:
   - **All** - enables all (user defined and system) active validation suites.
   - **System** - enables validation suites that are critical to the model. For example, all projects have the UML Standard Profile, which has the Composition Integrity system validation suite. This suite contains validation rules allowing for detecting shared project usage, recovered element, or illegal reference related problems.

     Info
     You can check these rules under the **Auxiliary Resources** > **UML Standard Profile** > **Validation Profile** > **Composition Integrity**.
     To show the auxiliary resources

     - In the Model Browser(see page 29), click and enable the **Show Auxiliary Resources** option.
3. Click **OK**.
   Selected validation suites are active for all projects, not just the current project.

Specifying the active validation period

You can control how often, in seconds, a modeling tool will analyze the rules to be executed and how often the modeling tool will execute these rules. After opening a project, all required validation rules will be executed, including system or all (user and system), and next time these rules will be checked only after the specified period.

To specify the active validation period

278 https://docs.nomagic.com/display/SYSMLP190SP4/SysML+validation+suites#SysMLvalidationsuites-startStartingthevalidation

1. On the main menu, select **Options** > **Environment.**
2. On the left of the **Environment Options** dialog, select **Active Validation** group.
3. Select the **Active Validation Period (seconds)** option.
4. Type its value in seconds.

   Specify a higher time period to ensure better performance of the application.

5. Click **OK.**
   The specified period is used for all projects, not just the current project.

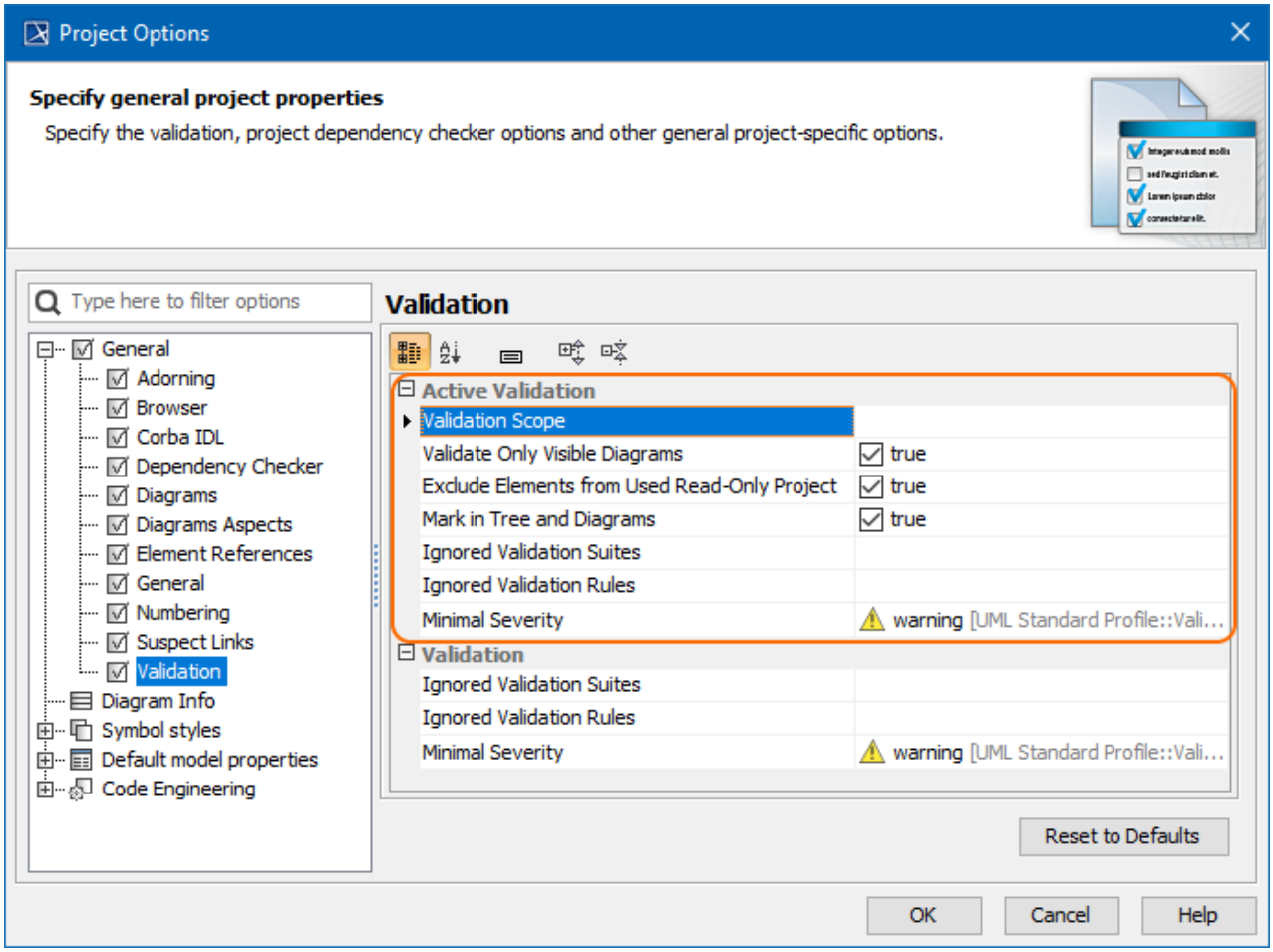
Customizing active validation options

You can customize the Active Validation options in the **Project Options** dialog.

To open the Active Validation options

---

1. Open the **Project Options** dialog in one of the following ways:
   - On the main menu, select **Analyze** > **Validation** > **Active Validation Options**.
   - On the main menu, select **Options** > **Project**. In the **Project Options**(see page 214) dialog, select **General** > **Validation**.
2. Specify the options you need in the **Active Validation** area. How to use the **Project Options** dialog >>(see page 214)
3. Click **OK**.



Optimizing performance

- Set the **Validate Only Visible Diagrams** option value to *true*. This limits a validation scope to visible diagrams only.
- Set the **Exclude Elements from Used Read-Only Project** option value to *true* to reduce the number of elements to check in your model.
- Smart package content (separate elements) will be validated only if Smart package is set directly as a scope of the active validation.
- If smart package is set indirectly for the validation scope, modeling tool will automatically exclude it from the active validation scope, because elements in the smart package are owned by other elements.

Turning off the parameters synchronization

To turn off the parameters synchronization

1. On the main menu, select **Options** > **Project**.
2. In the **Project Options** dialog, select **General** > **General**.
3. Find the **Synchronize Parameters and Arguments** option and set its value to *false*.

Related pages

- MagicDraw predefined validation suites(see page 1308)
- SysML validation suites[279]
- SysML active validation suites[280]
- Requirements validation suites[281]

Starting the validation

Validation can run on:

- The entire model, or its part (e.g. several packages).(see page 1314)
- Diagram, table, matrix, or relation map(see page 1315)
- Elements(see page 1315)

To start the validation of the entire model or its part

1. Open the **Validation** dialog. How to open the Validation dialog >>(see page 1321)
2. Specify the following options:
   - **Validation Suite**.
   - **Validate For**.

279 https://docs.nomagic.com/display/SYSMLP190SP4/SysML+validation+suites
280 https://docs.nomagic.com/display/SYSMLP190SP4/SysML+active+validation+suites
281 https://docs.nomagic.com/display/CRMP190SP4/Requirements+validation+suites

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- **Minimal Severity**.
  Learn more about each of the option >>(see page 1321)
3. Click **Validate**.
   The model or its part is validated against the selected validation suite.

To validate the diagram, table, matrix, or relation map

- On the toolbar of diagram, table, matrix, or map, click the **Validate Diagram** button .
- Right-click the diagram, table, matrix, or map in the Containment tree(see page 32), select **Validation** > **Validate Diagram**.

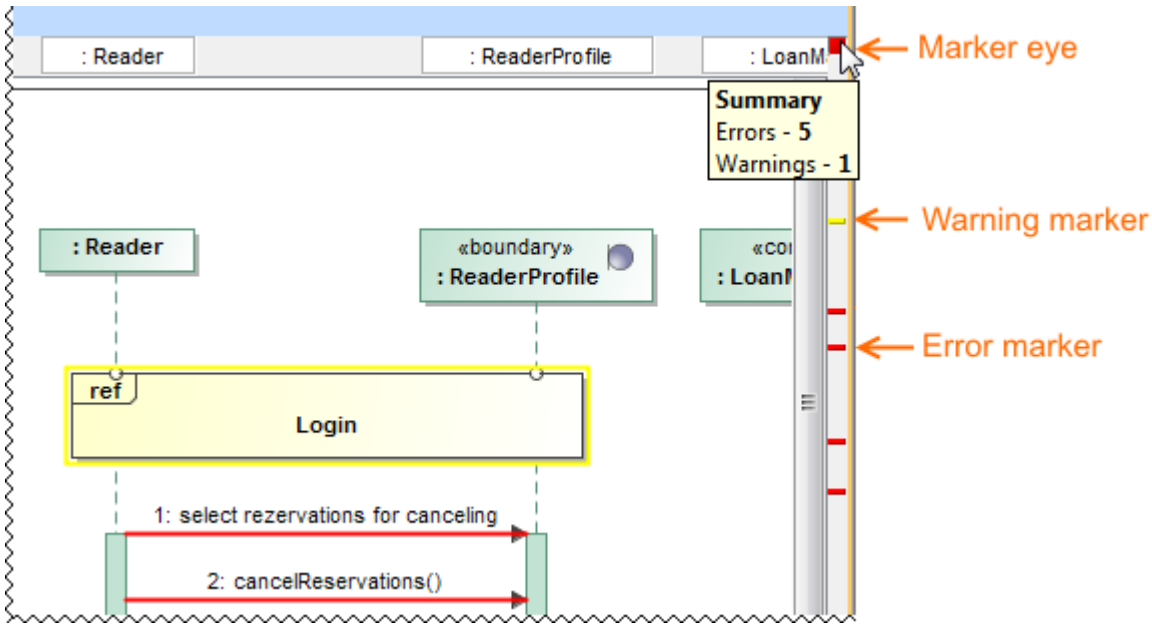
To validate the element(s)

---

1. Right-click the element in the Containment tree(see page 32).
2. In the shortcut menu, point to **Validation** and select:
   - **Validate Element** - to validate only selected element.
   - **Validate Elements Recursively** - to validate all child elements of the selected element.

Validation results marker bar

The validation results marker bar appears at the right side of a diagram pane, if the diagram contains symbols of invalid elements.



The markers are shown even for the items that are not currently visible on the screen (you may need to scroll down to see them). The marker bar shrinks to fit the screen and displays relative positions of annotated symbols on the axis of ordinates. Every marker is colored according to the violation severity.

Clicking a marker on the bar selects the appropriate symbol on the diagram pane. Moving the mouse over the marker eye (the red square at the top-right corner of the diagram pane) opens the summary of the diagram validation results.

Validation Results panel

On this page

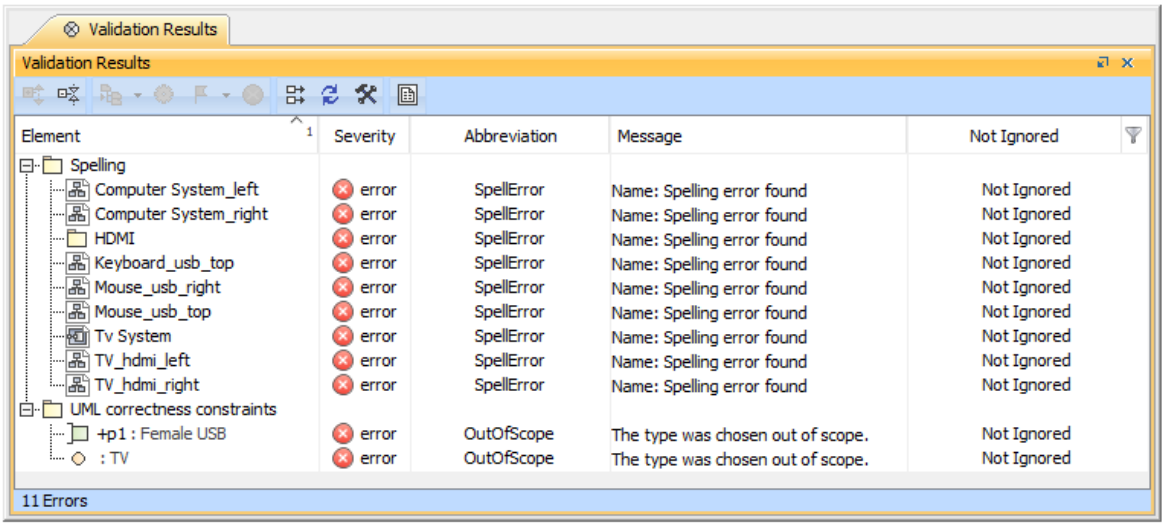
- Introduction(see page 1316)
- Opening the Validation Results panel(see page 1317)
- Description of Validation Results panel areas(see page 1317)
  - Toolbar(see page 1318)
  - Validation results area(see page 1320)
  - Summary of severity area(see page 1321)

Introduction

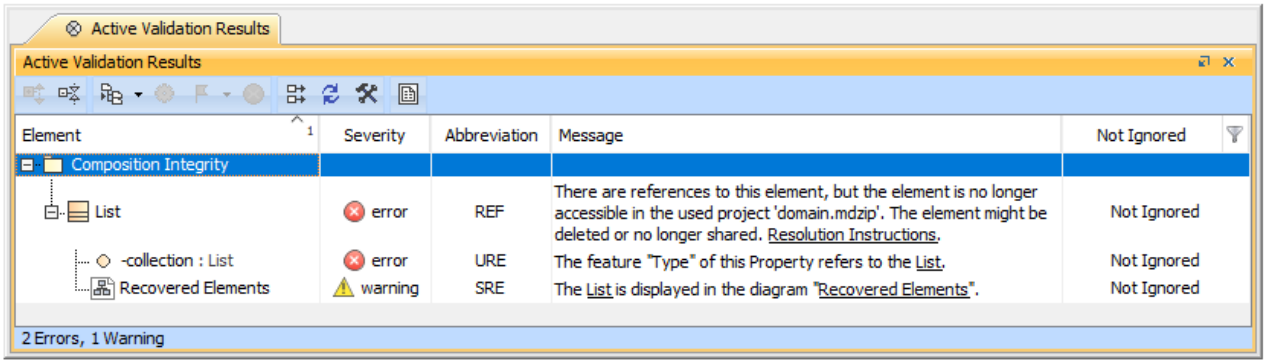
The **Validation Results** panel and **Active Validation Results** panel display the results of the validation: invalid element, severity, explanation of the validation rule and its abbreviation.The **Active Validation Results** panel shows the results of active validation suites(see page 1310), while the **Validation Results** panel shows predefined validation suites(see page 1308).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Validation Results panel



Active Validation Results panel



The Validation Results and Active Validation Results panels.

Opening the Validation Results panel

The **Validation Results** panel opens automatically after the validation has ended. How to start the validation >>(see page 1314)

Since the active validation suites always check your model's correctness, you must open the **Active Validation Results** panel to see the results of active validation.

To open the **Active Validation Results** panel

- On the status bar(see page 24), click the failure indicator (e.g.

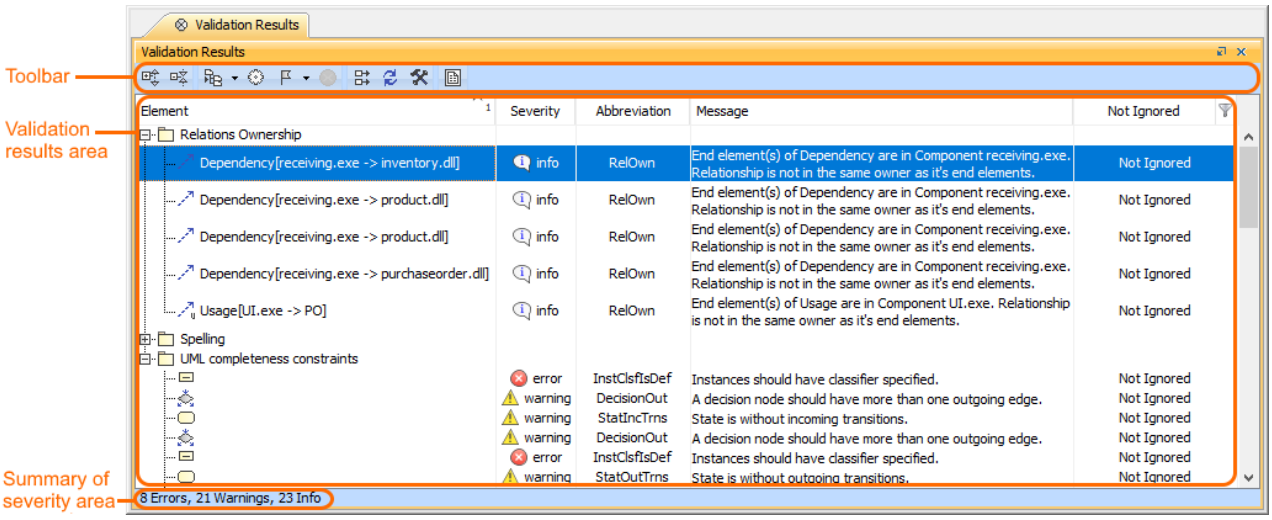
).

> If you cannot see the failure indicator on the status bar, your model is correct according to active validation suites.

Description of Validation Results panel areas

The **Validation Results** panel consists of the following areas:

-
  - Toolbar(see page 1318)
  - Validation results area(see page 1320)
  - Summary of severity area(see page 1321)



The Validation Results panel areas: toolbar, validation results, and summary of severity areas.

Toolbar

The toolbar allows you to manage the validation results. All buttons are described in the following table.

Button icon	Button name	Description
	Expand All	Click to expand recursively all the elements in the validation results area.
	Collapse All	Click to collapse recursively all the elements in the validation results area.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Button icon	Button name	Description
	Navigate to Validated Object (Alt+B)	Click the button to select the element in the Containment tree(see page 32). Click the arrow next to the button to open the menu of the following navigation commands: • **Open Specification** - opens Specification window(see page 913) of the selected element in the validation results area. • **Go To** - opens a submenu whose commands allow you to track and navigate to the elements. • **Select Rule in The Containment Tree** - selects the failed validation rule in the Containment tree(see page 32) of the selected element in the validation results area. • **Add to Search Results** - selects an element in the Search Results tree(see page 47). • **Open Project Usage Map** - opens the **Project Usage Map**. This command is available only for Teamwork Server(see page 1316) projects.
	Solve	Click to open the list of available commands to solve a selected element violation in the validation results area.
	Ignore	Click to ignore a selected element violation in the validation results area. In server projects, you can select whether to ignore the violation for you as a project user or for all users who works on that server project. The button is available when you select an unignored element in the validation results area. Click the arrow next to the button to open the menu of the following commands: • **Ignore Validation Rule** - ignores the validation rule of a selected element in the validation results area. • **Ignore Validation Suite** - ignores the validation suite of a selected element in the validation results area.

Button icon	Button name	Description
	Unignore	Click to make a selected element in the validation results area violation unignored. The button is available when you select an ignored element in the validation results area.
	Display the full path of the elements	Click to show the full path of all elements in the validation results area.
	Run validation with current settings	Click to run the validation with current settings.
	Run validation with new settings	Click to open the **Validation** dialog(see page 1321) and run the validation with new settings.
	Export	Click to export the list of validation results either to .html or .csv file.

Validation results area

The validation results area provides a list of violated elements. This area contains the following columns:

- **Element** - elements that violate the validation rule.
- **Severity** - severity level of the validation rule.
- **Abbreviation** - a short word, phase, or acronym of the validation rule used for sorting/filtering.
- **Message** - description of the invalid situation when a validation rule fails.
- **Is Ignored** - indicator of whether the violation is ignored or not.

You can use the validation results area to:

- Manage validation results(see page 1320)
- Sort validation results(see page 1321)
- Filter validation results(see page 1321)

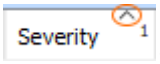
To manage validation results

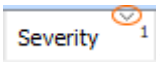
- Right-click the element in the validation results area and select one of the following commands from the shortcut menu:
    - Solver name

- **Ignore**
- **Navigate to Validated Object**
- **Open Specification**
- **Go To**
- **Select Rule in The Containment Tree**

All those commands are described in the table above(see page 1318).

To sort the validation results

- Click the header of the column you want to sort the validation results. A small arrow appears on the column header:
  - 

- ascending sorting.
  - 

- descending sorting.

To filter the validation results

---

1. Point to the header of a column whose value you want to select for filtering the validation results.
2. Click at the right of the column header and select a value from the drop-down list.

Summary of severity area

The summary of the severity area displays the number and severity of violated elements in expanded packages. If you collapse the package in the validation results area, the number of violated elements are excluded from the count.

Validation dialog

On this page

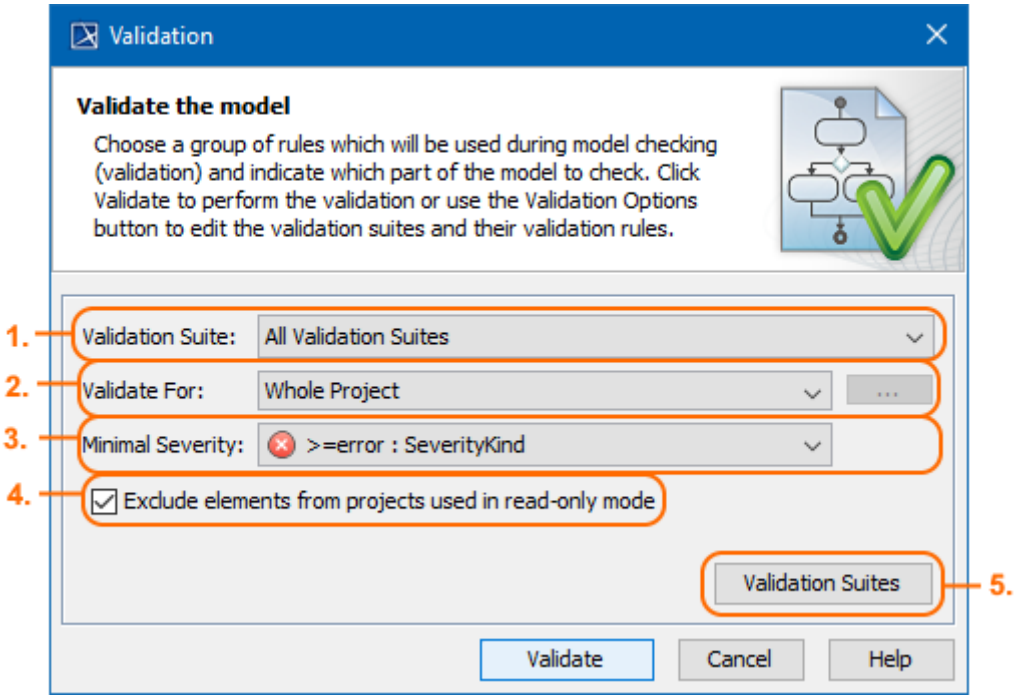
- Introduction(see page 1321)
- Opening the Validation dialog(see page 1322)
- Description of Validation dialog options(see page 1322)

Introduction

The **Validation** dialog allows you to limit the scope of the constraints to validate against your model. Since the active validation suites(see page 1310) check your model all the time, the validation according predefined validation suites (see page 1308)must be started by using this dialog. If you select the **All Validation Suite** as the **Validation Suite** in the **Validation** dialog, your model is validated against all

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

predefined(see page 1308) and active validation suites(see page 1310) at the same time. To limit the scope of the constraints to be validated against, select a more specific validation suite, such as *Numbering Validation*. The model or its part will be validated against the selected validation suite only. The **Validation Results** panel(see page 1316) shows the results, listing all elements that do not conform to some constraints (invalid elements) in the selected validation suite.



The Validation dialog. Its options (1,2,3,4,5) are described below.

Opening the Validation dialog

To open the **Validation** dialog

1. Do one of the following:
   - On the main menu, select **Analyze** > **Validation** > **Validate**.
   - On the Validation toolbar, click

   .
2. In the **Validation** dialog, specify the options. Descriptions of options >>(see page 1322)
3. Click **Validate**.
   The model or its part is validated against the selected validation suite. The results are shown in the **Validation Results** panel(see page 1316).

> You can run the validation for the entire diagram, table, matrix, relation map, or elements.
> Learn how to start the validation >>(see page 1314)

Description of Validation dialog options

The **Validation** dialog consists of the following:

- The **Validation Suite** drop-down list (#1 in the image above) allows you to select a validation suite that you want to validate the model or its part against.

  > ⚠ The list of available validation suites depends on the open project; the validation suites and validation rules are stored in the model as regular model elements.

- The **Validate For** drop-down list (#2 in the image above) allows you to choose:
  - **Whole Project** to run validation on the entire model.
  - **Validation Selection** to run validation on special packages and/or elements. Then click the ... button and select the scope for validation.

    > ⚠ Validation is always recursive. If you select a package for validation, you do not need to select its inner elements.

- The **Minimal Severity** drop-down list (#3 in the image above) allows you to select the minimal severity level: **debug**, **info**, **warning**, **error**, **fatal**. Debug is the lowest possible severity level.
- Clear the **Exclude elements from projects used in read-only mode** check box (#4 in the image above) if you want to run validation on the elements stored in the the read-only project usages.
- The **Validation Suites** button (#5 in the image above) opens the **Validation Suites** dialog(see page 1325).

Custom validation

You can extend existing validation suites that come with Profile (see page 816)by customizing existing suites or creating your own validation suites and rules.

Find an appropriate topic:

- Creating validation suites(see page 1323)
  - Validation Suites dialog(see page 1325)
- Creating new validation rules(see page 1328)
  - Creating new Constraint element(see page 1329)
  - Applying validationRule stereotype(see page 1330)
  - Specifying Constraint Element property(see page 1331)
  - Specifying the validation rule properties(see page 1331)
    - Customizing severity levels(see page 1332)
  - Defining Constraint condition(see page 1337)
  - Global validation rules(see page 1339)

Creating validation suites

The validation suite defines the set of validation rules applied when running the validation. The purpose of the validation suites is to group validation rules (constraints) without duplicating them. When creating a new validation suite you must choose the active or inactive validation suite you want to create, depending on the applied stereotype:

- «validationSuite» is dedicated for inactive validation rules. Learn more about predefined validation suites >>(see page 1308)
- «activeValidationSuite» is dedicated for active validation rules. Learn more about active validation suites >>(see page 1310)

You can:

- Create your own validation suites.(see page 1324)
- Clone existing validation suites.(see page 1324)

- Include/exclude the validation rules to/from validation suites.(see page 1324)

> **Sharing**
> You can share validation suites by exporting validation suites Package(see page 237) and then use it in another project.(see page 245)

To create a new validation suite

1. Open the **Validation Suites** dialog. How to open the **Validation Suites** dialog >>(see page 1326)
2. Do the one of the following:
   - In the toolbar(see page 1325), click and select **Validation Suite** or **Active Validation Suite** command.
   - Right-click the Validation Suites area(see page 1325) and select **Create Validation Suite** or **Create Active Validation Suite** command.
3. In the **Select Owner** dialog, choose the owner for the new validation suite Package, and click **OK**.
4. In the **Rename** dialog, type the name of the new validation suite Package, and click **OK**.
   An empty validation suite Package is created and is located in your model. You can include validation rules from other validation suites. How to include validation rules >>(see page 1324)

   > **Alternative**
   > An alternative way is to create a new or select existing Package element and apply one of the following stereotypes:
   > - «validationSuite»
   > - «activeValidationSuite»
   > How to apply a stereotype >>(see page 857)

To clone existing validation suites

1. Open the **Validation Suites** dialog. How to open the **Validation Suites** dialog >>(see page 1326)
2. In the Validation Suites area(see page 1325), select the validation suite Package you want to modify.
3. Do the one of the following:
   - In the toolbar(see page 1325), click .
   - Right-click it and select **Clone Validation Suite** command.
4. In the **Select Owner** dialog, choose the owner for the cloned validation suite Package, and click **OK**.
5. In the **Rename** dialog, type the name of the cloned validation suite Package, and click **OK**.
   The clone of the existing validation suite Package is created and contains the same validation rules as the original. You can include or exclude validation rules from the cloned validation suites Package. How to include/exclude validation rules >>(see page 1324)

To include/exclude the validation rules to/from a validation suite Package

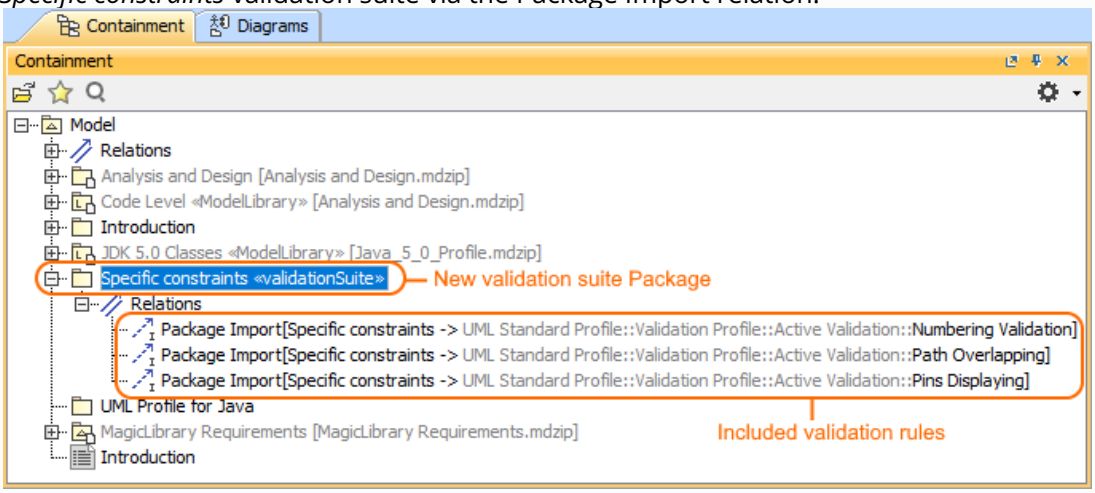
1. Open the **Validation Suites** dialog. How to open the **Validation Suites** dialog >>(see page 1326)

2. In the Validation Suites area(see page 1325), select the newly created or cloned validation suite Package.

   You cannot include/exclude the validation rules to/from read-only validation suites.

3. In the Validation Rules area(see page 1325), select/clear the check boxes for the entire Package or an appropriate validation rule to include/exclude validation rules to/from the new validation suites Package.
   The validation suites Package contains only selected validation rules.

   Additional information
   After you include/exclude the validation rules (constraints), the appropriate Package Import(see page 704) relation is created/deleted in the model. You can check that in the Containment tree(see page 32), under newly created or cloned validation suites Package, **Relations** group. In the image below, the *Numbering Validation*, *Path Overlapping*, and *Pins Displaying* validation suites with all Constraints are included in the newly created *Specific constraints* validation suite via the Package Import relation.



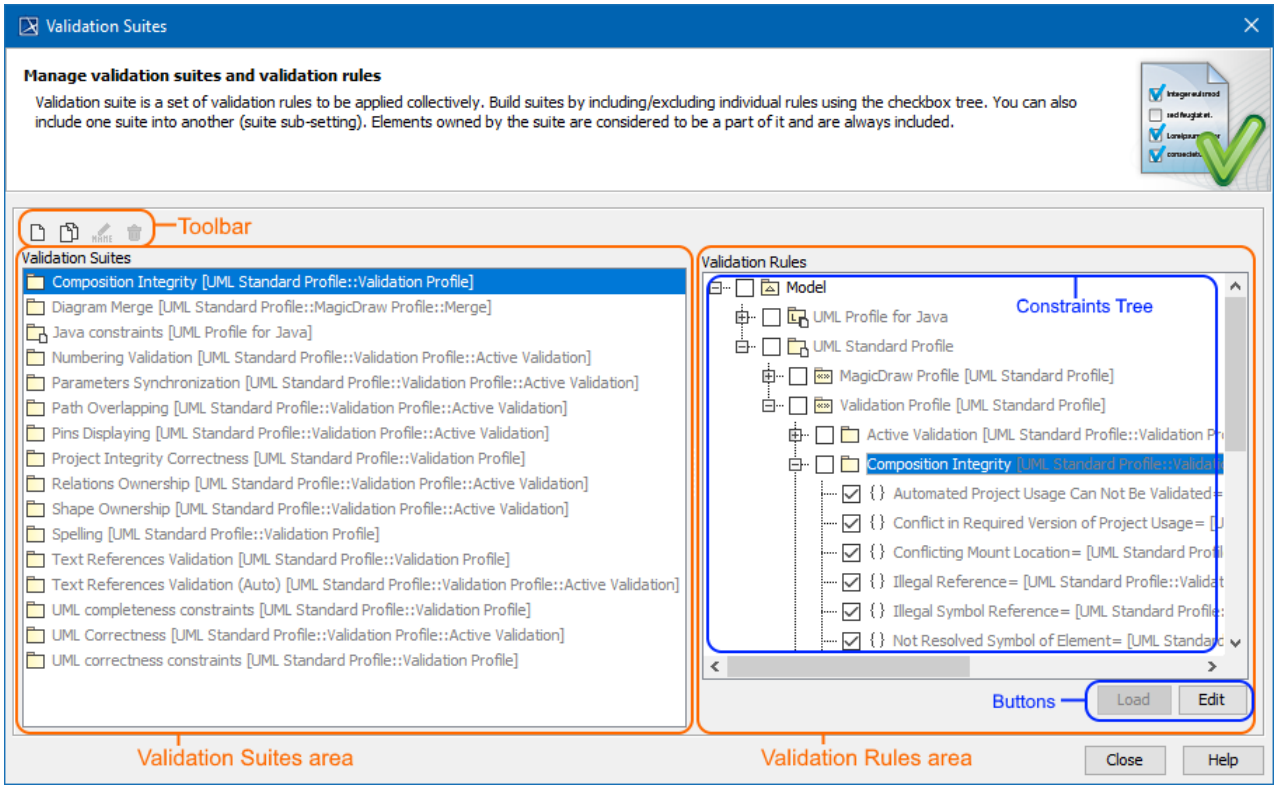
Validation Suites dialog

On this page

- Introduction(see page 1325)
- Opening the Validation Suites dialog(see page 1326)
- Description of Validation Suites dialog areas(see page 1326)
  - Toolbar(see page 1326)
  - Validation Suites area(see page 1327)
  - Validation Rules area(see page 1327)

Introduction
The **Validation Suites** dialog allows you to create new suites, clone read-only validation suites, include one suite to another, and include/exclude already existing validation rules in validation suites.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The Validation Suites dialog areas: toolbar, Validation Suites, and Validation Rules areas.

Opening the Validation Suites dialog

To open the **Validation Suites** dialog

- Do either:
  - From the main menu, select **Analyze** > **Validation** > **Validation Suites**.
  - Open the **Validation** dialog(see page 1322) and click the **Validation Suites** button.

Description of Validation Suites dialog areas

The **Validation Suites** dialog consists of the following areas:

-
  - Toolbar(see page 1326)
  - Validation Suites area(see page 1327)
  - Validation Rules area(see page 1327)

Toolbar

The toolbar allows you to manage the validation suites. All buttons are described in the following table.

Button icon	Button name	Description
	Create New Validation Suite	Click to create a new validation suite. This button contains the following commands: • **Validation Suite** - creates a new validation suite, stored in your model as a Package with «validationSuite» stereotype applied. The validation rules stored in this suite will check if a model is correct and complete only when you start your model validation manually. How to start validation >>(see page 1314) • **Active Validation Suite** - creates a new active validation suite stored in your model as a Package with «activeValidationSuite» stereotype applied. The validation rules stored in this suite will be active and check if a model is correct and complete immediately. Learn more about active validation suites >>(see page 1310)
	Clone Validation Suite	Click to clone an already existing validation suite with its validation rules. It is stored in your model as a Package with «validationSuite» stereotype applied.
	Rename (F2)	Click to rename the package of new or cloned validation suites.
	Remove	Click to delete the new or cloned validation suites from the model.

Validation Suites area
The Validation Suites area shows a list of all validation suites that comes with a profile (read-only) or your own created validation suites (editable).

> Each profile contains its own validation suites.

You can manage validation suites directly in the Validation Suites area by using the toolbar buttons(see page 1326) or shortcut menu commands: **Create Validation Suite**, **Create Active Validation Suite**, **Clone Validation Suite**, **Rename (F2)**, or **Remove**.

> All those commands are described in the table above(see page 1326).

Validation Rules area
The Validation Rules area contains:

- *Constraints Tree* displaying the structure of the validation suite Packages with their validation rules. It allows you to include/exclude validation rules to/from a newly created or cloned validation suite Package. For this select/clear the check box before the entire Package or an appropriate validation rule.

You cannot include/exclude the validation rules to/from read-only validation suites.

- *Buttons*:

Button	Description
Edit	Click to open the Specification window(see page 913) of the selected element in the *Constraint Tree*.
Load	Click to load the used project(see page 245) with its validation rules.

Creating new validation rules

You can create your own validation rules in the model and use them to check your model or keep them only for documentation purpose. You can locate new validation rules anywhere in your model, but usually they are created for the classifier, stereotype, or metaclass.

To create a new validation rule

1. Create new Constraint element.(see page 1329)
2. Apply «validationRule» stereotype.(see page 1330)
3. Specify Constraint Element property.(see page 1331)
4. Specify the validation rule properties.(see page 1331)
5. Define Constraint expression.(see page 1337)
   The new validation rule is created in the model.

Managing new validation rule
You can include a new validation rule to your own created validation suite.
- How to create validation suite >>(see page 1323)
- How to include the validation rules to validation suite Package >>(see page 1324)

You can manage new validation rules in your model as any other elements.
- How to work with model elements >>(see page 909)

Related pages

- Creating new Constraint element(see page 1329)
- Applying validationRule stereotype(see page 1330)
- Specifying Constraint Element property(see page 1331)
- Specifying the validation rule properties(see page 1331)
- Defining Constraint condition(see page 1337)
- Global validation rules(see page 1339)

Related pages in Developer Guide

- Creating validation rules(see page 2335)

Before creating a new Constraint element, you must decide the Constraint type to create. The Constraint type defines a target for which the validation rule will be evaluated.

There are three types of Constraints(see page 689) that can be evaluated:

- **Classifier level Constraints**. Constraints placed on the classes, datatypes and other classifiers (see page 461)of the model are evaluated on all the instances of these classifiers (i.e. those Instance Specifications(see page 735) that have the particular classifier set as their type). Constraints defined on some particular classifiers are evaluated on the instances of these particular classifiers when validating. Inheritance is taken into account; instances of the subclasses of the class are also validated.
- **Constraints on Metaclasses**. When a Constraint is placed on a metaclass(see page 780) (one of the classes in the UML Standard Profile > UML2 Metamodel), this Constraint is evaluated on all the model elements of that kind (e.g. if the Constraint is placed on the Actor metaclass, then this Constraint applies to all actor elements in the model). The following is an example of a rule (specified in OCL2.0) mandating that all actor names in the model must be capitalized. These Constraints are useful for specifying generic rules that must apply on all the model elements of a particular kind.

  ```
  context Actor inv capitalize:
  let startswith:String = name.substring(1,1) in
  startswith.toUpper() = startswith
  ```

- **Constraints on Stereotypes**. When a Constraint is placed on some Stereotypes(see page 856) of the Profile(see page 816), that Constraint applies to all the model elements that have these Stereotypes applied to them. These Constraints are useful when creating domain specific profiles. When adapting UML to some specific modeling domain, a Profile is usually created with extensions for that domain - Stereotypes, tags etc. The Constraints on these Stereotypes allow enforcing the rules of that domain.

Recommendation
It is strongly recommend not to mix the Constraints from different metalevels into one validation suite. Learn more how to create validation suite >>(see page 1323)

Examples of Constraint types
The examples of each Constraint type can be found in the **model validation** sample model. To open this sample do one of the following:
- Download model validation.mdzip[282].
- Find in your modeling tool *<modeling tool installation directory>\samples\product features\model validation.mdzip.*

You can create a new Constraint for :

- Classifier(see page 461), which is constrained – classes, datatypes, etc. (for classifier level Constraints). How to create a validation rule for a classifier >>(see page 1330)

28 https://docs.nomagic.com/download/attachments/55847477/model%20validation.mdzip?api=v2&modificationDate=1513353053644&version=1

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Stereotype (see page 856)(for meta-classifier level Constraints). How to create a validation rule for a Stereotype >>(see page 1330)
- Metaclass (see page 780)located in a read-only profile. How to create a validation rule for a metaclass >>(see page 1330)

To create Constraint for a classifier or Stereotype

1. Open the Specification window of a classifier or Stereotype. How to open the Specification window >>(see page 913)
2. On the left side of the window, select the **Constraints** property group.
3. On the right side of the window, click the **Create** button and select **Constraint**.
4. Type its name.
   The new Constraint is created in the model.

To create Constraint for a metaclass

1. Select a package where you want to store the rule.
2. Right-click this package, select **Create Element** > **Constraint**.
3. Type its name.
   The new Constraint is created in the model.

   > If you create a new Constraint for a metaclass, you must specify the **Constrained Element** property in the Constraint Specification window. How to specify Constrainted Element property >>(see page 1331)

Applying validationRule stereotype

After the Constraint element is created, you must apply:

- «validationRule» stereotype. It is derived from the «invariant » stereotype. It expands the Constraint properties with the ability to specify the following information: severity level (for sorting/filtering), abbreviation string (a short string, for easy recognition) and error message (complete description of error explanation). How to specify the validation rules properties >>(see page 1331)

To apply a stereotype, do one of the following:

- In the corresponding element's Specification window(see page 913), click the **Applied Stereotype** property specification cell, then click , and select stereotypes from the open list. After you have selected, click **Apply**.
- In the element's shortcut menu, select **Stereotype** and select a stereotype that you wish to apply. Click **Apply** when you are done.
- On the diagram pane, select an element and in that element's name area, type the stereotype name between brackets « ». The element's name comes after the stereotype.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

If you want to name the element Books and assign it the «table» stereotype, in the element's name area type the following: «table» Books. The name completion for the stereotypes works in the name editing mode, press Ctrl+Spacebar or Ctrl+Backspace to get a list of available stereotypes.

You can change the order of the stereotypes applied to the element. The symbol style of the first stereotype in the list will be applied to the element.

Specifying Constraint Element property

You must ensure that the constrained element is specified and points to the necessary classifier. Each validation rule modeled as a Constraint has a **Constrained Element** property. This property determines the type of element this rule applies to. If you create the Constraint for a classifier or stereotype(see page 1329), this property is filled automatically. Otherwise, you need to specify it manually, shown in the following procedure:

To specify a constrained element manually

1. Open the Constraint Specification window. How to open the Specification window >>(see page 913)
2. Find the **Constrained Element** property and select it's value box.
3. Click ... and select an appropriate element from the **Select Elements** dialog.

   For constraints on metaclasses, select an appropriate element from the **UML Standard Profile** > **UML2 Metamodel**. Only the single **Constrained Element** property value is supported for validation rules.

4. Click **OK**.
   The **Constrained Element** property value is specified.

Specifying the validation rule properties

If you applied the «validationRule» stereotype to the Constraint element, you must specify the following validation rule properties in its Specification window: **Abbreviation**, **Error Message**, and **Severity**.

To specify the validation rule properties

1. Open the Constraint Specification window. How to open the Specification window >>(see page 913)
2. On the right side of the window, find the **Validation Rule** group.
3. Specify the following properties:
   - **Abbreviation**. Select the **Abbreviation** property value box, click ... and type a short word, phase, acronym of the validation rule. It allows you to distinguish the validation rules among other rules when sorting or filtering.
   - **Error Message**. Select the **Error Message** property value box, click ... and type the description of the invalid situation when this validation rule fails.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

If you need to define the variable error message, you can embed expressions directly in the **Error Message** value box by using curly brackets - {}. Everything between them is treated as an expression and evaluated for each validation result. Expressions are treated as OCL2.0 expressions by default, but you can also use binary expressions: {bin: <binary expression>}. For example, in the following error message the budget is calculated according to variables. The result of the expression is provided as number in the **Validation Results** panel(see page 1316), e.g. Budget not balanced - overbudget by $500.

```
Budget not  balanced - overbudget by {
ResearchProject::allInstances().budget->sum() -
Sponsor::allInstances().contribution->sum()}$
```

You can insert a hyperlink. Learn more about defining hyperlinks >> (see page 1008)

- **Severity**. Select the **Severity** property value and from the drop-down list select one of the following severity levels:
  - **debug** - for missing information of a constrained element that is dedicated for the developer.
  - **info** - for missing information of a constrained element that is dedicated mostly for the end-user.
  - **warning** - for incorrect situations that can cause errors. It is used for less severe situations than the error.
  - **error** - for incorrect situations that must be solved.
  - **fatal** - for critical errors such as model corruption or model is invalid according to the UML metamodel. This severity level is mostly reserved for future use.

    The icons on invalid elements in the model depend on the selected severity level described above. Learn more about invalid elements representation >>(see page 1307)
    You can customize severity levels. How to customize severity levels >>(see page 1332)

4. Click **Close**.
   The validation rule properties are specified.

Customizing severity levels

You can customize the default severity levels (such as **debug**, **info**, **warning**, **error**, **fatal**) by creating new and and additional levels. For this you must edit the *SeverityKind* Enumeration (see page 697)element, located in the UML Standard Profile.

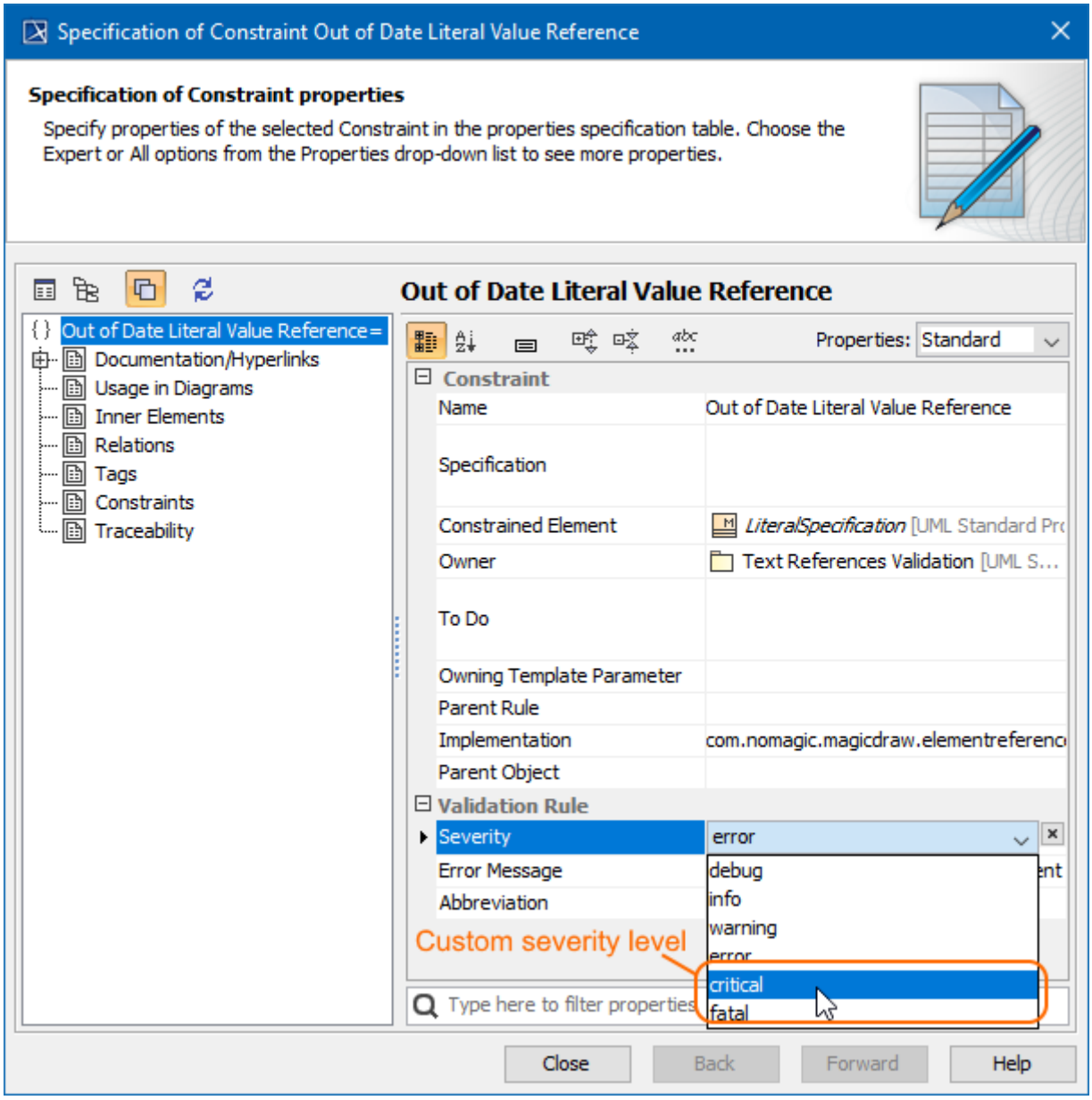
We do not recommend changing the UML Standard Profile.

To customize severity levels

1. Edit the UML Standard Profile.(see page 1333)
2. Create new or modify default severity levels.(see page 1334)
3. Define new icon for severity level.(see page 1335)
4. Apply new icon for severity level.(see page 1336)

After you create the custom severity level or modify the default severity, you can select it from from the drop-down list of the **Severity** property when specifying the validation rule properties(see page 1331). In the following figure the new severity level named *critical* can be selected as a value of the **Severity**

property in the Specification window of validation rule (Constraint element). The value order in the **Severity** drop-down list is the same as they are listed in the *SeverityKind* Enumeration element.



The custom severity level named critical in the drop-down list of the Severity property in the validation rule specification.

Editing the UML Standard Profile
To edit the UML Standard Profile

1. Open a project.
2. In the Containment tree(see page 32), click and enable to show the **Show Auxiliary Resources** in the Model Browser(see page 29).
3. Right-click the *UML Standard Profile* Package.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

4. Select **Project Usages** > **Open**. Learn how to manage project usages >>(see page 1441)
   The UML Standard Profile is now editable.

Creating new or modifying default severity levels

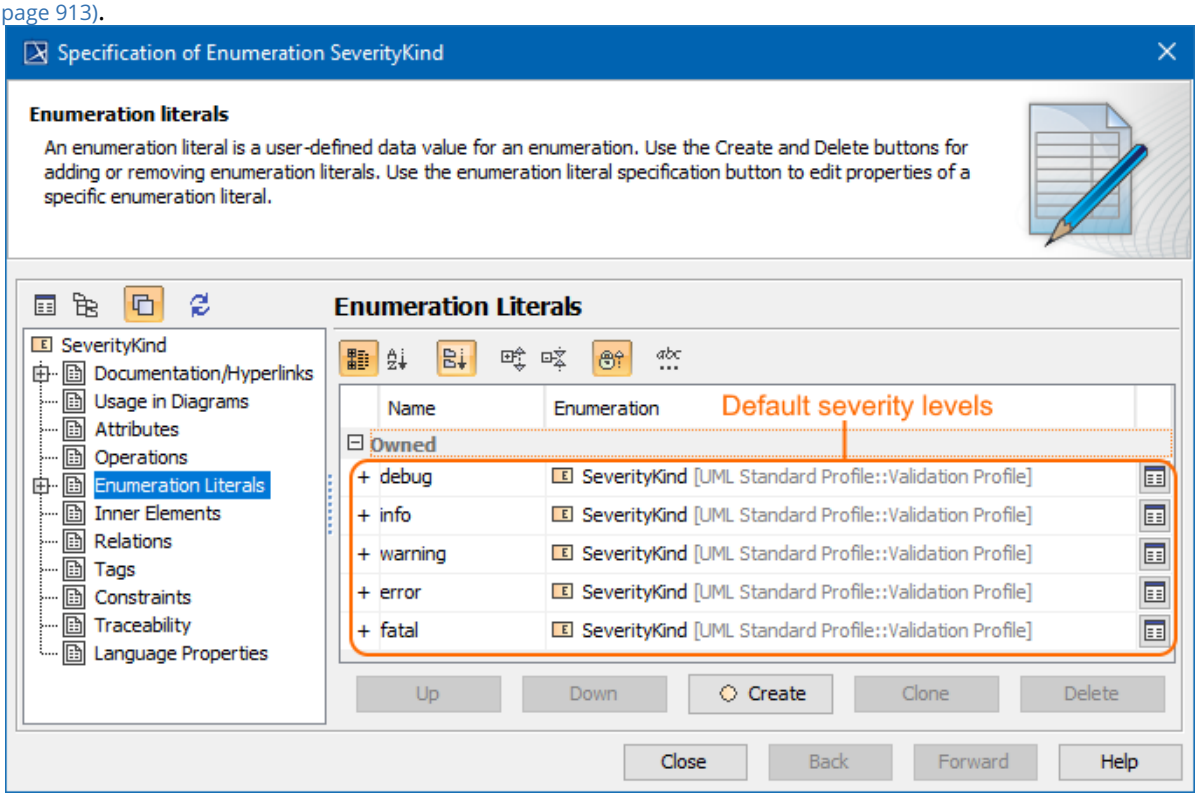
> You can create new or modify default severity levels only if the UML Standard Profile is editable. To see how to edit the profile read the procedure above(see page 1333).

Each Enumeration Literal created for the Enumeration element corresponds to available severity levels.

To modify default severity levels (Enumeration Literals)

---

1. In the Containment tree(see page 32), click + to expand the *UML Standard Profile* Package > *Validation Profile* Package.
2. Double-click the *SeverityKind* Enumeration element to open its Specification window(see page 913).
3. On the left side of the Specification window, select the **Enumeration Literals** property group(see page 913).



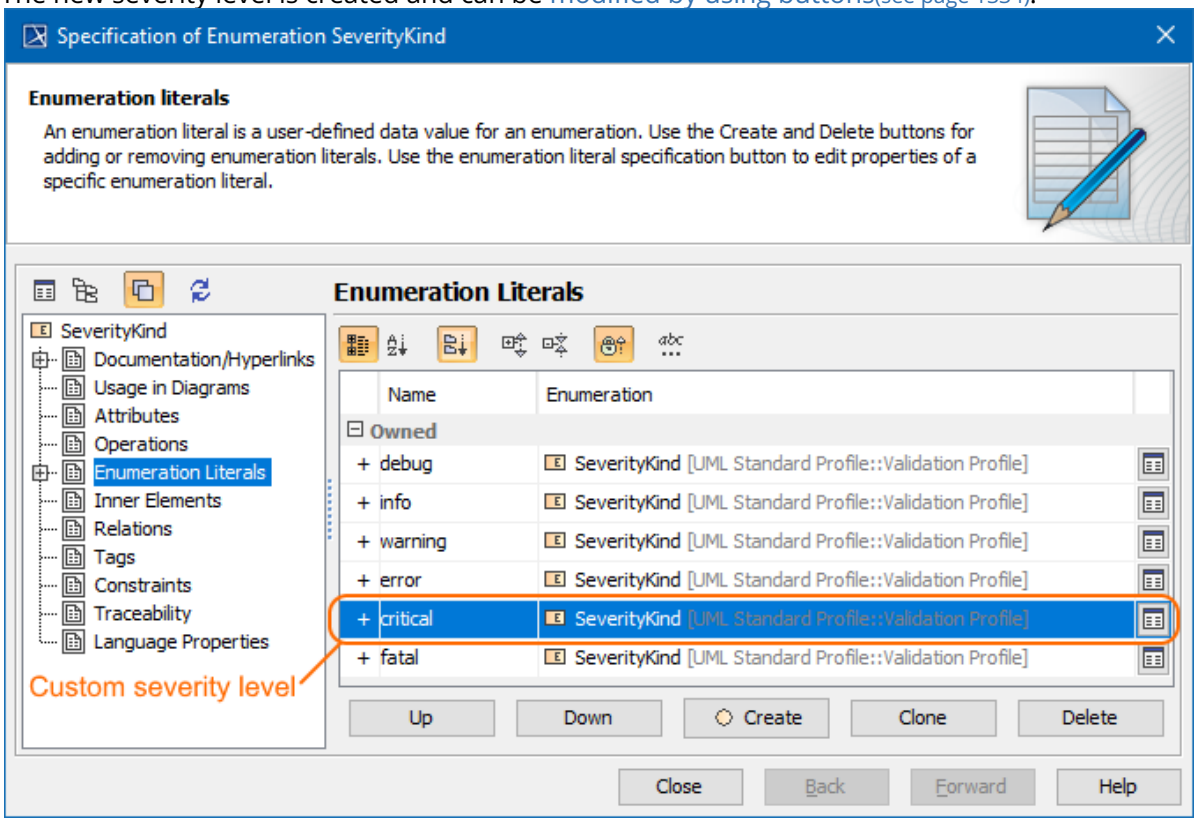
4. On the right side of the Specification window, select severity level from the list:
   - Click to open Specification window(see page 913).
   - Click the **Up**/**Down** button to move the severity level up/down in the list.

   > Severity levels are listed in ascending order according to severity importance.

   - Click the **Clone** button to create the same severity level. The Specification window of newly created Enumeration Literal opens. Type its unique name.
   - Click the **Delete** button to remove the selected severity level from the model.
5. Click **Close**.
   The default severity levels are modified.

To create a new severity level (Enumeration Literal)

1. In the Containment tree(see page 32), click + to expand the *UML Standard Profile* Package > *Validation Profile* Package.
2. Double-click the *SeverityKind* Enumeration element to open its Specification window(see page 913).
3. On the left side of the Specification window, select the **Enumeration Literals** property group(see page 913).
4. On the right side of the Specification window, click the **Create** button. The Specification window of the newly created Enumeration Literal opens.
5. In the **Name** property, type the name of the new severity level (e.g. critical).
6. Click **Close**.
   The new severity level is created and can be modified by using buttons(see page 1334).

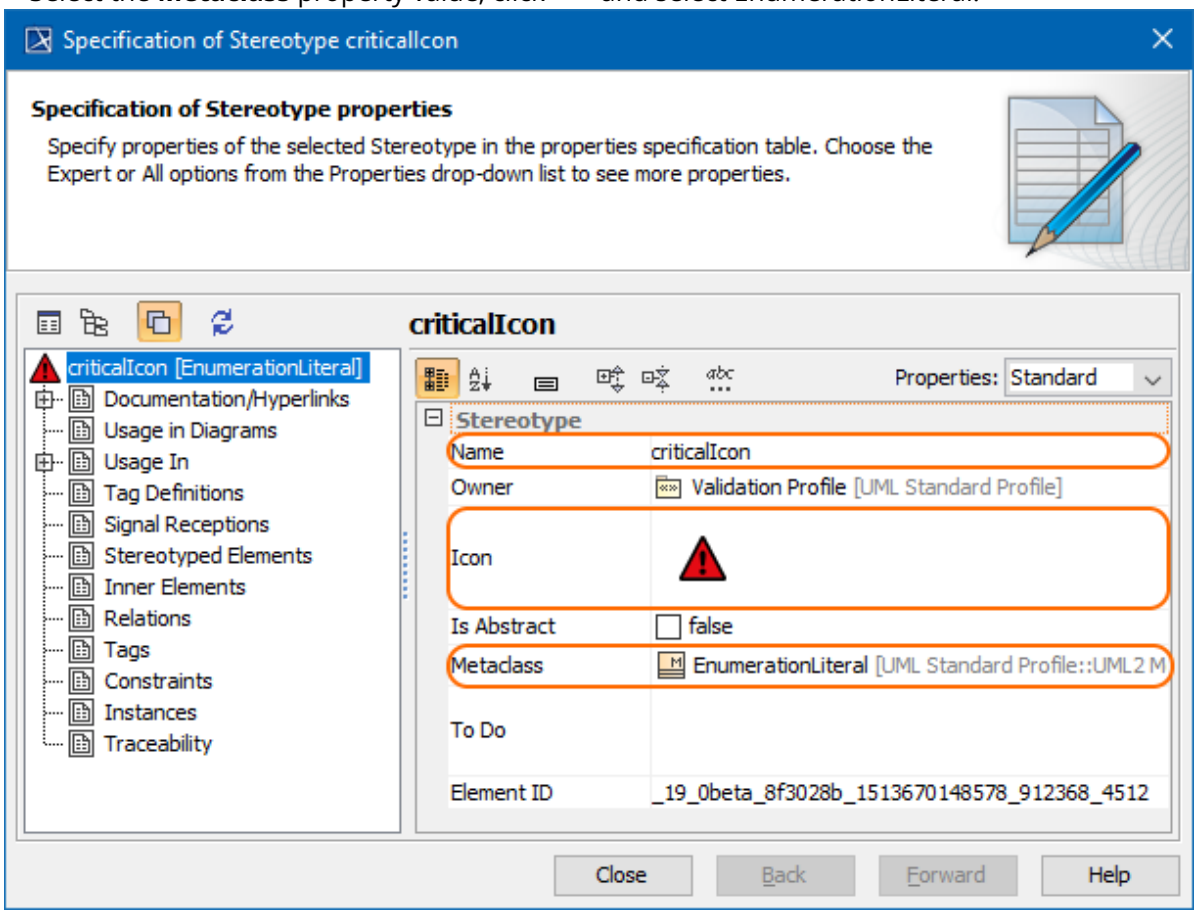


Defining new icon for severity level
You can define a different icon for a custom or default severity levels (Enumeration Literals). For this, you must create a new stereotype and specify its **Icon** and **Metaclass** properties.

To define a new icon for the severity level (Enumeration Literals)

1. In the Containment tree(see page 32), right-click the *Validation Profile* Package, and select **Create Element** > **Stereotype**.
2. Type its name (e.g. criticalIcon) and press **Enter**.
3. Double-click newly created Stereotype.
4. In the Specification window(see page 913) of Stereotype specify the following properties:
   - Select the **Icon** property value, click to select an image from the Image Library(see page 57), or to select an image from your file system.

- Select the **Metaclass** property value, click ... and select EnumerationLiteral.



5. Click **Close**.
   The new stereotype with a new icon is created and ready to be applied to a severity level. How to apply it >>(see page 1336)

Applying new icon for severity level
After you define the new icon for severity level, you can apply it to custom or default severity levels (Enumeration Literals). This icon is shown when representing invalid elements(see page 1306) of your model.

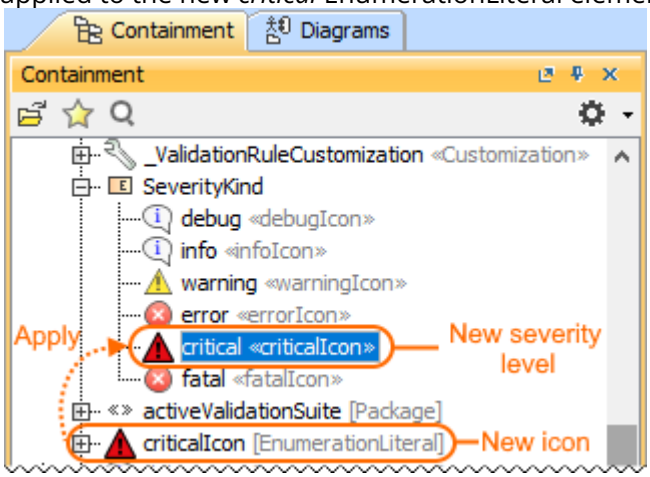
To apply a new icon to a severity level

---

1. In the Containment tree(see page 32), click + to expand the *UML Standard Profile* Package > *Validation Profile* Package > *SeverityKind* Enumeration element.
2. Double-click the severity level (Enumeration Literals element) to which you want to apply a new icon.
3. In its Specification window(see page 913), select the **Applied Stereotype** property value.
4. Click ... and select the newly created stereotype (e.g. criticalIcon) dedicated for this severity level.
5. Select **Apply**.
6. Click **Close**.
   The new icon for the severity level is applied. In the figure below, the *criticalIcon* stereotype is

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

applied to the new *critical* EnumerationLiteral element.



Defining Constraint condition

You must specify the validation condition for a validation rule. This condition must be true when evaluated in order for the Constraint to be satisfied. The result of a validation rule must be of the boolean type.

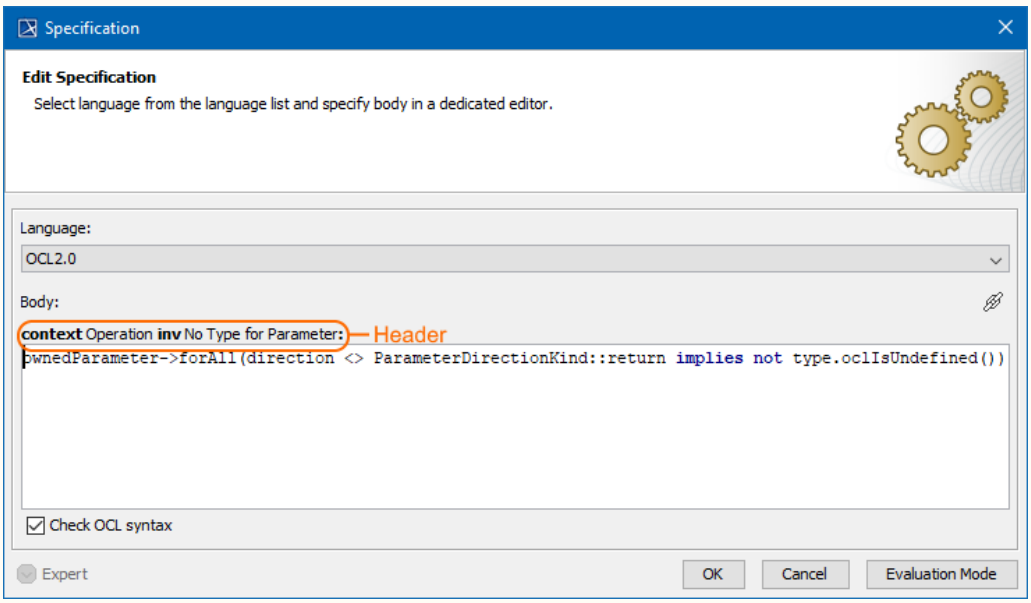
To define a Constraint condition

---

1. Open the Constraint Specification window. How to open the Specification window >>(see page 913)
2. Find the **Specification** property and select its value box.
3. Click ... .The **Specification** dialog opens.
4. From the **Language** list , select one of the following languages:
   - OCL(see page 1147) **2.0** is used for validation rules, specified in OCL language. Learn how to create OCL2.0 validation rule in Developer Guide >>(see page 2337)

OCL Header

If you select the OCL2.0 language, the header of the expression from the constraint information is generated automatically
according to the following rules: **context** <constrained element> <constraint type> <constraint name if any>:. See the following image.



Constraint types

Since the Constraint is stereotyped by «validationRule» which is derived from «invariant » stereotype, **inv** is shown in the header.
Only **inv** constraints can be evaluated.
Other types of Constraints
are not evaluated, butcan be modeled for documentation purposes:

- **def** – for the expression of the constraint with «definition» stereotype applied.
- **init**, **derive** - for the expression of the default value of the property.

  The **derive** expressions can be evaluated indirectly when the validation rule (**inv** constraint) is referencing the property and the validation rule is evaluated.

- **pre**, **post**, **body** - for the expression of the appropriate fields of operation.

OCL Performance

When evaluating the validation rule defined in OCL language, the validation on the first run can have a delay of 20-30 seconds (depending on the computer performance) while the Java compiler is loading. Subsequent validations will run faster than the first one.
If the validation process is run heavily on medium-large projects, increasing the default Java VM size is advisable. By default, the VM size is set to 400MB in MagicDraw; increasing this to 600 (or 800 if the computer has sufficient RAM) might improve the performance.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- **Binary** is used for more advanced expressions not easily expressed in OCL.These expressions are written in Java, compiled, and specified in the MagicDraw classpath. These expressions can then be specified as validation rule expressions. Learn more about binary validation rule in Developer Guide >>(see page 2339)
- Scripts of **JavaScript**, **Jython**, **JRuby**, **Groovy**, and **BeanShell**. Learn more about script writing in Creating executable opaque behaviors(see page 1265).
- Since constraints cannot have parameters, you may skip the information about managing parameters.
- **StructuredExpression**. Learn more about specifying criteria for querying model >>(see page 1227)

5. In the **Body** box, type the expression of the selected language syntax.

> MagicDraw can evaluate only those validation rules whose expression is defined in one of the languages listed above. The other languages can be used only for documentation purposes.

> You can add hyperlinks in the **Body** box. How to define hyperlinks >>(see page 1008)
> You can define expressions for global validation rules. Learn more about global validation rules >>(see page 1339)

6. (optional) Click the **Evaluation Mode** button to to execute an expression on the actual testing model while editing. How to use the **Evaluation Mode** >>(see page 1273)
7. Click **OK**.
   The Constraint condition is defined.

Global validation rules

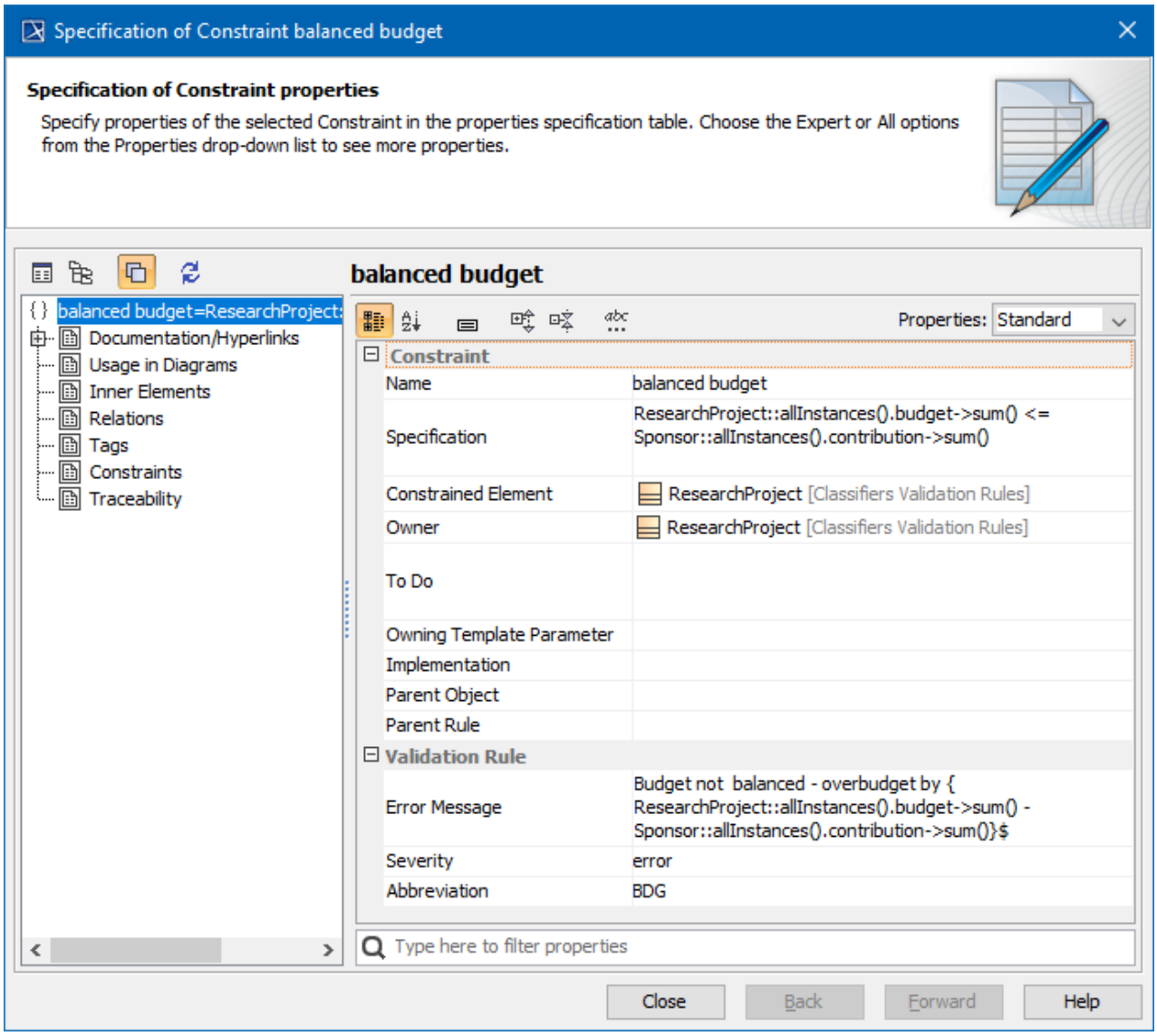
Global validation rules check the result of an element set, while validation rules contained in predefined validation suites(see page 1308) or active validation suites(see page 1310) are dedicated to check a separate element. You can create global validation rules in the same way as other validation rules by defining an appropriate condition in the **Specification** property.

- How to create validation rules >(see page 1328)
- How to define Constraint expression >>(see page 1337)

For example, the *balanced budget* validation rule (Constraint specification is shown in the following figure) is a global validation rule. It is created for the classifier of the *ReseachProject* Class. The *balanced budget* validation rule condition, defined by using OCL2.0 language in the **Specification** property, expresses that the sum of the expenses of all projects must be less than the sum of all *Sponsor* contributions. This validation rule is global because it includes all instances in the model, rather than only the particular instance of *ResearchProject* that violates the rule.

Sample model
An example of a global validation rule can be found in the **model validation** sample model > *Classifiers Validation Rules* Class diagram(see page 297). To open this sample do one of the following:

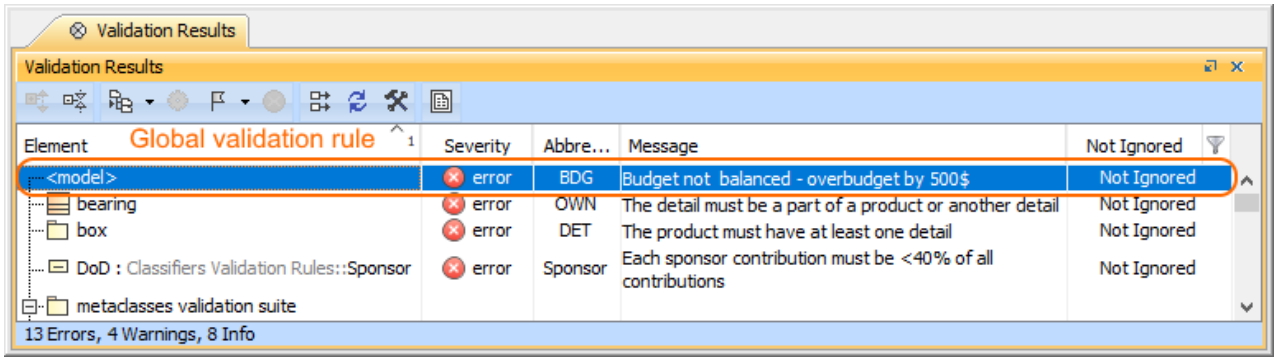
- Download model validation.mdzip[283].
- Find in the modeling tool *<modeling tool installation directory>\samples\product features\model validation.mdzip.*



The specification of the global validation rule named balanced budget.

28 https://docs.nomagic.com/download/attachments/55847477/model%20validation.mdzip?api=v2&modificationDate=1513353053644&version=1

After the validation finishes, the global validation rule is marked as <model> in the **Validation Results** panel(see page 1316) > **Element** column. This means that the entire model, not the separate element, violates the rule.



The representation of global validation rule in the Validation Results panel.

Pre-commit validation

On this page

- Required options for pre-commit validation(see page 1341)
- Customize validation options(see page 1342)
- Warning and error dialogs(see page 1343)
- Pre-commit Validation Results panel(see page 1344)
  - Toolbar(see page 1344)
  - Validation results area(see page 1346)

The pre-commit validation runs just before a committing server project to the Teamwork Cloud. This validation protects against violations that are not allowed according to the preset validation suites. You will receive a notification prior to commit completion if the validation fails, according to setup. The notifications can be either a warning (with an ability to continue the commit) or error messages (to disallow commit). Pre-commit validation does not run validations on auto-commits (e.g. when upgrading profiles, on element move to the used project, update from a local file and other).

Required options for pre-commit validation

To enable pre-commit validation, you must set up at least two project options:

- Validation Suites(see page 1342)
- Minimal Severity That Prohibits Commit(see page 1343)

or

- Validation Suites(see page 1342)
- Minimal Severity That Produces Warning(see page 1343)

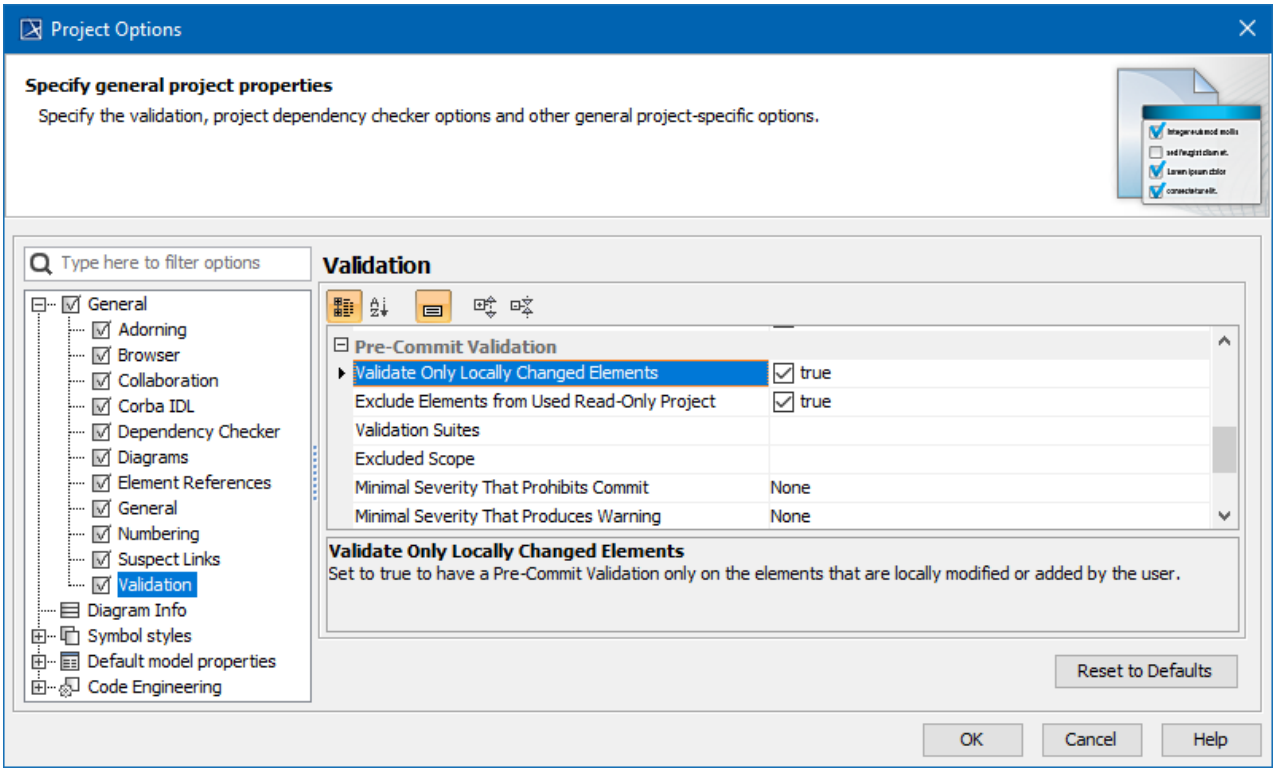
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Customize validation options

Customize validation options in the **Project Options** dialog.

To open the **Validation Options**

1. Open the **Project Options** dialog in one of the following ways:
   - On the main menu, select **Analyze** > **Validation** > **Validation Options**.
   - On the main menu, select **Options** > **Project**. In the **Project Options**(see page 214) dialog, select **General** > **Validation**.
2. Specify the options you need in the **Validation** area. How to use the **Project Options** dialog >>(see page 214)
3. Click **OK**.



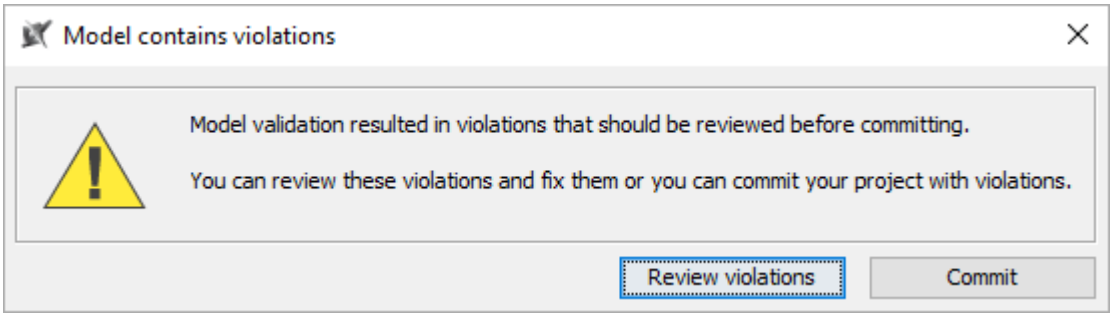
Option	Values	Description
Validate Only Locally Changed Elements	True or False	When set to *true*, pre-commit validation will only run on the elements that are locally modified or added by the user.
Exclude Elements from Used Read-Only Project	True or False	When set to *true*, it will exclude projects that are used in read-only mode and elements that exist in these projects.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

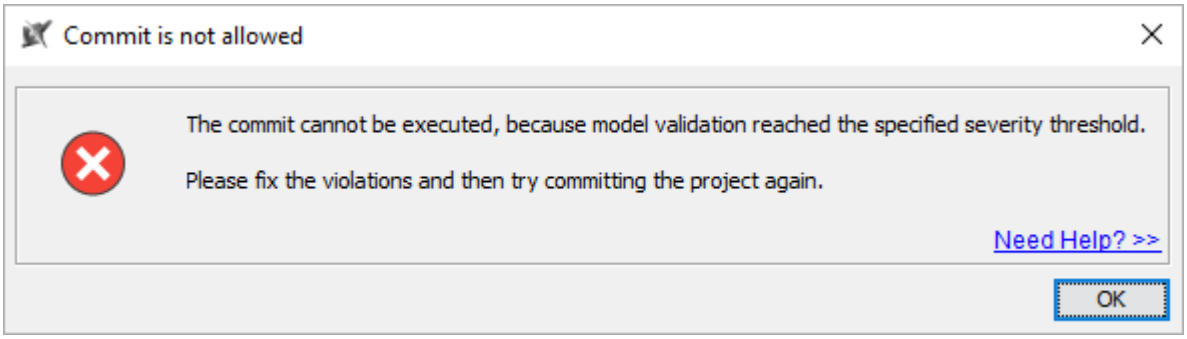
Option	Values	Description
Validation Suites	Validation suites	Specify a validation suites list that must be run before committing changes to a server.
Excluded scope	Excluded scope	Specify the validation scope to be excluded from the rules that belong to pre-commit validation suites. Elements in the selected scope will not be validated before committing changes to a server.
Minimal Severity That Prohibits Commit	• None • - debug. • - info. • - warning. • - error. • - fatal.	Specify the minimal severity violation severity threshold that prohibits commit. Pre-commit validation will be performed before executing commit to a server. It will validate the model and check if there are validation results with reached or exceeded selected severity levels. Commit will be prohibited until these violations are fixed.
Minimal Severity That Produces Warning	• None • - debug. • - info. • - warning. • - error. • - fatal.	Specify the minimal severity threshold that produces a warning. Pre-commit validation will be performed before executing commit to a server. It will validate the model and check if there are validation results with reached or exceeded selected severity level. If these violations are found, a warning dialog will appear, allowing you either to review warnings or to execute project commit.

Warning and error dialogs

- If a warning appears, you are allowed to choose whether to review violations or to continue to commit. If you choose to review violations, the pre-commit validation results panel will be open; otherwise, your project will be committed without showing violations.



- If there is an error, you are not allowed to choose. You will get a list of the validation results.

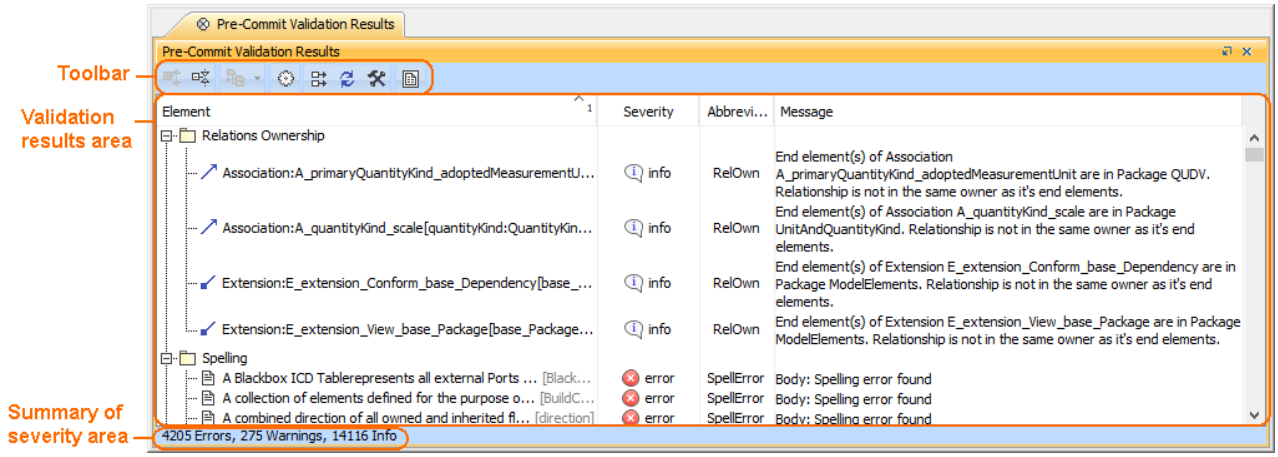


> When the pre-commit validation is supposed to show a warning and an error at the same time, the tool will only show an error due to its behavior being more restricted.

Pre-commit Validation Results panel

The **Pre-commit Validation Results** panel opens after you either get an error message(see page 1343) or you select to review violations in the warning message(see page 1343).

> It is impossible to ignore Pre-Commit Validation results.



Toolbar

The toolbar allows you to manage the validation results. All buttons are described in the following table

Button icon	Button name	Description
	Expand All	Click to recursively expand all the elements in the validation results area.

Button icon	Button name	Description
	Collapse All	Click to recursively collapse all the elements in the validation results area.
	Navigate to Validated Object (Alt+B)	Click the button to select the element in the Containment tree(see page 32). Click the arrow next to the button to open the menu of the following navigation commands: • **Open Specification** - opens the Specification window(see page 913) of the selected element in the validation results area. • **Go To** - opens a submenu whose commands allow you to track and navigate to the elements. • **Select Rule in The Containment Tree** - selects the failed validation rule in the Containment tree(see page 32) of the selected element in the validation results area. • **Add to Search Results** - selects an element in the Search Results tree(see page 47). • **Open Project Usage Map** - opens the **Project Usage Map**. This command is available only for Teamwork Server(see page 1341) projects.
	Solve	Click to open the list of available commands to solve a selected element violation in the validation results area.
	Display the full path of the elements	Click to show the full path of all elements in the validation results area.
	Run validation with current settings	Click to run the validation with the current settings.
	Validation Options	Click to open the **Project Options** dialog with the Validation node open.
	Export	Click to export the list of validation results either to an .html or .csv file.

Validation results area

The validation results area provides a list of violated elements. This area contains the following columns:

- **Element** - elements that violate the validation rule.
- **Severity** - severity level of the validation rule.
- **Abbreviation** - a short word, phrase, or acronym of the validation rule used for sorting/filtering.
- **Message** - description of the invalid situation when a validation rule fails.

Model Visualizer

This functionality is available in Standard, Professional, Architect, and Enterprise editions only.

The **Model Visualizer** helps to create elements from existing data and analyze the relationships between elements created in a UML model. It is also possible to analyze the inheritance and dependency relationships between classes.

All model visualizing and analyzing tools are presented as Wizards, with several steps that should be followed in order to accomplish the desired results.

All Wizards have the following common buttons:

- **Add** – add selected model elements from the **All** list to the **Selected** list.
- **Add All** – add all elements located in the same hierarchy level as the selected element to the **Selected** list.
- **Add Recursively** – add all elements in the selected packages and all elements from nested packages to the **Selected** list.
- **Remove** – remove the selected element from the **Selected** list.
- **Remove All** – remove all selected elements.
- **Back** – return to the previous dialog.
- **Next** – proceed to the next step.
- **Finish** – finish the configuration. All other options will be set by default. The **Wizard** exits and results are displayed.
- **Cancel** – cancel the wizard.
- **Reset To Defaults** – if changes were made to the element properties, values will be set to default.

Related pages

- Class Diagram Wizard(see page 1347)
- Package Dependency Diagram Wizard(see page 1347)
- Package Overview Diagram Wizard(see page 1349)
- Hierarchy Diagram Wizard(see page 1350)
- Realization Diagram Wizard(see page 1350)
- Activity Decomposition Hierarchy Wizard(see page 1350)
- Content Diagram Wizard(see page 1351)
- Generic Table Wizard(see page 1352)

Sequence Diagram from Java Source Wizard(see page 1352)

Class Diagram Wizard

This functionality is available in Standard, Professional, Architect, and Enterprise editions only.

The **Class Diagram Wizard** helps you create a class diagram[284] when all classes and their relationships are already created and specified. You can select which classes, packages, and relationships will be included in a new class diagram[285], as well as the details of the class representation to be configured (attributes, operations, accessibility). The **Class Diagram Wizard** then guides you through several steps and collects information along the way. It will automatically create a class diagram[286] with your selected elements, freeing you from creating the class diagram[287] manually.

To create a class diagram[288] using the **Class Diagram Wizard**

1. Open the **Class Diagram Wizard** by doing one of the following:
   - On the **Diagrams** menu, select **Diagram Wizards** > **Class Diagram Wizard**.
   - From the **Analyze** menu, select **Model Visualizer**. In the open Model Visualizer dialog, select to open the **Class Diagram Wizard** and click **Start.**
2. Follow the steps as directed by the wizard.

Related pages

- Class diagram[289]

Package Dependency Diagram Wizard

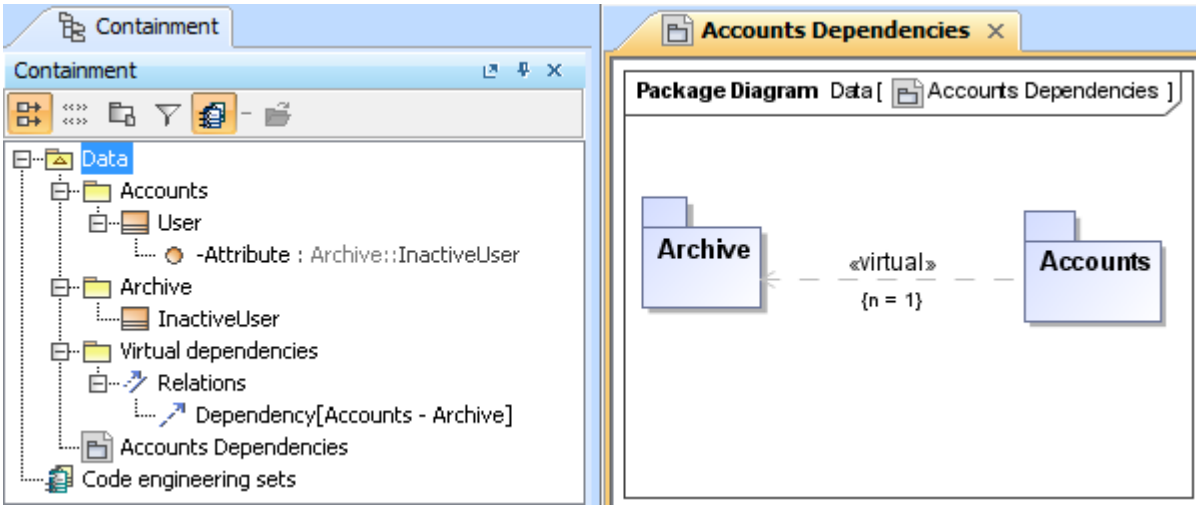
This functionality is available in Standard, Professional, Architect, and Enterprise editions only.

The **Package Dependency Diagram Wizard** creates the Package diagram[290], that contains packages (created within a project) and shows the relationships between them. The diagram may reflect all packages in the project, or only selected ones. The **Package Dependency Diagram Wizard** collects all information needed for both analyzing dependencies and generating a new diagram.

If a package owns one or more elements used by or dependent on other package elements, the **Package Dependency Diagram Wizard** analyzes dependencies and creates «virtual» relations between the dependent packages, as shown in the following example.

284 https://docs.nomagic.com/display/MD190/Class+diagram
285 https://docs.nomagic.com/display/MD190/Class+diagram
286 https://docs.nomagic.com/display/MD190/Class+diagram
287 https://docs.nomagic.com/display/MD190/Class+diagram
288 https://docs.nomagic.com/display/MD190/Class+diagram
289 https://docs.nomagic.com/display/MD190/Class+diagram
290 https://docs.nomagic.com/display/MD190/Package+diagram

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The **Virtual dependencies** package containing dependency links is created after finishing the wizard. If more than one dependency exists between elements in the packages selected to analyze, the number of dependencies is assigned to the tagged value number {n=... } on the virtual dependency.

To create a package diagram[291] using the **Package Dependency Diagram Wizard**

---

1. Open the **Package Dependency Diagram Wizard** by doing one of the following:
   - On the **Diagrams** menu, select **Diagram Wizards** > **Package Dependency Diagram Wizard**.
   - From the **Analyze** menu, select **Model Visualizer**. In the open Model Visualizer dialog, select the **Package Dependency Diagram Wizard** and click **Start**.
2. Follow the steps as directed by the wizard.

Related pages

- Displaying dependency details(see page 1348)
- Dependency(see page 699)
- Dependency Matrix(see page 334)
- Package diagram(see page 321)

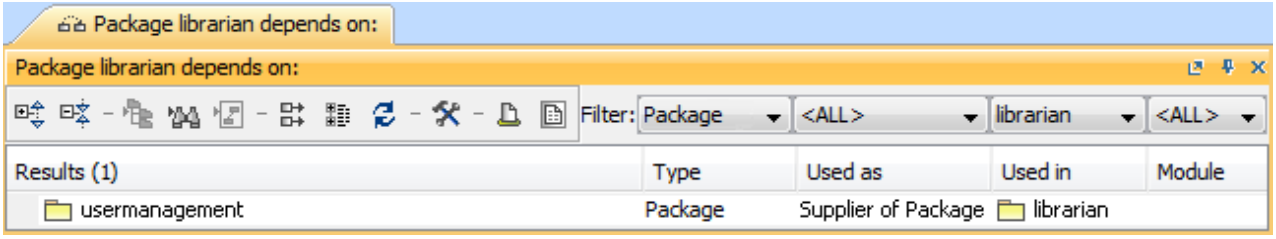
Displaying dependency details

In the Package diagram[292], the virtual dependencies (dependency links) are created. All virtual dependencies are stored in the **Virtual dependencies** package, which you can find in the model browser.

You can use the **Show Dependency Details** command to see the dependencies of a particular package. Dependency results are displayed in the dependency details panel (see the following example).

---

291 https://docs.nomagic.com/display/MD190/Package+diagram
292 https://docs.nomagic.com/display/MD190/Package+diagram



To display the dependency details

1. In the Model Browser or on the diagram pane, right-click the dependency with the details you want to display.
2. In the shortcut menu, select **Related Elements** > **Show Dependency Details**. The dependency details panel opens.

Related pages

- Dependency(see page 699)
- Package diagram(see page 321)

Package Overview Diagram Wizard

This functionality is available in Standard, Professional, Architect, and Enterprise editions only.

The **Package Overview Diagram Wizard** allows you to create a diagram for every package from the selected scope (reversed packages). The created diagram displays the content of the packages – inner packages with inner elements connected with available relations.

To create a diagram using the **Package Overview Diagram Wizard**

1. Open the **Package Overview Diagram Wizard** by doing one of the following:
   - On the **Diagrams** menu, select **Diagram Wizards** > **Package Overview Diagram Wizard**.
   - From the **Analyze** menu, select **Model Visualizer**. In the open Model Visualizer dialog, select to open the **Package Overview Diagram Wizard** and click **Start**.
2. Follow the steps as directed by the wizard.

Related pages

- Package(see page 800)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Hierarchy Diagram Wizard

This functionality is available in Standard, Professional, Architect, and Enterprise editions only.

The **Hierarchy Diagram Wizard** collects the largest hierarchies and allows each of them to be displayed either as separate diagrams or all in one diagram.

To create a diagram using the **Hierarchy Diagram Wizard**

---

1. Open the **Hierarchy Diagram Wizard** by doing one of the following:
   - On the **Diagrams** menu, select **Diagram Wizards** > **Hierarchy Diagram Wizard**.
   - From the **Analyze** menu, select **Model Visualizer**. In the open Model Visualizer dialog, select to open the **Hierarchy Diagram Wizard** and click **Start**.
2. Follow the steps as directed by the wizard.

Realization Diagram Wizard

This functionality is available in Standard, Professional, Architect, and Enterprise editions only.

The **Realization Diagram Wizard** shows a table of the largest element groups that realize an interface.

To create a diagram using the **Realization Diagram Wizard**

---

1. Open the **Realization Diagram Wizard** by doing one of the following:
   - On the **Diagrams** menu, select **Diagram Wizards** > **Realization Diagram Wizard**.
   - From the **Analyze** menu, select **Model Visualizer**. In the open Model Visualizer dialog, select to open the **Realization Diagram Wizard** and click **Start**.
2. Follow the steps as directed by the wizard.

What you should know when selecting Implementations:

- The **Add all implementations into one diagram** check box creates only one diagram for all selected realizations. This option is available only if the selected realizations can be added into one diagram (the same diagram type).
- The **Show outside specializations** check box shows realizations when derived interfaces are in the scope, but specializations are outside the scope.
- The **Show outside implementations** check box counts outside derived elements from displayed realizations. Otherwise, the realization will not be fully displayed and the diagram will not be valid.

Activity Decomposition Hierarchy Wizard

This functionality is available in Standard, Professional, Architect, and Enterprise editions only.

The **Activity Decomposition Hierarchy Wizard** allows you to convert an activity into a class and SysML Block Definition Diagram. This provides the ability to represent, analyze, and document activity hierarchies in the structure diagrams.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Diagram generation rules include:

- Behaviors will be connected with contained object node types by compositions. The name of the object node that corresponds to the composition will be used as the end name of the association on the end towards the object node type.
- Pins are not included in the calculation.
- CallBehaviorActions that are not directly in the Activity, but are in the Structured Activity Nodes contained by the Activity, for example, are also included in the calculation.
- An activity will be connected by composition association with other behaviors that are called by CallBehaviorActions. The part end name must be the same as the name of a CallBehaviorAction in the composing activity. If the action has no name, then the end name is as same as that of the invoked activity.
- Hierarchical layout - Top to Bottom is used to arrange the generated diagram.
- If CallBehaviorAction calls the same activity, the composition to self will be displayed on the generated diagram.
  Recursive structure analysis will be stopped after reaching a behavior that has already been analyzed. This requirement prevents an endless cycle.
  - In such a case, the composition will be created in a previously analyzed activity in the diagram. The new behavior symbol will not be created.
  - There will be as many compositions from one activity into another as different CallBehaviorActions call this activity.

To create a diagram using the **Activity Decomposition Hierarchy Wizard**

---

1. Open the **Activity Decomposition Hierarchy Wizard** by doing one of the following:
   - On the **Diagrams** menu, select **Diagram Wizards** > **Activity Decomposition Hierarchy Wizard**.
   - From the **Analyze** menu, select **Model Visualizer**. In the open Model Visualizer dialog, select to open the **Activity Decomposition Hierarchy Wizard** and click **Start**.
2. Follow the steps as directed by the wizard.

What you should know when selecting structure:

- The **Add contained Object Nodes** check box is selected by default. Types of object nodes are displayed and connected to the composition with activities containing object nodes.
- The **Search recursively** check box is selected by default. The search will be conducted in the selected activity and those activities invoked by CallBehaviorActions that are in the selected activity. This search is recursive. If not selected, the search will be conducted in only one level of the selected activity.

Content Diagram Wizard

This feature is available in Standard, Professional, Architect, and Enterprise editions.

The **Content Diagram Wizard** generates the content of diagrams used in the project.

To create a content diagram using the **Content Diagram Wizard**

---

1. Open the Content Diagram Wizard by doing one of the following:
    - On the **Diagrams** menu, select **Diagram Wizards** > **Content Diagram Wizard**.
    - From the **Analyze** menu, select **Model Visualizer**. In the open Model Visualizer dialog, select to open the **Content Diagram Wizard** and click **Start**.
2. Follow the steps as directed by the wizard.

Related pages

- Content diagram(see page 399)

Generic Table Wizard

> **Editions**
> This functionality is available in Standard, Professional, Architect, and Enterprise editions only.

A Generic Table(see page 441) allows you to display a list of selected elements and their properties in a single place.

The Generic Table Wizard helps you create a table when model elements and their relationships are already created and specified. You can select which elements (e.g., classes, packages, use cases, and relationships) will be included in a new table. You can also select properties of the selected element types, which will be displayed as columns in the table.

The Generic Table Wizard guides you through several steps and collects information along the way. It will automatically create a table with your selected elements, freeing you from creating the table manually.

To create a Generic Table using the Generic Table Wizard

---

1. Open the **Generic Table Wizard** by doing one of the following:
    - On the **Diagrams** menu, select **Diagram Wizards** > **Generic Table Wizard**.
    - From the **Analyze** menu, select **Model Visualizer**. In the open The Model Visualizer dialog, select to open the **Generic Table Wizard** and click **Start**.
2. Follow the steps as directed by the wizard.

Related pages

- Generic table(see page 441)

Sequence Diagram from Java Source Wizard

> This feature is available only in the Enterprise edition.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

The **Sequence Diagram from Java Source Wizard** allows for visualizing Java method implementation within UML Sequence diagrams[293]. At this time, the UML Sequence diagram[294] cannot show Java code with 100% accuracy; however, MagicDraw provides a mechanism for generating a diagram that reflects the essence of Java method content.
If you want to create a sequence diagram[295] from the Java source, you must first reverse (see page 2407)the Java source code to a model.

You can also create a model corresponding to your Java code structure manually.

> Ensure the model you use to create a sequence diagram fully corresponds the Java source code you want to represent in the diagram.

One sequence diagram[296] can represent one method in a Java code. Classes are represented as lifelines, and method calls are represented as messages in the diagram. The sequence diagram[297] can also be used to visualize dependencies for all classes used in this method.

To create a sequence diagram using the **Sequence Diagram from Java Source Wizard**

1. Open the **Sequence Diagram from Java Source Wizard** by doing one of the following:
   - On the **Diagrams** menu, select **Diagram Wizards** > **Sequence Diagram from Java Source Wizard**.
   - From the **Analyze** menu, select **Model Visualizer**. In the open Model Visualizer dialog, select to open the **Sequence Diagram from Java Source Wizard** and click **Start**.
   - In the Code Engineering Set, select an operation of the class and from the shortcut menu, select **Reverse Implementation**.
2. Follow the steps as directed by the wizard.

What you should know when selecting classes for the diagram:

- **Analyze and split long expressions in the diagram**: Select to display every call as a separate call message with a temporary variable initialization, if the expression containing calls cannot be displayed as a call message. In the final expression message, these calls are replaced with appropriate temporary variable names.
- **Create reply message**: Select to display the return message for every call message.
- **Wrap message text**: Select to wrap message text in the diagram. In the **Maximum wrapped messages name length (in pixels)** box, specify the maximum message text length in pixels.

Related Pages:

- Extending sequence diagrams(see page 1354)
- Method Implementation Reverse[298]
- Sequence diagram(see page 304)

293 https://docs.nomagic.com/display/MD190/Sequence+diagram
294 https://docs.nomagic.com/display/MD190/Sequence+diagram
295 https://docs.nomagic.com/display/MD190/Sequence+diagram
296 https://docs.nomagic.com/display/MD190/Sequence+diagram
297 https://docs.nomagic.com/display/MD190/Sequence+diagram
298 https://docs.nomagic.com/display/MD190/Method+Implementation+Reverse

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Code Engineering(see page 2399)

Extending sequence diagrams

Creating a sequence diagram[299] from a Java code allows you to analyze dependencies between the methods represented on the diagram and the classes they are referencing to and/or the other methods they call for. This means you can create a sequence diagram[300] for any method selected in the existing sequence diagram[301].

To create a new sequence diagram for a selected method

1. Select the method (call message with an operation assigned) whose details you want to see.
2. Right-click the message and, from the shortcut menu, select **Reverse Implementation**.
   The **Sequence Diagram from Java Source Wizard** opens to create a sequence diagram for the selected method.

   > ⚠ If you have started the wizard via the operation assigned to the message, be aware that in this case the first step of the wizard will be skipped, and the wizard will start from Step #2.

All selected method dependencies will be represented in a newly created diagram.

Related pages

- Sequence diagram(see page 304)
- Code Engineering(see page 2399)

Suspect Links

Working on a server project requires an up-to-date knowledge of all relevant project changes. For this, we introduce a new set of validation rules for suspect link analysis. These rules help you track changes in requirements that are linked to design elements (such as SysML Blocks, Test Cases, Use Cases, etc.) by Satisfy, Derive, Refine or Verify relationships, and stay informed about unlinked and deprecated requirements in the model.

> ⚠ Before you start tracking changes using the Suspect Link analysis, make sure your model (or its particular elements) are locked(see page 1403).

To learn about the requirements tracking, read the following topics:

- Tracking elements affected by requirement changes(see page 1355)
- Tracking deprecated requirements(see page 1357)
- Tracking unlinked requirements(see page 1357)

---

299 https://docs.nomagic.com/display/MD190/Sequence+diagram
300 https://docs.nomagic.com/display/MD190/Sequence+diagram
301 https://docs.nomagic.com/display/MD190/Sequence+diagram

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Solving untracked links issue(see page 1358)

You can also create your own suspect links customization to keep track of the changes in the elements of the specified type. To learn more, read the following topic:

- Customizing suspect links tracking(see page 1359)

Tracking elements affected by requirement changes

To enable tracking

1. In the main toolbar, go to **Analyze** > **Suspect Links** and select **Enable Tracking**.
2. In the **Enable Tracking** message dialog
   - choose to track **Entire Model**, or
   - click **Specific Scope** to choose packages, where the requirements and design elements you want to track are.
3. Click **OK**.

   If you're working on a Teamwork Server or TWCloud project, commit it for the changes to apply.

To change scope

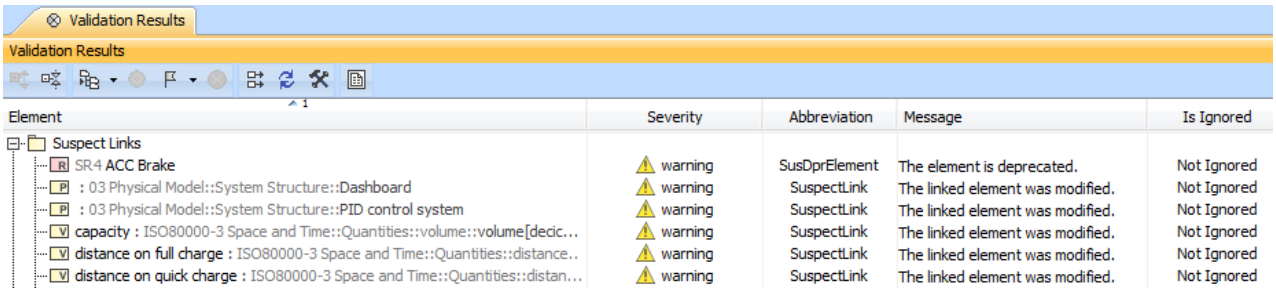
1. Go to **Analyze** > **Suspect Links** > **Tracking Scope**.
2. In the **Select Package** dialog, choose packages, where the requirements and design elements you want to track are, then click **OK**.

   If you're working on a Teamwork Server or TWCloud project, commit it for the changes to apply.

To check for suspicions

1. Modify the linked requirements as per your needs.
2. If you're working on a Teamwork Server or TWCloud project, commit it for the changes to apply.
3. In the main toolbar, go to **Analyze** > **Suspect Links**, and select **Show Suspicions**.

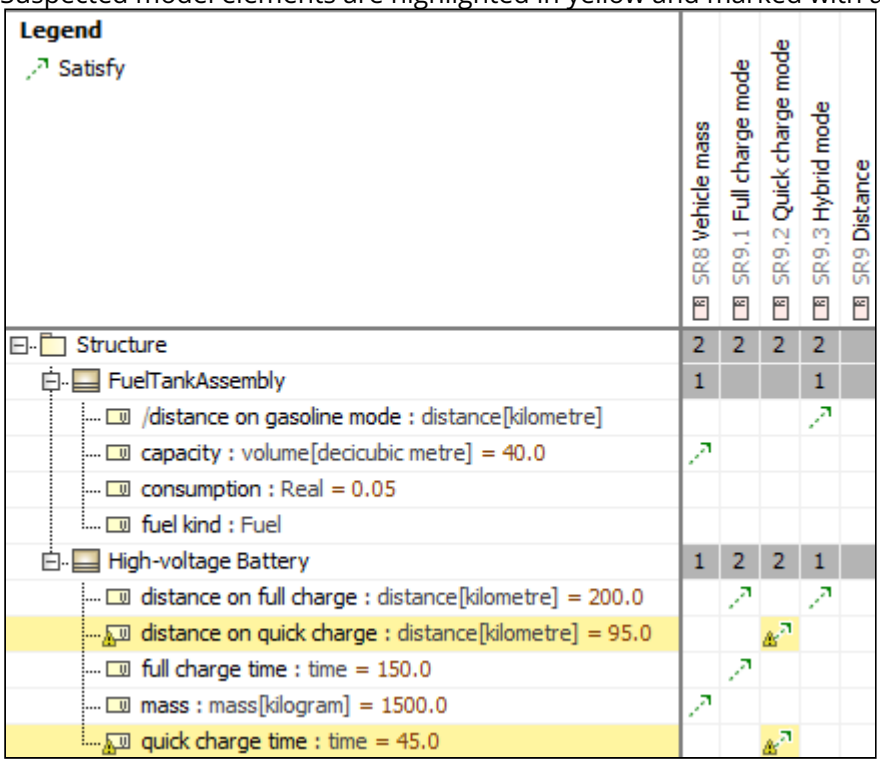
The detected suspect links are provided in the **Validation Results** panel.



To handle suspicions

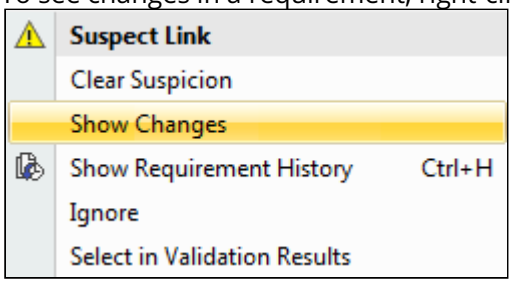
* marked steps are only available in the TWCloud project.

1. Suspected model elements are highlighted in yellow and marked with a small triangle symbol.

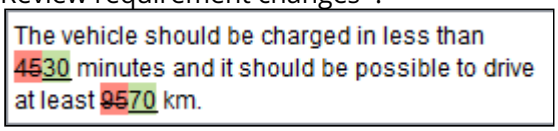


Legend: Satisfy	SR8 Vehicle mass	SR9.1 Full charge mode	SR9.2 Quick charge mode	SR9.3 Hybrid mode	SR9 Distance
Structure	2	2	2	2	
FuelTankAssembly	1			1	
/distance on gasoline mode : distance[kilometre]				✓	
capacity : volume[decicubic metre] = 40.0	✓				
consumption : Real = 0.05					
fuel kind : Fuel					
High-voltage Battery	1	2	2	1	
distance on full charge : distance[kilometre] = 200.0		✓		✓	
distance on quick charge : distance[kilometre] = 95.0			✓		
full charge time : time = 150.0		✓			
mass : mass[kilogram] = 1500.0	✓				
quick charge time : time = 45.0			✓		

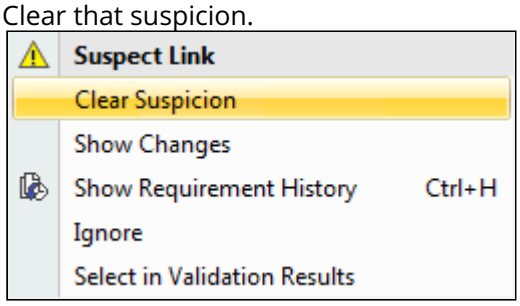
2. To see changes in a requirement, right-click the validated cell and select **Show Changes***.



3. Review requirement changes*.



4. Update design elements according to changes in the requirement.
5. Clear that suspicion.



« Suspect Links(see page 1354)

Tracking deprecated requirements »(see page 1357)

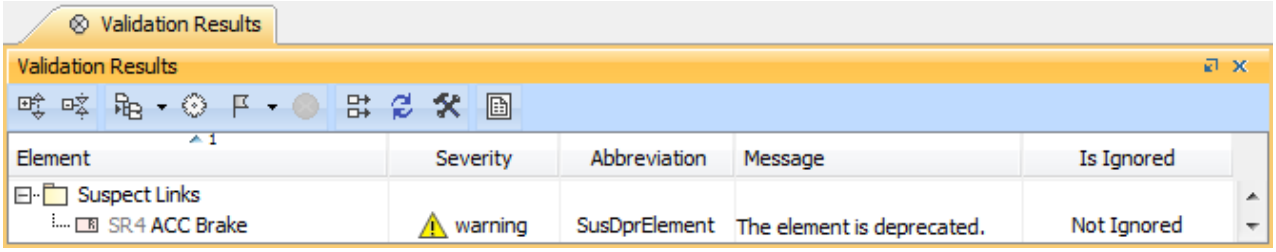
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Tracking deprecated requirements

Suspect links analysis allows tracking deprecated requirements. This is useful when you are working with requirements and some of them are no longer used. In that case, apply the *«deprecated»* stereotype to mark the requirements of this type.

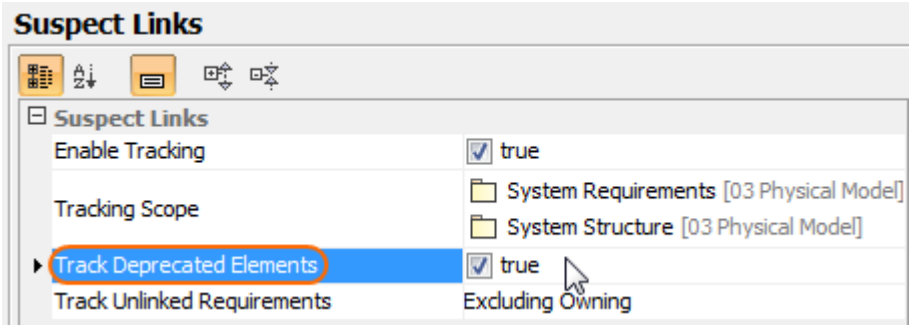
Deprecated requirements are tracked by default.

All deprecated requirements are provided in the **Validation Results** panel.



To check whether deprecated requirements are tracked

- In the main toolbar, go to **Analyze** > **Suspect Links** and select **Options**. If the check box next to the **Track Deprecated Elements** property group is checked and set to *true*, tracking deprecated elements is enabled.



« Tracking elements affected by requirement changes(see page 1355)

Tracking unlinked requirements »(see page 1357)

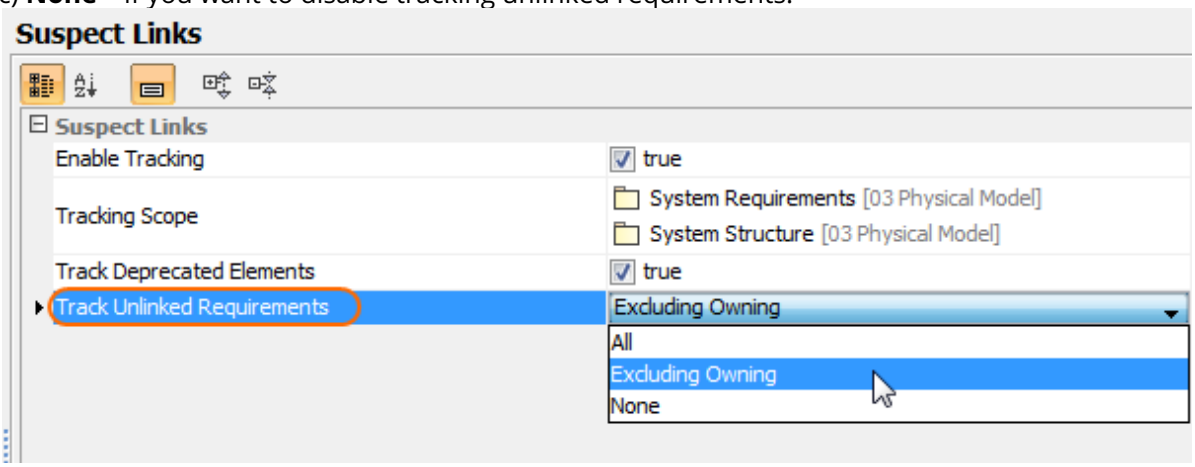
Tracking unlinked requirements

Suspect links analysis allows tracking unlinked requirements. This is useful when you're working with structure elements and you need to know which requirements remain unsatisfied.

To check whether unlinked requirements are satisfied

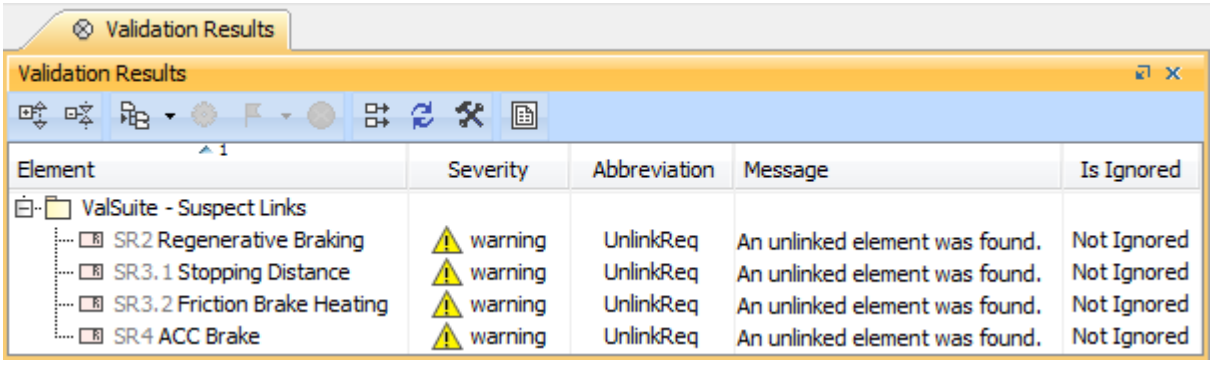
1. In the main toolbar, go to **Analyze** > **Suspect Links** and select **Options**.

2. Click the right column next to the **Track Unlinked Requirements** property group and choose:
   a) **All** – if you want to track all the unlinked requirements
   b) **Excluding Owning** – if you want to track only those unlinked requirements, that have no sub-requirements
   c) **None** – if you want to disable tracking unlinked requirements.



3. Click **OK**.

All unlinked requirements are provided in the **Validation Results** panel.



« Tracking deprecated requirements(see page 1357)

Solving untracked links issue »(see page 1358)

Solving untracked links issue

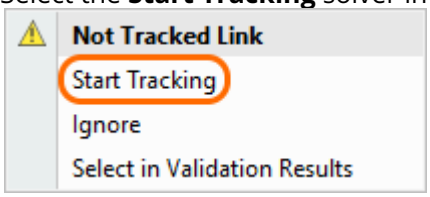
If you notice that some links in your model are not tracked for some reason, you can start tracking them manually.

To start tracking links in your model

1. Right-click the untracked link (it is marked with an exclamation mark in a yellow triangle).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

2. Select the **Start Tracking** solver in the opened menu.



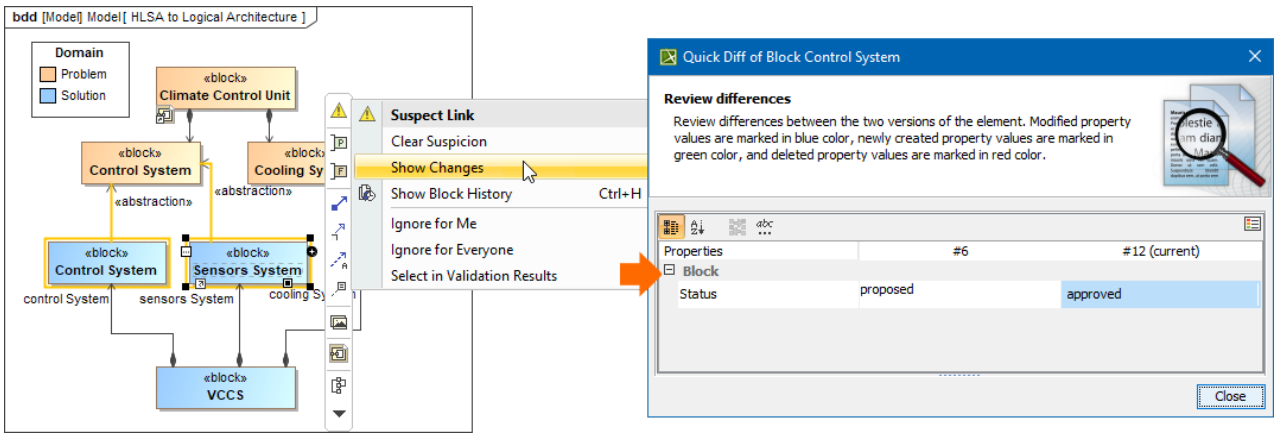
Customizing suspect links tracking

On this page

- Customizing tracking(see page 1359)
- Configuration properties(see page 1360)

In addition to tracking the changes directly affecting requirements linked to other model elements, you can now create your own custom suspect link configurations to keep track of the changes applied to selected model elements and their properties at the chosen end (source or target) of the specified dependencies.

> A newly created suspect links configuration comes into effect immediately and overrides a configuration previously used in the project.



Suspect links tracking in the model

Customizing tracking

To create a custom suspect link configuration

1. In the Model Browser(see page 29), right-click a package to create your configuration in, and select **Create Element** > **Class**. Name it.

2. Right-click the configuration Class(see page 666), select **Stereotype,** and apply the «SuspectLinkConfiguration» stereotype.
3. Open the created element's Specification window(see page 913).
4. Specify the following configuration properties(see page 1360):
    - **Tracked Dependency Type**
    - **Tracked End**
    - **Tracked Element Type**
    - **Tracked Element Properties**
5. Close the window when you are done and press F5 to refresh your project.
   The configuration is created. Please note that it is not automatically applied to relationships that existed in the model prior to creating a configuration for suspect links tracking. In such a case, you need to start tracking(see page 1358) links manually.

> The suspect links tracking must be enabled(see page 1355) (**Analyze** > **Suspect Links** > **Enable Tracking**) in your project.

Configuration properties

These are the properties you need to specify for your custom suspect link configuration:

Property name	Description
Tracked Dependency Type	Specify the types of dependencies to keep track of their source/target element changes.
Tracked End	Specify whether the source or the target element of the selected dependencies should be tracked. The target end-related changes are tracked by default.
Tracked Element Type	Specify the element types to keep track of their changes.
Tracked Element Properties	Specify the properties of the selected model elements to keep track of their changes. If left empty, all element properties are tracked by default.

« Solving untracked links issue(see page 1358) Collaborative modeling »(see page 1360)

Collaborative modeling

On this page

- Basic concepts(see page 1361)
    - Repository(see page 1361)
    - User(see page 1361)
    - TWCloud project(see page 1361)
    - Teamwork Server project(see page 1362)
    - Server project(see page 1362)
    - Version(see page 1362)
    - Comment(see page 1362)
    - Tag(see page 1362)

    - Commit(see page 1362)
    - Author(see page 1362)
    - Update(see page 1362)
    - Category(see page 1362)
    - Locked item(see page 1362)
    - Server project usage(see page 1362)
    - Branch(see page 1362)
    - Trunk(see page 1363)
- Scenario to get started(see page 1363)

Team members can individually develop separate parts of a single model and then merge them together to form a complete model. In comparison to individual modeling, collaborative modeling is more effective and produces better results because each member employs his/her best skills and experience.

No Magic offers you two servers for your team's collaboration:

- Teamwork Cloud(see page 1367) (formerly known as Cameo Enterprise Data Warehouse)
- MagicDraw Teamwork Server(see page 1479)

Both servers provide a repository for storing projects and users. They also support the management of users' permissions as well as sharing and versioning projects.

Projects stored in the server repository can be accessed through the network from multiple clients who have MagicDraw or any Cameo Suite product installed. Different users, depending on their role in a team or enterprise, can have different permissions to the projects.

The same model or even the same diagram can be accessed and modified in parallel. Every user may instantly obtain the newest version of the model as well as commit his/her own changes.

Basic concepts

For better understanding further material, get acquainted with basic concepts of collaborative modeling.

Concept	Description
Repository	A place for storing projects and users managed by Teamwork Cloud (TWCloud) or MagicDraw Teamwork Server.
User	A collaboration team member, who has credentials to log into TWCloud or Teamwork Server and holds permissions to access one or more server projects.
TWCloud project	A project stored in TWCloud repository.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Concept	Description
Teamwork Server project	A project stored in MagicDraw Teamwork Server repository.
Server project	A common name for TWCloud and Teamwork Server projects.
Version	A snapshot of a project at a particular point in time. A new version of a project is created after successful commit of changes made in this project. All versions of the same project have unique numbers.
Comment	An optional description about changes in a project version.
Tag	Information about the status of a project version (for example, *approved* or *initially tested*) or other.
Commit	A process of sending the changes made in the project to the server. Each commit creates a new project version.
Author	A user who has committed a particular project version.
Update	A process of getting from the server the latest version of a project you are working with.
Category	A concept for grouping projects in both TWCloud and MagicDraw Teamwork Server repositories.
Locked item	A part of the model that can be edited by the user, who has locked it. Other users cannot edit locked items. The locks can be released during the commit of a new version.
Server project usage	Another server project used in the project you are working with. It can be a library, profile or regular project.
Branch	A duplicate of a server project version. Branching allows the users to work in parallel on the same project version.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Concept	Description
Trunk	A permanent branch of a server project that is a base for all other branches. It cannot be deleted.

Scenario to get started

Independent of what server your team or enterprise uses, we offer you to follow these steps to get started with collaborative modelling:

1. Obtain user's credentials.

   > Creating users and deciding the access rights of the users against projects is an administrative task. For more information, see Teamwork Cloud User Guide[302].

2. Log into a server.
3. Add a project to the server and/or open it.
4. Lock the elements you want to edit (to prevent other users could not change the elements you are working with) and edit them.

   > You can skip this step if you are working in the Lock-Free Editing mode(see page 1409).

5. Edit these elements and/or create new ones.
6. Commit the changes to the server so that other users could see these modifications.

Related pages

Starting a collaboration session

Before logging in to any of No Magic's servers, you must:

- Have a user account on the server.
- Be sure the server is running.
- Ensure the server and the client application use the same Java version. Small Java version updates may differ.

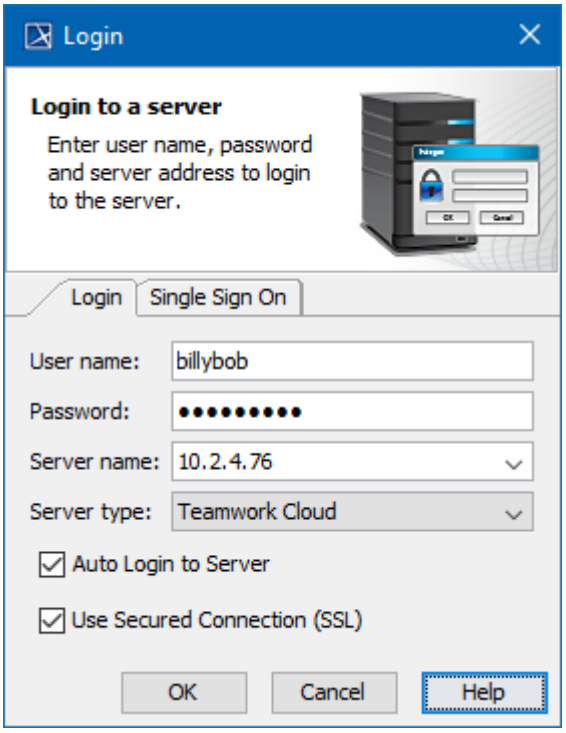
  > The server Java version 1.8.0_45 will work with the client Java version 1.8.0_51.

302 https://docs.nomagic.com/download/attachments/792966/Cameo%20Enterprise%20Data%20Warehouse%20UserGuide.pdf?api=v2&modificationDate=1438322880484&version=5

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

1. From the **Collaborate** menu, select **Login**. The **Login** dialog opens.
2. Login to a server:
   - To login using "username/password" authentication:

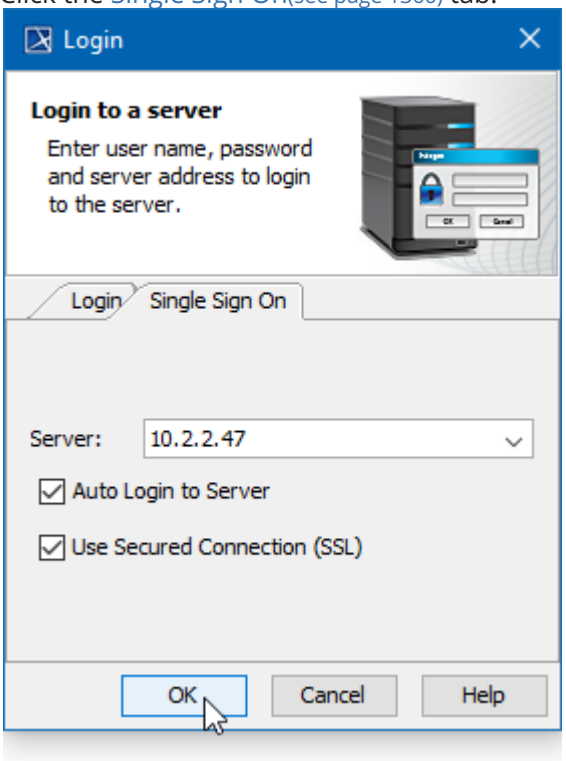
     i. Click the **Login** tab.
     ii. Type your user name and password.



   - For users having a single ID and password for multiple software systems (valid only for Teamwork Cloud Server):

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

i. Click the Single Sign On(see page 1366) tab.



3. Provide the server name (address) and select server type. If you need to specify a server port, type *<server* name> : *<port number>*, for example *10.2.2.47:3579*.

> **For Teamwork Cloud Server users**
> If a Single Sign On property is selected to login, the Teamwork Cloud Server is automatically selected as a type.

4. If you want the client application to remember your credentials and automatically log in to the server next time upon starting the client application, select the **Auto Login to Server** check box.
5. If you need to connect to the server using the SSL connection, select the **Use Secured Connection (SSL)** check box.

> The SSL connection must be established on both the server and the client side.

6. Click **OK**.
7. Wait while you are connected to the server, and then you can start performing collaborative tasks.

To log out of the server

- From the **Collaborate** menu, select **Logout**.

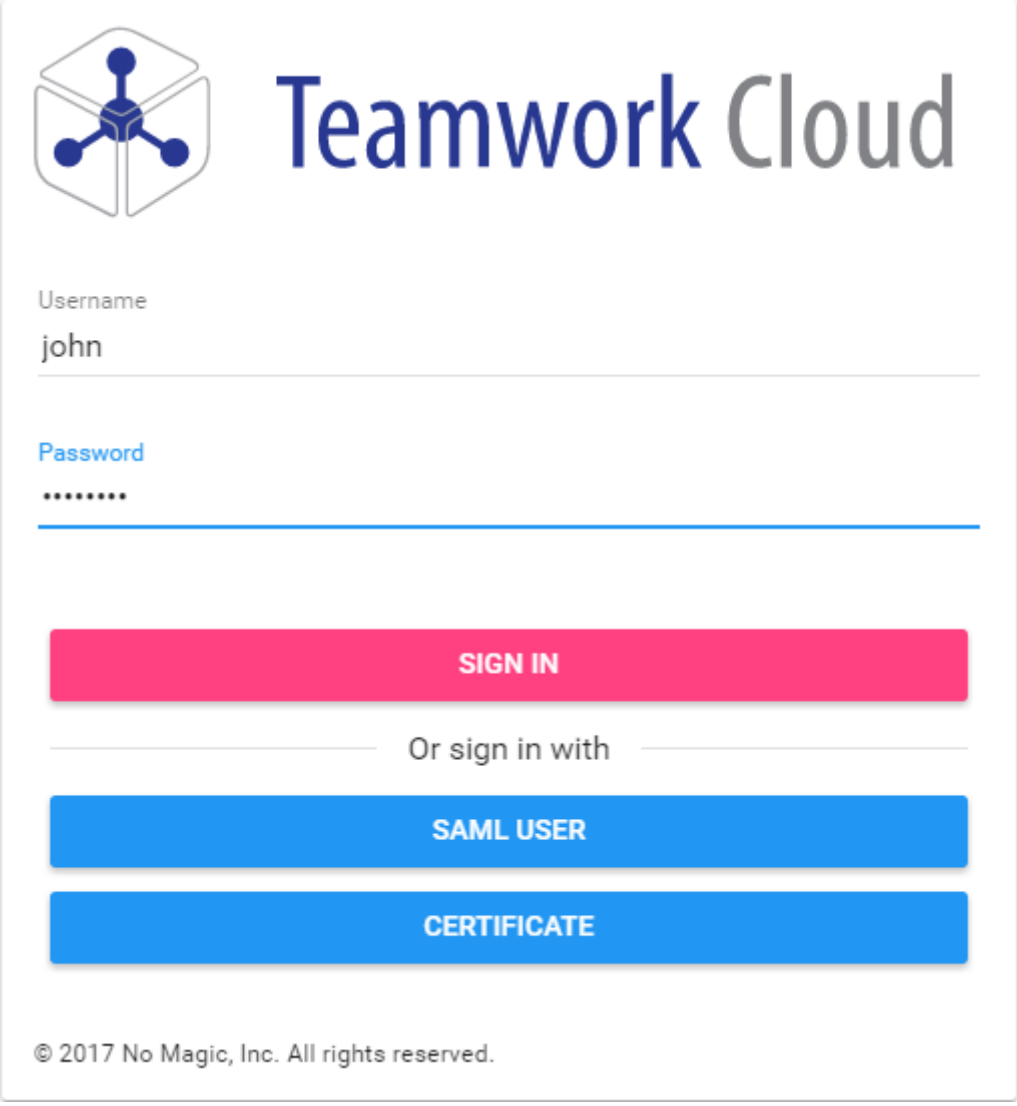
Related pages

- Single Sign-On Authentication(see page 1366)
- Working with Teamwork Cloud projects(see page 1367)
- Working with Teamwork Server projects(see page 1479)

Single Sign-On Authentication

The Single Sign-On (SSO) property is available to log on to a Teamwork Cloud server (TWC). For now, you can log using SAML authentication (e.g., ForgeRock, Ping, WSO2). And for you it means:

- No need to remember and type credentials each time you logging to a server.
- No issues about weak passwords and updating it.



Logging to TWC using SSO.

SSO implementation is based on the JxBrowser platform[303], and is supported by following OS:

- **Windows (32-bit & 64-bit)**
  - Windows 10
  - Windows 8 & 8.1

303 https://www.teamdev.com/jxbrowser

    - Windows 7
    - Windows Server 2008 R2
    - Windows Server 2012
    - Windows Server 2016
- **Linux (32-bit & 64-bit)**
    - Ubuntu 12.04+
    - Debian 7.7+
    - Fedora 20
    - openSUSE 13.1
    - RedHat Enterprise Linux 7

- **macOS (64-bit)**
    - macOS 10.9.x - 10.12.x (Intel)

For more information about supported OS, please refer to JxBrowser[304] system requirements[305].

Related pages

- Administrating the authentication server(see page 1366)

Using Teamwork Cloud

Introduction

Teamwork Cloud (TWCloud) is No Magic's next-generation repository for collaborative development and versioned storage of models. TWCloud is intended and developed to improve the areas of model governance, model analysis, and integration with third-party tools. The benefits to enterprise users include:

- Role-Based Access Control is a major feature that enables efficient management of users' access rights across the enterprise. Additionally, as distributed systems can be challenging to administer, we have built a convenient web-based administrative interface to TWCloud[306], which provides a single way to manage user accounts, licensing, access control, LDAP integration, and SSL setup.
- TWCloud is designed from the ground up to support working with large models. In particular, the efficient delta-based protocol used when users commit and retrieve updates to a model will save time for those users who access the repository through poor network connections. Likewise, the amount of data that is transmitted depends only on the size of the changes being made to the model.
- A database-driven model storage paradigm in TWCloud unveils powerful client-side features for model changes and history tracking at element level.

304 https://www.teamdev.com/jxbrowser
305 https://jxbrowser.support.teamdev.com/support/solutions/articles/9000013733-system-requirements
306 https://docs.nomagic.com/display/TWCloud190SP4/Using+TWCloud+Admin

The main differences between Teamwork Cloud and Teamwork Server in the modeling tool consist of the following:

- **Project Usages location**: rather than mounting everything under the main model root element, the Model Browser has a dedicated root for Project Usages in Teamwork Cloud.
- **Shared Packages**: there are no shared packages in the TWCloud Project Usages; all content is available for reference. However, the TWCloud project can be stripped using the Hiding packages(see page 1457) feature.
- The mounting of Used Projects in TWCloud is achieved via a model-based Mount relationship, rather than local persistence and Teamwork server. In order to ensure the same Containment tree structure when adding/migrating projects to TWCloud, original mounting is re-created by adding the Mount relationship automatically.
- Teamwork Server allows you to turn off the auto-load of used projects if performance with all the loaded projects is slow. Teamwork Cloud, on the other hand, does not allow you to choose the Used Project loading mode as it uses the lazy loading technology to determine which elements from Used Projects need to be loaded and when.
- Teamwork Cloud architecture supports the sticky versions (used project versions that remain used until they are changed in the main project) approach of used projects. However, you can easily update the used project to the latest version by setting up a notification that will inform you about the appearance of a new used project version. Alternatively, you can use the Teamwork Cloud Used Project Updater plugin(see page 1473) that allows the automatic update of used projects.
- In Teamwork Cloud, every new version of the project gets a unique consecutive number, even between different branches, which makes it easier to keep track of the commit history and identify the specific project versions. In Teamwork Server, every branch number starts from 0 and may repeat between branches.
- Teamwork Cloud used projects are read-only(see page 1441). This helps to prevent cyclic dependencies that may occur in Teamwork Server while working on multiple projects at the same time. We recommend that you open a used project as a stand-alone and edit it there; however, you can move elements to other projects using the cross-project refactoring functionality(see page 1456) in Teamwork Cloud as well. This feature allows transferring data between projects in just a few steps.

You can find more information about the TWCloud's key benefits, required resources, and the latest news on its official product page here http://www.nomagic.com/products/teamwork-cloud.html.

Related pages

- Adding projects to Teamwork Cloud(see page 1369)
- Managing server projects(see page 1374)
- Opening projects from Teamwork Cloud(see page 1383)
- Collaboration between disconnected teams(see page 1386)
- Offline modeling(see page 1395)
- Package permissions(see page 1399)
- Locking model for edit(see page 1403)
- Using Lock-Free Editing mode(see page 1409)
- Unlocking model(see page 1416)
- Committing changes to Teamwork Cloud(see page 1420)
- Updating changes from Teamwork Cloud(see page 1423)
- Reviewing model history(see page 1427)
- Managing project usages(see page 1441)
- Moving elements to other projects(see page 1456)
- Hiding packages(see page 1457)
- Managing Teamwork Cloud users(see page 1462)
- Used Project Switch Impact Analysis(see page 1463)

Used Projects Auto Update Plugin(see page 1473)
- Troubleshooting(see page 1478)

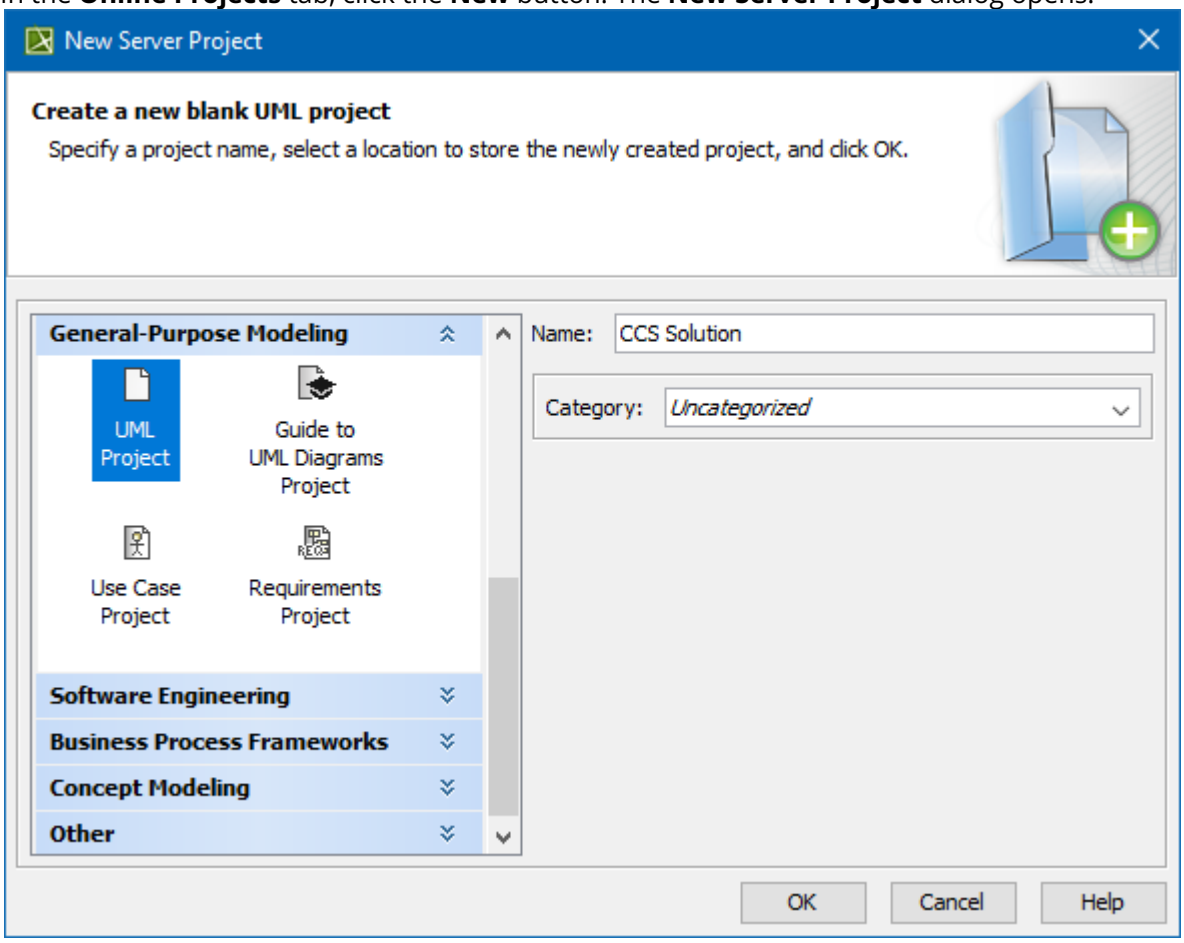
Adding projects to Teamwork Cloud

Projects can be added to Teamwork Cloud (TWCloud) in one of the following ways:

- By creating a project in TWCloud from scratch(see page 1369)
- By adding a local project to TWCloud(see page 1370)

To create a project in TWCloud from scratch

1. From the **Collaborate** menu, choose **Projects**. The **Manage Projects** dialog opens.
2. In the **Online Projects** tab, click the **New** button. The **New Server Project** dialog opens.



3. In the **Name** box, type a new project name.
4. From the **Category** drop-down list, select the category in which you want to store your project.

> You can create a new category in the server repository by typing its name directly in the **Category** box.

5. Click **OK**.

To add a local project to TWCloud

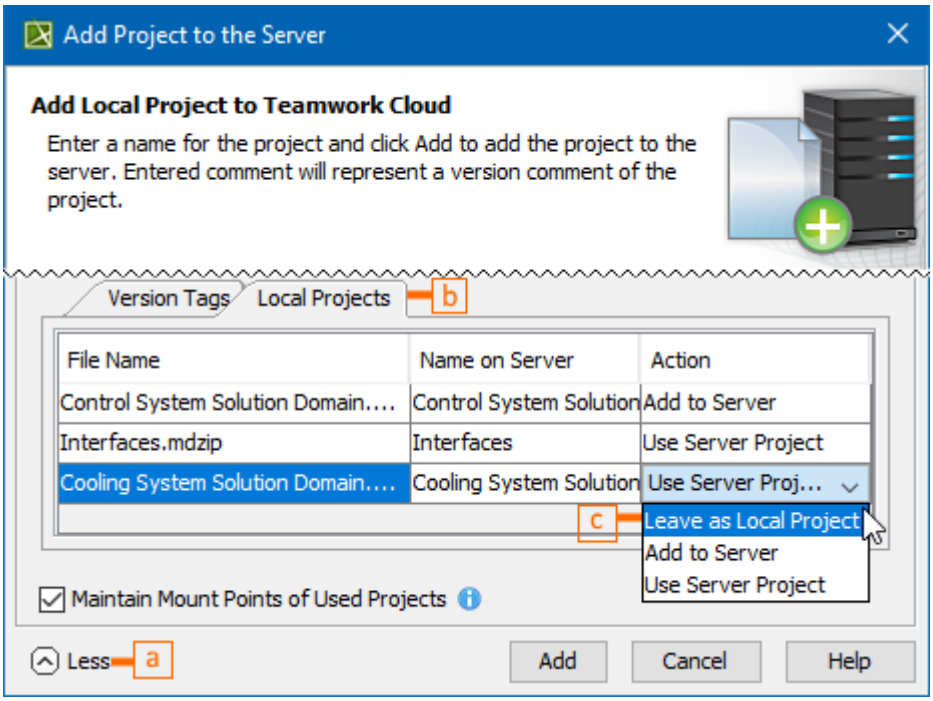
1. Log in to TWCloud(see page 1363).
2. Open the local project you want to add to TWCloud.
3. From the **Collaborate** menu, choose **Add Project to Server**. The **Add Project to the Server** dialog opens.
4. If you want to give the project a new name, type the name directly in the **Name** box when it is added to the server.
5. From the **Category** drop-down list, select the category in which you want to store your project.

   > You can create a new category in the server repository by typing its name directly in the **Category** box.

6. If you have some explanations or notes about this project, provide them in the **Comment** box.
7. If the project has no used projects, or if it has some that you want to be added to the server, skip this step. If you don't want any of the used projects to be added, do the following:

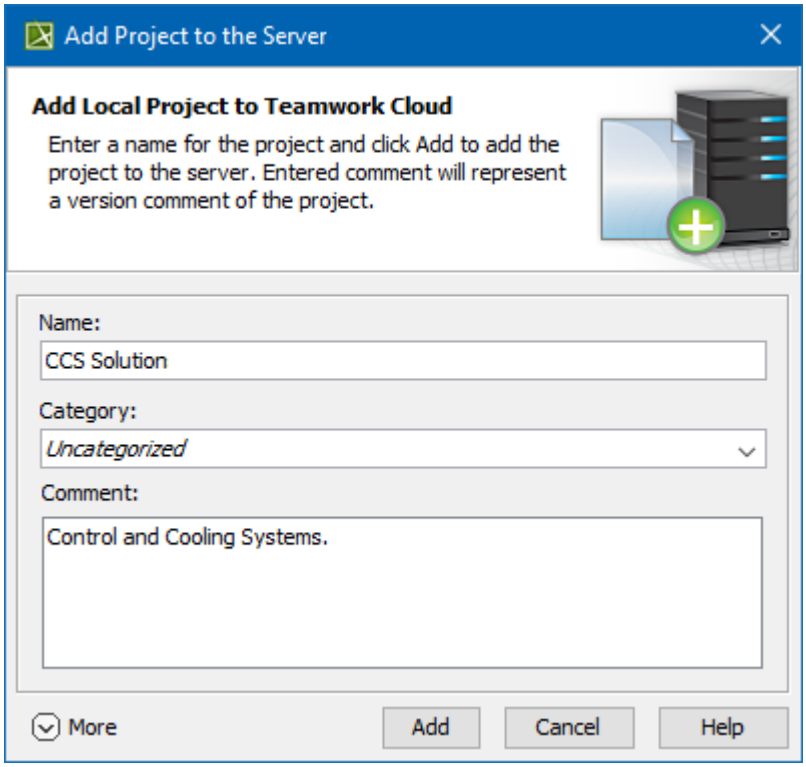
   a. Click **More** in the **Add Project to the Server** dialog for more options.
   b. Click the **Local Projects** tab. The list of used local projects opens.
   c. In the **Action** column of each local project that you don't want to add to the server, select **Leave as Local Project**.

   > If you exclude all or some of the used projects from the addition to TWCloud (see step #7), you can always link them to the project later.



   > You can disable the **Maintain Mount Points of Used Projects** option in the **Add Project to the Server** dialog to prevent the automatic creation of **Mount** relationships(see page 702) in your project.

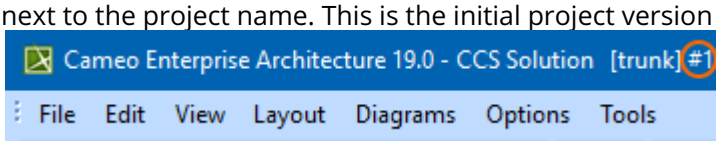
8. Click **Add**.



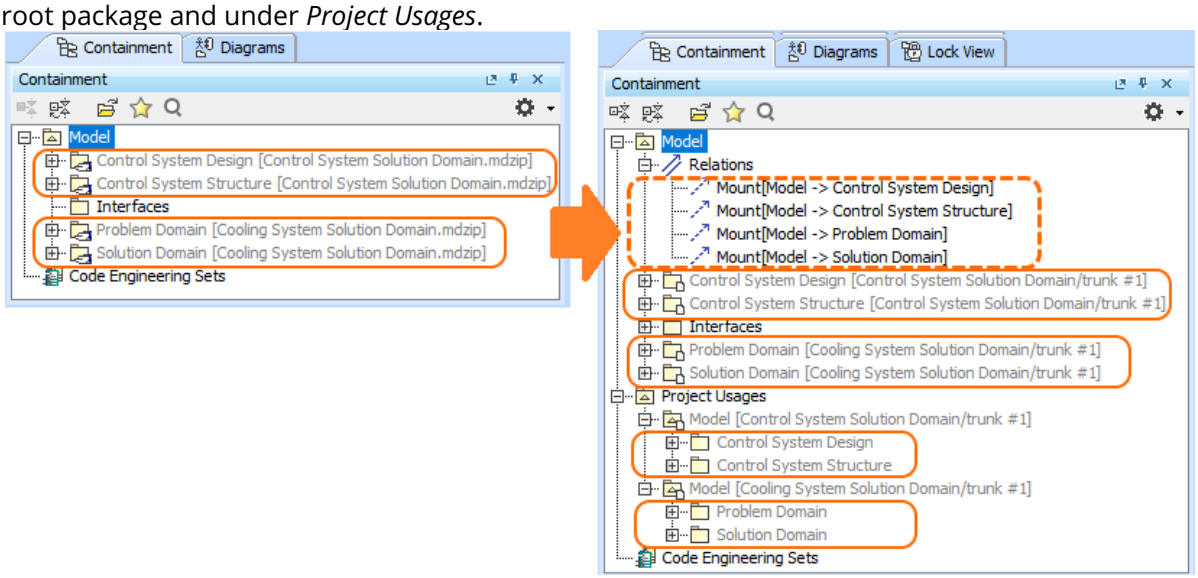
9. The project is now added to TWCloud.

When the project is added to the server:

- The title of the client application, e.g., MagicDraw, Cameo Systems Modeler, window displays*#1* next to the project name. This is the initial project version number.



- The contents of the Model Browser are gray(see page 1404). The model is unlocked.
- If the project is added to the used projects, you can see them in the Model Browser under the root package and under *Project Usages*.



- To view the project, select **Collaborate** > **Recent Server Projects.**

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related pages

- Migrating projects from Teamwork Server(see page 1372)
- Creating and managing categories(see page 1380)

Migrating projects from Teamwork Server

On this page

- Pre-conditions(see page 1372)
- Actions before migration procedure(see page 1372)
- Migration Procedure(see page 1372)
- Troubleshooting(see page 1373)

The following instructions describe how to migrate projects from MagicDraw Teamwork Server to the Teamwork Cloud (TWCloud). These instructions cannot be applied to users' and permissions' migration. The users and permissions can be migrated only manually.

Pre-conditions

- Server, which you want to migrate from, and the server, which you want to migrate to, are connected to the same network (recommended).
- TWCloud repository has enough space on disc to store migrated projects. We recommend to allocate at least 100 MB per project.
- Client and TWCloud server versions are the same. For example, both 19.0.
- The user who performs the migration has the **Administer project** or **Read model** permission on Teamwork Server and the **Resource Creator** role on TWCloud.

Actions before migration procedure

1. Make a back up[307] of the Teamwork Server repository.
2. Install a temporary Teamwork Server of the same version as the Teamwork Cloud.

   ⚠ Please contact your sales executive for getting a temporal licence key.

3. Start the temporary Teamwork Server with the backup repository.
4. Resave projects in this temporary Teamwork Server(see page 1496). After that, projects are ready to migrate to the Teamwork Cloud. This step is recommended for smoother and faster project migration to TWCloud.

Migration Procedure

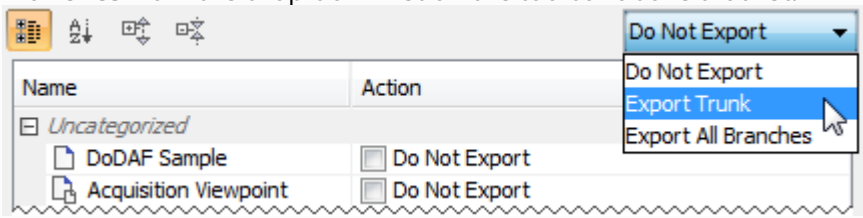
1. Log in to the temporary Teamwork Server(see page 1363).
2. From the **Collaborate** menu, select **Export Projects to TWCloud**. The **Project Export Wizard** opens.
3. Select the projects you want to migrate to TWCloud repository – in the **Action** column of each project, select one of the following:

   - **Export Trunk**, if you want export the last trunk version of that project.

307 https://docs.nomagic.com/display/TWCloud190SP4/Backup+and+restore+data+procedures

- **Export All Branches**, if you want to export the last trunk version of that project and the last versions of all its branches.
- **Export Custom Branches**, if you want to export the last versions of selected branches and/or trunk of that project.

> ⚠
> - If you select a project that is already exported to TWCloud, it will be skipped. The project cannot be exported twice.
> - To ensure project integrity, it is recommended to export all dependent projects (projects with its used projects) in one go. In this case you will not need to link used projects manually.
> - You can export projects in separate batches, but before exporting, you should identify projects related via used project. After export is done, used projects will end up as separately embedded projects instead of being correctly linked. Thus, you will need to link them manually.

> ⓘ If you want to export all the projects in the list, select **Export Trunk** or **Export All Branches** from the drop-down list on the toolbar above that list.
>
> 
>

4. Click **Next**.
5. Type your user name and password on TWCloud, and then provide the TWCloud server address.
6. Click **Finish**.
7. Wait while project copies are created and exported to TWCloud. This may take 5 to 15 minutes per project.

   > ⚠ Only the last versions of the selected projects are exported. Project history cannot be migrated.

8. Log out of Teamwork Server.
9. Log in to TWCloud(see page 1363).
10. From the **Collaborate** menu, select **Open Server Project**.
11. In the **Open Server Project** dialog, find the exported to TWCloud projects, open them, and perform collaborative tasks (see page 1479).

> ✓ The projects exported to TWCloud will have original commit details from the Teamwork Server:
> - Who committed the last commit in the Teamwork Server,
> - The date of the last commit in the Teamwork Server,
> - The comment of the last commit in the Teamwork Server.
>
> These details can be found in the Version Properties dialog.

Troubleshooting

Projects migration cannot be completed
This may happen because of many reasons, for example:

- Project has recovered elements
- Some profile/plugin is not installed
- Project elements are locked by other users (in case the project is of older format)

To get the thorough report of the particular issue, open the project and read the message that instantly appears on the project load. Fix the reported issue and try to export again.

If the problem persists, report the issue with the log file attached to No Magic Customer Support team.

Project migration is terminated because of lack of memory
To solve this issue, increase the size of memory allocated for the client application, that is, MagicDraw, Cameo Systems Modeler, or other. For this, do the following:

1. Open for edit the *.properties* file of the client application, for example, *magicdraw.properties*, which can be found in *<MagicDraw installation directory>\bin*.
2. In the JAVA_ARGS line, replace *-Xmx800M* with *-Xmx3700M* , which stands for 3,7 GB allocated, or greater value.
3. Restart the client application.
4. Try to export again.

User rights are insufficient to create projects in TWCloud
To export projects to TWCloud you need to have the **Resource Creator** role. Contact the TWCloud Administrator to obtain this role.

Managing server projects

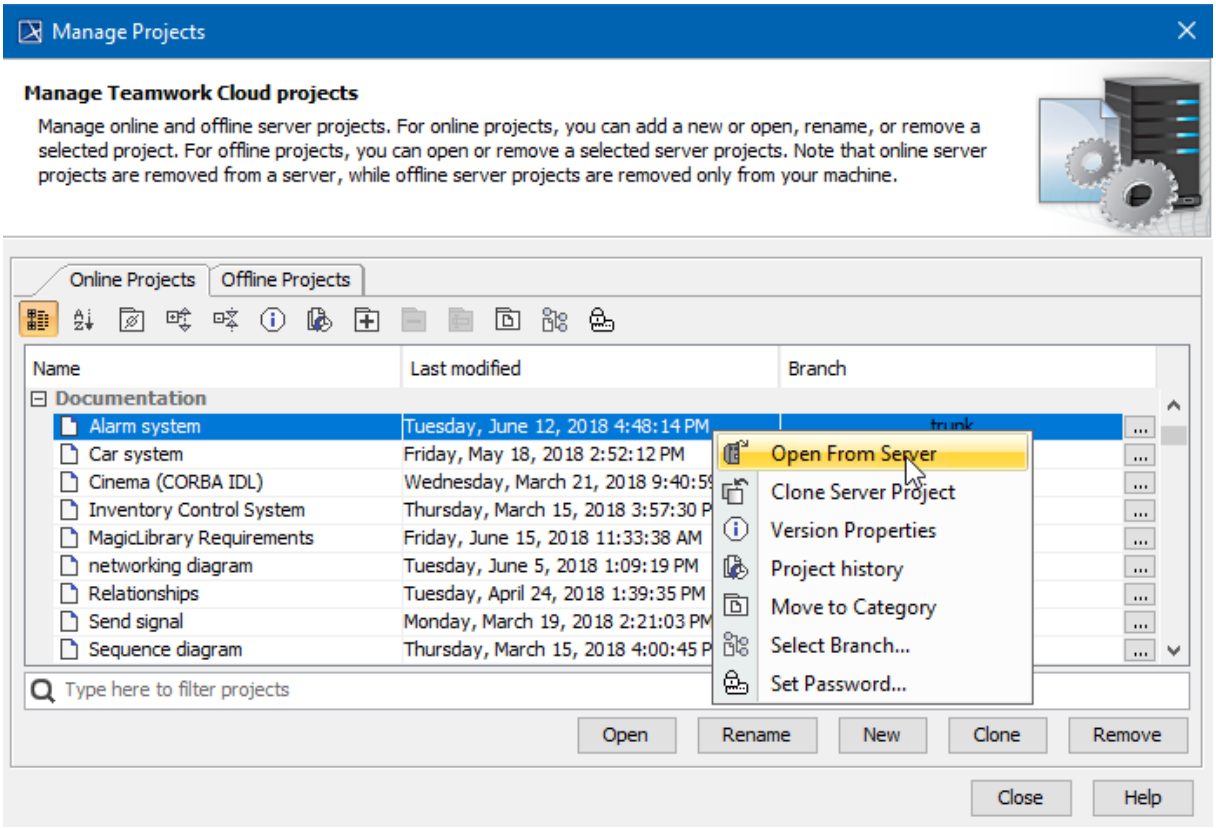
You can manage your server projects, as well as locally saved server projects (known as offline server projects(see page 1395)), in the **Manage Projects** dialog.

Currently open project is automatically selected in the **Manage Projects** dialog, making it easier to find it in the project list.

To open the **Manage Projects** dialog

---

1. Log on to the Teamwork Cloud server.(see page 1363)
2. On the main menu, click **Collaborate** > **Projects**.



You can use the search tab to quickly filter used projects by name.

In the **Manage Projects** dialog, you can:

- For server projects:
  - Add a new server project(see page 1369)
  - Open a server project(see page 1383)
  - Rename a server project
  - Remove a server project
  - Clone a server project(see page 1378)
  - Review and change project versions(see page 1428)
  - Manage server project categories(see page 1380)
  - Manage server project branches(see page 1375)
  - Set a password for a server project(see page 1381)

- For offline server projects:
  - Open a server project(see page 1395)
  - Remove a server project

Branching projects

Permissions
To create, remove, or rename a branch, you need to have an **Administer Resources** permission (see page 1375)assigned.

Project branching duplicates versions of a project and enables users to work simultaneously on them. Branching increases productivity for those, who have several development teams working on a single product and have to merge several development branches before releasing the product.

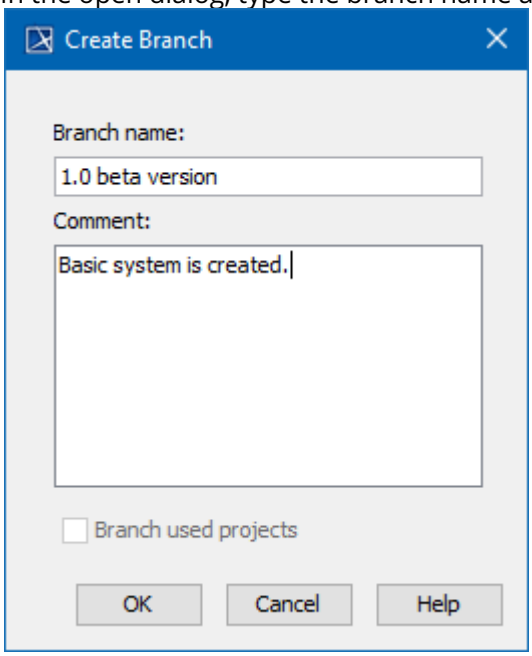
Branches can be created from any version of a Teamwork Cloud project or its branch. The branch version holds the information about the project version it is derived from. Every branch version can be tagged. Two versions of the same of different project branch can be compared (see page 1291)and merged (see page 1083)into a single one.

Projects that have usages can be branched *with* or *without* these usages. In the later case, the branch version of the project references versions of the projects used in the project, which that branch is derived from (it can be the latest version or some specific version selected by the user).

To create a project branch

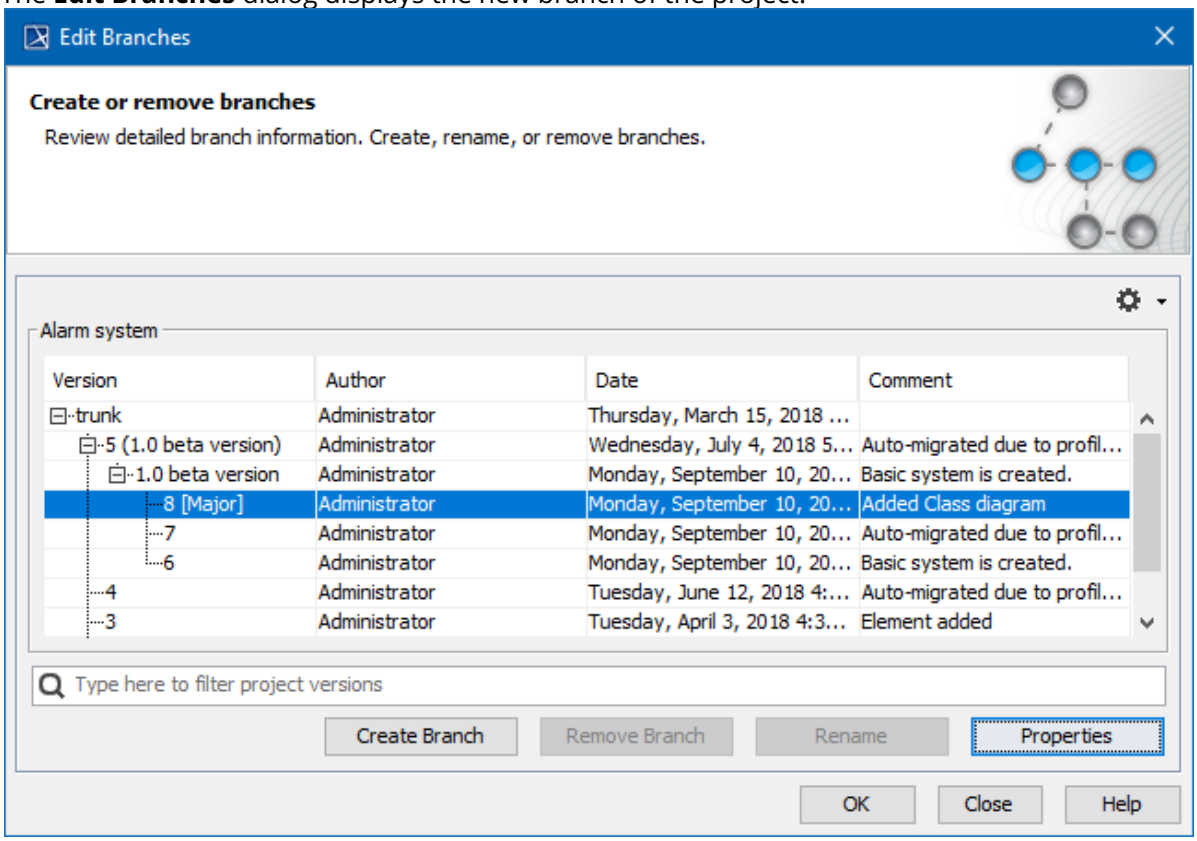
---

1. From the main menu, select **Collaborate** > **Projects**. The **Manage Projects** dialog opens.
2. Select a project you want to branch.
3. On the toolbar of the **Manage Projects** dialog, click the Select Branches button . The **Select Branch** dialog opens.
4. In the open dialog, click the **Edit Branches** button. The **Edit Branches** dialog opens.
5. Select a project version you want to branch.
6. Click the **Create Branch** button. The **Create Branch** dialog opens.
7. In the open dialog, type the branch name and comment.



8. Click **OK**. The new branch is created, and the dialog closes.

9. The **Edit Branches** dialog displays the new branch of the project.



Search bar
Use the search bar to filter projects according to:
- Project Version (together with the commit tags)
- Author
- Comment

10. Click **OK**. The dialog closes.
11. If you want to open the newly created branch, do the following:
    a. Select it in the **Select Branch** dialog and click **OK**. The dialog closes.
    b. In the **Manage Projects** dialog, see that branch instead of the branched project.
    c. Click **Open**. The branch opens.

Archived branch

The branches that are not being worked with anymore and are only maintained for historical purposes can be hidden from users eyes.

Performance and disk space are not affected because archived branches are hidden only graphically.

To archive a project branch

1. From the main menu, select **Collaborate** > **Projects**. The **Manage Projects** dialog opens.
2. Select a project you want to archive branch.

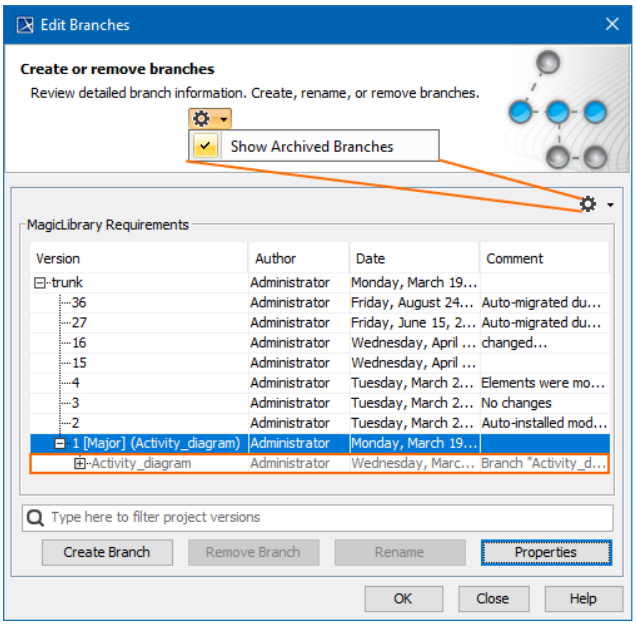
3. On the toolbar of the **Manage Projects** dialog, click the Select Branches button . The **Select Branch** dialog opens.
4. In the open dialog, click the **Edit Branches** button. The **Edit Branches** dialog opens.
5. Right-click the branch you want to archive and, on the shortcut menu, click **Mark as Archived.**

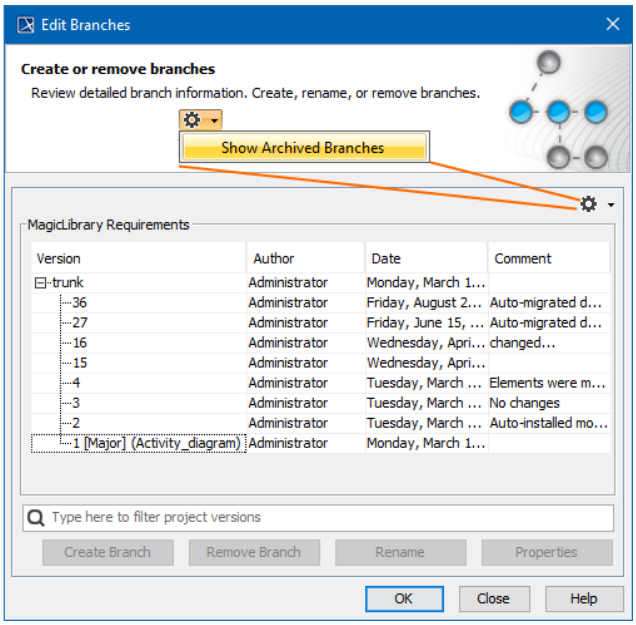
The archived branches can be recognized from the grayed out text. Same way you can uncheck archived branch and it will be no more archived.

To hide or show archived branches from the dialog

- Click and deselect **Show Archived Branches**, to hide archived branches.
- Click and select **Show Archived Branches**, to show archived branches.

> This action can be done in **Edit Branches** and **Select branch** dialogs.





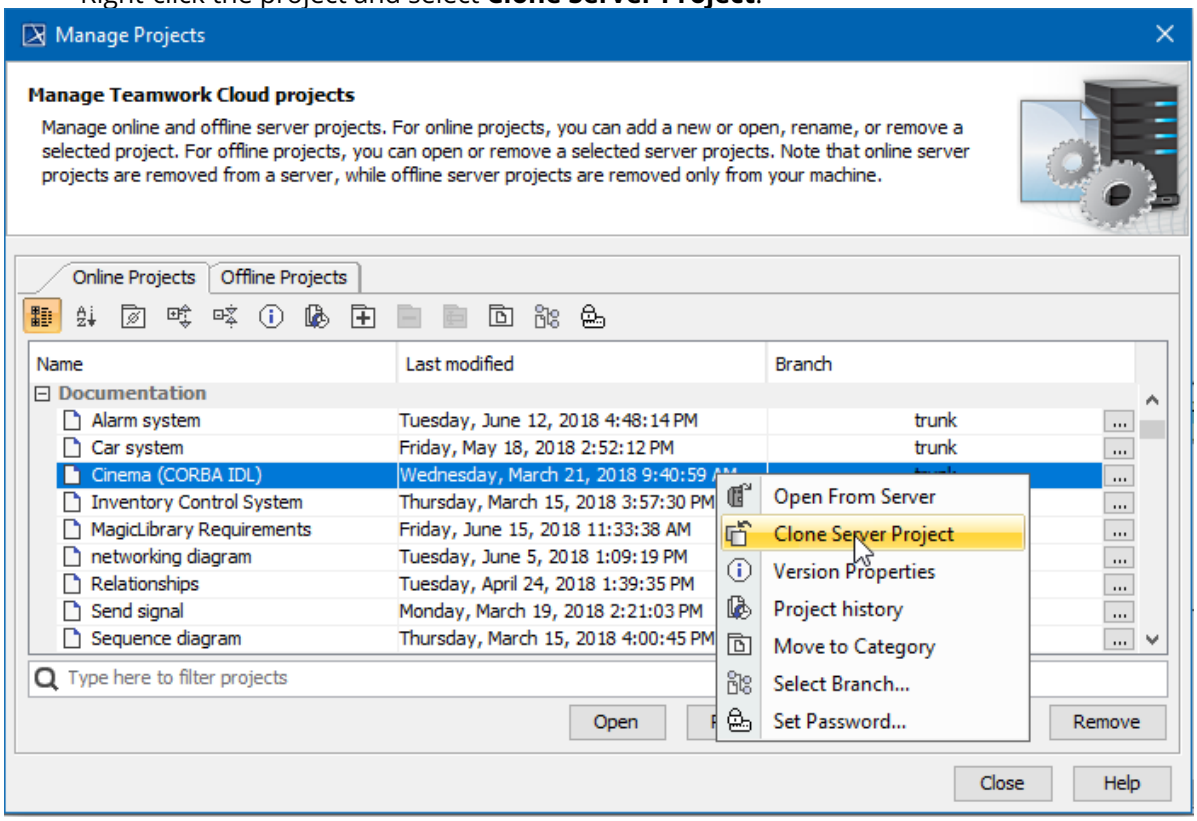
Cloning projects

You can clone the latest version of your server project as a separate project. The project being cloned can be used as a "template" to create a new project or as an independent base for new development. IDs of a cloned project and elements are reset automatically. Project permissions are reset as well; the project is available only for a user who has cloned the project. All references to used projects (if there are any) are maintained.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To clone a project

1. On the main menu, click **Collaborate** > **Projects**. The **Manage Projects** dialog opens.
2. Select a project version you want to clone and do one of the following:
   - Click the **Clone** button.
   - Right-click the project and select **Clone Server Project**.



You can use the search bar to faster filter used projects by name.

3. The **Clone Server Project** dialog opens, wherein you need to type a name of the cloned project, select a category to save the project and/or add a comment. Click **More**, if you want to add a tag for the project.
4. Click **Clone** when you are done.
5. Close the **Manage Projects** dialog.

Related pages

- Adding projects to Teamwork Cloud(see page 1369)

Creating and managing categories

Using categories and grouping projects into them, you can organize the structure of your server projects' repository according to your needs.

By default, there is a single category *Uncategorized*, but you can create as many categories as you need. A project can be created directly in a selected category and later easily moved to another one. The category can be created in advance or while adding the project to the server(see page 1369). Categories can be renamed and removed. However, they cannot have sub-categories.

> ⚠ The TWCloud project categories in the modeling tool dialogs will be hidden if user does not have *Categorize Resources* permission or *Read-Only* permission to at least one project inside of these categories.

Read the following to learn how

- To create a single category(see page 1380)
- To rename a category(see page 1380)
- To move a project to another category(see page 1381)
- To remove the category(see page 1381)
- To hide empty categories(see page 1381)

> ⓘ Categories can also be managed via the Resource application. Learn more about this in Teamwork Cloud User Guide[308].

To create a single category

1. From the **Collaborate** menu, choose **Projects**. The **Manage Projects** dialog opens.
2. Be sure the Categorized View is turned on in the dialog. If it is not, click on the toolbar of the dialog.
3. Choose one of the following:
   - Click on the toolbar of the open dialog.
   - Right-click anywhere in the project list area and then select **Create Category**.
   - Press Insert.
4. After the **Create Category** dialog opens, type a category name.
5. Click **OK** to confirm the creation.
   The category is created. You can find it in the project list ordered alphabetically.

To rename a category

1. From the **Collaborate** menu, choose **Projects**. The **Manage Projects** dialogs opens.
2. Select a category or a project that belongs to the category you want to rename and then do either:
   - Click on the toolbar of the open dialog.

308 https://docs.nomagic.com/display/TWCloud190SP4

- Right-click the selection and then select **Rename Category**.

  The *Uncategorized* category cannot be renamed.

3. When the **Rename Category** dialog opens, type a new category name.
4. Click **OK**.
   The category is renamed.

To move a project to another category

1. From the **Collaborate** menu, choose **Projects**. The **Manage Projects** dialogs opens.
2. Select the project you want to move and do one of the following:

   - Click on the toolbar of the open dialog.
   - Right-click the selection and then select **Move to Category**.

3. When the **Move to Category** dialog opens, select the category to which you want to move the project.
4. Click **OK**.
   The project appears in the selected category.

To remove the category

1. From the **Collaborate** menu, choose **Projects**. The **Manage Projects** dialogs opens.
2. Select the category or the project that belongs to the category you want to remove and then do either:
   - Click on the toolbar of the open dialog.
   - Right-click the selection and then select **Remove Category**.

     The *Uncategorized* category cannot be removed.

3. Click **Yes** to confirm your decision.
   The category is removed from the server repository. As a result, projects from that category are automatically moved to the *Uncategorized* category.

Empty categories are by default shown in the **Manage Projects** and **Open Server Project** dialogs, but you can simply hide them.

To hide empty categories

- Click on the toolbar of the open dialog. Click the button once more to show the empty categories again.

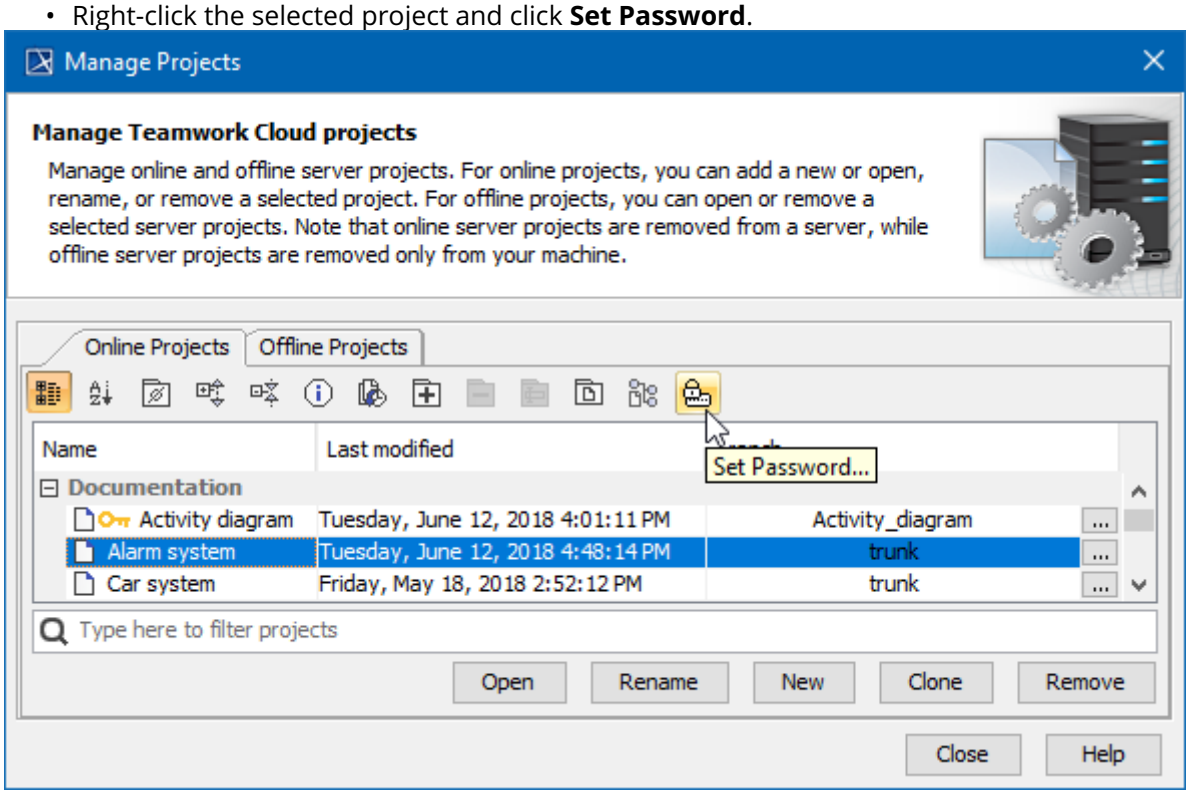
Password-protected projects

You can protect your server projects with a password. The password covers all project versions, and can be set or removed by a project owner or a user having **Administer Projects** permission(see page 1381).

You must be logged to a server to set a password for a particular project.

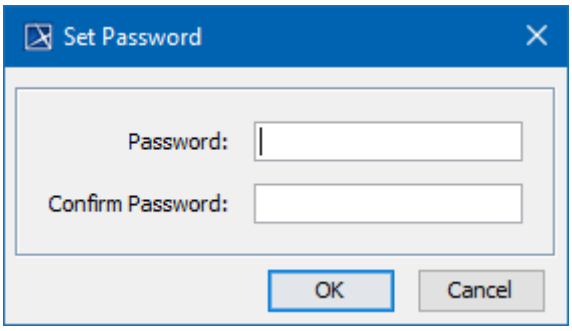
To set a password for a project

1. On the main menu, click **Collaborate** > **Projects**. The **Manage Projects** dialog opens.
2. In the **Online project** tab, select a project, you want to protect by a password and do one of the following:
   - On the dialog toolbar, click the **Set Password** button.
   - Right-click the selected project and click **Set Password**.



You can use the search bar to faster filter used projects by name.

3. The **Set Password** dialog opens, wherein you need to type and confirm the password.



The icon indicates password-protected projects.

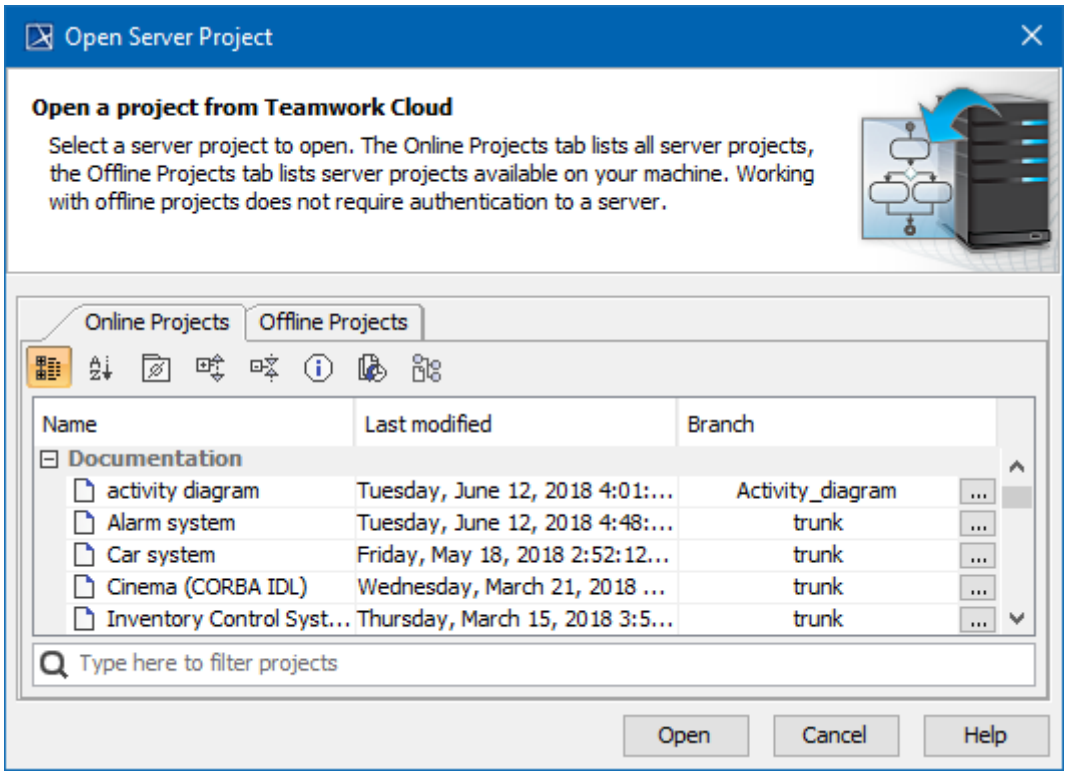
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Opening projects from Teamwork Cloud

The **Open Server Project** dialog is used for opening and managing projects from the Teamwork Cloud (TWCloud).

To open the Open Server Project dialog

- From the **Collaborate** menu, select **Open Server Project** or **Projects...**.
- On the **Collaboration** toolbar, click the button.
- Press Ctrl+Shift+O.

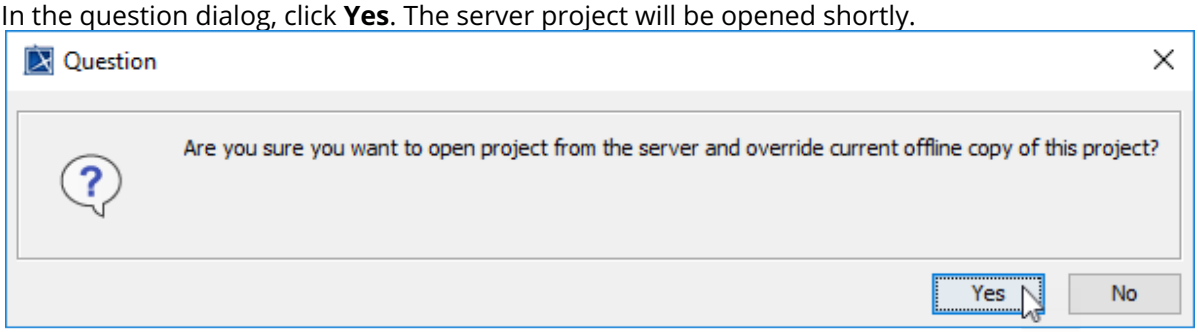


The Open Server Project dialog.

In the dialog, you may select whether to open an online project (that is a server project from the server) or an offline project(see page 1395) (that is a server project form your machine). However, if an online project has an offline project version, the offline project will be opened even if you choose to open the project from the **Online Projects** tab. This mechanism ensures faster project load and ensures that your offline not yet committed changes are not lost. On the other hand, you can force open the online project (see instructions(see page 1384) below), but, in this case, all changes made and not committed in the offline project will be lost.

To open the project from the server

1. From the **Collaborate** menu, select **Open Server Project** or **Projects...**.
2. In the new dialog, open the **Online Projects** tab.
3. Right-click the project you want to open.
4. From the shortcut menu, select **Open From Server**.
5. In the question dialog, click **Yes**. The server project will be opened shortly.



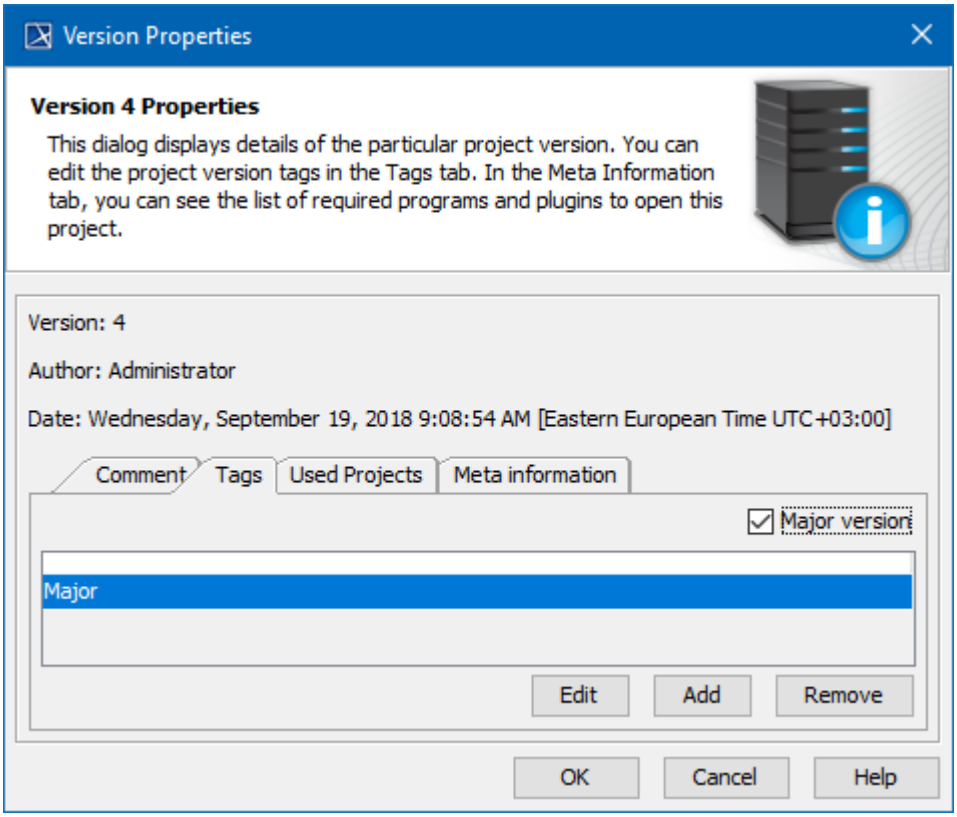
Password-protected projects

- You may be prompted to enter a password if the selected project is password-protected(see page 1381).

- You can use the search bar to filter used projects by name.
- Currently, an open project is automatically selected in the **Open Server Project** dialog, making it easier to find it in the project list.

Button	Description
Show empty categories	When selected, shows empty categories. Cancel the selection if you do not want empty categories to be displayed.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Button	Description
Version Properties	Opens the **Version Properties** dialog. In this dialog, you can see project information, such as comments, tags, modules, and meta information. You can find out the author, version number, and date including time zone. The button is available only when the project is selected. In the Tags tab, the **Major version** checkbox was added, to easily add tag *Major.* 
Project History	Opens the **History** dialog. In this dialog, you can find the project change history. You can find out who made the changes, what and when has been changed. Also, you can open earlier versions of the project or even restore your project if any changes have corrupted your project. The button is available only when the project is selected.

Button	Description
Select Branches	Opens the **Select Branch** dialog. In this dialog, you can see if the project has branches. If the project has branches, you can select them. Furthermore, you can create and manage project branches. Click ... in a project line to select a particular project branch.

Related pages

- Offline modeling(see page 1395)

Collaboration between disconnected teams

Teamwork Cloud server allows for contributing on huge projects, supporting multiple contractors who are not able to work on the same Teamwork Cloud installation. This feature allows you to import *.mdzip* files that have been previously exported back to the server project. Thus, you can simply save your project locally(see page 1387) as a *.mdzip* file and send this file to another contractor. The contractor is then able to update the very same server project from the local project. Once the contractor completes project editing, he or she can send the project back to you. This life cycle may continue between any amount of contractors. The feature works even when multiple contractors work with separate Teamwork Cloud installations as well as local work environments. Teams can either use Teamwork Server in addition to Teamwork Cloud, or select not to use Teamwork Server or Teamwork Cloud at all.

The main contribution workflow is as follows:

1. Contractor A saves the project locally.(see page 1387)
2. Contractor B gets the locally saved project for editing, modifies it and sends back to Contractor A. (see page 104)
3. Contractor A receives the modified project and updates the server project.(see page 1387)

> ⚠ To update the server project from the local one, you must have **Administer Project** permission[309].

« Opening projects from Teamwork Cloud(see page 1383)

Saving a local copy of Teamwork Cloud project »(see page 1387)

Related pages

- Saving a local copy of Teamwork Cloud project(see page 1387)
- Branching projects(see page 1375)
- Model Merge(see page 1083)

309 https://docs.nomagic.com/display/TWCloud190SP4/Permissions

Saving a local copy of Teamwork Cloud project

To assign a Teamwork Cloud (TWCloud) project to a user who cannot access the project on the server, or to move the project to another server, you can save a local copy on your computer. This enables you to send the local copy by email or add it to another server.

> **Note**
> The local copy of the project will not have locking and project version information. You will not be able to commit local project changes to the server.

To save a local copy of a TWCloud project

1. From the **File** menu, select **Save Project As**.
2. The question box appears. If you are sure you want to save a local copy of the server project, click **Save**.
3. Specify a name and location to save the project.
4. Wait while the local copy is saved to your computer.

« Collaboration between disconnected teams(see page 1386)

Updating a server project with local changes »(see page 1387)

Related pages

- Collaboration between disconnected teams(see page 1386)
- Offline modeling(see page 1395)

Updating a server project with local changes

Once a third-party has modified the locally saved project and sent it back to you, you can update your server project with local changes. To update your project, you need to import the received file back to the Teamwork Cloud server as a new version of the project.

> To update a server project from a local project, you need to have the Administer Resources[310] permission assigned to you.

> The branch selected for the update cannot contain any locks. They must be released before updating the project.

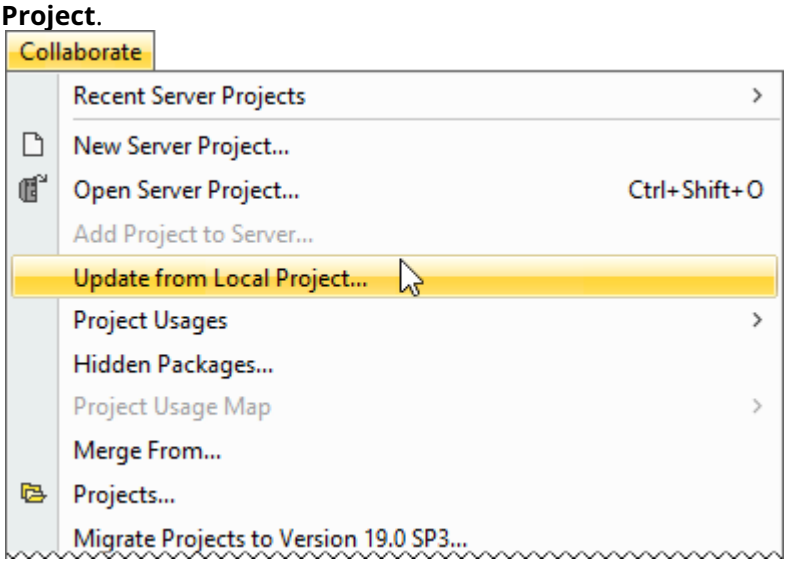
To update a server project with local changes of the same project

1. Do one of the following:

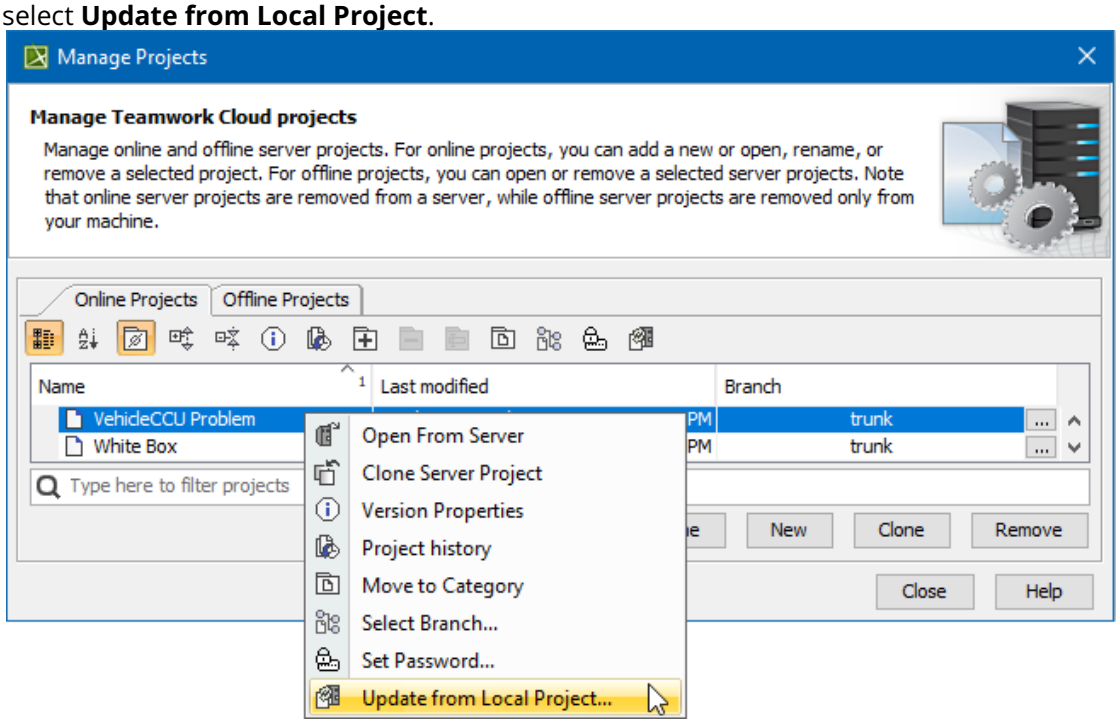
310 https://docs.nomagic.com/display/TWCloud190SP4/Permissions

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Open the server project. In the main menu, click **Collaborate** > **Update from Local Project**.

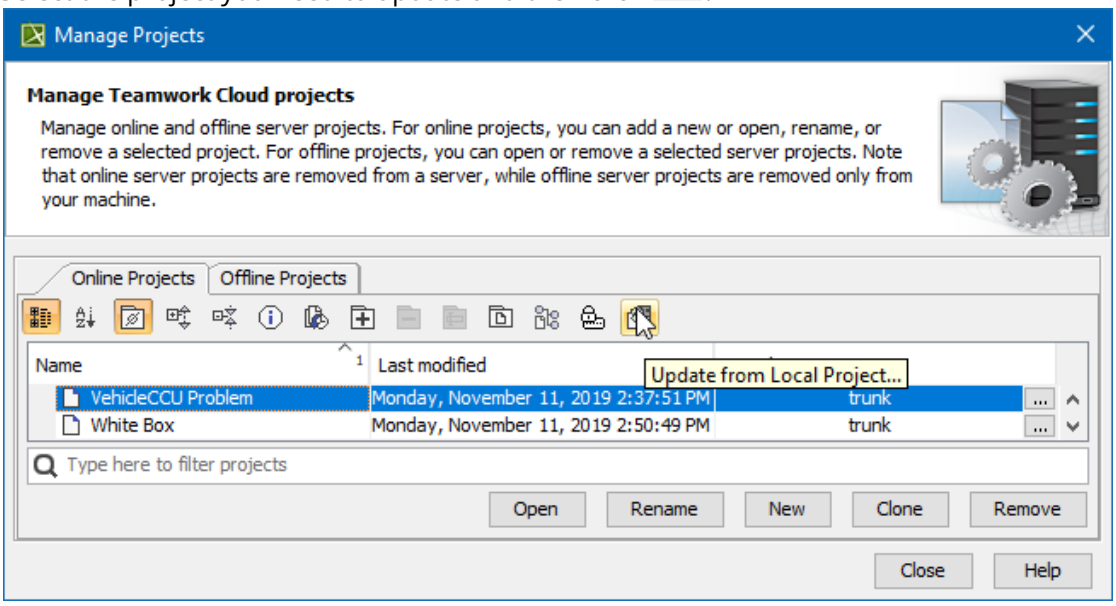


- On the main menu, click **Collaborate** > **Projects.** The **Manage Projects** dialog opens. Right-click the project you need to update under the **Online Projects** tab, and then select **Update from Local Project**.



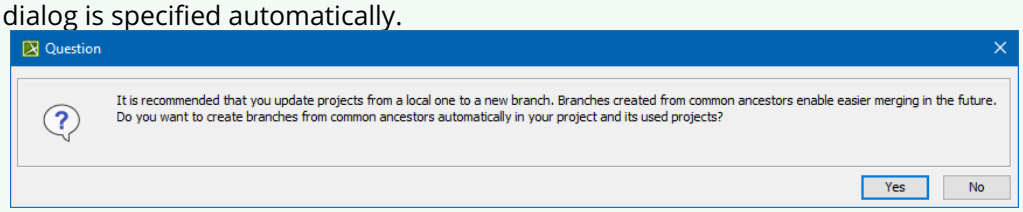
- On the main menu, click **Collaborate** > **Projects.** The **Manage Projects** dialog opens.

  Select the project you need to update and then click .



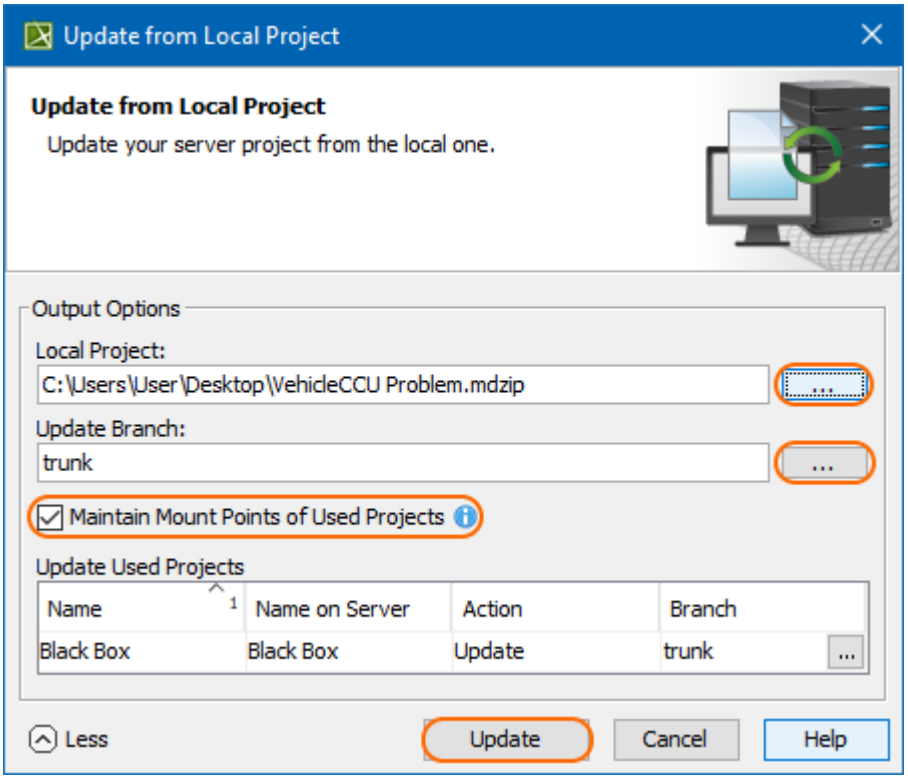
2. When the **Update from Local Project** dialog opens:
   - select a local file exported from the server project you are working on
   - specify the branch to update

   It is highly recommended that you create a new branch from the locally exported version of the server project. This will ensure that the changes made in Teamwork Cloud can be merged(see page 1083) with local changes by using a correct common ancestor.
   If the **Question** dialog appears, click **Yes** to automatically create a new branch from the locally exported version.
   If you click **No**, the branch that is opened or selected in the **Manage Projects** dialog is specified automatically.



   - Disable the **Maintain Mount Points of Used Projects** option if you do not want **Mount** relationships(see page 1393) to be automatically created upon the project update.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

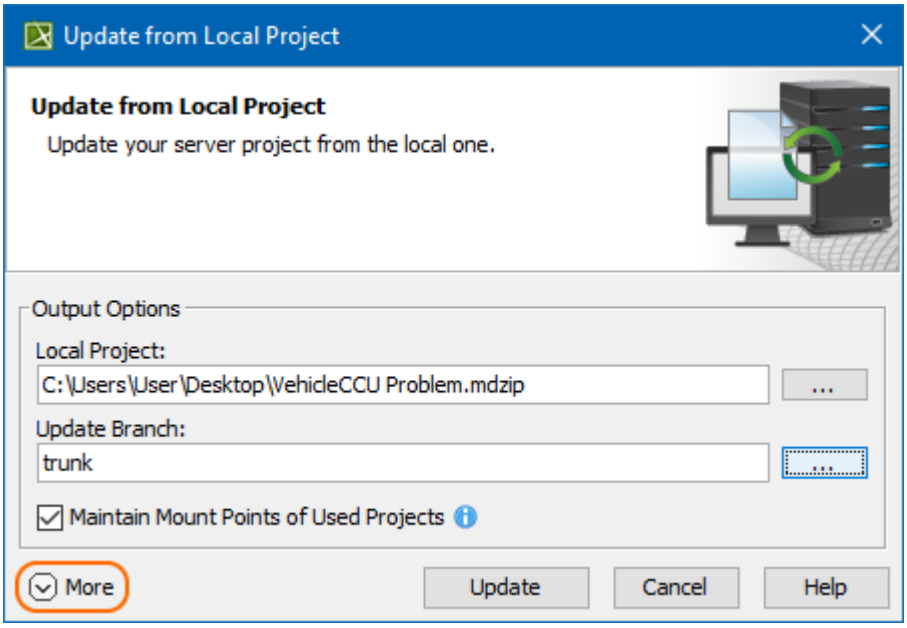


3. Click **Update** when you are done. An updated version of the project with the contents of the local project is created in the selected branch.

> ⚠ The server project will not be updated if the tool recognizes that the local project is the same version as the server project. The same rule goes for used projects.

Used projects

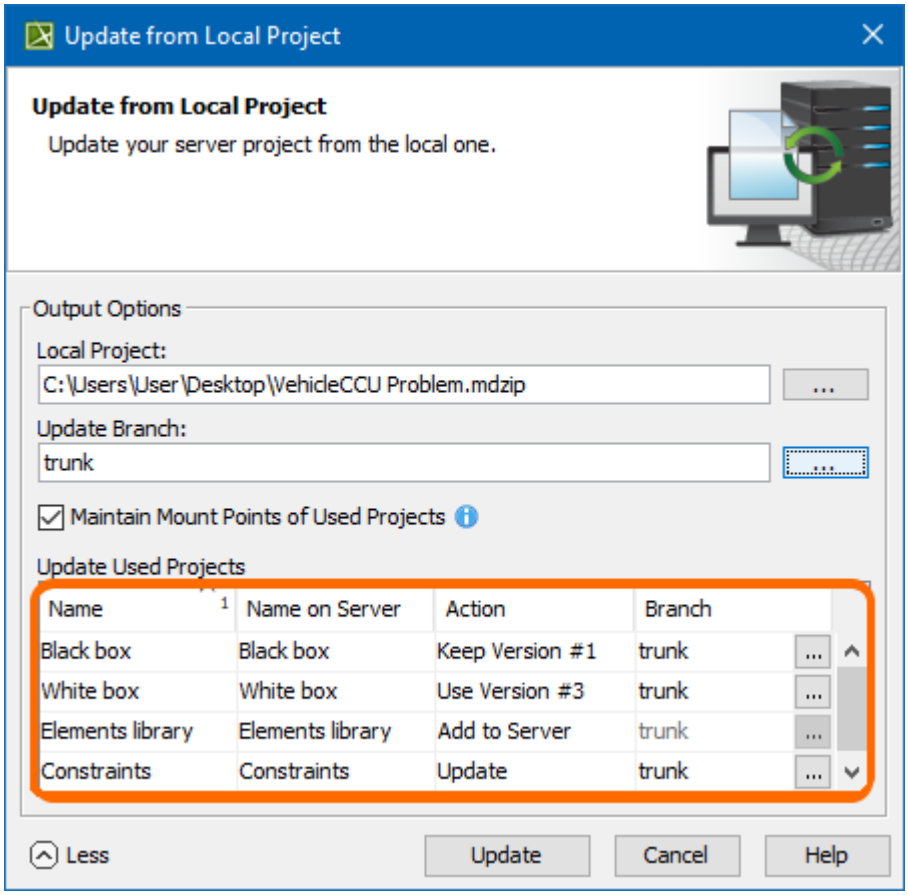
If the exported version of the server project contains used projects, their status can be reviewed in the **Update from Local Project dialog** when the project updates. To review updates, click **More** in the bottom-left corner of the dialog.



Depending on the changes made in the used project, the following actions can be performed when the project updates:

- **Keep Version #<number>**: if the used project version is the same as the one on the server (for example, it has not been modified locally), the same version of that used project is kept when the main project updates.
- **Use Version #<number>**: if the used project version on the server differs from the one in the local file, the version that is available on the server and matches the local file contents is used.
- **Add to server**: if a new used project is added to the local file and it does not exist in the server project yet, it is added to the server upon the main project update.
- **Update**: if the local file contains used project-related changes, the changes are applied to the server project upon the main project update.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



During a server project update from a local project, commits to branches being updated are not allowed, meaning other users working on the same project branch cannot commit the changes they have made.

« Saving a local copy of Teamwork Cloud project(see page 1387)

Understanding differences in local and server project structure »(see page 1392)

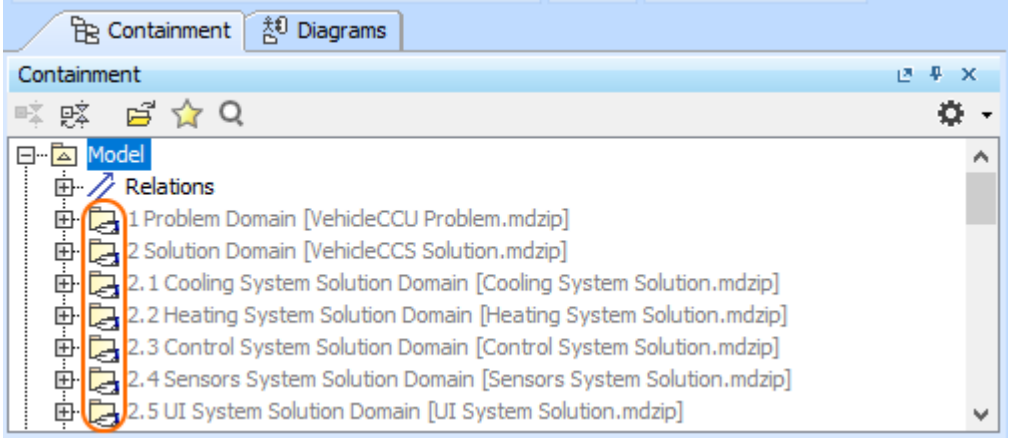
Understanding differences in local and server project structure

The structure of your model may change when switching back and forth between local and server projects. Usually, this happens when the local project is added to the server, the server project is saved locally, or the server project is updated with local changes.

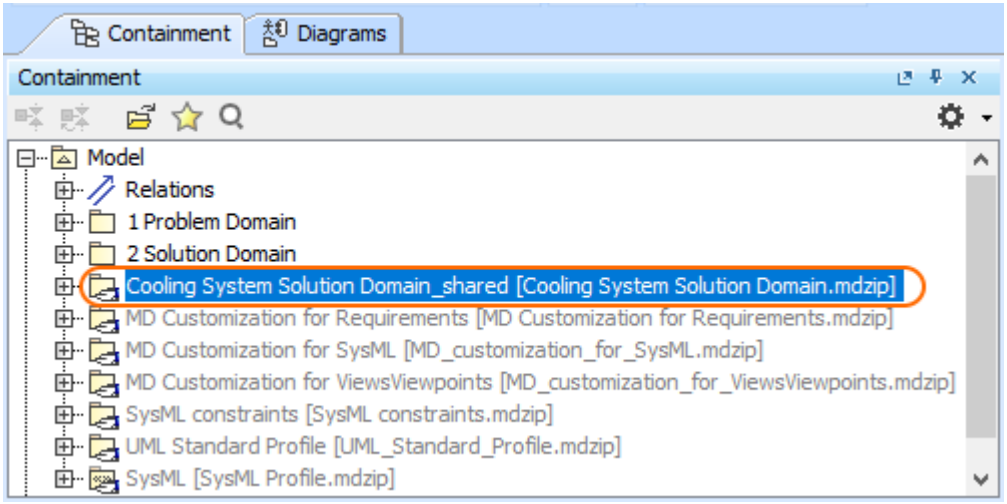
From Teamwork Cloud to local project

- When the Teamwork Cloud project containing used projects is saved locally, packages from used projects are mounted under the model root (Model) and marked with the (shared package) symbol. The original model structure (as it is in the server project) cannot be

maintained due to model architecture limitations, i.e., the absence of the dedicated root node for used projects.



- In the local environment, only shared packages will be visible in the used project. Therefore, if the used project contains elements under the model root, a new package **<userProjectName>_shared** is created when you save the project to make it possible to share those elements.



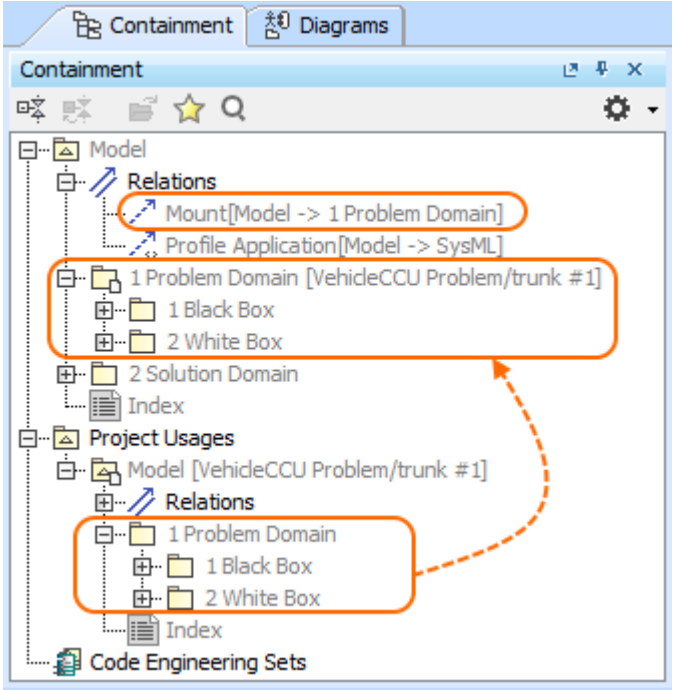
From local project to Teamwork Cloud

Mount relationship

- The **Mount** relationship may be created when the server project containing used projects is saved locally, modified, and then added back to the server.

It is used to recreate the same structure of the Containment tree as it was originally in the local project. This may be important for features that use a defined scope of the model. Only elements from used projects that are mounted under the Model root (Model) or any of its packages are considered part of the model scope (when no used projects are set as the scope).

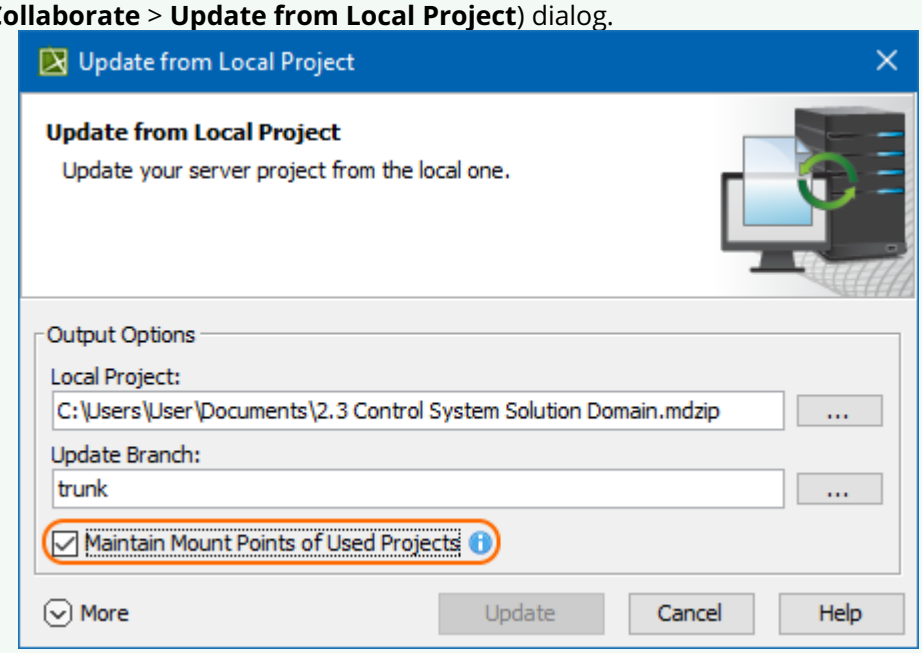
In general, the Mount relationship enables you to create a visual copy/representation of the chosen package in any place in the model.



- You can delete **Mount** relationships from your project if, for example, you do not want to have duplicate packages in the Containment tree of your server project.

Deleting Mount relationships may affect diagrams, elements or other options that require specifying the scope since packages from used projects will no longer be under the main project model root (they will still be available under **Project Usages**).

If you do not want Mount relationships to be created by default, remove the tick from the **Main tain Mount Points of Used Projects** checkbox in the **Update from Local Project** (**Collaborate** > **Update from Local Project**) dialog.

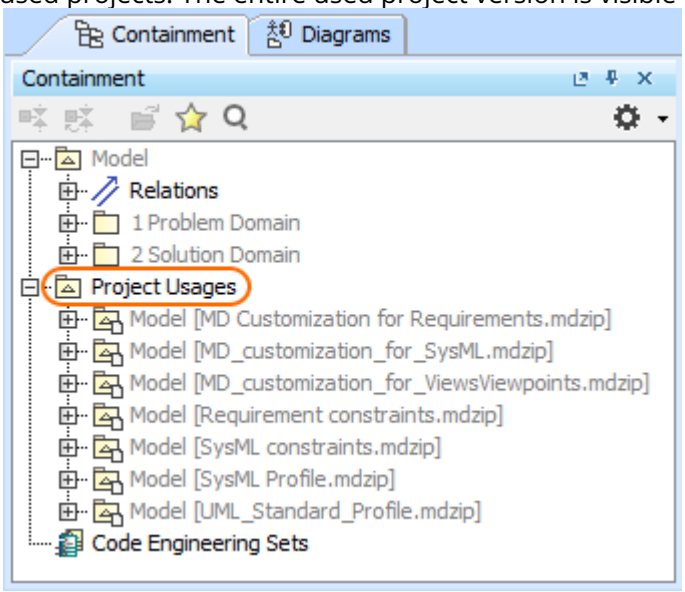


Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Packages from used projects mounted under the model root or any of its packages via the Used Projects(see page 245) dialog are kept in their original locations by using Mount relationships. They are automatically created once the project is added to the server.

Project Usages

The root node **Project Usages** is created in the Containment tree in Teamwork Cloud projects to list all used projects. The entire used project version is visible in the project.



Symbols used

Symbol	Description
	Indicates that a package belonging to the used project is shared.
	Indicates that a shared package belongs to the opened project.
/ /	Indicates that there is a package/profile/model from the used project in the Teamwork Cloud project.
	Packages from used projects have the project name they come from written in brackets.

Offline modeling

The Teamwork Cloud server supports offline modeling by allowing you to:

- Work with the project (read/edit) when a connection to the server is not available.
- Save intermediate changes locally (privately), and commit changes to the server when required.
- Save time by saving changes locally, which is much faster than committing directly to the server.

In an offline project, you can:

- Edit elements locked by a current user.
- Edit element symbols of elements locked by a current user.
- Create new elements/diagrams in the model.
- Review a project structure (used projects), project options, and styles.
- Save changes locally.
- Use a local project (if permissions allow).

When a connection is established, you can commit and/or update project changes, and lock or unlock project elements.

Make project available offline

To make an online project available offline

1. Open a server project.
2. From the main menu, select one of the following:
    - **Collaborate** > **Saves Changes Locally / Make available offline**.
    - **File** > **Save Project**.

> **Not a local copy of a server project**
> This is **not** a local copy of a server project! All user locks and changes are saved within the project, so the offline project can be committed to the server later.

Open offline project

To open an offline project

1. From the **Collaborate** menu, select either **Open Server Project** (or press Ctrl+Shift+O), or **Projects**.
2. In the open dialog, click the **Offline Projects** tab and select the offline project to open.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

In the title bar, you can see information about a project opened from TWCloud. This information allows you to distinguish Online from Offline projects:
If you open the Online project, the title bar will show:

- Modeling tool name and version,
- Project name,
- Project trunk or branch,
- Project version,
- Server name.

MagicDraw 19.0 - Alarm system/trunk #1 [10.2.2.47:3579]

If you open the Offline project, the title bar will show:

- Project name,
- Project trunk or branch,
- Project version,
- Server name,
- Last user who saved the project,
- Available Offline.

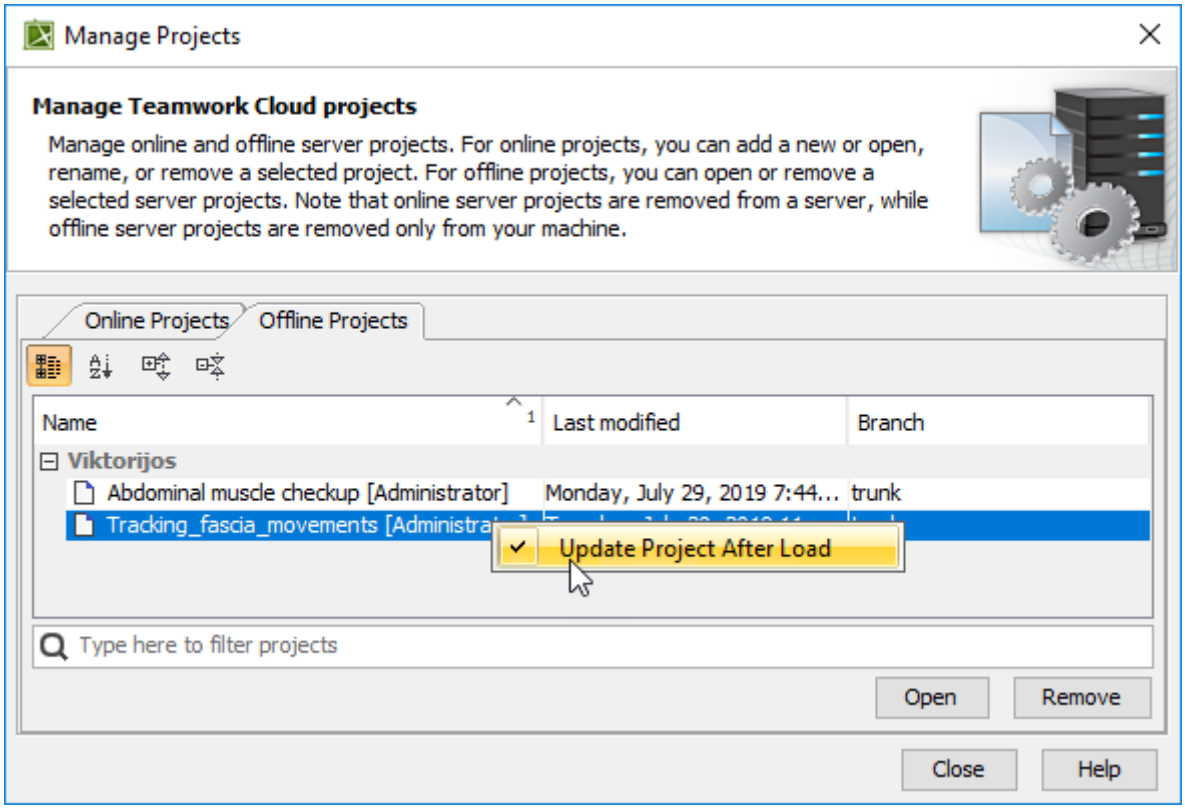
MagicDraw 19.0 - Alarm system #1 [10.2.2.47:3579, Saved by User: Administrator] Available Offline

Update project after load

You can open the project as it is saved locally (without checking for the latest updates) or open the project and update it automatically from the server, if a connection is available.

To include updates, right-click the selected project and select **Update Project After Load** for each project you want to update when loading.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The selection to update is persistent
Keep in mind that if you select to update the project after loading it is persistent and remains selected after reloading the project.

Remove offline project

To stop using the offline project version you need to remove it. This action does not remove a project from the server; it only removes the offline copy from the actual local machine. However, if you do not want that offline project would be removed, you can force open server project(see page 1383). In this case, changes made to the offline project and not committed will be lost.

To remove an offline project

1. From the **Collaborate** main menu, select **Projects...**. The **Manage Projects** dialog opens.
2. In the **Manage Projects** dialog open **Offline Projects** tab.
3. Select an offline project and click the **Remove** button.

Related pages

- Committing changes to Teamwork Cloud(see page 1420)
- Locking model for edit(see page 1403)
- Unlocking model(see page 1416)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Package permissions

By default, all packages in the project can be reviewed and modified by any Teamwork Cloud user.

There are two levels of permissions:

- The **Global permission** specifies which permission is applied for all project packages for all users:
  - Select **Read-Write** as the **Global permission** to restrict editing of the selected package, leaving the rest of the model editable.
  - Select **Read-Only** as the **Global permission** to restrict editing of the entire model, leaving only the selected package editable.

  For each package, the global permission can be over-specified by the package permission.

- **Package permissions** specify the permissions applied to a particular package for a particular user. If no permission is specified for the package, the global permission is valid.

If you want to restrict editing of a package for a specific user or user group, you can easily do this by modifying the package permissions(see page 1400). You can modify package permissions if you have the Manage model permissions(see page 1399) permission on that project. Select:

- **Read-Only** to restrict editing of the package for the selected user.
- **Read-Write** to allow editing of the package for the selected user.

By default, **Read-Write** as the **Global permission** is assigned to the project and the **Read-Only** permission is assigned to the package.

Project-level and package-level permissions
The TWCloud project-level **Read-Write** Permissions(see page 1399) can be overridden by package permissions. For example, if a user was granted project-level **Read-Write** permissions, but was assigned **Read-Only** permissions for a particular package, that user will not be allowed to make changes to the elements owned by the package and the package itself.
If a user has project-level **Read-Only** permissions, package permissions cannot override them.

Permissions conflict logic

There are some cases when permissions for users or user group might be conflicting, these cases and the final outcome are described below:

1. If the user belongs to two groups and one group is granted Read-Only permission, while another group - Read-Write permission. In this permission conflict, the user will be granted with higher permission, which is Read-Write permission.
2. User is granted different permissions through the Users and Groups tabs, if permissions are different, User permission wins.
3. If permission is granted simultaneously for both users and groups, then this permission will be granted for standalone users and for users within the selected groups.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

4. In case of nested packages with each having different permissions (Read-Only/Read-Write), owned element modification mode is decided from the closest root package permissions in the tree.

Related pages

- Setting package permissions(see page 1400)

Setting package permissions

On this page

- Modify package permissions(see page 1400)
  - Inheritable permissions(see page 1401)
- Reviewing package permissions(see page 1402)

Modify package permissions

To modify the permissions of the package

---

1. Right-click the package or its shape and select **Permissions**. The **Permissions of Package** dialog opens.

   > ⓘ The full name of the dialog depends on the selected package name. Thus, if you select to modify the permissions of the package *domain*, the **Permissions of Package domain** dialog opens.

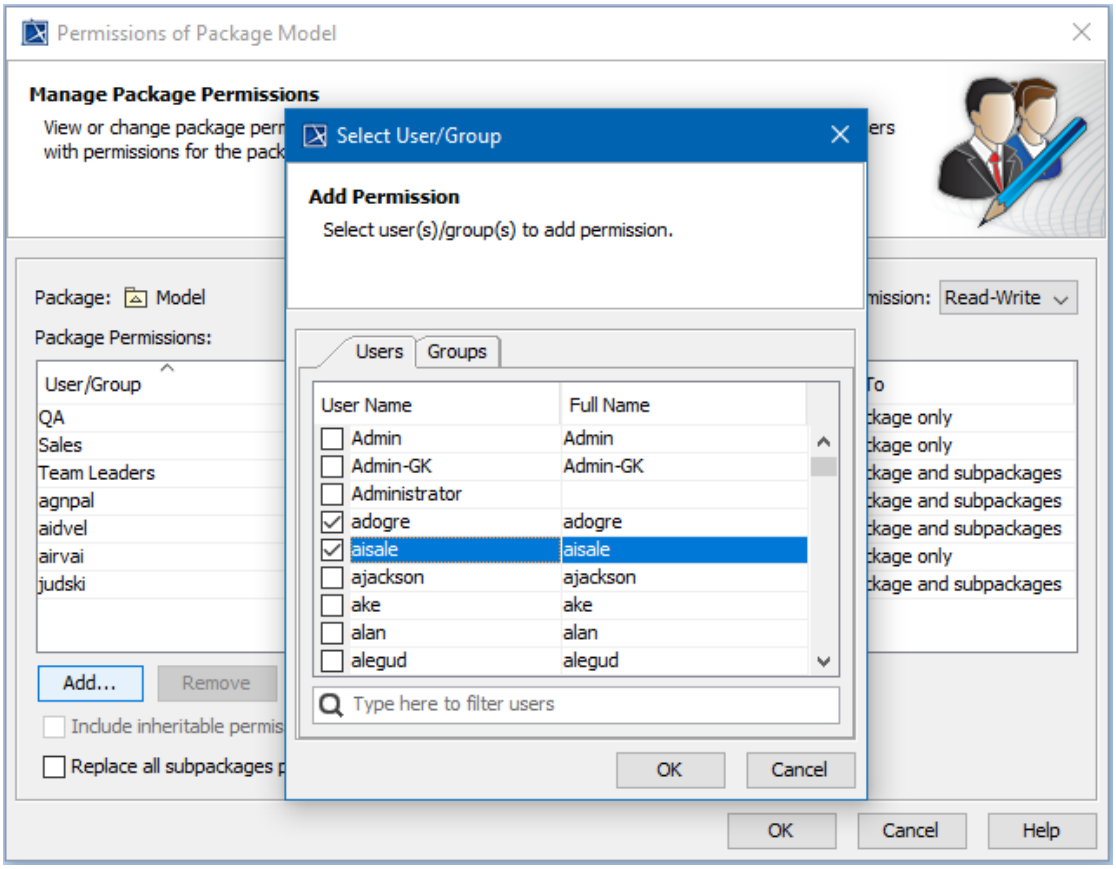
2. Change global permissions appropriately:

   - To restrict editing of the selected package leaving the rest of the model editable, select **Read-Write** as **Global permission**.
   - To restrict editing of the entire model leaving only the selected package editable, select **Read-Only** as **Global permission**.

3. Add a new package permission:

   a. Click the **Add** button under the **Package permissions** table. The **Select User/Group** dialog opens.
   b. Use search bar to filter users or groups, then click to select the check box near the user or group name.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



c. Click **OK** and see the new row appeared in the **Package permissions** table.
d. Click the **Permission** cell and select:

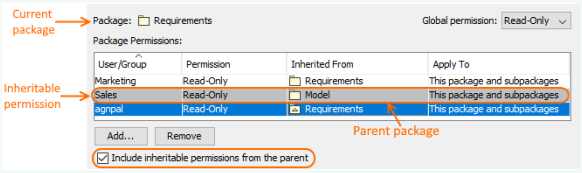
   - **Read-Only** to restrict editing of the package for the selected user/group.
   - **Read-Write** to allow editing of the package for the selected user/group.

e. If you don't want to apply the same permission on the subpackages of this package, click the **Apply To** cell and select **This package only**.

4. Add as many permissions as you need by repeating step 3.
5. Close the **Permissions of Package** dialog.

Inheritable permissions

The **Permissions of Package** dialog includes a few options for handling inheritable permissions. The following table provides their descriptions.

Option	Description
Include inheritable permissions from the parent	*Inheritable permissions* of the package are those that apply to its subpackages as well. By default, these permissions are displayed together with individual permissions of every subpackage, but cannot be modified or deleted there. If you want to hide them, click to clear the check box. Let's say you have a package *Model* and a subpackage *Requirements*. If you set on the package *Model* the Read-Only permission for certain user and leave the default setting to apply this permission on subpackages as well, the **Permissions of Package domain** dialog also displays the permission, if the relevant check box is selected. The **Inherited From** cell value of the inherited permission is *Model*. 
Replace all subpackages permissions with inheritable permissions from this package	*Inheritable permissions* of the package are those that apply to its subpackages as well. If the package, whose permissions you modify, has subpackages, you can choose to replace their individual permissions with those of the parent package. Make sure the appropriate permissions are set as inheritable and click to select the check box.

Reviewing package permissions

You can use the **Package permissions** dialog to review the permissions set on all packages in a project. This dialog allows you to change the global permissions affecting all users for a project. You can also change the user permission for a selected package, as well as remove the user permission for the package.

To review package permissions

- From the **Options** main menu, select **Permissions**. The **Package permissions** dialog opens.

Locking model for edit

On this page

- Locking without forced project update(see page 1404)
- Representation of locked elements(see page 1404)
- Locking elements and diagrams(see page 1404)
- Locking symbol styles(see page 1407)
- Locking project usages(see page 1409)

You must lock the part of the model you are going to edit to prevent other users from editing it at the same time. Locking is allowed if that part of the model is not locked by another user, because the same item can only be locked by one user at a time.

The following table explains what you can edit after you lock a single element, single diagram, diagram with elements, and so on.

When you lock...	You can...
Element	Edit element specification properties (in the Specification window)
Diagram	Edit: • Diagram specification properties (in the Specification window) • Diagram properties (in the **Diagram Properties** dialog) • Symbol layout on the diagram pane • Symbol properties (in the **Symbol Properties** dialog)
Diagram with elements	Edit: • Diagram specification properties (in the Specification window) • Element specification properties (in the Specification window and on the diagram pane) • Diagram properties (in the **Symbol Properties** dialog) • Symbol properties (in the **Symbol Properties** dialog) • Symbol layout on the diagram pane
Symbol styles	Edit properties of symbol styles (in the **Project Options** dialog)
Project usages	• Import used projects as packages into the main project • Stop using projects • Change versions of used projects

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Locking without forced project update

You can lock elements without forcing an update of a project to the latest version if the selected elements are not locked or do not have changes in a newer version on the server.

This feature can be disabled/enabled in the **Environment Options** dialog.

To disable/enable locking without updating

---

1. Select **Options** > **Environment** > **Collaboration** node.
2. Find **Enable Locking Without Forced Project Update** option. Do one of the following:
   - Set value to *true,* to enable locking without updating;
   - Set value to *false,* to disable locking without updating.

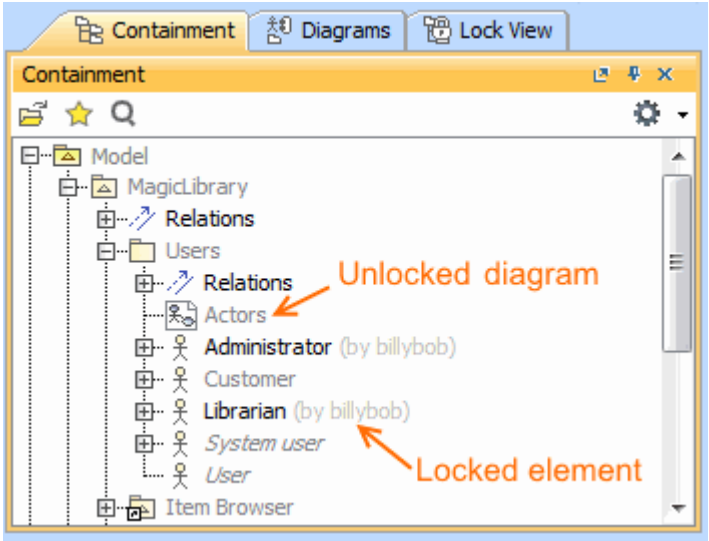
There is a 60-second threshold; if locking takes longer than 60 seconds, a project update will be required.

An example
You are trying to lock package B, which contains many elements, recursively. The modeling tool cannot check all contained elements' status in the later versions of the project in 60 seconds. After 60 seconds, a project update will be required to lock the package.

Representation of locked elements

In the Containment tree, element names are represented in two colors:

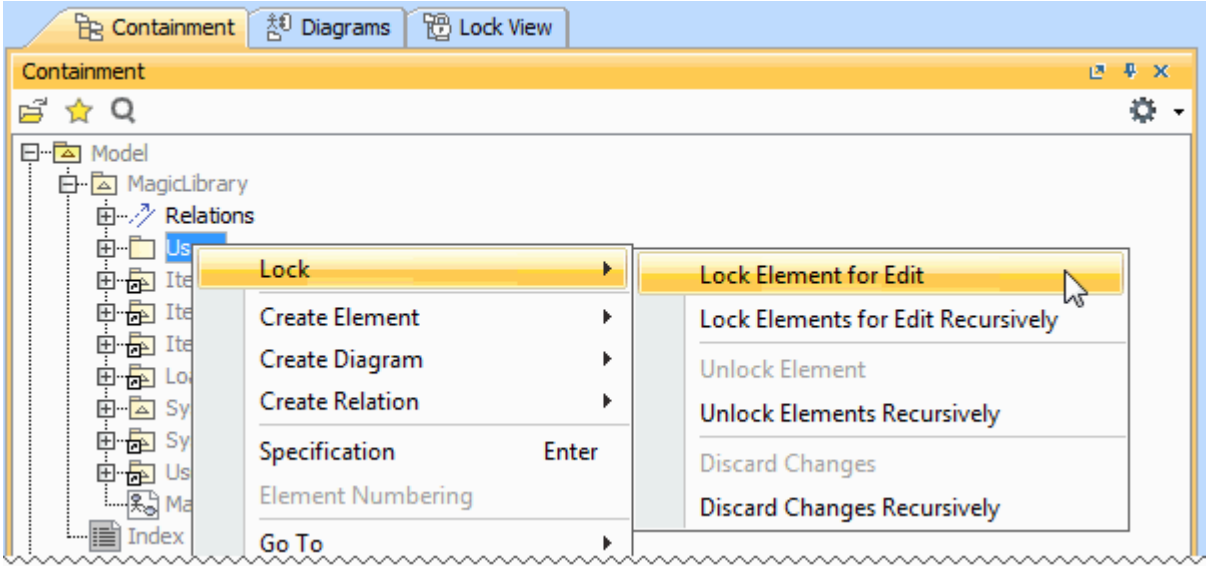
- Unlocked elements are represented in black, meaning they are editable.
- Locked elements are represented in gray, meaning they are not editable.



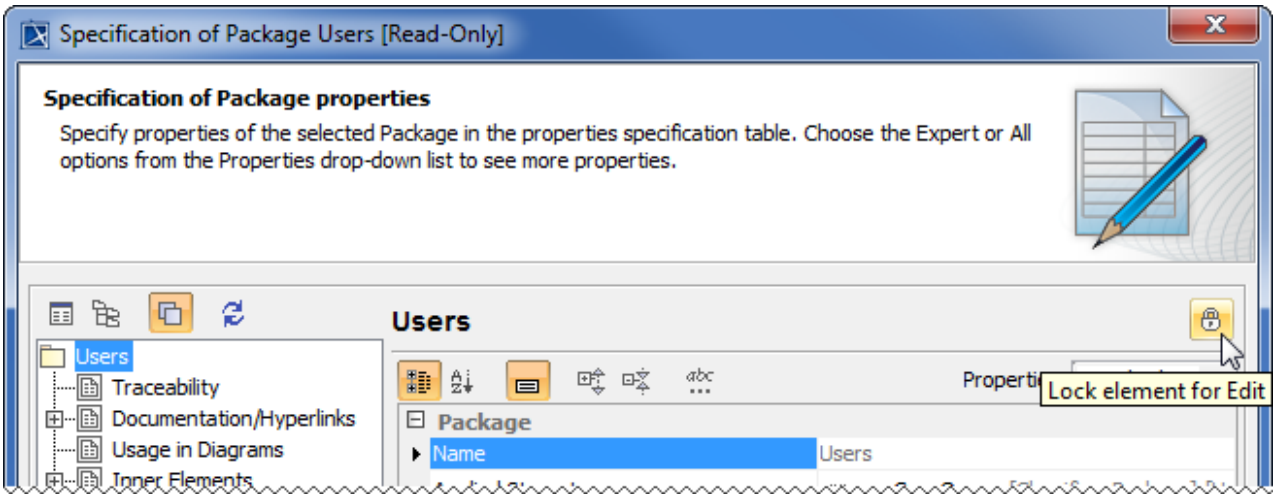
Locking elements and diagrams

Elements and diagrams can be locked by using the commands from their shortcut menu.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Elements can also be locked by clicking the Lock element for the Edit button on their Specification window.



The following table provides suggestions on which locking commands to use for different purposes.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

If you need to edit...	Do the following
Specification properties of a single element	1. Right-click either: • This element in the Model Browser. • A symbol of this element on a diagram pane. 2. From the shortcut menu, select **Lock** > **Lock Element for Edit**. OR 1. Double-click either: • This element in the Model Browser. • A symbol of this element on a diagram pane. 2. In the Specification window, click the Lock element for Edit button .
Specification properties of both an element and the elements it contains (owns) For example, a project has a Package *Analysis*, containing two inner Packages - *Design* and *Implementation*. Each inner Package also contains elements. Lock the Package *Analysis* recursively to edit all the elements in this Package.	1. Right-click either: • This element in the Model Browser. • A symbol of this element on a diagram pane. 2. From the shortcut menu, select **Lock** > **Lock Elements for Edit Recursively**
At least one of the following: • Diagram representation properties (including diagram layout) • Symbol properties of elements represented on that diagram pane	1. Right-click this diagram in the Model Browser. 2. From the shortcut menu, select **Lock** > **Lock Diagram for Edit**. OR 1. Right-click a free space on this diagram pane. 2. From the shortcut menu, select **Lock Diagram** > **Lock Diagram for Edit**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

If you need to edit...	Do the following
At least one of the following: • Diagram specification properties • Diagram representation properties (including diagram layout) • Properties of elements represented on that diagram pane • Symbol properties of elements represented on that diagram pane	1. Right-click this diagram in the Model Browser. 2. From the shortcut menu, select **Lock** > **Lock Diagram Content for Edit**. OR 1. Right-click a free space on this diagram pane. 2. From the shortcut menu, select **Lock Diagram** > **Lock Diagram Content for Edit**.
At least one of the following: • Symbol properties of a single element represented on a diagram pane • Position of the symbol on the diagram pane	1. Right-click the symbol of that element on a diagram pane. 2. From the shortcut menu, select **Lock** > **Lock Element for Edit in Diagram**.

Locking symbol styles

You can lock symbol styles via the **Lock View** tab or the **Project Options** dialog.

To lock symbol styles via the **Lock View** tab

1. In the **Lock View** tab on the Model Browser, expand *Project Options*.

   Where is the Lock View tab?

   It is the last tab on the Model Browser. If the tab is closed, you can simply open it by selecting **Collaborate** > **View Locked Elements**.

2. Right-click *Symbol Styles* and from the shortcut menu select **Lock Symbol Styles for Edit**.

To lock symbol styles via the **Project Options** dialog

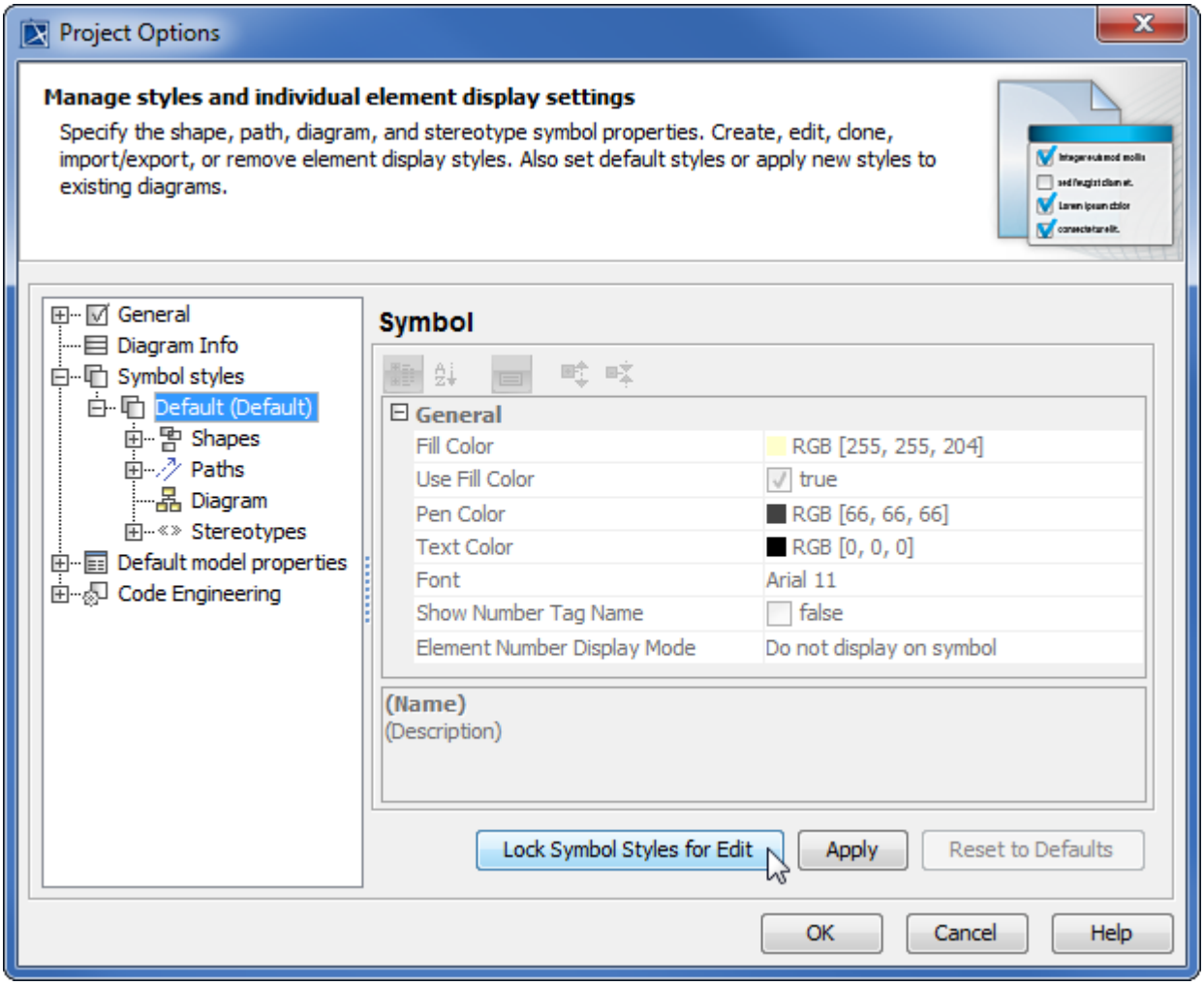
1. Do either:
   - Select **Options** > **Project**.
   - In the **Lock View** tab on the Model Browser, right-click *Project Options* and from the shortcut menu, select **Symbol Styles**.

     Where is the Lock View tab?

     It is the last tab on the Model Browser. If the tab is closed, you can simply open it by selecting **Collaborate** > **View Locked Elements**.

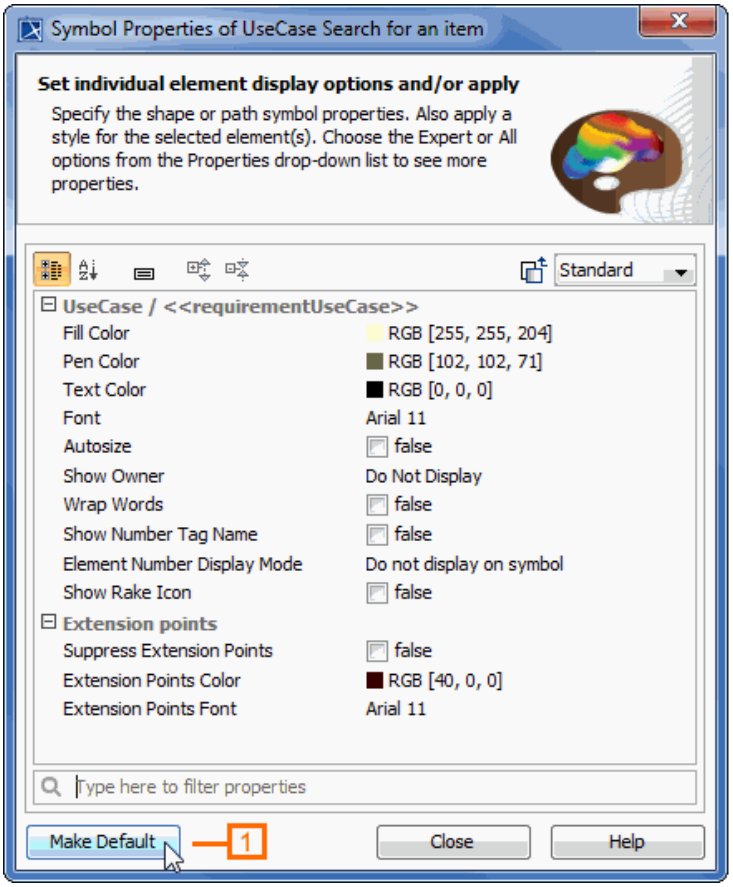
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

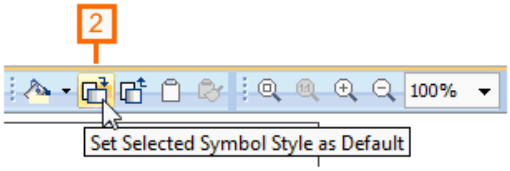
2. In the **Project Options** dialog, click the **Lock Symbol Styles for Edit** button.



As a result, symbol styles become editable. Additionally, the **Make Default** button (1) in all **Symbol Properties** dialogs and the **Set Selected Symbol Style as Default** button (2) on all the diagram toolbars become available.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.





Locking project usages

To lock a used project

1. From the **Options** menu, select **Project Usages**. The **Project Usages** dialog opens.
2. Select a used project on the left side of the dialog.
3. Click the button to lock the used project.

> **Shortcut menu**
> This step can be done via shortcut menu. Right-click project and then select **Lock**.

You can then import this used project as a package to the main project, stop using it, or change its version.

Related pages

- Unlocking model(see page 1416)
- Committing changes to Teamwork Cloud(see page 1420)

Using Lock-Free Editing mode

A Lock-Free Editing mode enables users to work with model elements without the need to lock them. Once the option is enabled in **Project Options**, you can edit elements freely; however, keep it in mind

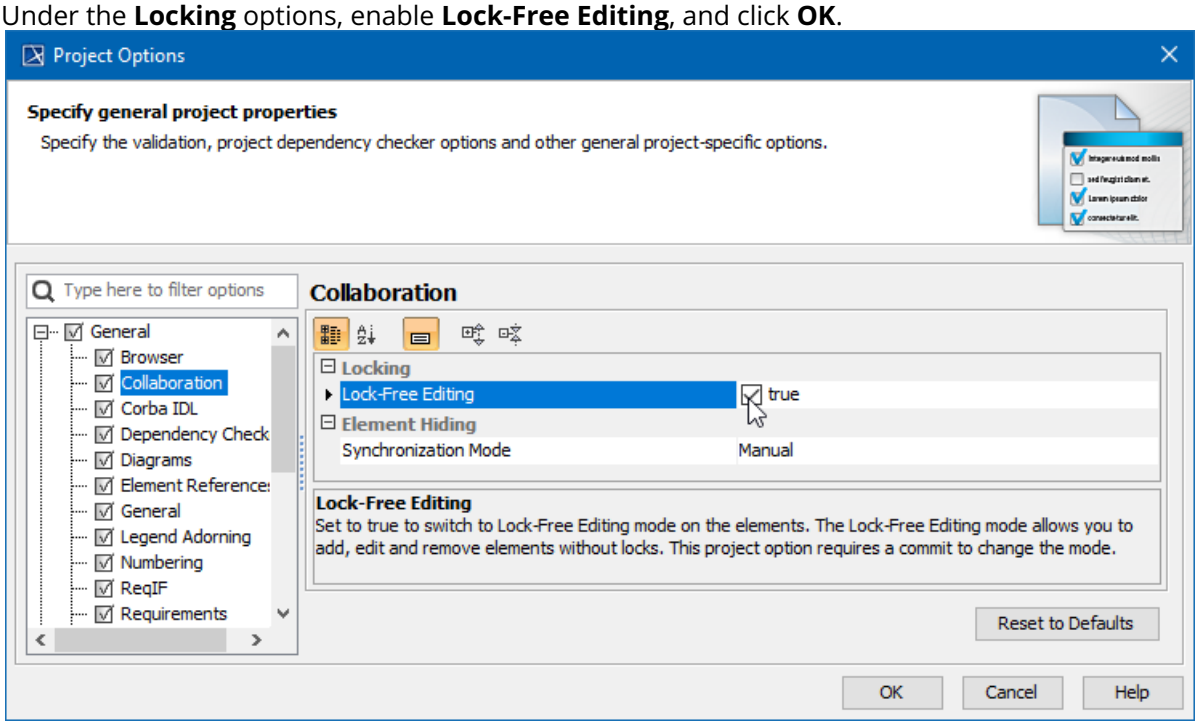
that this is applicable to model elements only. Locks for diagrams, project options, and used projects will persist. In other words, you will still have to lock them before editing.

Recommendation
Assigning different scope of the model to different users is the recommended way to work in the Lock-Free Editing mode to prevent conflicts that may occur when users' changes interfere with each other.

To enable the Lock-Free Editing mode in the project, you need to have the **Release Resource Locks**[311] permission assigned to you.

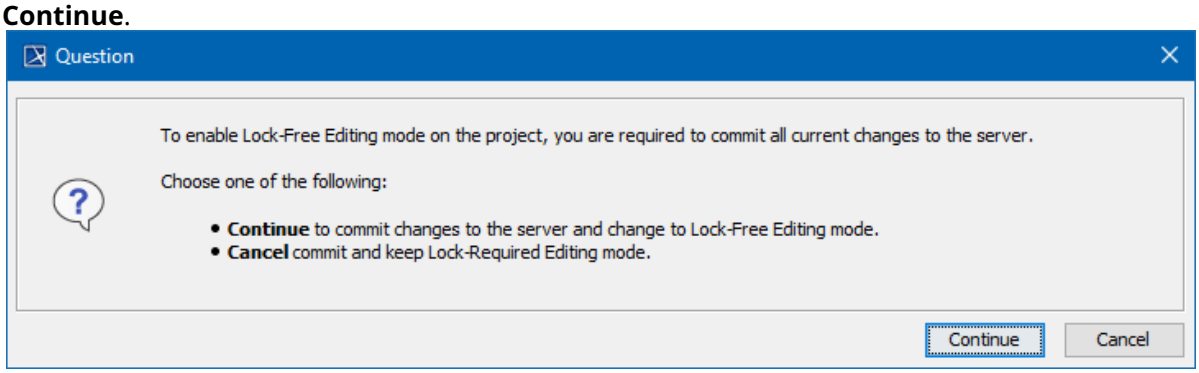
To start using the Lock-Free Editing mode

1. In the main menu, click **Options** > **Project**.
2. In the panel on the left, select **General** > **Collaboration**.
3. Under the **Locking** options, enable **Lock-Free Editing**, and click **OK**.



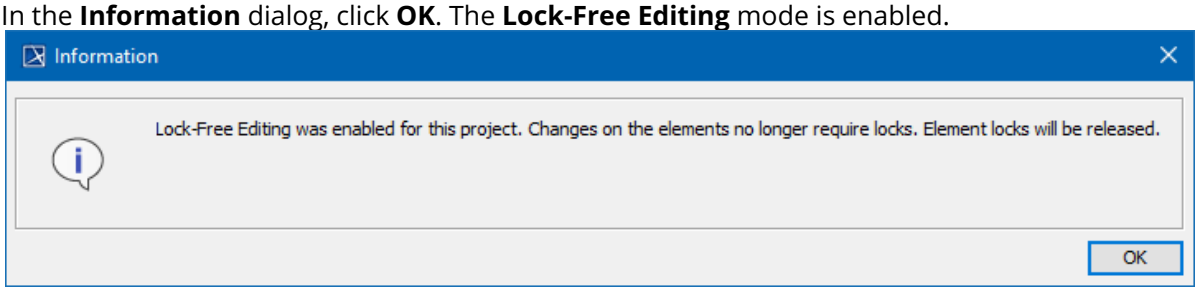
311 https://docs.nomagic.com/display/TWCloud190SP4/Permissions

4. In the **Question** dialog asking whether you want to commit the changes to the server, select **Continue**.



As soon as the Lock-Free Editing mode is enabled, all element locks are released.

5. In the **Information** dialog, click **OK**. The **Lock-Free Editing** mode is enabled.



If the Lock-Required mode is enabled in the project, users who worked without locks will not be able to commit their changes to the server. Therefore, we recommend making sure that all the users working on the model do not have uncommitted changes before switching modes.

« Locking model for edit(see page 1403)

Reviewing conflicting changes »(see page 1411)

Reviewing conflicting changes

On this page

- Conflicting Changes panel(see page 1412)
- Composition of the Conflicting Changes panel(see page 1413)
- Symbols used in the Conflicting Changes panel(see page 1413)
- Resolving conflicting changes(see page 1414)

Even though users usually work on different parts of the model, situations when they start editing the same model elements and, as a result, conflicting changes are detected may occur. In such a case, you will be asked to review these conflicts in the **Conflicting Changes** panel and then take certain action, for instance, confirm the update result or discard the local changes.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

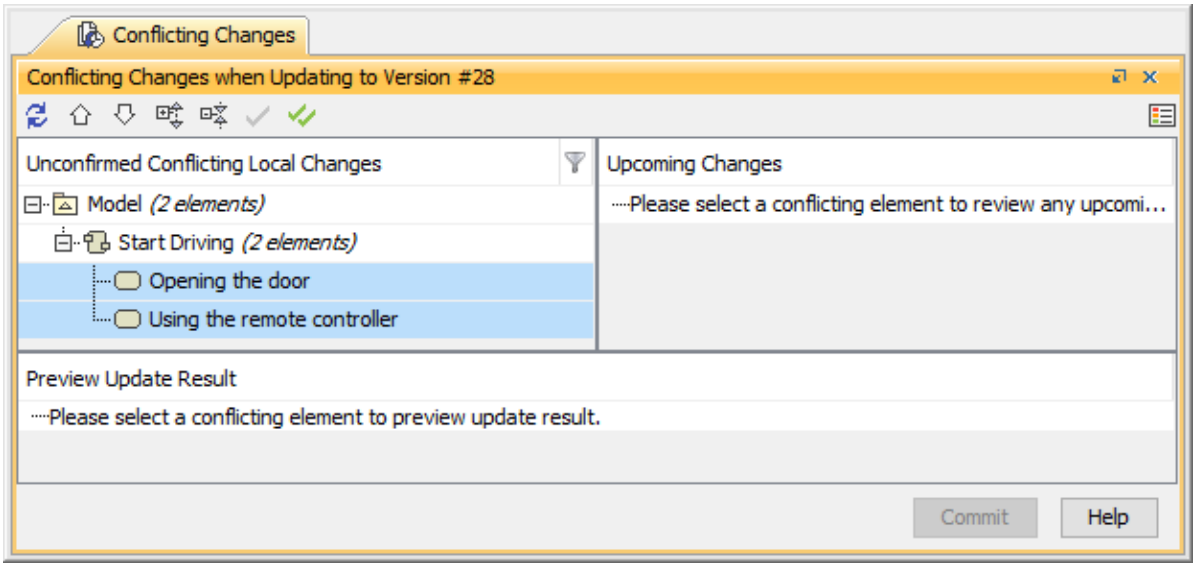
Example #1
User 1 and User 2 are working on the same model version: #24. The model contains Class A.
User 1 deletes Class A from the model and commits the project to the server. The model version #25 is created.
User 2 renames Class A to Class B and commits/updates the project to the server. The conflicting changes occur.

Example #2
User 1 and User 2 are working on the same model version: #24. The model contains Class A.
User 1 renames Class A to Class B and commits the project to the server. The model version #25 is created.
User 2 renames Class A to Class C and commits the project to the server. The conflicting changes occur.

Conflicting Changes panel

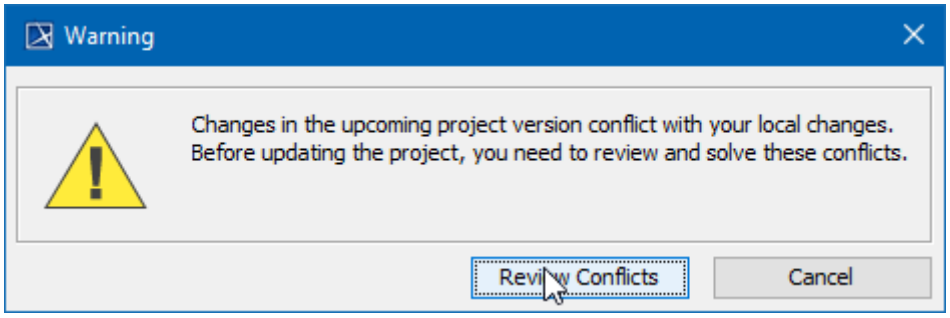
To open the panel

1. In the main menu, click **Window**.
2. Select **Conflicting Changes** from the drop-down menu. The **Conflicting Changes** panel opens.



The Conflicting Changes panel.

Alternatively, you can open it by clicking the **Review Conflicts** button directly on the warning that appears when conflicting changes are detected.



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

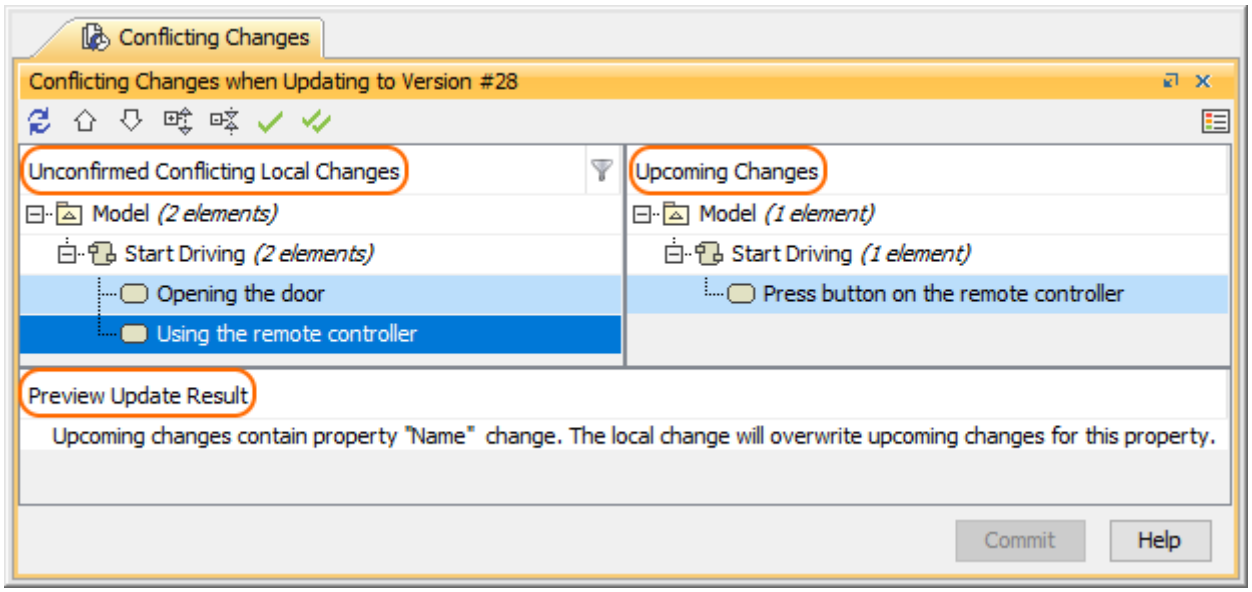
The Conflicting Changes warning.

> ⚠ Please note that the possible occurrence of conflicting changes is being checked upon the project update or a commit that requires an update.

Composition of the Conflicting Changes panel

The **Conflicting Changes** panel consists of three smaller panels that provide different pieces of information to the user:

- **Conflicting Local Changes**: shows locally modified elements whose changes conflict with the changes already committed to the server.
- **Upcoming Changes**: shows the upcoming changes that conflict with the selected local change.
- **Preview Update Result**: informs about the changes that will be applied after the project update.

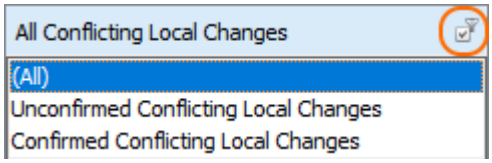
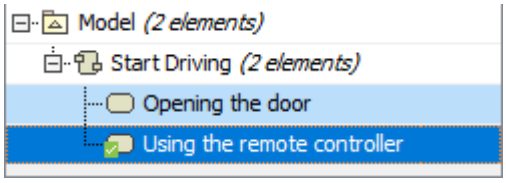


Conflicting Changes panels that inform users about the changes that conflict in the project.

Symbols used in the Conflicting Changes panel

Icon	Description
	Checks the presence of the conflicting changes in the model.
	Selects the previous conflicting change on the list.
	Selects the next conflicting change on the list.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Icon	Description
	Expands elements in the **Conflicting Local Changes** panel.
	Collapses elements in the **Conflicting Local changes** panel.
	Confirms the update result affecting the selected element.
	Confirms all update results of the conflicting changes at once.
	Filters confirmed/unconfirmed conflicting changes. 
	Shows that the conflicting change has been confirmed by the user. 

Confirming update results

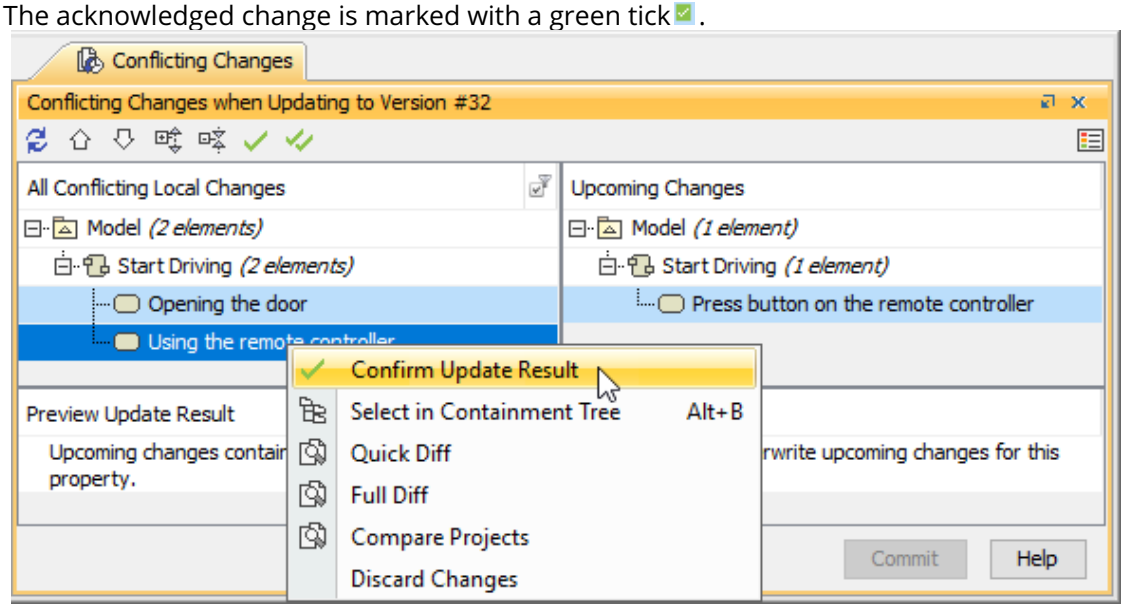
Once you have reviewed the conflicting changes, you can confirm the update result that will be applied after the project update. It should be emphasized that remote deletions always take precedence over the local changes.

To confirm the update result

1. In the **Conflicting Changes** panel, do one of the following:
   - select the specific conflicting element and click in the toolbar at the top.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- click in the toolbar to confirm all conflicting changes at once.
- right-click the conflicting element and select **Confirm Update Result** in the shortcut menu.
  The acknowledged change is marked with a green tick.



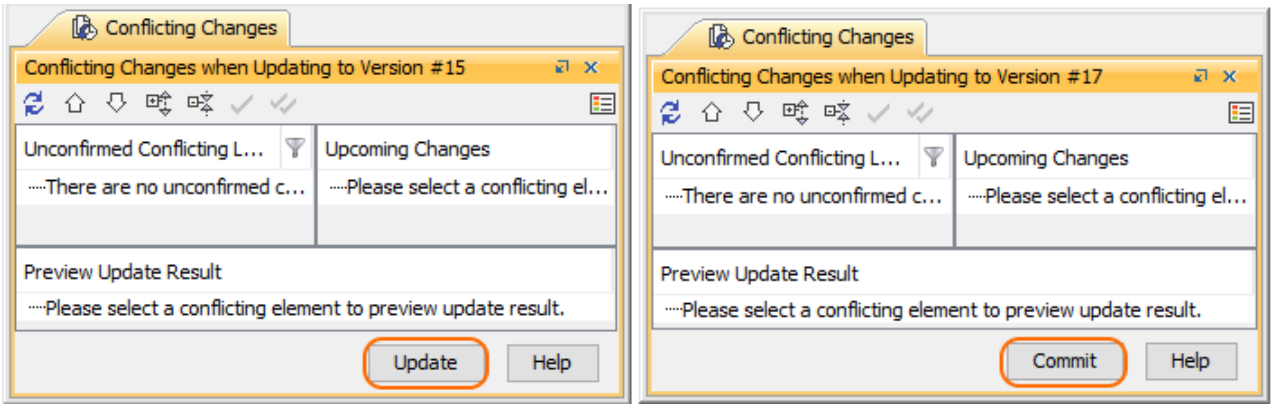
Confirming the conflicting changes via the shortcut menu.

There is another group of actions you can perform to handle the detected conflicting changes:

- **Select in Containment Tree**: click to select the element in Containment tree.
- **Quick Diff**: select to inspect the conflicting changes in a compact form.
- **Full Diff**: select to inspect the conflicting changes in a detailed form.
- **Compare Projects**: select to compare the modified projects.
- **Discard Changes**: select to discard the local changes.

Please note that the discarded changes cannot be undone.

Depending on the action you have performed prior to the detection of conflicting changes in your model, you will see either the **Commit** or the **Update** button in the **Conflicting Changes** panel after confirming the conflicting changes in your project.



Update or Commit button in the Conflicting Changes panel.

« Using Lock-Free Editing mode(see page 1409)

Unlocking model »(see page 1416)

Related pages

- Reviewing element history and inspecting changes(see page 1431)
- Change Sets(see page 1439)

Unlocking model

Unlocking the model is useful in one of the following cases:

- After you choose to keep locks, when committing changes(see page 1420).
- If you locked the model(see page 1403), but haven't made any changes in the model.
- When you want to clean other user locks.

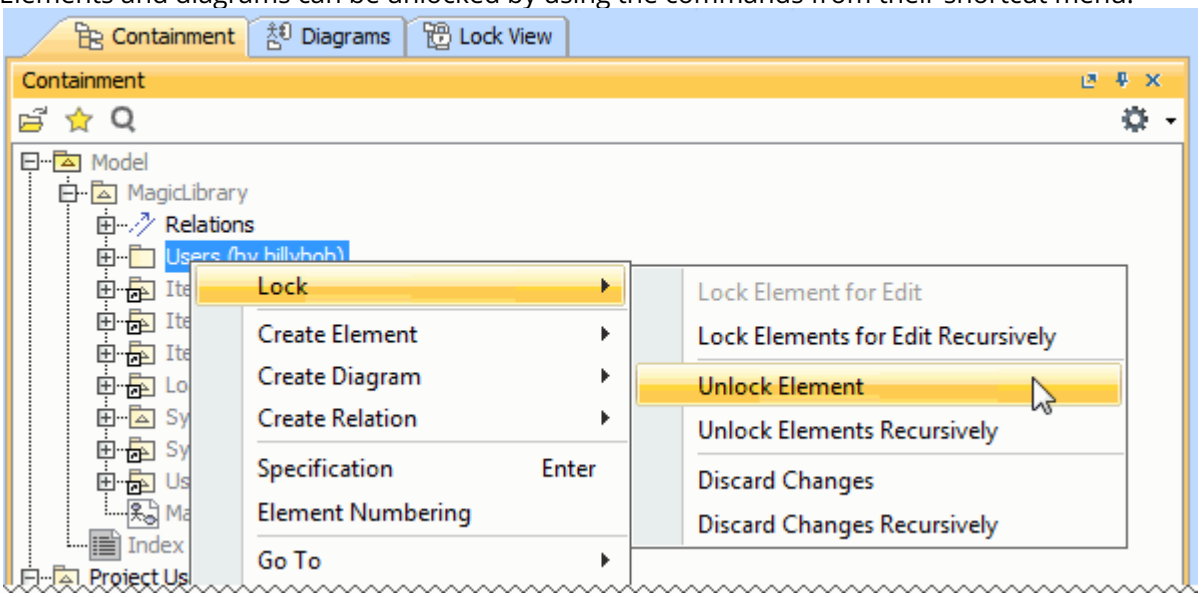
> ⚠ To unlock other users' locks, you must have the **Project Locks Administrator** permission.

On this page

- Unlocking elements and diagrams(see page 1416)
- Unlocking elements locked by other users(see page 1418)
- Unlocking symbol styles(see page 1419)
- Unlocking project usages(see page 1420)
- Unlocking project structure(see page 1420)

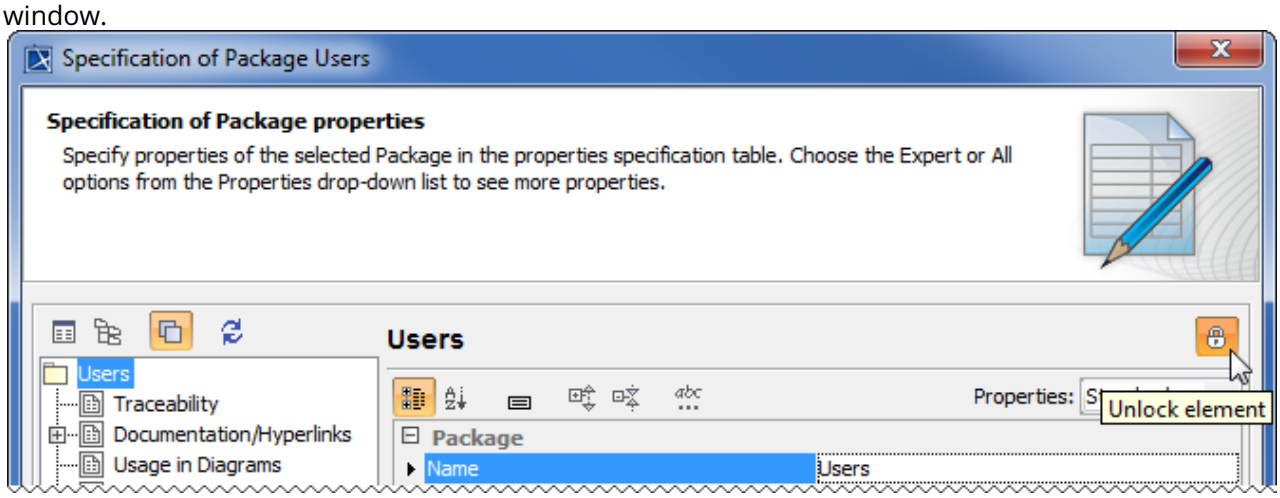
Unlocking elements and diagrams

Elements and diagrams can be unlocked by using the commands from their shortcut menu.



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Elements can also be locked by clicking the Lock element for Edit button on their Specification window.



The following table provides suggestions on what unlocking commands to use for different purposes.

If you finished editing...	Do the following
Specification properties of a single element	1. Right-click either: • This element in the Containment tree of Lock View tree on the Model Browser. • A symbol of this element on a diagram pane. 2. From the shortcut menu, select **Lock** > **Unlock Element**. OR 1. Double-click either: • This element in the Containment tree of Lock View tree on the Model Browser. • A symbol of this element on a diagram pane. 2. In the Specification window, click the Unlock element button .
Specification properties of both an element and all the elements it contains (owns)	1. Right-click either: • This element in the Containment tree of Lock View tree on the Model Browser. • A symbol of this element on a diagram pane. 2. From the shortcut menu, select **Lock** > **Unlock Elements Recursively**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

If you finished editing...	Do the following
At least one of the following: • Diagram representation properties (including diagram layout) • Symbol properties of elements represented on that diagram pane	1. Right-click this diagram in the Containment tree of Lock View tree on the Model Browser. 2. From the shortcut menu, select **Lock** > **Unlock Diagram**. OR 1. Right-click a free space on this diagram pane. 2. From the shortcut menu, select **Lock Diagram** > **Unlock Diagram**. Use this command after you have finished editing the diagram representation, but you still need to change the specification properties of the elements represented on that diagram pane.
At least one of the following: • Diagram specification properties • Diagram representation properties (including diagram layout) • Properties of elements represented on that diagram pane • Symbol properties of elements represented on that diagram pane	1. Right-click this diagram in the Containment tree of Lock View tree on the Model Browser. 2. From the shortcut menu, select **Lock** > **Unlock Diagram Content**. OR 1. Right-click a free space on this diagram pane. 2. From the shortcut menu, select **Lock Diagram** > **Unlock Diagram Content**.
At least one of the following: • Symbol properties of a single element represented on a diagram pane • Position of the symbol on the diagram pane	1. Right-click the symbol of that element on a diagram pane. 2. From the shortcut menu, select **Lock** > **Unlock Element in Diagram**.

Unlocking elements locked by other users

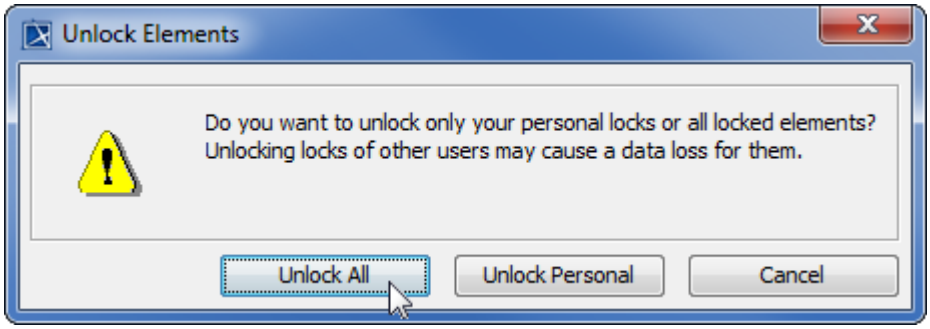
Forced unlocking is helpful, when elements are left locked by the user who is currently unavailable.

For this you must have the **Project Locks Administrator** permission.

To unlock the elements locked by another user

1. From the **Collaborate** menu, select **Unlock All**.
2. If the message opens asking you to choose what to unlock, click **Unlock All**. The **Commit Project to the Server** dialog opens.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



3. In the dialog, do the following:
   a. If there is a need, type a comment.
   b. Click **Commit**.

As a result, all the locks made by another user are released, and the changes he/she made are lost forever.

Unlocking symbol styles

Symbol styles can be unlocked via the **Lock View** tab or the **Project Options** dialog.

To unlock symbol styles via the **Lock View** tab

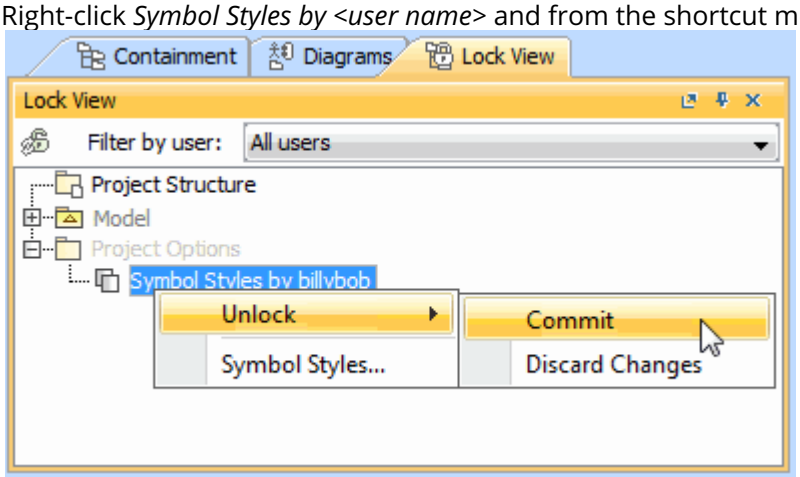
---

1. In the **Lock View** tab on the Model Browser, expand *Project Options*.

   Where is the Lock View tab?

   It is the last tab on the Model Browser. If the tab is closed, you can simply open. For this, select **Collaborate** > **View Locked Elements**.

2. Right-click *Symbol Styles by <user name>* and from the shortcut menu select **Unlock** > **Commit**.



3. In the **Commit Project to the Server** dialog do the following:
   a. If there is a need, type a comment.
   b. Click to clear the **Keep Locks** check box.
   c. Click **Commit**.

To unlock symbol styles via the **Project Options** dialog

---

1. Do either:
    - Select **Options** > **Project**.
    - In the **Lock View** tab on the Model Browser, right-click *Project Options* and from the shortcut menu, select **Symbol Styles**.

      Where is the Lock View tab?

      It is the last tab on the Model Browser. If the tab is closed, you can simply open. For this, select **Collaborate** > **View Locked Elements**.
2. In the **Project Options** dialog, click the **Unlock** button and then select **Commit**.
3. In the **Commit Project to the Server** dialog do the following:
    a. If there is a need, type a comment.
    b. Click to clear the **Keep Locks** check box.
    c. Click **Commit**.

Unlocking project usages

To unlock a used project

---

1. From the **Options** menu, select **Project Usages**. The **Project Usages** dialog opens.
2. Select a locked used project on the left side of the dialog.
3. Click the **Edit** button and then select **Unlock** > **Commit**. The **Commit Project to the Server** dialog opens.
4. In the dialog:
    a. Type a comment, if there is a need.
    b. Click to clear the **Keep Locks** check box.
    c. Click **Commit**.

Unlocking project structure

Project structure becomes locked during the merge(see page 1083) operation. The lock is automatically released after the merged project is committed to the server.

However, the user who started the merge operation can release the lock manually as well.

To unlock the project structure manually

---

1. From the **File** menu, select **Project Properties**. The **Project Properties** dialog opens.
2. Click the **Unlock (locked by <user name>)** button, for example, **Unlock (locked by billybob)**. The **Commit Project to the Server** dialog opens.
3. In the dialog:
    a. Type a comment, if there is a need.
    b. Click to clear the **Keep Locks** check box.
    c. Click **Commit**.

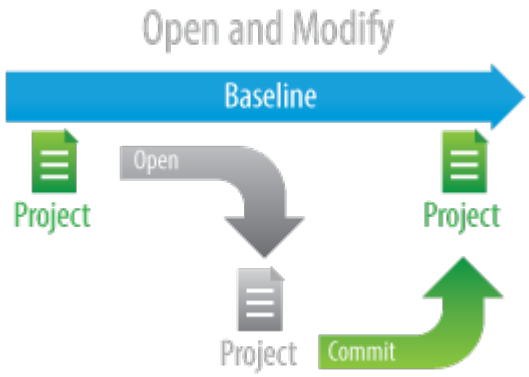
Other users cannot unlock the project structure manually.

Committing changes to Teamwork Cloud

After you finish editing the model, you must commit the changes to the server to enable other team members to see these changes when editing other parts of the model. When you commit the changes, new data from your client application (for example, MagicDraw or a Cameo Suite product, such as

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Cameo Systems Modeler) is merged with the project on the server repository. As a result, a new version of that project is created. Other team members must update their projects(see page 1423) from the server to see the committed changes.



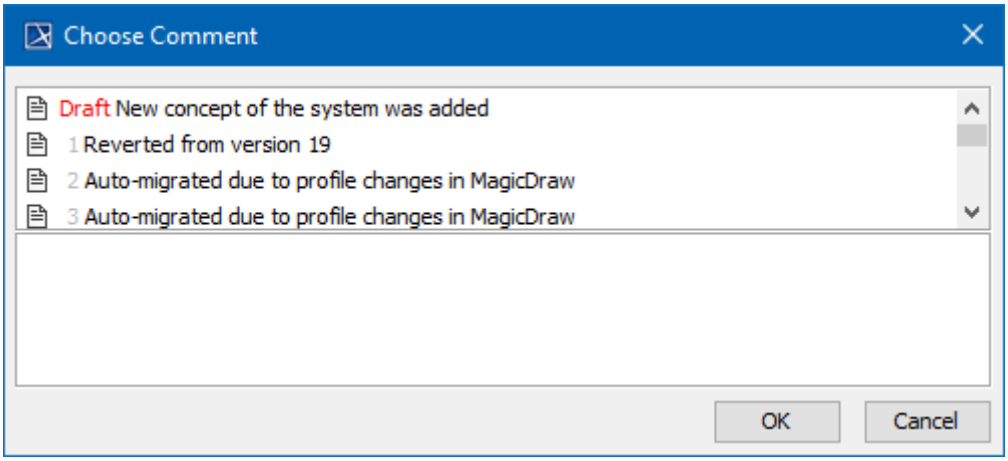
When committing changes, you can choose to release or keep the locks on the model(see page 1403). If you are going to continue editing after the commit is over, keep the locks.

To commit changes to Teamwork Cloud

---

1. From the **Collaborate** menu, select **Commit Changes To Server**, or press Ctrl+K. The **Commit Project to the Server** dialog opens.
2. If you need to describe the changes you are going to commit, type a comment in the open dialog.

   You can select a comment from the recently used comments list. Click the Choose Comment button above the **Comment** box, and select any comment from the list in the **Choose Comment** dialog.
   Last comment written in the **Comment** box is saved as draft after user clicks **Cancel** instead of **Commit** in a **Commit Project To the Server** dialog. Next time when user opens **Commit Project To the Server** dialog, he will be notified about saved comment draft, with Choose Comment button . In addition, in the **Choose Comment** dialog user will see saved comment marked as a draft.

3. If you want to keep locks on the model and continue editing after the commit is over, select the **Keep locks** check box.

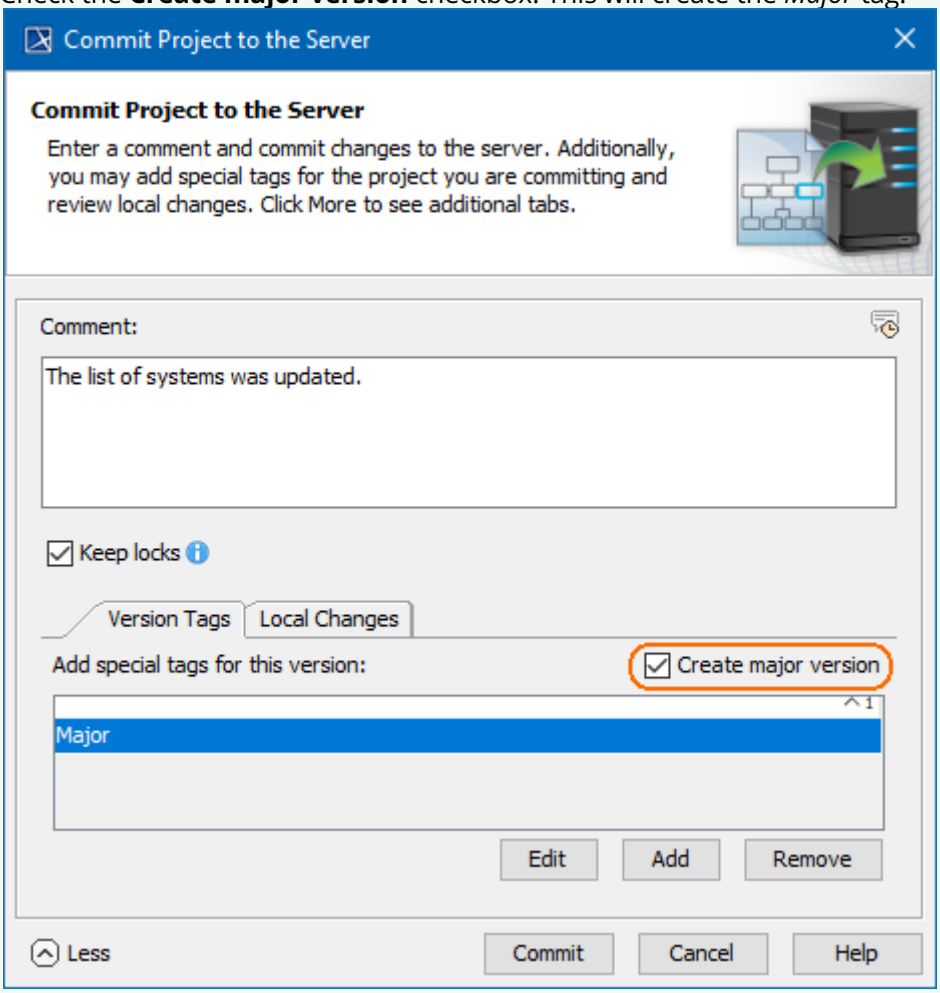
The **Keep locks** check box remembers your last choice. Thus, if you want to release the locks every time when committing changes, once you make your choice during the first commit you don't need to unselect this check box each time.

4. If you want to tag the new project version, such as an *Loan final* tag, do the following:

   a. Click **More**. The **Version Tags** tab appears on the **Commit Project to the Server** dialog.
   b. Click the **Add** button.
   c. In the open dialog, type a text and click **OK**.

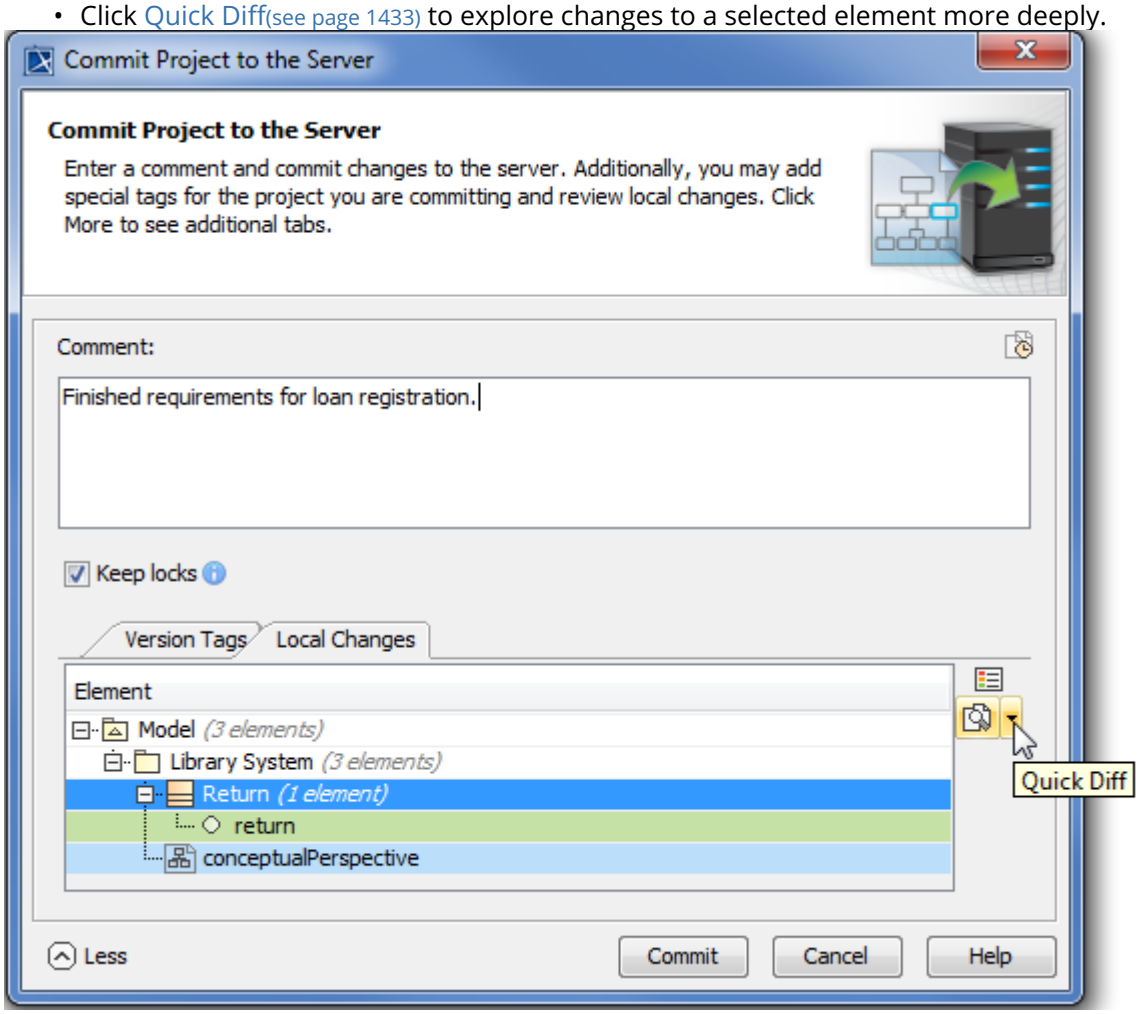
If you want to tag the new project version with a **Major** tag, do the following

a. Click **More**. The **Version Tags** tab appears on the **Commit Project to the Server** dialog.
b. Check the **Create major version** checkbox. This will create the *Major* tag.



Notifications on new version
Now you can specify tags that will trigger a notification if a new used project version with a specified tag is committed to the Teamwork Cloud. Learn more about Setting for notification about version changes(see page 0).

5. You can also see local changes of the new project version. Click **More** and the **Local Changes** tab appears on the **Commit Project to the Server** dialog. All project elements with changes are listed in this tab. You may do one of the following:
   - Click the legend to see changes to elements.
   - Click Quick Diff(see page 1433) to explore changes to a selected element more deeply.



6. In the **Commit Project to the Server** dialog, click **Commit**.
7. Wait while a new project version containing the committed changes is sent to the Teamwork Cloud repository.

> ⚠ In case of commit failure commit message and commit tags are saved for next commit attempt, so user does not have to re-type commit information.

Updating changes from Teamwork Cloud

On this page

- Notifications about new versions(see page 1424)
- Update the project from the server(see page 1425)
- Display upcoming changes(see page 1425)

- Granularity level of project options(see page 1427)

Notifications about new versions

When several users work on the same project simultaneously, you will be notified if new changes are committed to the server project you are working on.

To turn notifications about new versions on/off

1. On the main menu, click **Options** > **Environment**. The **Environment Options** dialog opens.
2. In the options list, select **Collaboration** and then the property **Notify About New Project Versions on the Server.** Do one of the following:
    - Set to *true* to get notifications.
    - Set to *false* to turn the notification window off.

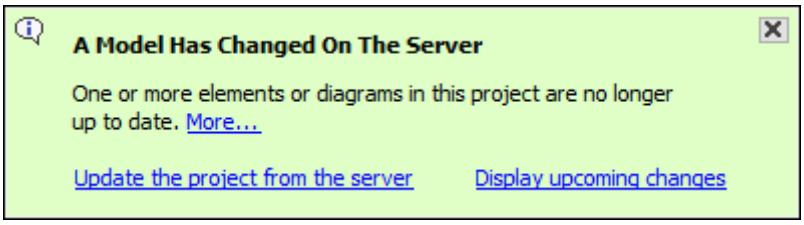
You can specify the interval you want the tool to check for new committed project versions on the server; by default, this time interval is two minutes.

To specify a time interval for checking new versions

1. On the main menu, click **Options** > **Environment**. The **Environment Options** dialog opens.
2. In the options list, select **Collaboration.**
3. Specify **Interval to Check For New Project Version (in seconds)** property value.

> Any changes to the interval will be applied to newly opened projects only.

After notifications are turned on and/or a time interval is specified, the tool will automatically check to see if there is a new project version available on the server. When a new version is detected, the following notification will appear:



You can choose these actions from this notification: Update the project from the server(see page 1425) or display upcoming changes(see page 1425) in the **Upcoming Changes** panel.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Update the project from the server

An update is the process of refreshing your current project version by merging changes other team members have made and committed to the server(see page 1420). You should update a project when you know that a new project version is available on the server.

To update the project with changes from Teamwork Cloud

- From the **Collaborate** menu, select **Update Project**, or press Ctrl+U.

  ⚠ Locked items cannot be updated.

Updating lock information is a special case of the update feature. Use it to refresh the list of elements locked by other users.

To update the lock information on elements locked by other users

- From the **Collaborate** menu, select **Update Lock Information**, or press Ctrl+Shift+U.

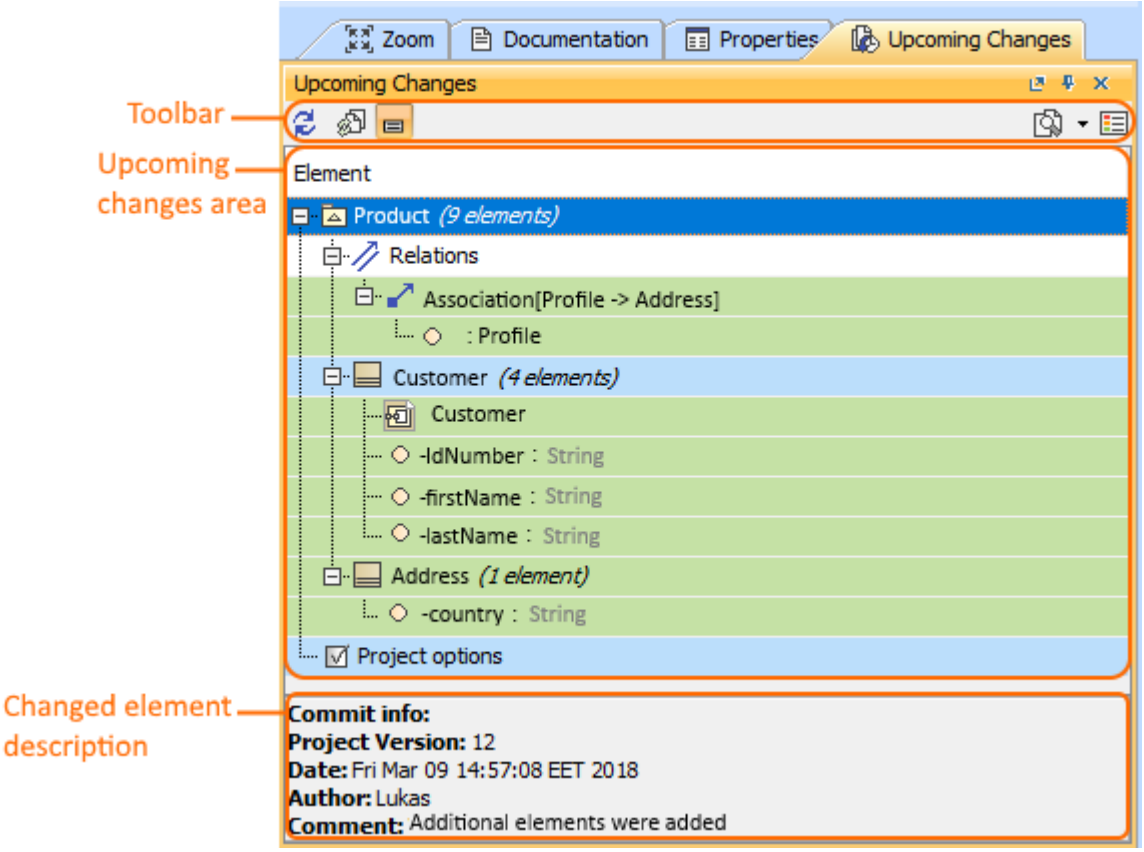
Display upcoming changes

If you want to see the latest changes, open the **Upcoming Changes** panel**,** which shows the changes committed to the server project while you were working on it.

To open the **Upcoming Changes** panel

- On the main menu, click **Window** > **Upcoming Changes**. The **Upcoming Changes** panel opens.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



This toolbar allows you to manage project changes. Its commands are described in the following table.

Button icon	Button name	Description
	Refresh	Refresh the view to check for changes in the latest project version.
	Update project	Update the project to the latest version (or press Ctrl+U).
	Show description	Select a changed element and click **Show description**. The latest changes committed for that element are displayed, including project version, date, author, and commit comment.
	Difference checker	Reviewing(see page 1431) element history and inspecting changes.

Button icon	Button name	Description
	Legend	- Green indicates that the element was added. - Red indicates that the element was deleted. - Blue indicates that the element was modified or moved.

Granularity level of project options

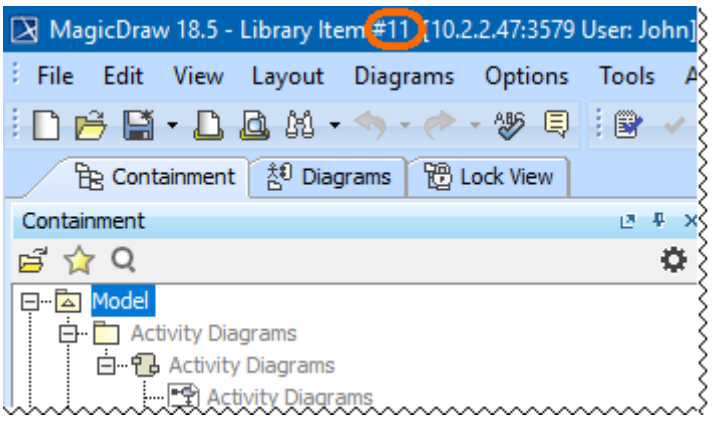
Every project option and its value under **General** and **Default Model Properties** nodes are treated as separate changes. However, options under the **Diagram Info** node are treated as single changes. After every update, project options get merged (or overwritten). When more than one user works on the same project version and changes different options, these options will be merged after updating. However, if all users changed the same options, the last user to commit will overwrite the option value and only that option will be seen in the last update. The user will be notified before overwriting the option value.

Related pages

- Reviewing element history and inspecting changes(see page 1431)
- Committing changes to Teamwork Cloud(see page 1420)

Reviewing model history

Every time a project is committed(see page 1420), a new version of that project is created. Each project can have many historical versions(see page 1428). Versions can be tagged.
The number of project version is shown in the title of the client application (for example, MagicDraw or a Cameo Suite product, such as Cameo Systems Modeler) window - see the highlighted area in the following figure.



The project version number is highlighted.

Every element in the model, including diagrams, has its own history, and you can review(see page 1431) it in any open project.

A new version of the element is created once you modify element properties, for example, update the documentation, change the multiplicity or type of the element, create a new attribute or operation. A new version of the diagram is correspondingly created after you modify its properties, for example, rearrange the diagram layout.

You can view changes in the model in a particular scope(see page 1437) at the element level - that is a selected package or a classifier element (a composed element). The latter is a big benefit for system engineers as now you can see the history of, e. g., a selected Block[312], Requirement[313], or other composed element.

While modeling, you can analyze model history by

Reviewing historical versions of the project

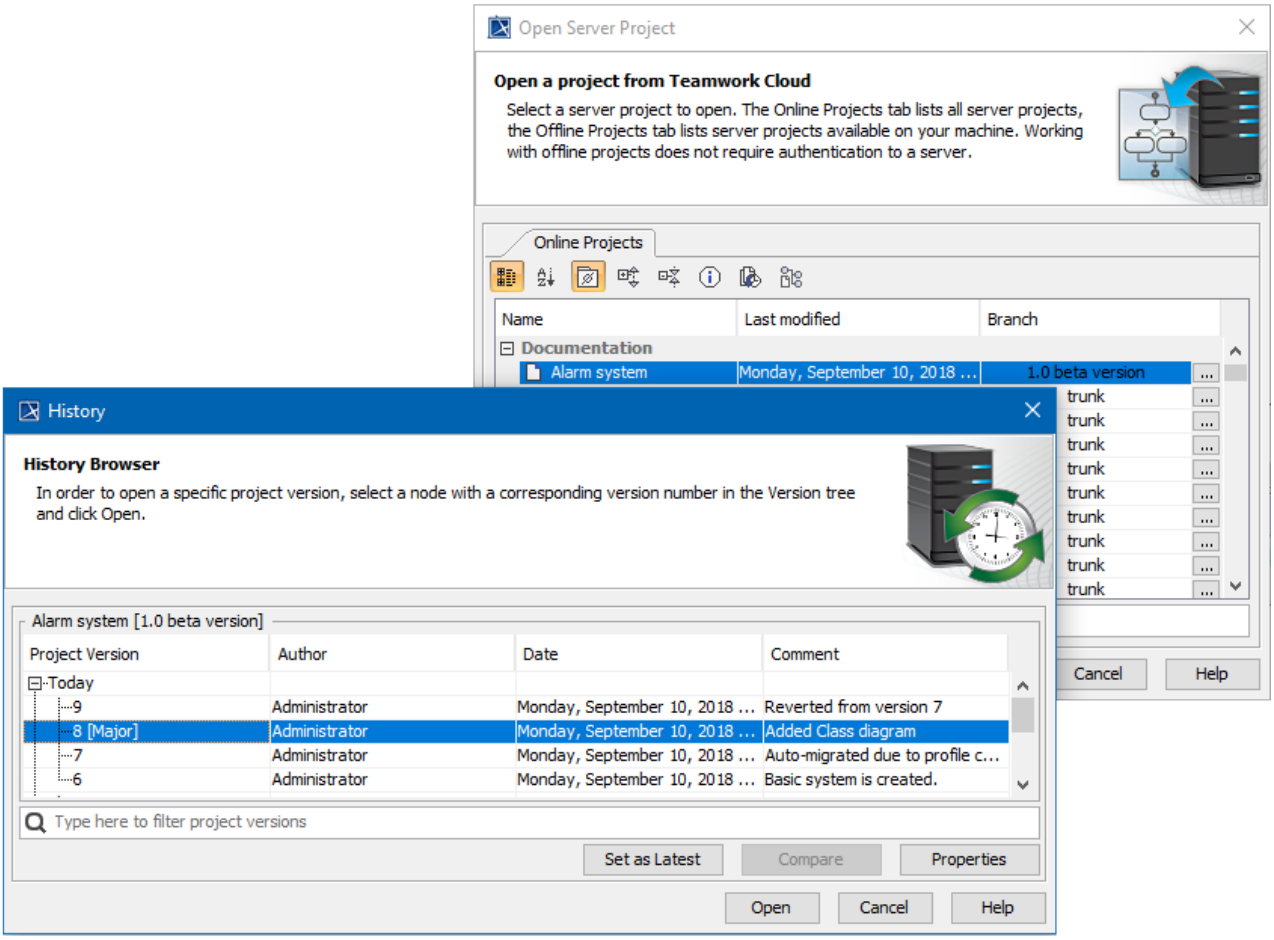
To review historical versions of a Teamwork Cloud project

---

1. From the **Collaborate** menu, select **Projects** or **Open Server Project**. The **Manage Projects** or **Open Server Project** dialog opens appropriately.
2. Select the project, whose historical versions you want to review.
3. Do either:
   - Click on the toolbar of the open project.
   - Right-click the selected project and from the shortcut menu, select **Project History**.
4. In the **History** dialog, review a list of all versions of the selected project. If a version has tags, you can see them next to the version number.

312 https://docs.nomagic.com/display/SYSMLP190SP4/Block
313 https://docs.nomagic.com/display/SYSMLP190SP4/Requirement

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Reviewing versions of a project.

Search bar
Use the search bar to filter projects according to:

- Project Version (together with the commit tags)
- Author
- Comment

Column name	Description
Project version	The version number of a project. In the example above, the last project version number is denoted as “3/1”. This is the result of setting a previous project version as the newest one. In this particular example, “3” denotes the current project version number and “1” denotes the number of the version, which has been set as the newest one.
Author	The login name of the user who created the project version.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Column name	Description
Project version	The version number of a project. In the example above, the last project version number is denoted as “3/1”. This is the result of setting a previous project version as the newest one. In this particular example, “3” denotes the current project version number and “1” denotes the number of the version, which has been set as the newest one.
Date	The date and time the project version was created.
Comment	The description of changes in the version. You can make a version comment when committing a version.
Button	**Description**
Set As Latest	Set the selected project version as the latest version.
Compare	Compares two selected branches. If there are differences between project branches, the result is presented in the Difference Viewer dialog. The button is available only when two project branches are selected (use the SHIFT or CTRL keys).
Properties	Opens **Version Properties** dialog.
Open	Open the selected version of the project. You can make changes only to the latest version of the project. Earlier versions are read-only.
Cancel	Closes the dialog.
Help	Documentation for the project version will be opened**.**

Also, you can review comments and tags of any historical version as well as edit or add new tags to it and set that version as the latest one.

To review comments and tags

1. In the **History** dialog, select a project version, whose properties you need to review and edit.

2. Click the **Properties** button. The **Version Properties** dialog opens with version comments displayed by default.
3. If you need to review and edit version tags, click the **Tag** tab. After the **Tab** opens, do ether:
    - If you need to edit a tag, select it and click **Edit**.
    - If you need to add a new tag, click **Add**.

Reviewing element history and inspecting changes

On this page

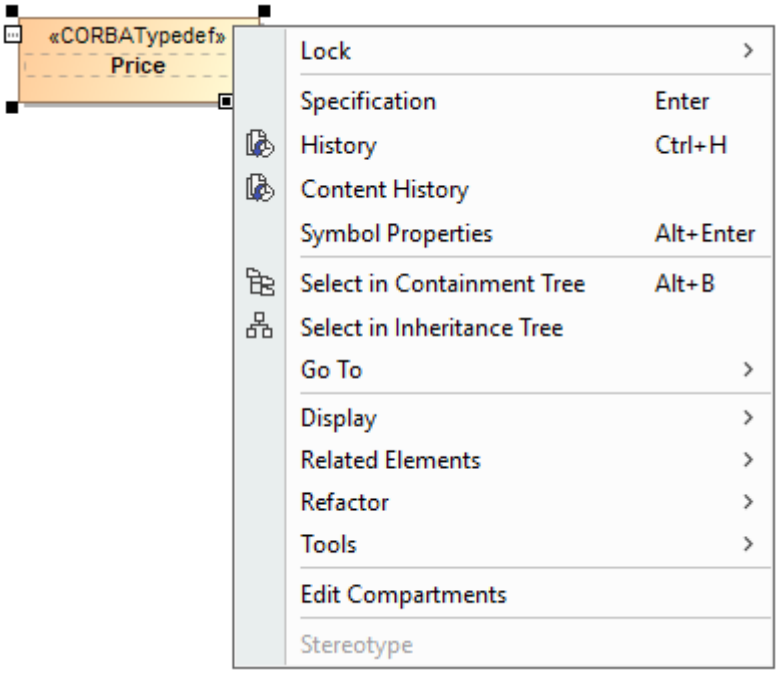
- Displaying element history (see page 1431)
- Comparing element versions(see page 1433)
- Generating reports(see page 1436)

Displaying element history

To display the history of an element, do one of the following

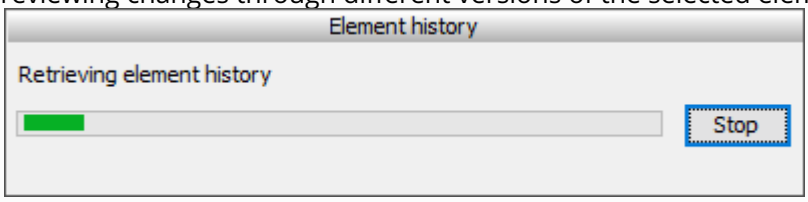
---

- Right-click the element or its symbol. From the shortcut menu, select **History**.

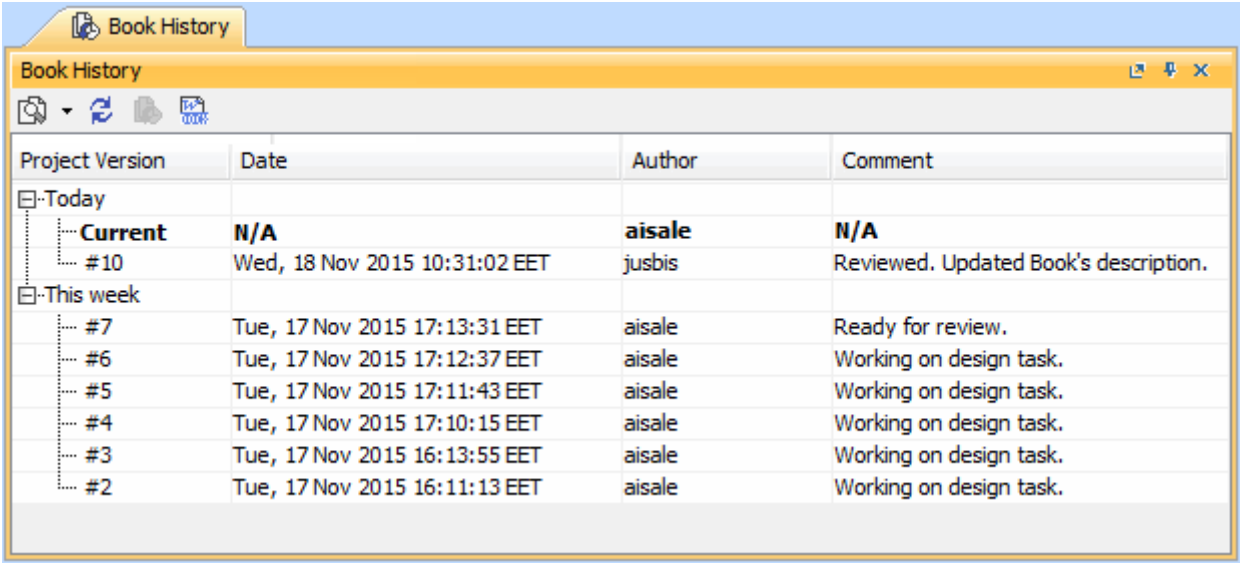


- Select the element or its symbol and press Ctrl + H.
- Open the Specification window(see page 913) of the element and click the History button .
- If it is a diagram, right-click an empty space on the diagram pane. From the shortcut menu, select **History**.

Stopping collecting changes
If retrieving the element history takes too much time, you can click the **Stop** button to complete reviewing changes through different versions of the selected element.



The **History** panel of the selected element then opens at the bottom of the client application window, such as MagicDraw or a Cameo Suite product (i.e., Cameo Systems Modeler). If you select an element in a used project, the corresponding panel title is **Usage History**. The highlighted row indicates current changes that are not yet committed. To refresh the list of versions, click the button in the panel's toolbar or perform the same actions as for opening the panel.



A list of project versions in the History panel.

By default, the list of versions is stripped to a a small number of the latest versions. To see more versions, click the **Show more results** button under the list.

You can compare any two versions of an element to see the differences between them.

To display the history of an element in a used project

1. Open the element's **Usage History** panel, as described above.
2. Select the project version.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

3. In the panel's toolbar, click the button or right-click the selected version and select the **Show History in Used Project** command.

Comparing element versions

To compare project versions in the Teamwork Cloud environment, you are required to have a **Read Resources** permission(see page 1431) for every used project in compared versions.

To compare element versions

---

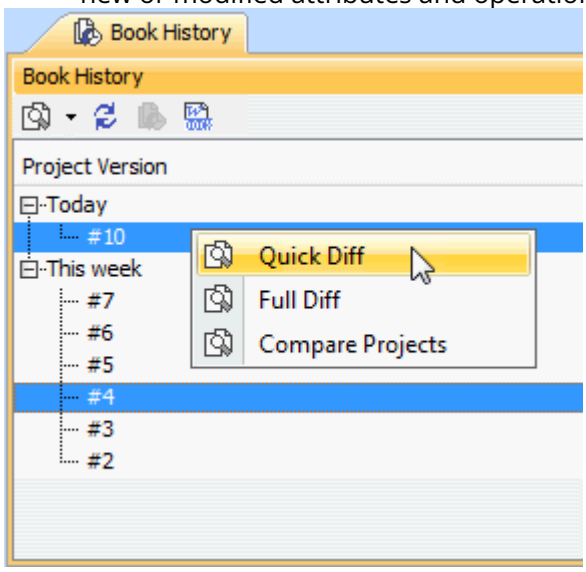
1. Hold down Ctrl and select the versions you want to compare.

   You can only compare two versions. Please note that if you select more than two items, only two (the latest and the earliest versions) will be compared.

   If you want to compare subsequent versions, you only need to select the later one.

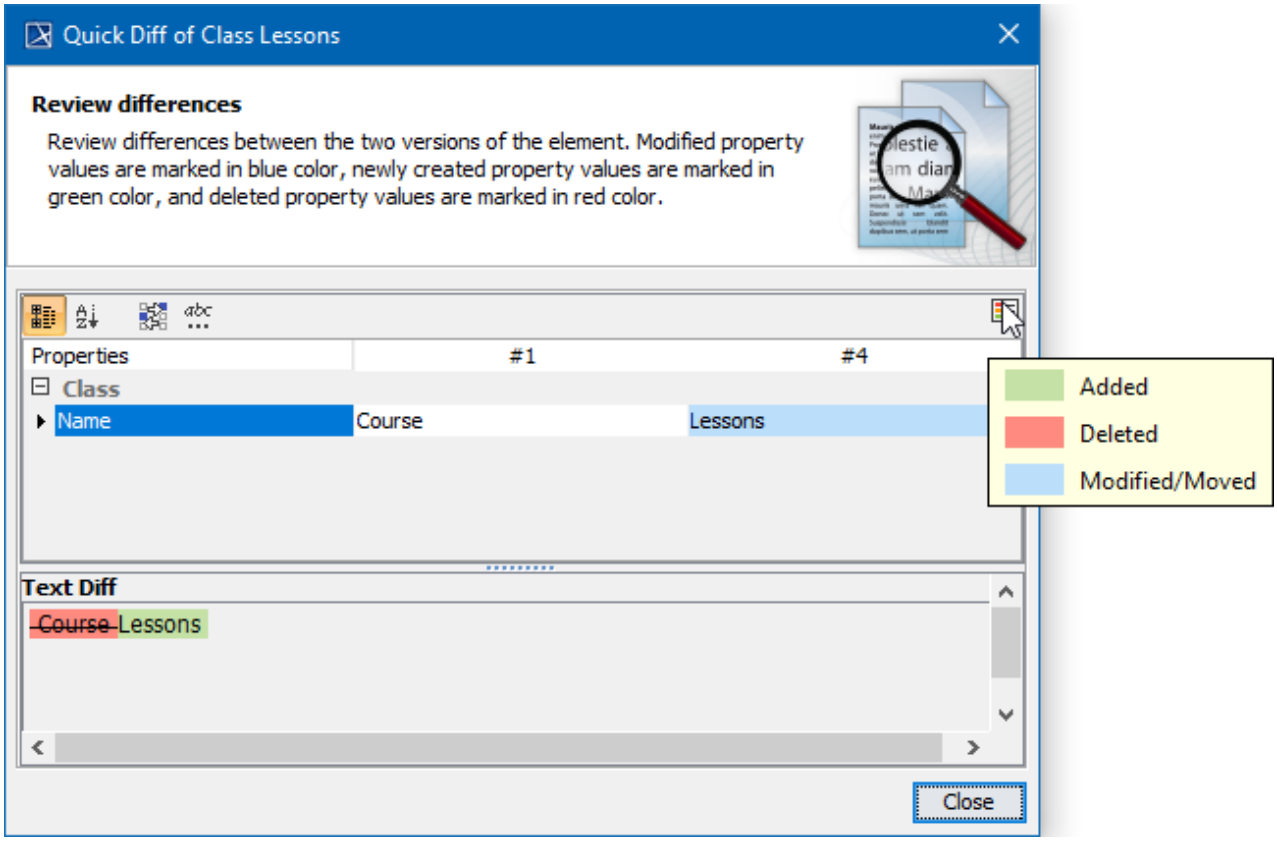
2. On the panel's toolbar, click the button or right-click the selection and choose either:

   - **Quick Diff** – to inspect changes of element property values (such as documentation, type, or multiplicity) in *a compact form*.
   - **Full Diff** – to inspect changes of diagrams or element property values in *a detailed form,* as well as look at the properties of new relationships and directly related elements (such as new or modified attributes and operations).



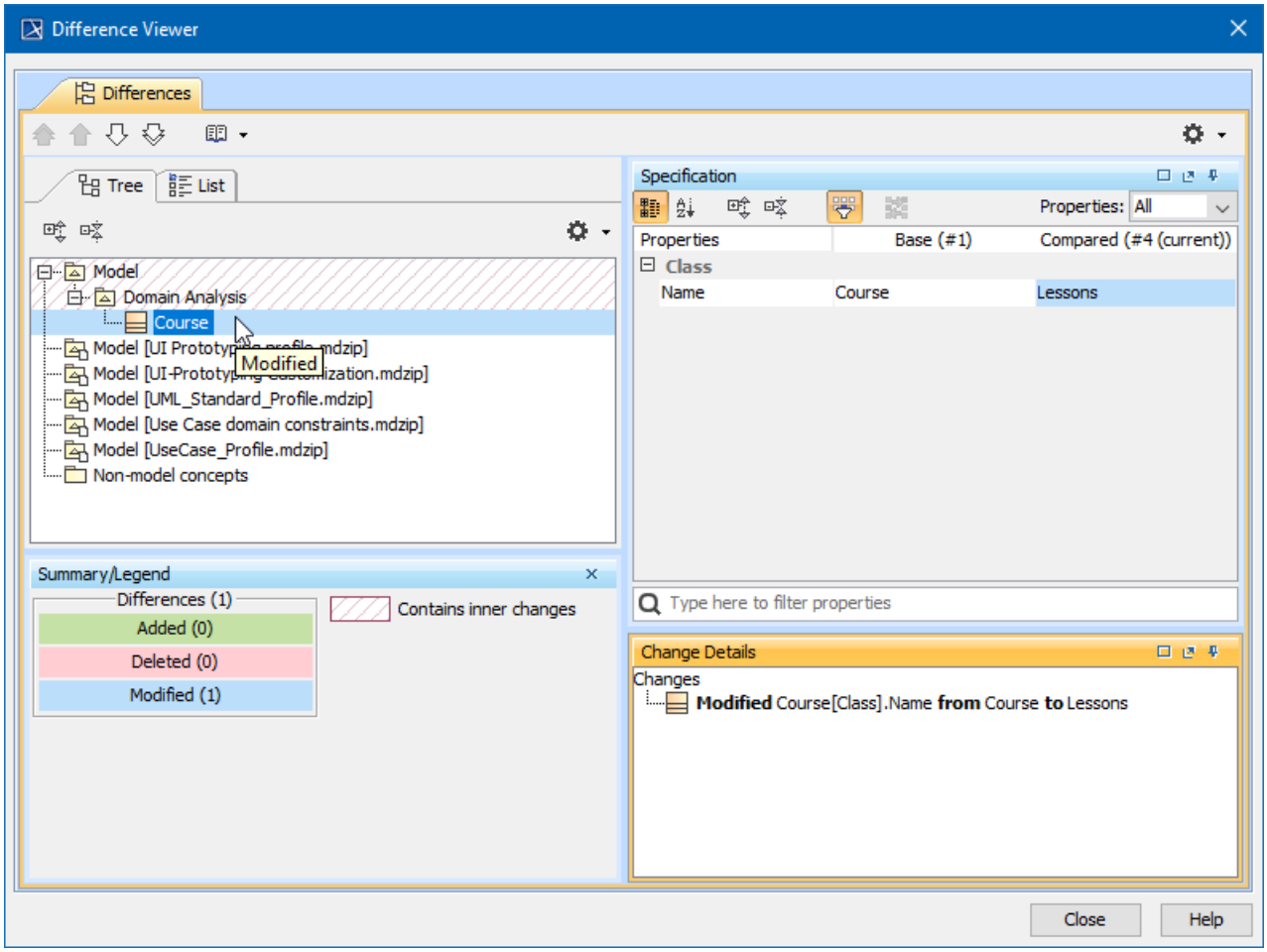
   You can also choose the same commands from the toolbar of the **History** panel.

After you select the **Quick Diff** command, a dialog opens that shows changes in element property values (see the following figure). Different colors help identify new, modified, and deleted values. You can analyze exact changes in the textual value of the selected property in the **Text Diff** box. If you want to see the original texts of both versions, click the Compare property values button on the toolbar of the dialog to open the **Compare property** dialog and then click **Original Texts**.



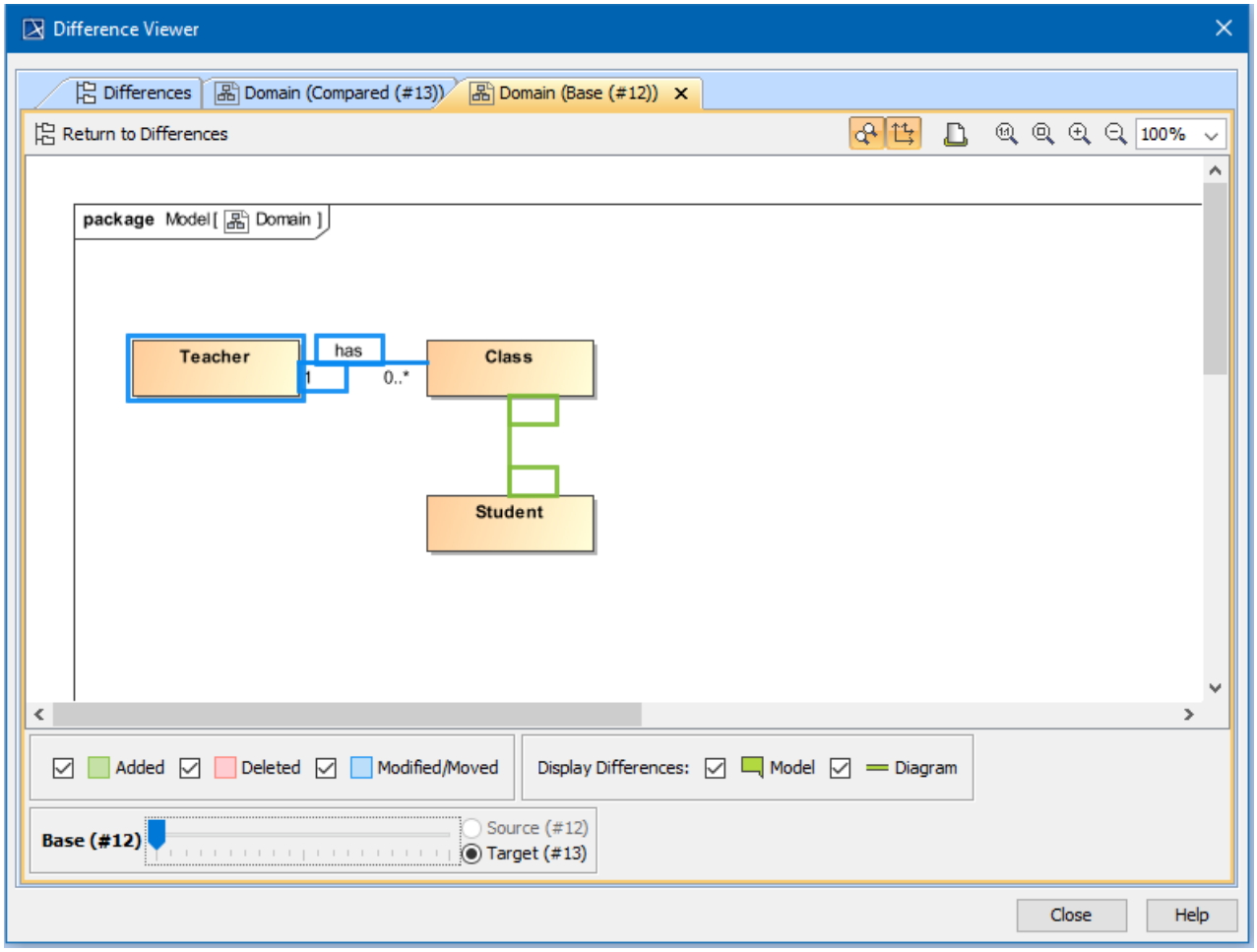
An example of the Quick Diff dialog of an element (class).

After you select the **Full Diff** command for an element, the **Difference Viewer** dialog opens, showing changes in element property values, as well as detailed information for directly related elements and new relationships (see the figure below). For more information about using this dialog, see Analyzing comparison results(see page 1295).



An example of the Difference Viewer dialog of an element.

After you select the **Full Diff** command for a diagram, the **Difference Viewer** dialog opens, showing all changes in the diagram (see the figure below). For more information about using this dialog, see Viewing and analyzing differences of modified diagrams(see page 1303).



An example of the Difference Viewer dialog of a diagram.

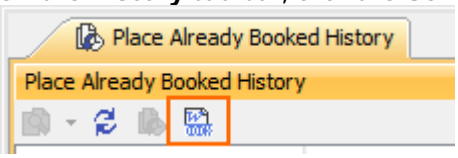
Be aware that Quick Diff is not available for displaying diagram changes. Diagram changes (as diagram layout changes) can be displayed only in the **Full Diff** mode. To see these changes you must initiate the comparison of entire project versions. For this, select a couple of project versions in the **History** panel a nd on the shortcut menu click **Compare Projects**. For more information, see Analyzing comparison results(see page 1295).

Generating reports

To generate an element history report

---

1. Perform element history as described above.
2. On the **History** toolbar, click the Generate Report button.



3. The **Save As** dialog opens to specify the location of the report file.

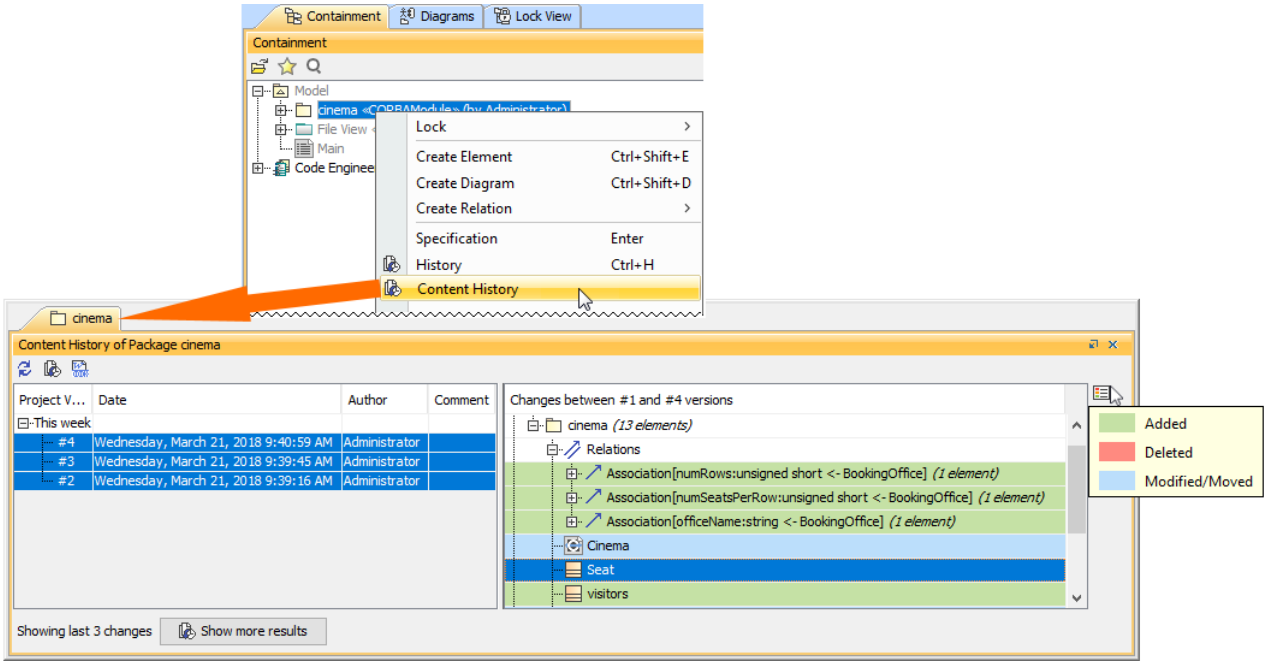
Related pages

- Reviewing model history(see page 1427)
- Analyzing comparison results(see page 1295)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Reviewing content history

You can view changes in the model in a particular scope at the element level; that is, a selected package or a classifier element (a composed element). The ability to see the history of a selected Block[314], Requirement[315], or another composed element is a big benefit for system engineers.



History of element changes in the scope of a *Cinema* package.

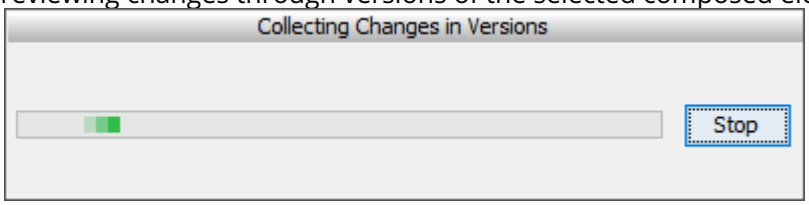
In the history review panel, you can see various details about the changes between selected versions, such as the list of elements that was changed, when changes were made, and who made changes. Different colors describe if a particular model element is added, deleted or modified. You can also export the content history of changes to a report in a *docx* format.

To view the changes of a particular project scope

1. Select a composed element in the Containment tree(see page 32) or diagram pane and right click it.
2. From the shortcut menu, select **Content History**. The history review panel opens, where you can analyze changes between selected server project versions.
3. To see differences between server project versions:
   - Double click the project version to see differences between the selected version and the earlier one.
   - Select the list of versions and click the button.

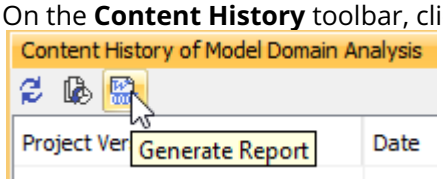
314 https://docs.nomagic.com/display/SYSMLP190SP4/Block
315 https://docs.nomagic.com/display/SYSMLP190SP4/Requirement

Stopping collecting changes
If retrieving the content history takes too much time, you can click the **Stop** button to complete reviewing changes through versions of the selected composed element.



To generate the content history report

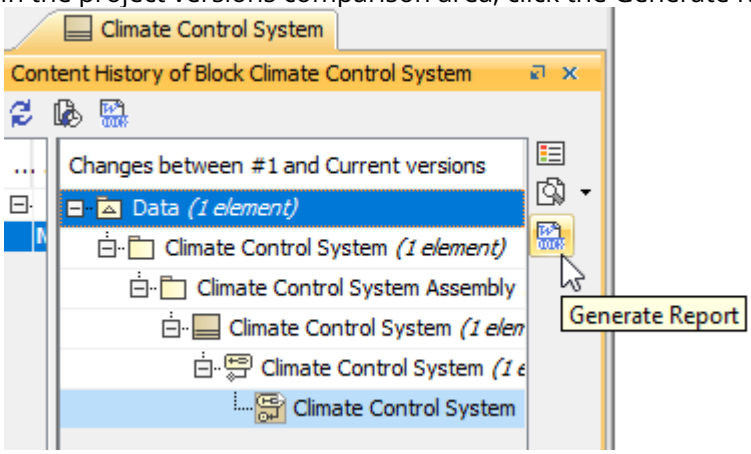
1. Perform content history as described above.
2. On the **Content History** toolbar, click the Generate Report button.



3. The **Save As** dialog opens to specify the location of a report file.

To generate a project Comparison report (Differences report)

1. Perform content history as described above.
2. In the project versions comparison area, click the Generate Report button.



3. The **Save As** dialog opens to specify the location of a report file.

In a Differences report you can find project scope and element scope:

- Project scope – name, version, the number of elements and diagrams in the compared project versions.
- Element scope – name, path and element type of the particular model element that was chosen as a content history scope.

This Differences report compared to the Full Diff (see page 1431)report only includes changes from the chosen model element scope and not the entire model. These changes are included in the sections **Diagram Differences**, **Model Differences**, and **Changes by Type** of the report.

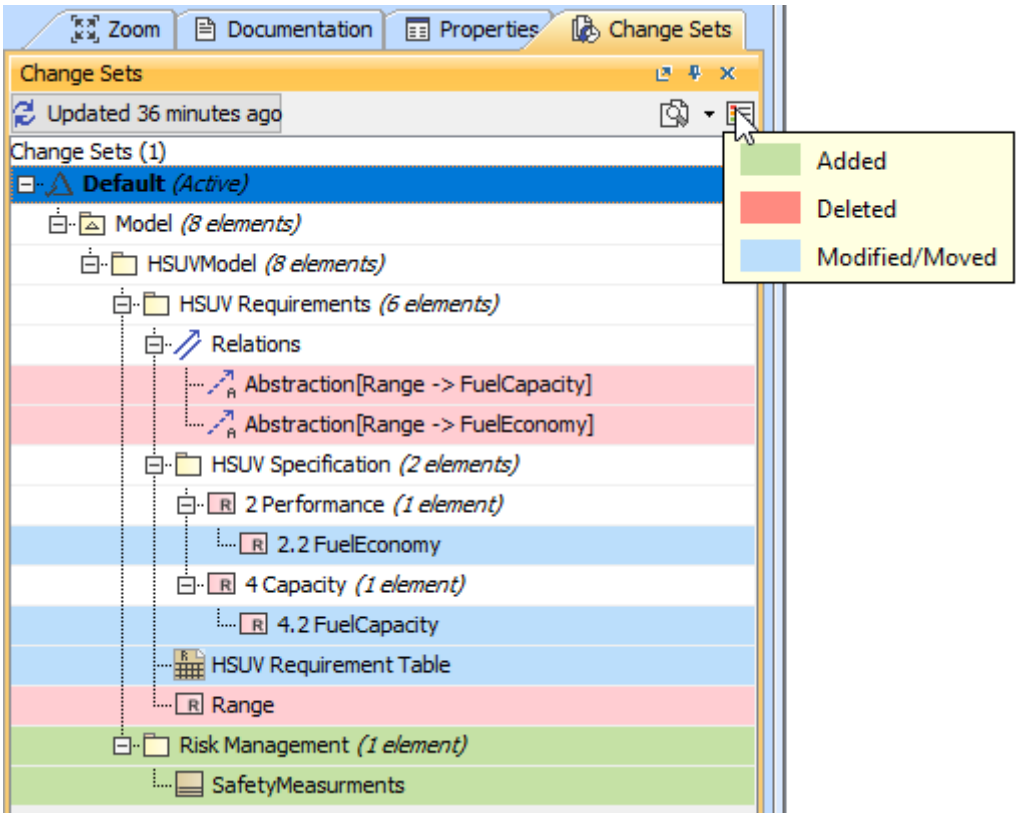
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related pages

- Analyzing comparison results(see page 1295)
- Reviewing model history(see page 1427)
- Reviewing historical versions of the project(see page 1428)
- Reviewing element history and inspecting changes(see page 1431)

Change Sets

While editing your server project, you can track project modifications in the **Change Sets** panel, located right after the Model Browser(see page 29). A change set is a related group of elements, diagrams and symbol modifications in a project that are created or modified locally and not yet committed to the server. You may have several sets for your project and commit them one by one. This feature is useful when, for example, you are working on a server project and receive a request for an immediate change. You can suspend your changes, complete the immediate task and commit the changes, after which you can resume your work on the previously suspended change set. This also applies when you are working simultaneously on different tasks that are required to be delivered as separate commits.



A set of changes made to a server project.

Editing elements
Ensure that you lock elements(see page 1403) to enable editing.

You can create as many change sets as you need. To organize and identify change sets, you may name each one. Only one change set can be active at a time. In order to create a new change set, all other change sets must be suspended. As shown in the preceding figure, colors on an element in the change set indicate whether the edit is an addition, modification or deletion. Also, you may compare differences(see page 1291) between the base project and a project with modifications applied.

To manage an *active* change set

- Right click an *active* change set and select:
  - **Commit** - to commit changes to the server
  - **Rename** - to name a change set
  - **Suspend** - to suspend a change set for further editions.
  - **Discard** - to discard the local changes in a change set. Note: all changes will be removed and this action cannot be undone.
  - **Tag version with Change ID after Commit** - for the Change ID to be automatically reflected in the Project History.

To manage *suspended* change sets

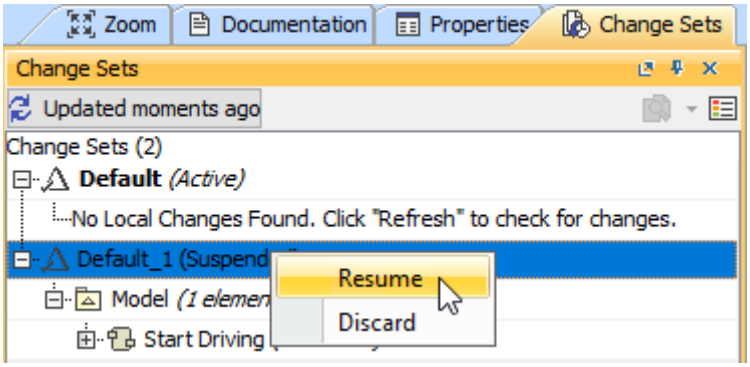
- Right click a *suspended* change set and select:
  - **Resume** - to resume working with a suspended change set. Ensure that the active change set is suspended before resuming work on another change set.
  - **Discard** - to discard the local changes in a change set. Note: all changes will be removed and this action cannot be undone.

To compare a project

1. Select the active change set.
2. On the toolbar of the **Change Sets** panel, click . The list containing three types of comparison appears. Click **Compare Projects**.
3. The **Difference Viewer** dialog(see page 1295) opens showing changes in element property values as well as detailed information of directly related elements and new relationships.

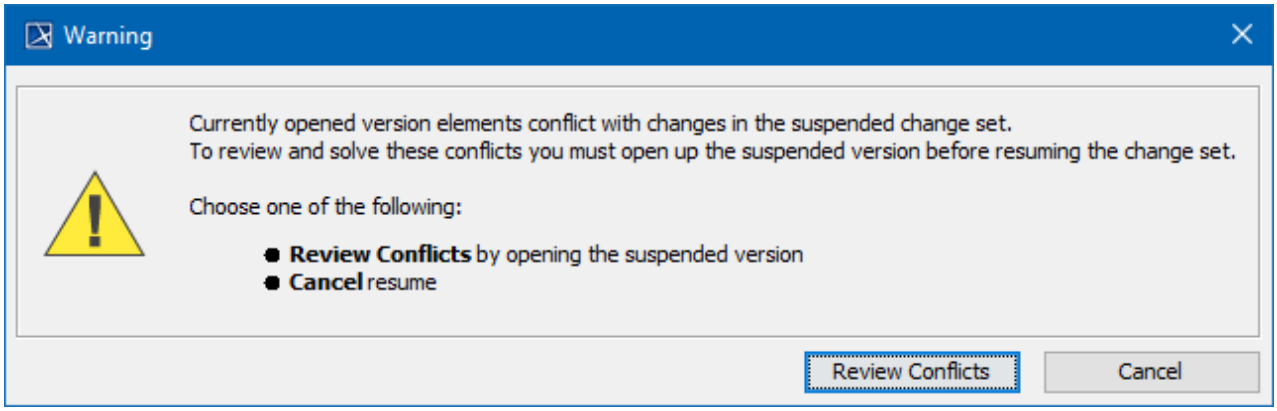
Resuming change sets in Lock-Free Editing mode

In some cases, it may not be possible to resume the suspended change sets in the Lock-Free Editing mo de without taking additional steps. The reason why this happens is the detection of conflicts among the suspended changes and present project version elements. In such a case, you need to review and resolve these conflicts(see page 1414) before you can resume the changes and continue working on them.



Resuming suspended change sets.

The project version with the suspended change set has to be opened to review and confirm the update result of the conflicting changes. You can simply do so by clicking the **Review Conflicts** button in the warning that appears:



Dialog asking to return to the project with the suspended change set version.

Related pages

- Analyzing comparison results(see page 1295)

Managing project usages

To manage project usages, you must have **Administer Resources** permission[316].

Large Teamwork Cloud (TWCloud) projects are built from smaller parts.

We recommend project partitioning when the model has weakly dependent parts, such as type libraries, models of different phases of software/system development, etc. A part of the model moved to a separate project can be used in many projects. From the perspective of the project using elements from another project, we refer to the other project as a *project usage* or *used project*.

You can easily detach a selected part of the model by exporting(see page 1442) it to a separate TWCloud project and re-use(see page 1443) it in other server projects. Since projects are used as *read-only*, you cannot modify their contents directly from the main project. To modify (see page 1450)a used project,

316 https://docs.nomagic.com/display/TWCloud190SP4/Permissions

you should open it as stand-alone. If there is a need, you can simply switch(see page 1447) to another used project version.

If you no longer need a fragment of the model in a separate project, you can import (see page 1451)the project into the main project. Additionally, you can stop (see page 1452)using the project.

You can manage TWCloud project usages in the **Project Usages** dialog, which opens after you select **Options** > **Project Usages**.

Related pages

- Exporting a package to a new server project(see page 1442)
- Start using a project in your project(see page 1443)
- Editing the content of a used project(see page 1450)
- Changing the used project version(see page 1447)
- Importing the content of a used project into your project(see page 1451)
- Stop using the project in your project(see page 1452)

Exporting a package to a new server project

To export packages, you must have an **Administer Project** permission[317].

Exporting a package to a new server project is necessary when you need to detach a part of the model from the project. The part can be a library or a fragment of the model simultaneously developed by another team. You can use the part of the model you moved in *more than one* project.

Exporting a package to a separate project is the first step of the model decomposition. It is followed by reusing(see page 1443) that project in other projects.

To export a package to a new server project

---

1. In the Model Browser, right-click the package to export.
2. Select **Project Usages** > **Export Packages to New Server Project**. The **Export Packages to New Project** dialog opens. The package is already selected in this dialog.
3. Click **OK**.
4. When the message asking to show used read-only projects appears, click **Yes** (this is our recommendation).
5. When the message asking to check for dependencies between the package and the rest of the model appears, click **Yes** (this is our recommendation). If the package depends on other projects, eliminate those dependencies before exporting it.
6. In the **Project Name** dialog, do the following:

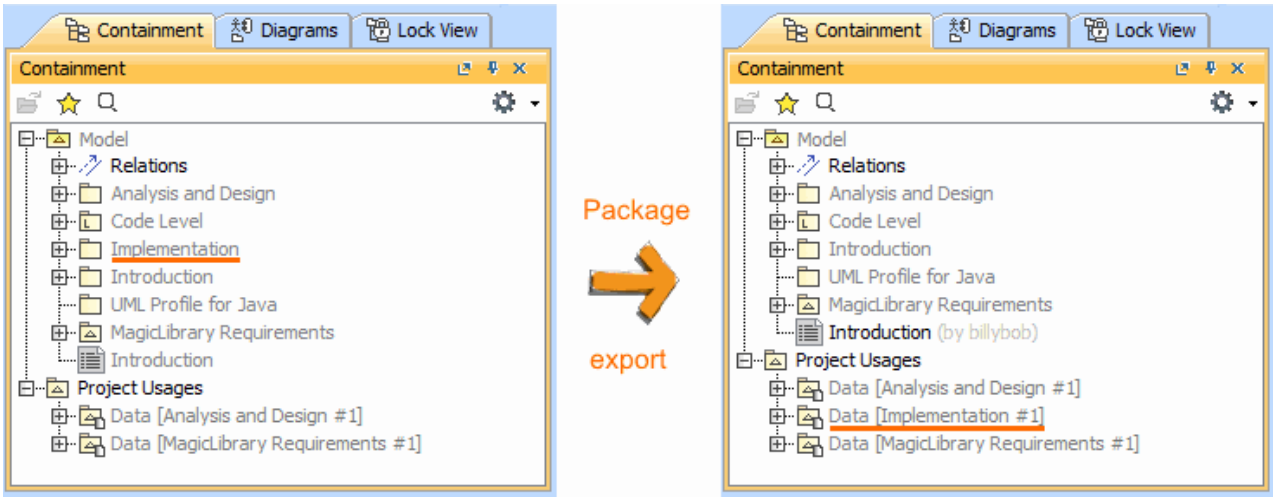
    a. Change the default name of the new server project, if you choose.
    b. Select a category to store the new project.
    c. Add a comment for the project.
    d. Click **OK**.

7. Wait while the export completes.

The newly created server project has the structure and the contents of the exported package and is automatically used in the project from which it was detached. The used project appears under the

317 https://docs.nomagic.com/display/TWCloud190SP4/Permissions

package *Project Usages* in the Model Browser, and the exported package no longer appears under the root package in the Model Browser.



You can reuse(see page 1443) the newly created server project in other projects.

Start using a project in your project

> ⚠ To begin using a project in your current project, you must have an **Administer Project** permission[318].

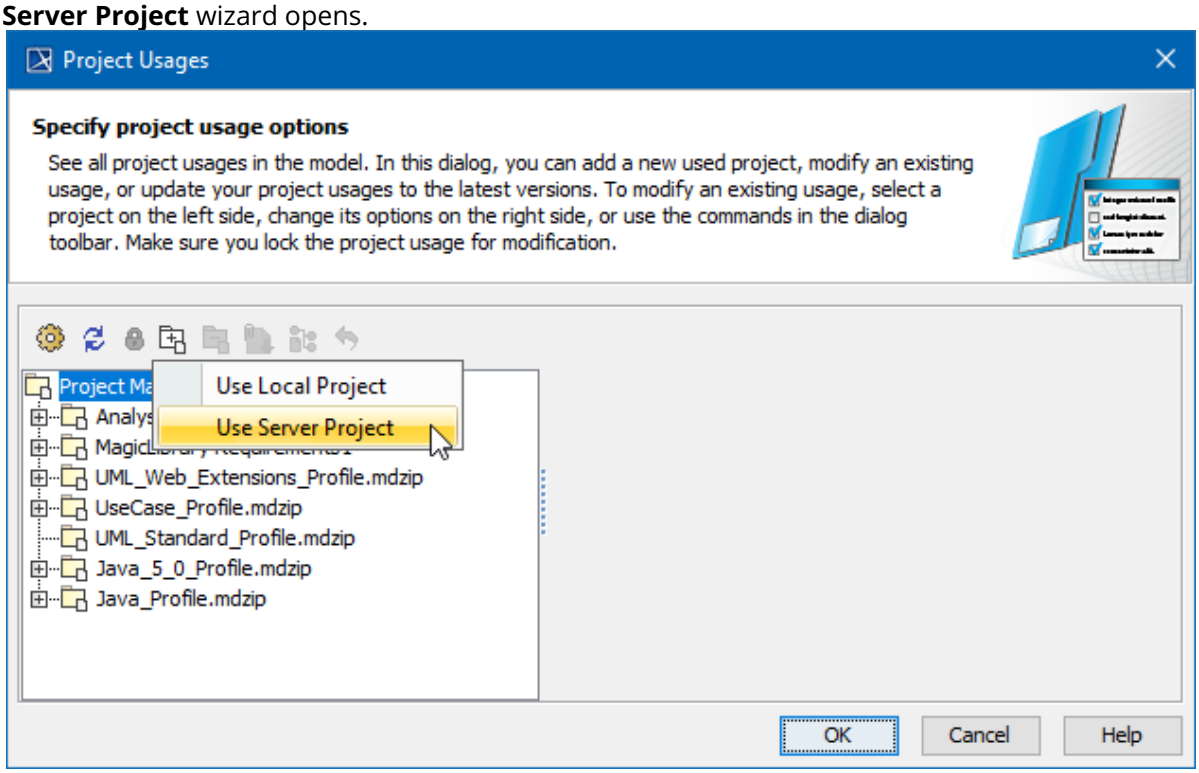
If you want to use an element from either a local or Teamwork Cloud (TWCloud) project, you must use that project in your project first. Projects can be used only in the read-only mode.

To start using a server project in your project

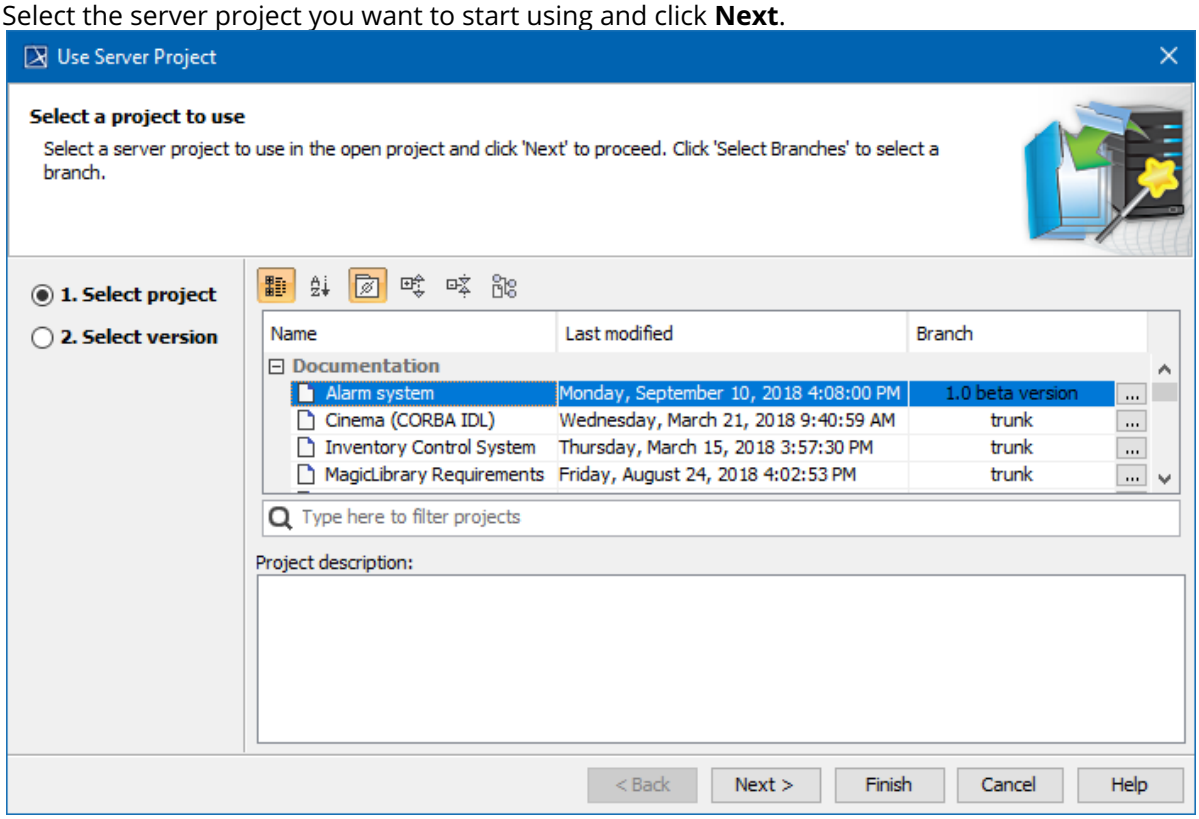
1. Select one of the following:
   - **Options** > **Project Usages**
   - **Collaborate** > **Project Usages** > **Project usages**
   The **Project Usages** dialog opens.

318 https://docs.nomagic.com/display/TWCloud190SP4/Permissions

2. Click the button and then select **Use Server Project** (see the following figure). The **Use Server Project** wizard opens.



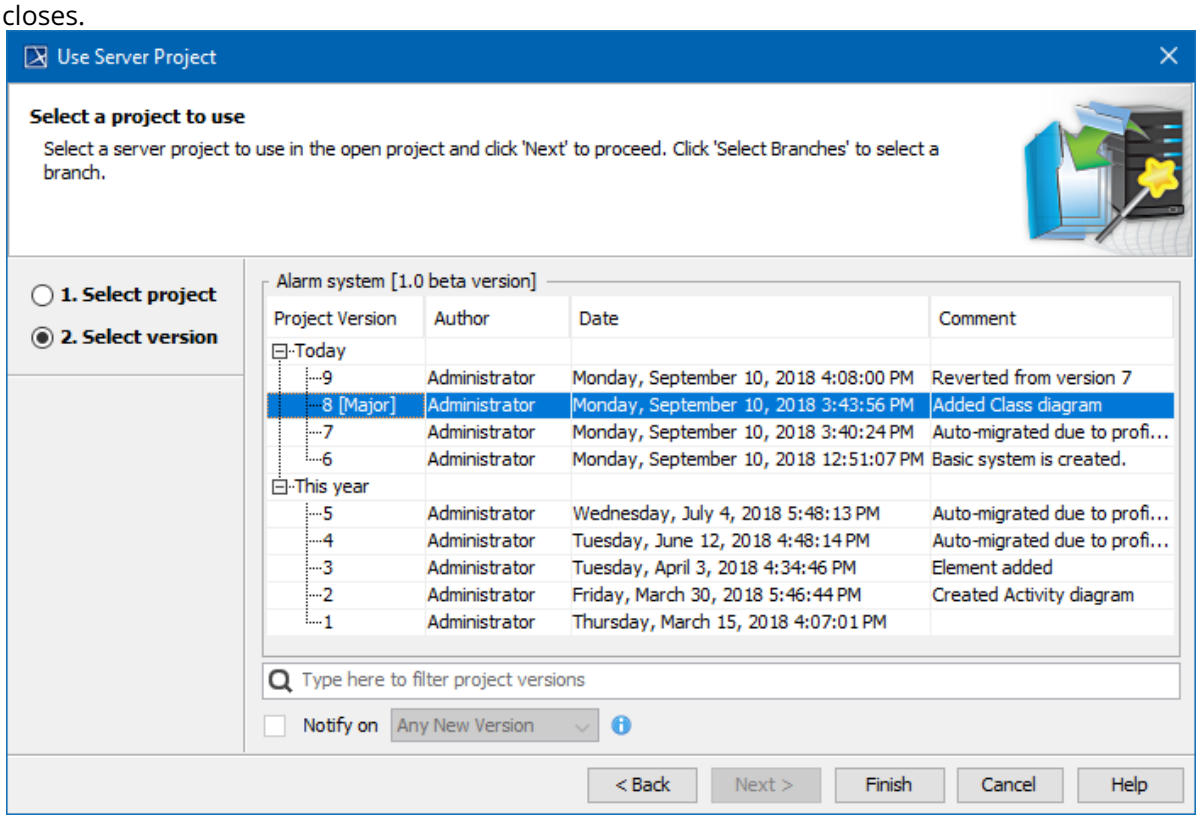
3. Select the server project you want to start using and click **Next**.



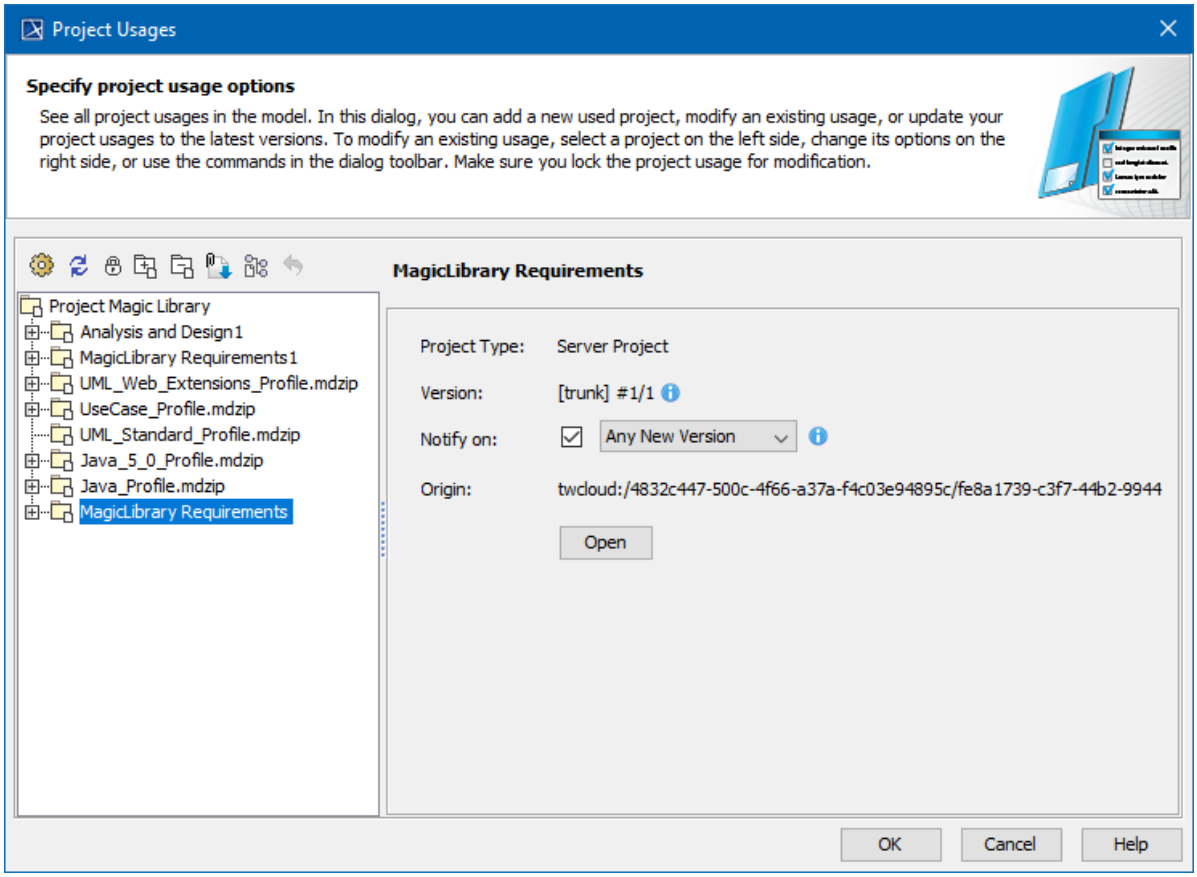
Directly/Indirectly used projects
The selected project is treated as a directly used project. A project used by a used project is treated as an indirectly used project.

Search bar
Use the search bar to filter projects according to:
- Project Version (together with the commit tags)
- Author
- Comment

4. Select a version of the server project to use and click **Finish** (see the following figure). The wizard closes.



5. Wait while the usage to the selected project is established.
6. View the selected server project added to the usages of your project In the **Project Usages** dialog.



The used project version shows which version is used and the latest project version (for example, #2/3 means project version 2 is used but version 3 is the latest project version).
Select **Notify on Any New Version** to get instant notifications about changes to used projects to ensure they are always up to date.

> **Used project**
> Please note, the whole content of a used project is added to the main project.

7. Click **OK** to close the dialog.

The selected server project also appears in the Model Browser, under the package *Project Usages*.

Once the usage to the project is established, you can start using its elements. For example, you can select them as types in your project.

Establishing usage to a local project is very similar. In this case, you don't have to select the project version, since local projects don't have them.

Related pages

- Editing the content of a used project(see page 1450)
- Changing the used project version(see page 1447)
- Importing the content of a used project into your project(see page 1451)
- Stop using the project in your project(see page 1452)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Changing the used project version

On this page

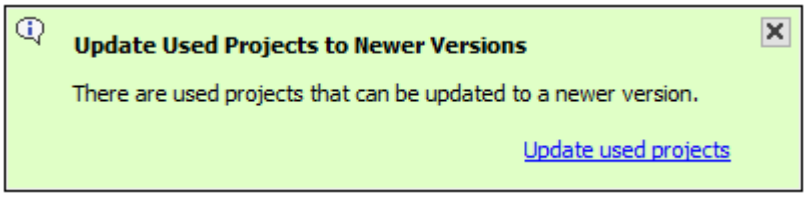
- Setting notifications about version changes(see page 1447)
- Notifications for any new version(see page 1448)
- Notifications for new tagged versions(see page 1448)
- Updating versions of used projects (see page 1449)
- Changing versions of used projects(see page 1450)

To change a used project version, you must have an **Administer Project** permission[319].

You can set an option to receive a notification when a new used project version appears. You may then choose to update the used project version to the latest version or keep working with the current one.

Setting notifications about version changes

By default, when you open your server project you will get a notification if there are used project versions later than the used one in your project that meet notification criteria. When this notification appears, click the **Update used projects** link and select the projects you want to update in the open **Update Used Projects** dialog.



You can change the default notification settings to receive notifications when a server project is opened and when updates are done, as well as disable them if they are not necessary.

To set when to show notifications about newer used project versions

1. On the main menu, click **Options** > **Environment** to open the **Environment Options** dialog.
2. In the options group list, select **Collaboration**, then select the **Notify About Newer Project Usages** option, and specify the commands to be notified:
    - on project opening and updating;
    - on project opening; or
    - never.

319 https://docs.nomagic.com/display/TWCloud190SP4/Permissions

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Notifications for any new version

If you do not need to specify a used project version with tags, you can choose to get notifications for any new version.

To get notifications for any new version

---

1. From the main menu, select **Collaborate** > **Project Usages** > **Project Usages**.
2. Select a used project and click to select the checkbox next to **Notify on**.
3. From the drop down list select **Notify for Any New Version**.
4. Click **OK** to close the dialog.

Notifications for new tagged versions

When you want to get notifications about updated used projects during your modeling process, you must define the used projects for routine version checking. You are allowed to specify certain commit tags for every used project. The modeling tool will show a notification when a new version of a used project with a specified commit tag is available.

To get notifications about changes to a certain tagged used project version

---

1. From the main menu, select **Collaborate** > **Project Usages** > **Project Usages**.
2. Select a used project and click to select the checkbox next to **Notify on**.
3. From the drop down list select **Notify on New Tagged Version**.
4. Click **Specify Tags**, near the drop-down list.
5. In the **Specify Tags** dialog, double click the first row. To add or delete rows, do one of the following:
   a. To add a row, click , or press Alt + Enter.
   b. To delete a row, click , or press Alt + Backspace.
6. Specify the tag, e.g., *2018Q* (tags are case insensitive), and then click **OK** to close the dialog.

If you do not specify tags, the modeling tool will show a warning icon.
It is recommended that tags specification be done in patterns, because the modeling tool searches for partial matches. See the example below for more information.

Example
After a new version of the used project is committed to the server with a tag that fully or partially matches the specified tag, you will be notified about updates of these tagged projects.
Let's say you specify these tags:
2018Q
2019Q
2020Q
Major
New versions and notification status:

- New version without tags - no notification
- New version with the tag [Major] – notification about a new version
- New version with the tag [2018q1] – notification about a new version
- New version with tags [Release][Major] – notification about a new version
- New version with the tag [Release] – no notification
- New version with the tag [Q_2018Q2T] - notification about a new version

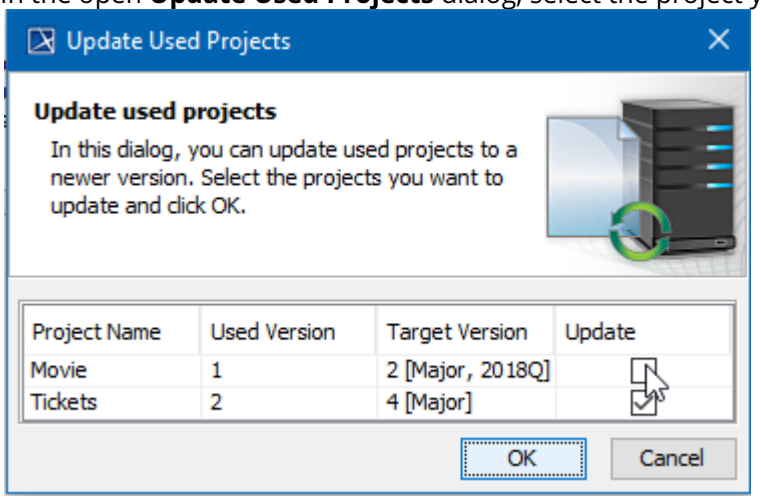
7. Click **OK** to close the dialog.

Updating versions of used projects

If a used project has tags specified for notification, only the version meeting the used projects notifications criteria (tags) will be shown in this dialog.

To update the used project version

1. Lock the used project(see page 1403).
2. From the main menu, select **Collaborate** > **Project Usages** > **Project Usages**.
3. Click the **Update Used Projects** button.
4. In the open **Update Used Projects** dialog, select the project you want to update.

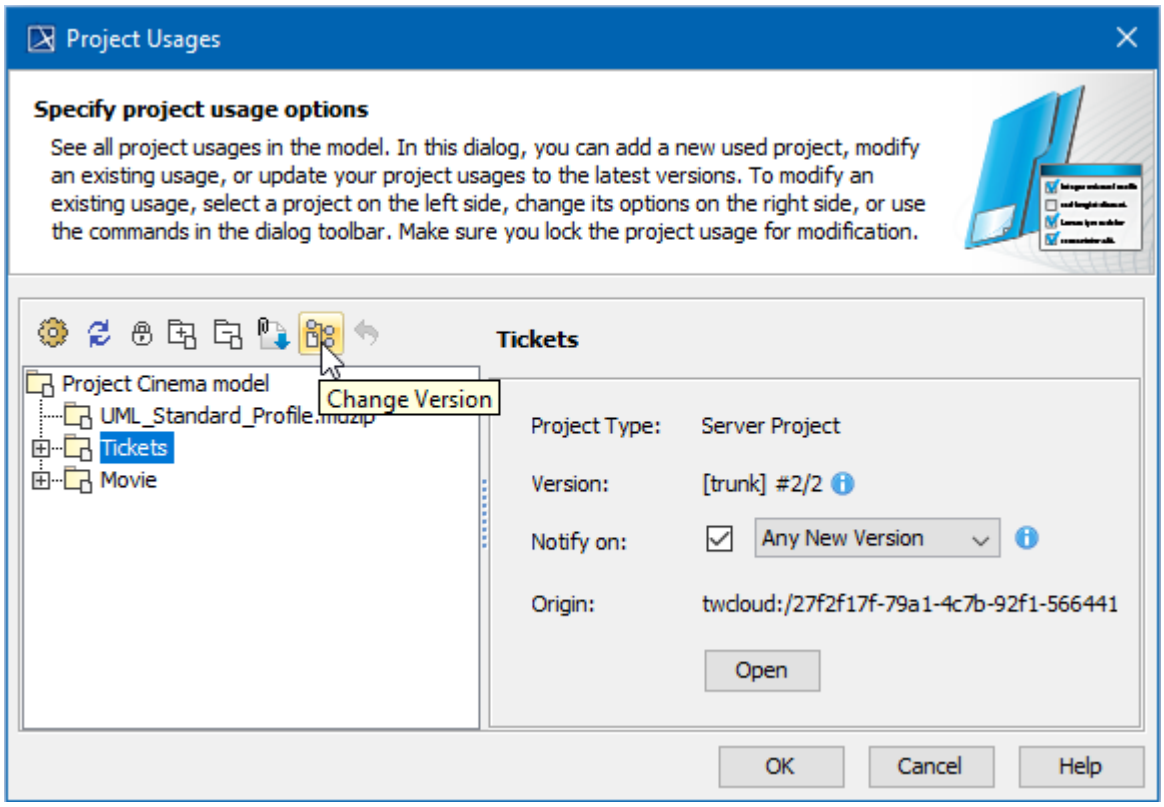


5. Wait while the version changes to the latest and click **OK** to close the dialog.

To change the used project version

1. Lock the used project(see page 1403).
2. From the main menu, select **Collaborate** > **Project Usages** > **Project Usages**.
3. Select a used project.
4. Click the button to change the version of a used project.

> **Shortcut menu**
> This step can be done via shortcut menu. Right-click project and then select **Change version.**



5. In the open dialog, select the used project version to switch to and click **OK**.
6. Wait while the version changes to the selected one and click **OK** to close the dialog.

Versions are displayed in the following format: *#<used version>/<latest version>.*

> The term #1/3 denotes that 1 is the used version and 3 is the latest version.

Editing the content of a used project

> To switch versions of the used projects you must have an **Administer Project** permission[320].

320 https://docs.nomagic.com/display/TWCloud190SP4/Permissions

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Teamwork Cloud (TWCloud) projects can only use other projects in the read-only mode. If you want to edit a used project, you must open it as stand-alone.

To open a used project for editing

1. In the Model Browser, right-click the project to open a shortcut menu under the package *Project Usages*.
2. Click **Project Usages** and select:
   - **Open Latest Version** to open the latest version of the used project for editing. Note that the version used in the project does **not** change after editing the used project.

     Updating used project versions
     If you select the check-box **Notify on New Version** in the **Project Usages** dialog(see page 1447) for a particular used project, you will be notified about the existence of a later used project version while reloading the main project.

   - **Open This version** to open the actual project version for editing. If the used version is not the latest one, the project will open in the Read-only mode.

Now you can modify the structure and content of the project.

Updating used project version
Please note, the used project version in the main project remains the same. You should update the usage manually(see page 1447).

Importing the content of a used project into your project

To import the content of a used project into your project, you must have an **Administer Project** permission[321].

If you no longer need to keep a part of the model in the used project, you can import the content of that used project into your project. After the import, all the elements of the used project are copied to the main project. The inner structure of the used project is included in the project as a package.

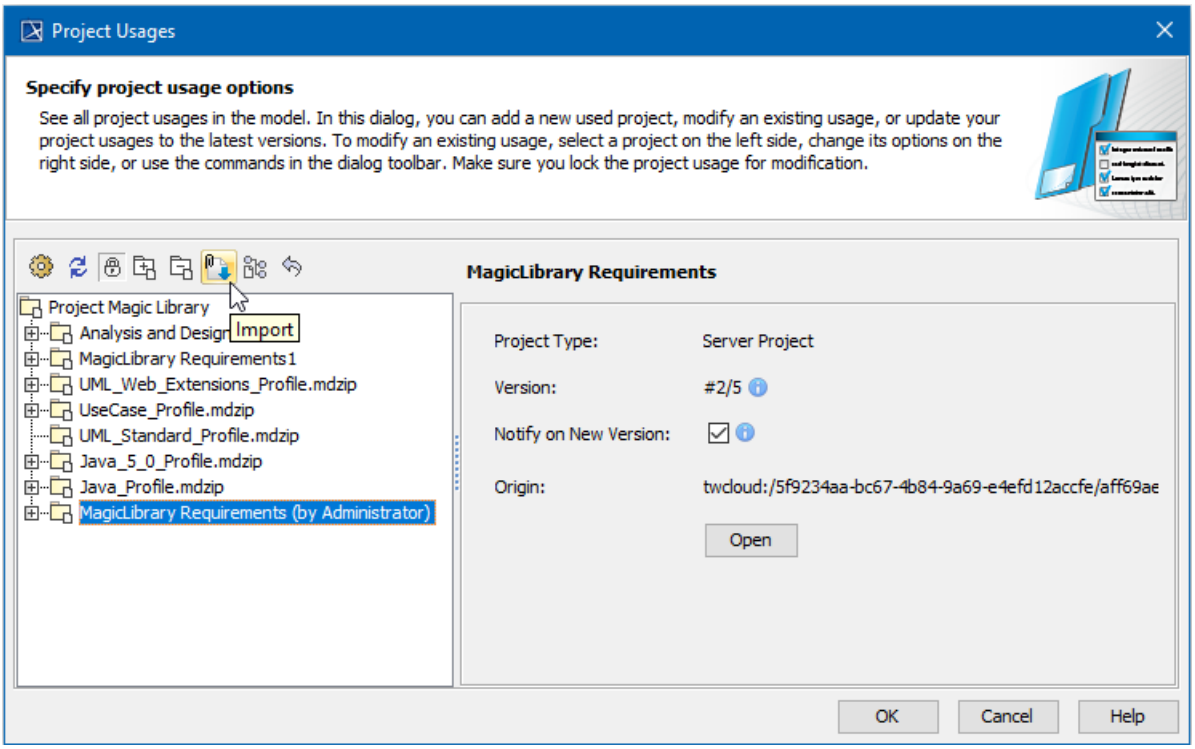
Standard/system profiles, such as *UML_Standard_Profile.mdzip*, cannot be imported.

To import the contents of a used project into your project

1. Lock the used project(see page 1403).
2. On the main menu, click **Collaborate** > Project Usages > **Project Usages**.
3. Select the used project and click the button.

   Shortcut menu
   This step can be done via shortcut menu. Right-click project and then select **Import**.

321 https://docs.nomagic.com/display/TWCloud190SP4/Permissions



4. In the **Import** dialog, choose how to import the elements of the used project: with new IDs or with old.
5. If you receive a message asking to commit changes made in your project before the import, click **Yes** and wait while the import completes. This may take some time.

> **Note**
> The imported project no longer appears in the **Project Usages** dialog.

6. Click **OK**.

The imported project also disappears from the package *Project Usages* and a new package named after the imported project appears under the root package in the Model Browser. If the imported project uses other projects, these projects will be directly used in your project.

After it is imported to the contents of your project, a used project will still exist in the Teamwork Cloud (TWCloud) repository.

Stop using the project in your project

> To stop using the project in your project, you must have an **Administer Project** permission[322].

If you no longer need to use a project, you can easily remove it from your project usages.

Removal can be:

- Complete, after which all references to the elements stored in the used project are eliminated.

322 https://docs.nomagic.com/display/TWCloud190SP4/Permissions

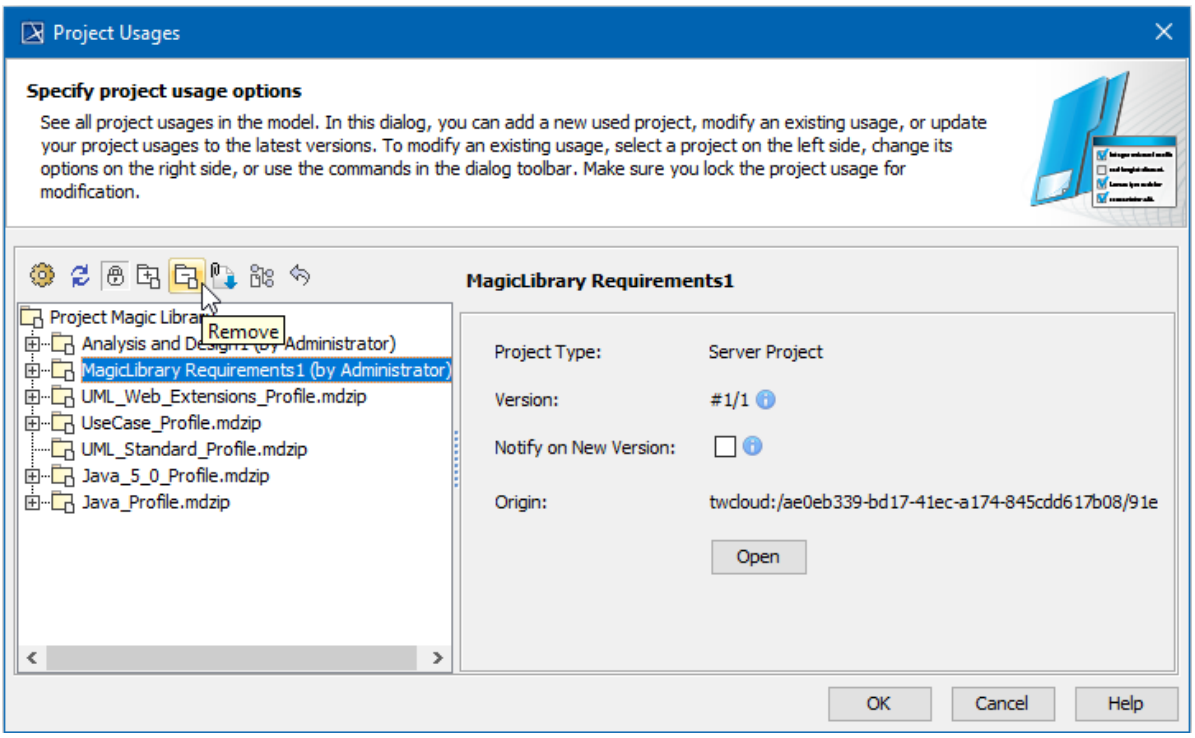
- Incomplete, after which all references are preserved. Choose this method when you need to replace the used project with an equivalent project containing the same elements with slightly different content.

Before you attempt to stop using a project, make sure other users are not currently creating usages to this project.

To stop using a project in your project

1. Lock the used project(see page 1403) you want to stop using.
2. On the main menu, click **Collaborate** > **Project Usages** > **Project Usages**.
3. Select the used project and click the button.

Shortcut menu
This step can be done via shortcut menu. Right-click project and then select **Remove**.



4. Wait while the removal completes.
5. Click **OK**.

The project is no longer used in your project. It will disappear from the Model Browser of your project.

Linking an embedded project with relevant server project

To link an embedded project with relevant server project, you must have an **Administer Project** permission[323].

323 https://docs.nomagic.com/display/TWCloud190SP4/Permissions

Linking replaces the privately embedded project (also referred to as the local project) with a relevant server project.

An embedded project appears in your project after you:

- Start using it(see page 1443) in your project.
- Refuse to add a used project(see page 1370) to Teamwork Cloud (TWCloud) together with the main project.

The embedded project can be:

- Used (referenced) in the main project
- Imported into the main project(see page 1451)
- Stopped being used(see page 1452)

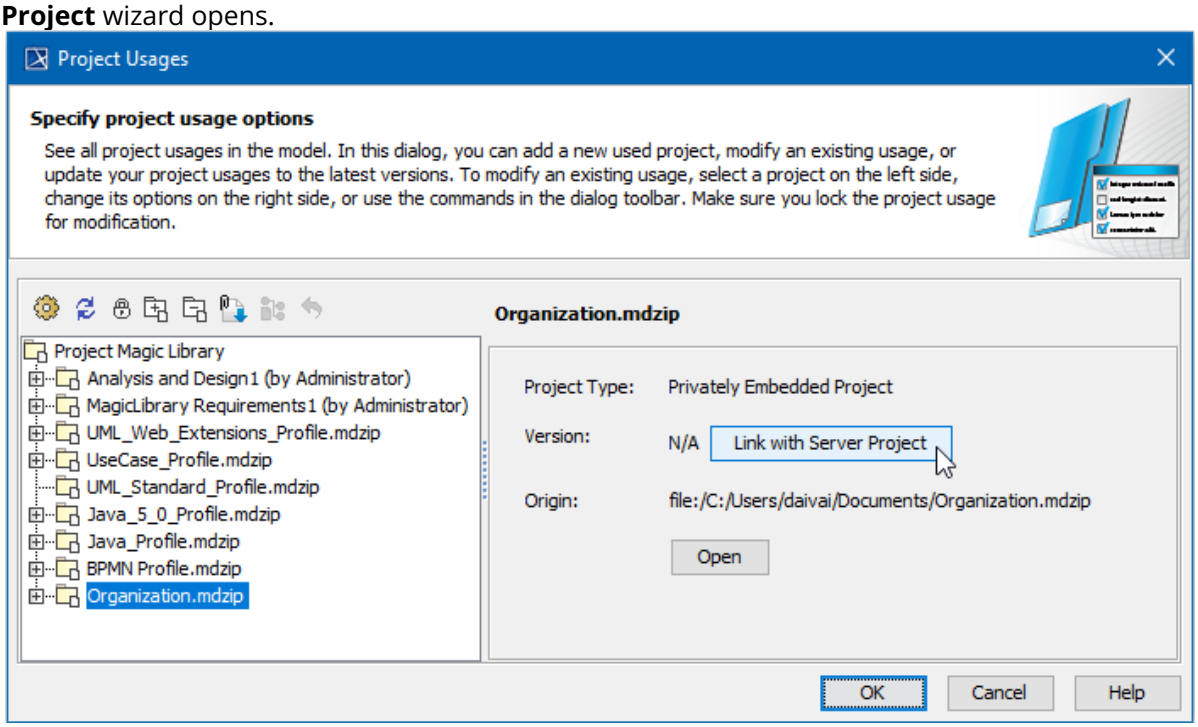
However, because it is not a server project, it has no versions and cannot be updated.

If a relevant project is stored on the server (it may be added to the server individually, separately from the project using it), you can easily link the embedded project with it.

To link an embedded project with a relevant server project

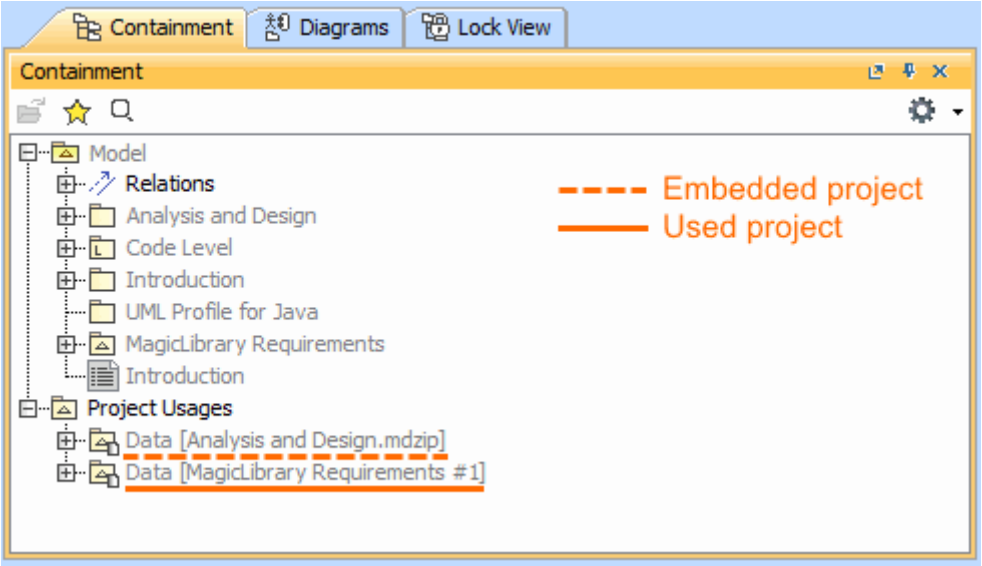
---

1. In the Model Browser, under **Project Usages**, right-click the embedded project.
2. Select **Project Usages** > **Project Usage Options**. The **Project Usages** dialog opens. The embedded project is already selected.
3. Click the **Link with Server Project** button (see the following figure). The **Use Server Project** wizard opens.



4. The same steps of using a project in the server project(see page 1443) are outlined below.

The following figure highlights the differences between displaying an unlinked embedded project and one linked with the server project (converted to the used project).



Notation differences in Model Browser for embedded and used projects.

Configure used projects

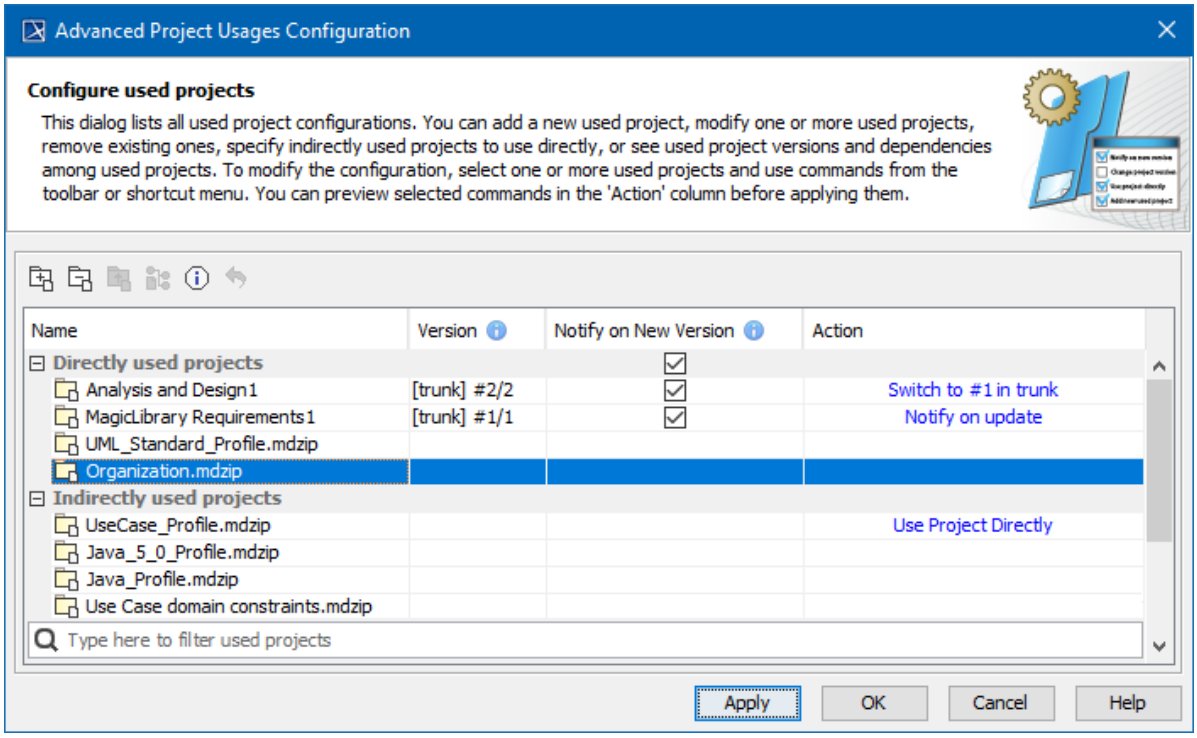
You can see all used projects and their versions In the **Advanced Project Usages Configuration** dialog.

To open the **Advanced Project Usages Configuration** dialog

1. Login to the Teamwork Cloud server.
2. Open your server project.
3. On the main menu, click **Collaborate** > **Project Usages** > **Project Usages**. The **Project Usages** dialog opens.
4. Click the button. The **Advanced Project Usages Configuration** dialog opens.

> **Search bar**
> You can use the search bar to faster filter used projects by name.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



In this dialog you can make various modifications for one or more projects at a time. Select one or more used projects and:

- Click the **Notify on New Version** check-box to get notifications when a new version of a particular project is available on the server.
- Use the commands in the toolbar or shortcut menu to:
  - Add a new project(see page 1443) to the server project you are working with
  - Remove a used project(see page 1452) from the server project you are working with.
  - Start using an indirectly used project directly.
  - Change the version(see page 1447) of a selected used project.
  - Review the information of a selected used project.
  - Reset changes to a selected used project. You must do this before applying selected actions!

In the **Action** column, you can see the action to be performed on a particular project after the **Apply** button is clicked.

To turn the action on, click one of the following:

- **Apply** - to apply changes and keep the dialog open in order to see what changes are applied. Please note, the applied changes cannot be undone.
- **OK** - to apply changes and close the dialog.

Moving elements to other projects

You can move selected elements from the main server project to a directly used project(see page 245). The relations are also moved with the element(s).

> **Moving cannot be undone**
> This action cannot be undone as a new version is automatically created and committed to the Teamwork Cloud server after the selected elements are moved to a new location.

To move elements from the main server project to a used one

---

1. In the Model Browser, select one or more elements.
2. Right-click the selected elements. From the shortcut menu, choose **Refactor** > **Move Element(s) to Other Project**.
3. The **Move Elements** dialog opens. Select a package or a classifier in the used project where you want to move the elements. You will see the classifier as a possible destination **if** only classifier elements are selected to move. Otherwise, only packages will be shown.
   - > **Drag the selection**
     > You can drag and drop the selected elements to the used project package or classifier instead of using the shortcut menu. This action can be performed if the following conditions are met:
     > - if the selected elements to move contain at least one non-classifier element, it is restricted to drop to other targets than packages.
     > - if the selected elements to move contain only classifiers, moving is allowed to either packages or classifiers.
4. Do one of the following:
   - Click **Move** to start moving elements or
   - click **Select More** to specify your selection. The **Select Elements** dialog elements opens, in which you can:
     - add or remove some elements from a selection or
     - change the destination (click ... at the end of the row). The **Select destination** dialog will list destination targets by the type of the element selected to move (i.e. if it is a classifier element, classifiers will be listed in the Tree and List tabs). Otherwise only packages will be listed.
   - Click **OK** when you are done.
5. A notification appears, asking if you want to check dependencies among projects(see page 1280). Select **Yes**/**No** to continue moving elements.
6. After you move the elements, new main project and used projects versions are created and automatically committed to the server. Additionally, the user project is switched to the latest version.

Related pages

- Identifying Package Dependencies(see page 1280)
- Start using a project in your project(see page 1443)

Hiding packages

On this page

- Hiding packages(see page 1458)

- Hiding and unhiding used projects(see page 1459)
- Dedicated branch for stripped model(see page 1459)
- Hidden package changes (see page 1460)
- Model synchronization(see page 1460)

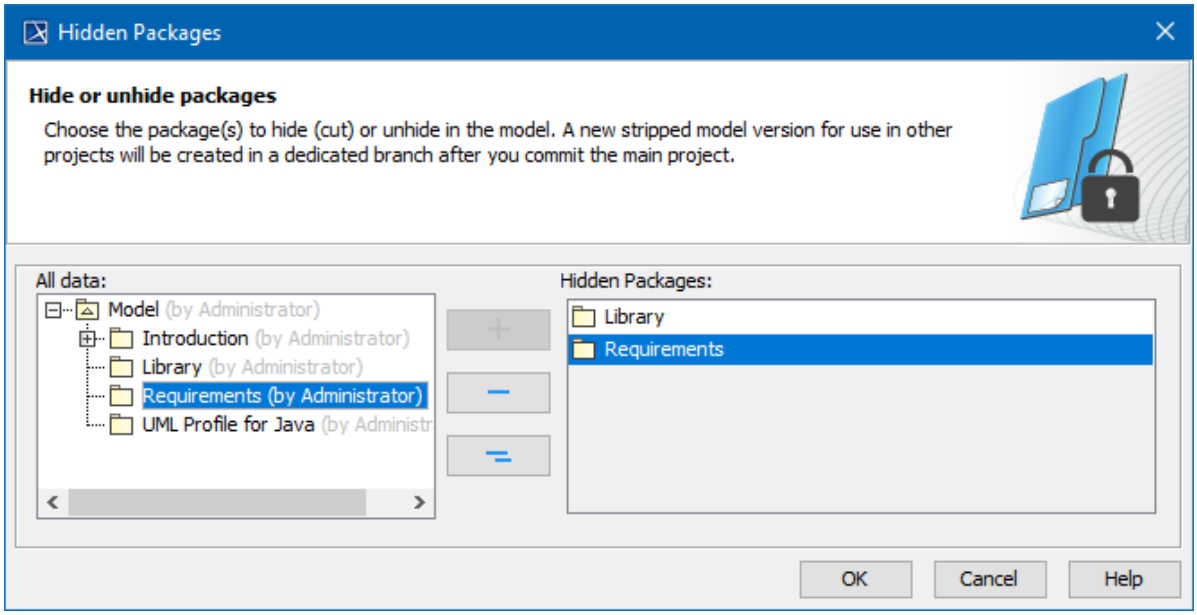
The Hiding packages utility lets you specify which model Packages,(see page 800) elements within the model packages, and Used projects(see page 1441) should be hidden. Once specified, the public version of a model will be created by cutting/deleting the hidden parts. You can strip your project to reuse only a particular part of the project in another project. To strip your project, you must hide packages you do not want to share.

Hiding packages

To hide packages

---

1. Open a server project.
2. Do one of the following:
   - Right-click the package you want to hide. On the shortcut menu, click **Project Usages** > **Hide Packages**.
   - On the main menu, click **Collaborate** > **Hidden packages**.
3. The **Hidden Packages** dialog opens. From the **All Data** side, select the packages you want to hide and add them to the **Hidden Packages** side. Move the selected package back to the **All Data** side to unhide it.



4. Click **OK** when you are done.
5. The package usage checking(see page 1192) is suggested. Usage checking may be time-consuming; however, it helps to identify elements that are related to the soon-to-be hidden parts. If any issues are found, they are displayed and described in the **Package Usage window**(see page 1199).
6. After you complete or skip Usage checking, you will see a notification that hiding the selected elements will be completed only after committing your project.
7. Commit (see page 1420)the project.
8. The stripped model version is created to be available to use in other projects. Hidden packages are marked with a lock sign in the Model Browser(see page 29).

Hiding and unhiding used projects

You can hide/unhide a used project the same way as other packages. When hiding a used project, you cannot hide separate used project parts: you need to hide the whole used project. After hiding is done, a dedicated branch is created with the stripped model version, where all the elements of the used project are hidden from the Containment tree due to being unused, and all references to these elements are cut from other model places.

To hide an indirectly used project, you need to hide it in the directly used project first.

> **Example**
> Let's say you have a project A that uses a project B, and this project uses a project C. You want to hide the project C inside the project A. In this case, you need to hide it in the project B directly first and then use the public branch of the project B.

Indirectly used projects can also be hidden/unhidden automatically. Typically, this happens when the status of the used project changes:

- when the used project is hidden/unhidden
- when the used project is updated
- when the used project is removed
- when the new used project is used
- when used or using projects are merged
- when the used project is exported
- when the used project is imported

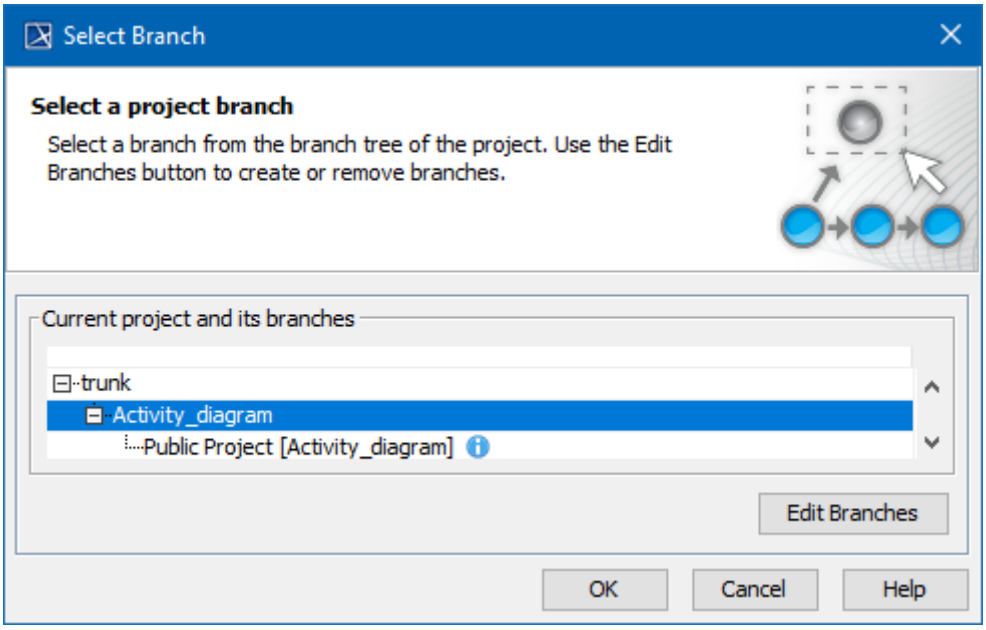
To hide an indirectly used project automatically, you need to hide all directly used projects that use the specific indirectly used project.

> **Example**
> Let's say you have a project A that uses a project B, and this project uses a project C. You hide the project B and then the project C is hidden as well; however, if you add a new usage D that also uses C, the hidden usage of the project C is automatically unhidden because not all directly used projects using the project C are hidden in this case.

Dedicated branch for stripped model

A stripped model is stored in a dedicated branch created when element hiding is performed for the first time. By default, this branch is named *Public Project*. You can rename the branch(see page 1375) if you need to.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Project with it`s Public Project branch. The public project now can be reused in another project.

If all packages from the model are unhidden, the dedicated branch will be deactivated.

If the main project is using a public branch of another project and that branch gets deactivated, you will be informed about this change with a warning icon in the main project with a warning icon in:

- Containment tree (under Project Usages root),
- Project Usages dialog,
- Advanced Project Usages Configuration dialog.

> The state of the used project is checked at the same time interval as checking for a new project version in the background, or by clicking the Project Usages Update button.

Hidden package changes

After a package or a used project is successfully hidden or unhidden, an icon with a lock on a package (or used project) is shown in a Difference viewer.

The icon with a lock on a package (or on a used project package) is also shown in:

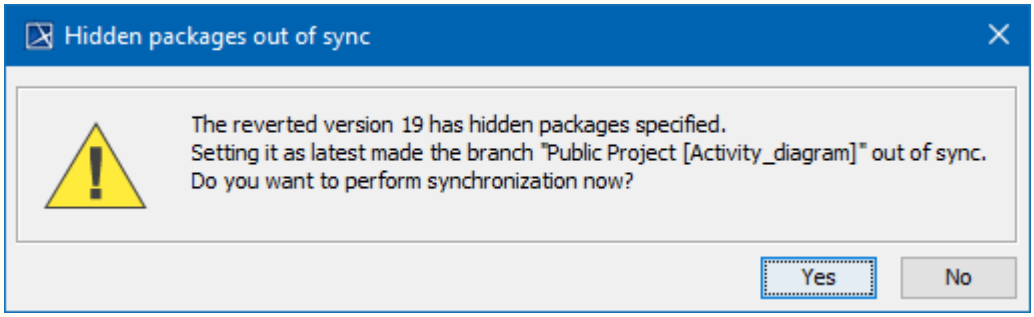
- Upcoming Changes browser panel,
- Local changes browser panel,
- Commit dialog,
- Content History panel,
- Element History.

Model synchronization

A stripped model is read-only. It can only be used for reuse in other projects, not for editing. However, you can edit the main project. On committing, checking is performed to see if the changes affect hidden and public parts of the model, and model synchronization is initialized. In the **Project Options** dialog(see page 214), you can define the model synchronization mode. The default mode is

Manual; that is, after committing project changes, you will be notified about local changes affecting a public model stored in a dedicated branch. You can then decide either to synchronize changes or not.

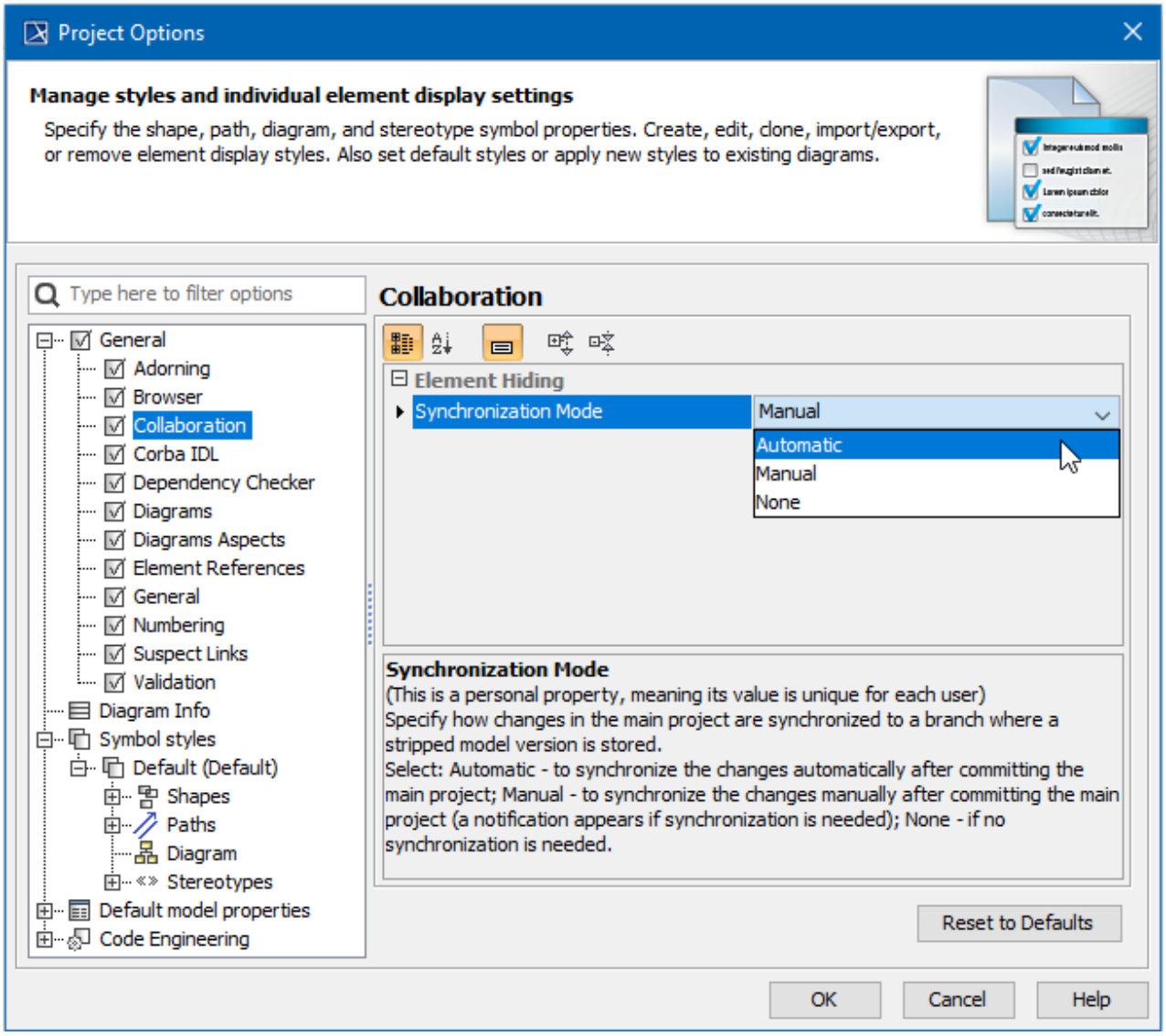
The server project with hidden packages (and public branches) might get out of sync with the current version of the model in the public branch after set as the latest functionality in the **History** dialog is used. Therefore, you will be asked to perform hidden package synchronization.



To select a synchronization mode

1. On the main menu, click **Options** > **Projects**. The **Project Options** dialog opens.
2. In the list of option groups, select **General** and then select **Collaboration** in the list of options.
3. For the option **Synchronization Mode**, specify one of the following values:
   - **Automatic** - to synchronize changes automatically on committing the project changes.
   - **Manual** (default) - to synchronize changes upon your decision.
   - **None** - do not perform any synchronization.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Selecting a synchronization mode to specify how changes in the main project are synchronized to a branch where a stripped model version is stored

Related pages

- Branching projects(see page 1375)
- Identifying Package Dependencies(see page 1280)
- Committing changes to Teamwork Cloud(see page 1420)

Managing Teamwork Cloud users

Managing Teamwork Cloud users includes:

- Creating and modifying user accounts
- Creating roles and assigning them to the users
- Assigning the users to projects

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

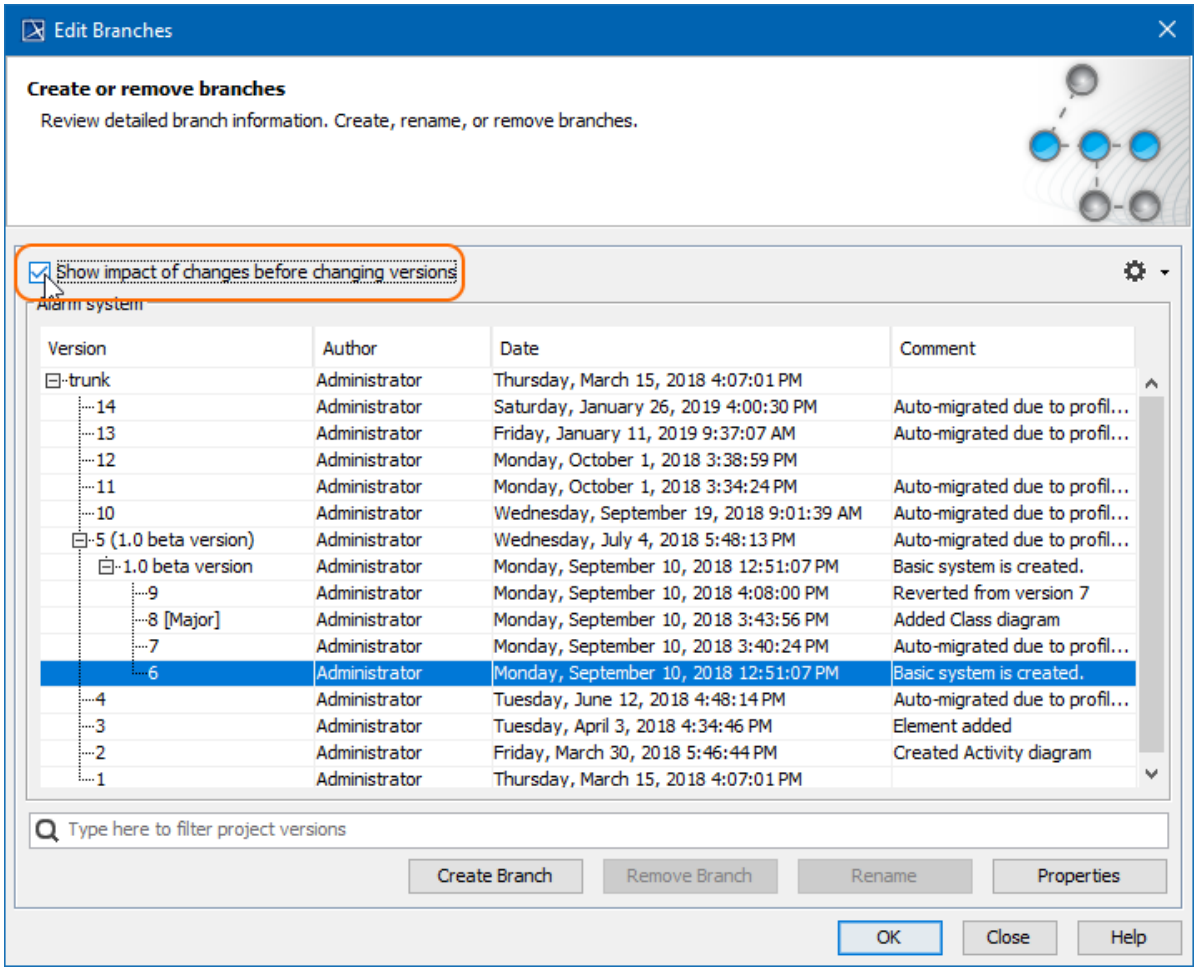
These tasks can be performed only in Teamwork Cloud Administrative Console. For more information, see Users Application[324] and Roles Application[325] in Teamwork Cloud documentation(see page 1462).

Used Project Switch Impact Analysis

The used project switch impact analysis enables you to review impacting changes before switching a used project version. In other words, this tool enables early insights over version changes that would have an impact on the main project once the switch is complete. You can use the analysis to inspect impacting changes at different granularity levels, allowing you to see the impact on element-to-element usages or to dive deeper into specific affecting/affected element properties.

To invoke the used project switch impact analysis

1. Right-click the used project version you want to change. The shortcut menu appears.
2. Click **Project usages > Change version**. The **Edit branches** dialog opens.
3. Select the **Show impact of changes before changing versions** check box, as shown below.



4. Choose the used project version and click **OK**. The Impact analysis window opens shortly.

324 https://docs.nomagic.com/display/TWCloud190SP4/Users+Application
325 https://docs.nomagic.com/display/TWCloud190SP4/Resources+Application

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

You can also invoke the used project impact analysis window using the following method:

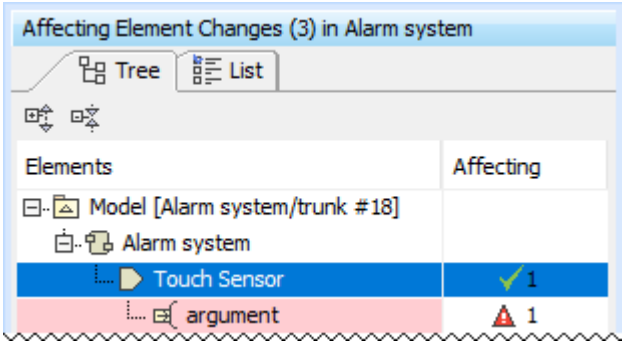
1. Go to the **Project usages** dialog, reached either from the context menu by choosing **Project usages > Project Usage Options...** or from the main menu **Options > Project usages**.
2. The **Edit Branches** dialog opens. Select **Show impact of changes before changing versions** check box.
3. Select the used project version and click **OK**. The Impact analysis window opens.

When the tool finds any impacting changes to be caused by a used project version switch the **Impact analysis** window shows up. Otherwise, you will get a notification that there are no impacting changes and used project version will be changed without invoking the dialog.

Used project switch impact analysis window

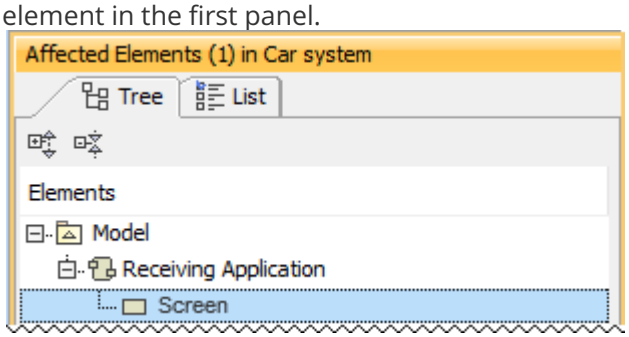
The **Impact analysis** window consists of three panels:

- **Affecting Element Changes** - the first panel, showing the changes in used project elements between the current used project version and the one that is to-be-switched to. Next to the panel title there are a number of elements that affect the main (using) project. This panel is separated into two columns:
  - The first column shows used project elements. It can be displayed as a tree or in list form.
  - The second column is named **Affecting**. This column shows how many elements are affected by the changed element and what kind of impact it might have (composition integrity errors or no errors).



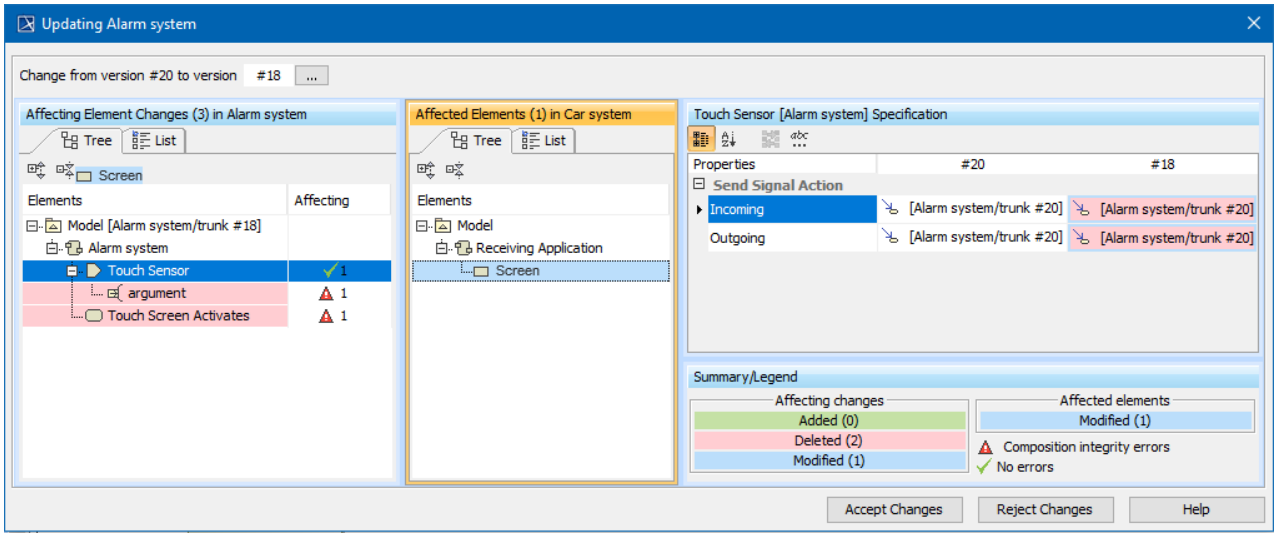
The **Affecting Element Changes** panel covers both direct and indirect usages, that is, if affecting change comes from an indirectly used project, it will be taken into account as well.

- **Affected Elements** - the second panel, which shows elements from the main project that are directly affected by element change(s) from the **Affecting Element Changes** panel. Next to the panel title are a number of elements that are affected in the main project by the selected element in the first panel.



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- **Element specification** - the third panel, showing the properties of selected element either from the **Affecting Element Changes** (changed element properties are shown) or **Affected Elements** (directly affected element properties are shown) panel. This panel works similarly to quick diff (see page 1433) and is separated into three columns:
- The first column shows the properties name.
- The second column shows the properties value from the current used project version when an element is selected in **Affecting Element Changes**. When an element is selected from **Affected Elements**, this column will show the current property value before the used project version switch.
- The third column shows properties value changes in the new used project version when an element is selected in **Affecting Element Changes**. When an element is selected from **Affected Elements**, this column will show the to-be property value after the used project version switch.



Impact analysis window for the *Alarm system* used project.

Derived properties are not considered in the impact analysis.
There are certain cases when element selection either in **Affecting Element Changes** or **Affected Elements** panels are not followed by a quick diff in the **Element Specification** panel:

- Whenever an element is deleted in the **Affecting elements** panel, its properties will no longer exist. Following this logic, the **Element Specification** panel will only show a notification indicating that the element is deleted and there are no properties to show.
- Whenever an affecting/affected element is selected in either of the first two panels against which a quick diff does not work, such as a diagram. In this case, you will be notified that there are no property changes for this type of element.

Toolbar

All three panels have toolbars to ease navigation through changes.

Icon	Name	Functionality
	Expand All	Click to expand all the elements in the tree.

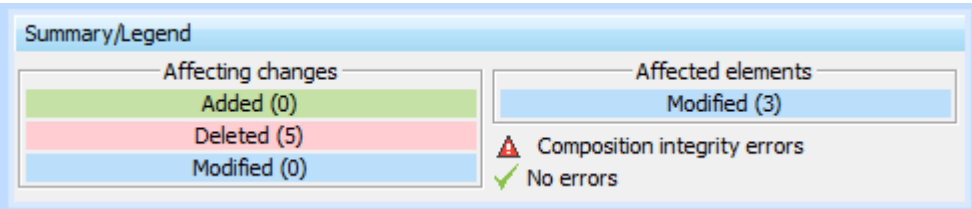
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

	Collapse All	Click to collapse all the elements in the tree.
Specific for Element specification panel toolbar		
	Categorized view	Click to show all property names in a category.
	Alphabetical view	Click to show all property names in alphabetical order.
	Compare values	Click to compare property values.
	Strip Multiline Text	Click to strip multiline text. If the text in the property cell (e.g., ToDo) covers more than five rows, it is finished with **...** showing only those five rows. The complete text is available in the tooltip or in the cell edit mode.

Affecting Element Changes and **Affected Elements** panels have a List view tab. Use this to list all changed elements. You can search for a specific element in these List views.

Summary/Legend

The Summary/Legend section introduces change types, the number of different affecting changes and the number of affected elements. It also includes marking composition integrity errors and a green tick to notify you that changes will not invoke any new composition integrity errors.



Understanding changes

The following definitions show the different change types.

Addition change
In the context of used project impact analysis, an addition change is treated to be the one when an element is restored from a recovered state to a normal one. Such element change is represented in the green rectangle in the **Affected elements** panel.

Deletion change
If an element has been removed, a deletion change occurs. The deletion change usually results in composition integrity errors, i.e., it causes the appearance of recovered elements, since the deleted

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

element is still used in the main project.

Modification change
If an element property has been modified, a modification change occurs.

Accept Changes or Reject Changes

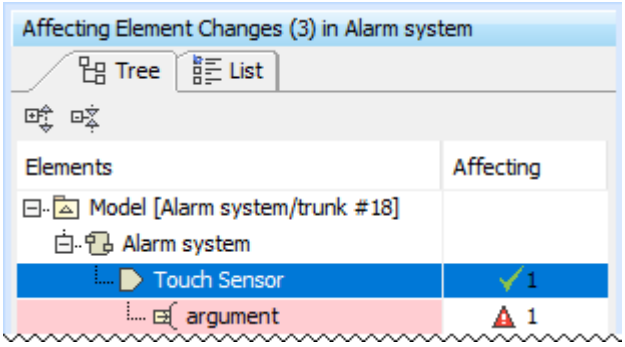
The **Accept Changes** button switches the used project version to the selected one. You will be warned if the selected version will cause or leave unresolved references.

Reject Changes restores the project to the state before invoking the Impact analysis with all of the local changes you had as well. The same applies for **X** (closing the dialog).

Used project switch impact analysis window

The **Impact analysis** window consists of three panels:

- **Affecting Element Changes** - the first panel, showing the changes in used project elements between the current used project version and the one that is to-be-switched to. Next to the panel title there are a number of elements that affect the main (using) project. This panel is separated into two columns:
  - The first column shows used project elements. It can be displayed as a tree or in list form.
  - The second column is named **Affecting**. This column shows how many elements are affected by the changed element and what kind of impact it might have (composition integrity errors or no errors).

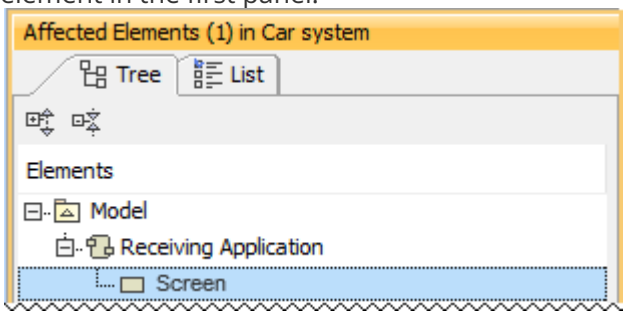


> The **Affecting Element Changes** panel covers both direct and indirect usages, that is, if affecting change comes from an indirectly used project, it will be taken into account as well.

- **Affected Elements** - the second panel, which shows elements from the main project that are directly affected by element change(s) from the **Affecting Element Changes** panel. Next to the panel title are a number of elements that are affected in the main project by the selected

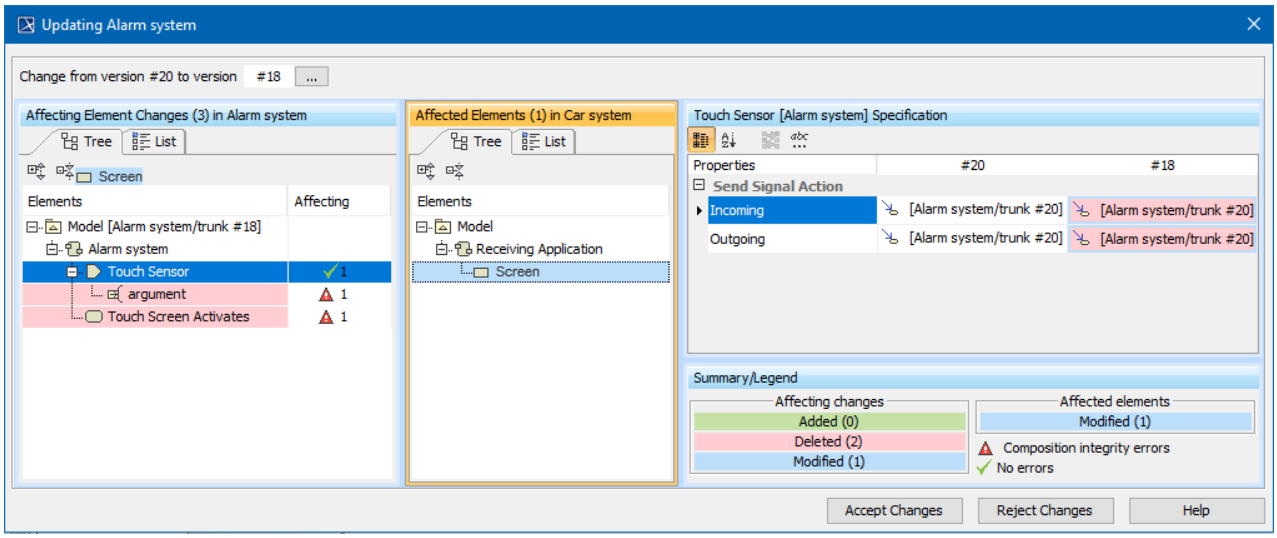
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

element in the first panel.



- **Element specification** - the third panel, showing the properties of selected element either from the **Affecting Element Changes** (changed element properties are shown) or **Affected Elements** (directly affected element properties are shown) panel. This panel works similarly to quick diff(see page 1433) and is separated into three columns:

- The first column shows the properties name.
- The second column shows the properties value from the current used project version when an element is selected in **Affecting Element Changes**. When an element is selected from **Affected Elements**, this column will show the current property value before the used project version switch.
- The third column shows properties value changes in the new used project version when an element is selected in **Affecting Element Changes**. When an element is selected from **Affected Elements**, this column will show the to-be property value after the used project version switch.



Impact analysis window for the *Alarm system* used project.

Derived properties are not considered in the impact analysis.
There are certain cases when element selection either in **Affecting Element Changes** or **Affected Elements** panels are not followed by a quick diff in the **Element Specification** panel:

- Whenever an element is deleted in the **Affecting elements** panel, its properties will no longer exist. Following this logic, the **Element Specification** panel will only show a notification indicating that the element is deleted and there are no properties to show.
- Whenever an affecting/affected element is selected in either of the first two panels against which a quick diff does not work, such as a diagram. In this case, you will be notified that there are no property changes for this type of element.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Toolbar

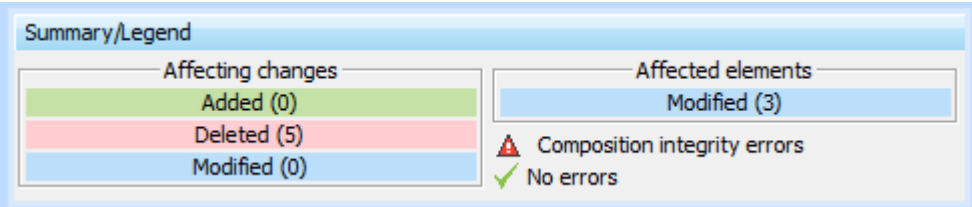
All three panels have toolbars to ease navigation through changes.

Icon	Name	Functionality
	Expand All	Click to expand all the elements in the tree.
	Collapse All	Click to collapse all the elements in the tree.
Specific for Element specification panel toolbar		
	Categorized view	Click to show all property names in a category.
	Alphabetical view	Click to show all property names in alphabetical order.
	Compare values	Click to compare property values.
	Strip Multiline Text	Click to strip multiline text. If the text in the property cell (e.g., ToDo) covers more than five rows, it is finished with **...** showing only those five rows. The complete text is available in the tooltip or in the cell edit mode.

Affecting Element Changes and **Affected Elements** panels have a List view tab. Use this to list all changed elements. You can search for a specific element in these List views.

Summary/Legend

The Summary/Legend section introduces change types, the number of different affecting changes and the number of affected elements. It also includes marking composition integrity errors and a green tick to notify you that changes will not invoke any new composition integrity errors.



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Understanding changes

The following definitions show the different change types.

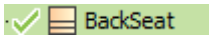
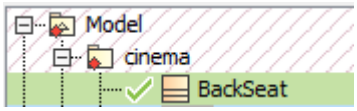
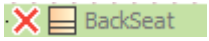
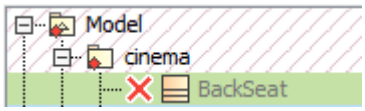
Addition change
In the context of used project impact analysis, an addition change is treated to be the one when an element is restored from a recovered state to a normal one. Such element change is represented in the green rectangle in the **Affected elements** panel.

Deletion change
If an element has been removed, a deletion change occurs. The deletion change usually results in composition integrity errors, i.e., it causes the appearance of recovered elements, since the deleted element is still used in the main project.

Modification change
If an element property has been modified, a modification change occurs.

Understanding element marking

When a change occurs on a single element, the element marking indicates the state of the change which can be either accepted or rejected.

Marking	Change state	Example
 (green tick before the element icon)	Accepted	 A new element had been created in one of the contributors and its addition was accepted.
 (red cross before the element icon)	Rejected	 A new element had been created in one of the contributors and its addition was rejected.

To understand the concept of change states, refer to Change states(see page 0).

Elements with resolved conflicting changes are decorated as shown in the following table. Owners of these elements are also decorated.

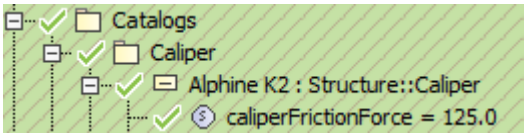
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Marking	Change applied by	Example
Pad:Pad (red diamond on the lower-left corner of the element icon)	System	 The element had a conflicting modification change which was resolved automatically.
Pad:Pad (yellow diamond on the lower-left corner of the element icon)	User	 The element had a conflicting modification change which was resolved by the user.

Recognizing addition changes

For the addition change description, refer to Addition change(see page 0).

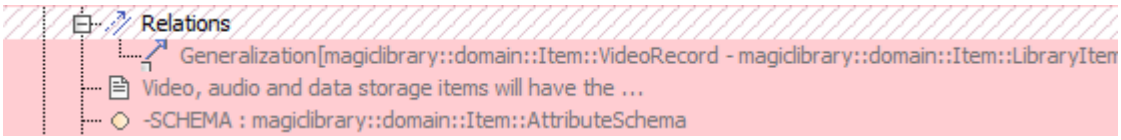
Added elements are highlighted in green background.



Recognizing deletion changes

For the deletion change description, refer to Deletion change(see page 0).

Deleted elements are highlighted in red background.

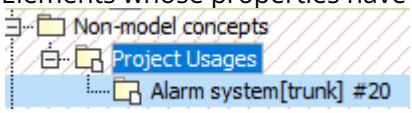


Recognizing modification changes

For the modification change description, refer to Modification change(see page 0).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Elements whose properties have been changed are highlighted in blue.



To see what properties have been modified, click the element and see changes in the **Specification** pan el.

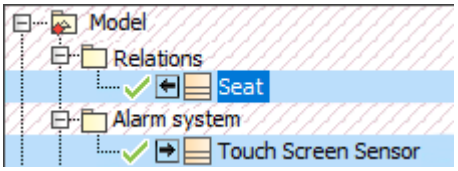
For more information about using the Specification panel, see Inspecting changes in the Change details panel(see page 1103).

Recognizing movement changes

For the movement change description, refer to Movement change(see page 0).

Elements with movement changes are highlighted in the same blue background that is used to highlight modification changes. Since movement changes are some kind of modification changes. Icon of an element with movement changes is represented with an arrow symbol on the left.

The following figure illustrates a movement change, when element X has been moved from package A to package B. The movement change had been accepted and element X became owned by package B.

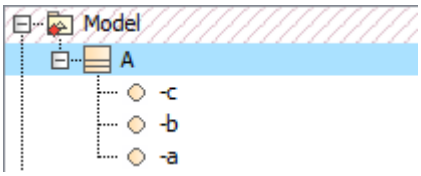


You can navigate from the element original location to the new one (and vice versa) using the commands on the element shortcut menu.

Recognizing order changes

For the order change description, refer to Order change(see page 0).

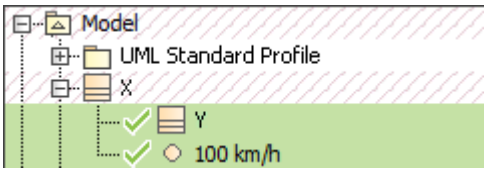
Elements with order changes are highlighted in the same blue background that is used to highlight modification changes, since order changes are some kind of modification changes.
After accepting the change, all the elements in the collection are reordered so that it would be the same as in one of the contributors.



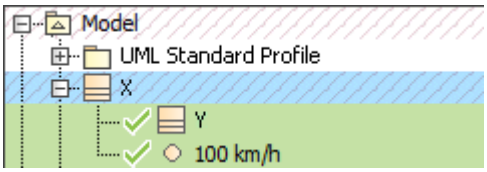
You can turn off the order changes detection. For more about it, see Turning on showing merged diagram annotations.(see page 0)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Recognizing elements with changed owned elements
Elements whose owned elements (in any nesting level) have changes are highlighted using red diagonals.



Modified elements with changed owned elements are highlighted in blue with red diagonals.



Accept Changes or Reject Changes

The **Accept Changes** button switches the used project version to the selected one. You will be warned if the selected version will cause or leave unresolved references.

Reject Changes restores the project to the state before invoking the Impact analysis with all of the local changes you had as well. The same applies for **X** (closing the dialog).

Used Projects Auto Update Plugin

The Teamwork Cloud Used Project Updater plugin is an internal modeling tool plugin. Users can set up the plugin to update all or selected used projects to the latest versions. With this plugin we are also introducing a wizard to help select used projects for update.

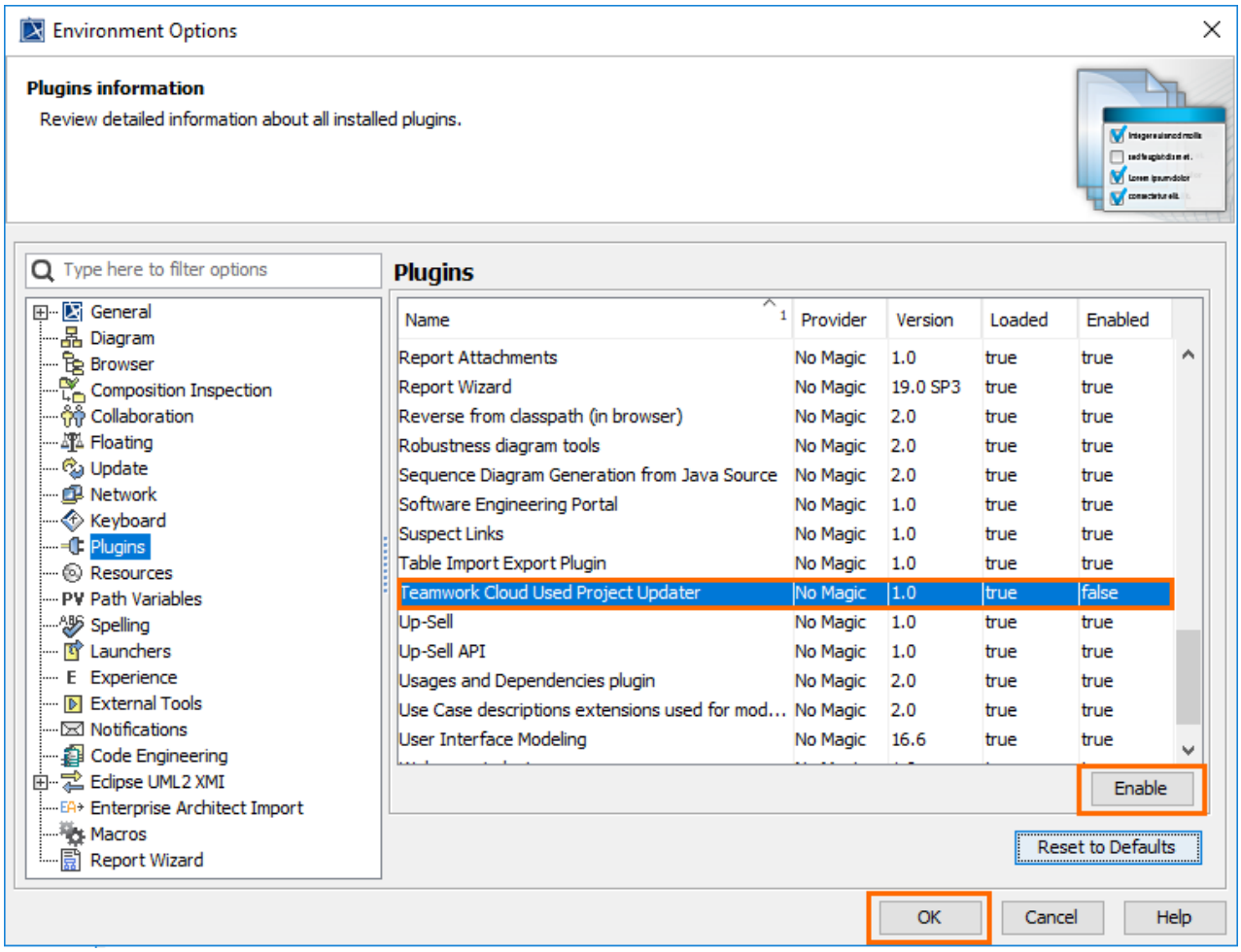
Projects within the update path will require these permissions: *Administer Resources*; *Edit Resource*; *Edit Resource Properties*; *Read Resource*. However, the target used project only needs the *Read Resources* permission.

An example
You have projects: A, B and C. These projects are in use as follows A->B->C (project A is using project B; project B is using project C). In order to update B project to the latest version you only need *Read Resource* on project B and *Administer Resources*, *Edit Resource*, *Edit Resource Properties* and *Read Resource* permissions on project A.
However, in order to update B and C used projects you need *Administer Resources*, *Edit Resource*, *Edit Resource Properties* and *Read Resource* permissions for projects A and B. You will need only *Read Resource* on project C.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To enable/disable the plugin

1. On the main menu, click **Options** > **Environment**. The **Environment Options** dialog opens.
2. In the options list, select **Plugins** and then the **Teamwork Cloud Used Project Updater** plugin**.** Do one of the following:
    - Set **Enabled** to *true* to enable the plugin.
    - Set to *false* to disable the plugin.



Enabling Teamwork Cloud Used Project Updater plugin

> **Create dedicated user**
> We recommend that you create a dedicated user, and use it in order to continue work with the Teamwork Cloud Used Project Updater plugin.

To set up a properties file do one of the following

- Using Used Projects Auto Update Wizard(see page 1475)
- Manually configuring properties file(see page 1477)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To update used projects

- Use command line to start updating used projects(see page 1478).

The plugin does not update used projects that are involved in cyclic dependencies.

Using Used Projects Auto Update Wizard

This section describes how to use the Used Projects Auto Update Wizard to configure and generate properties file.

Use dedicated user
After you enable the plugin, it is highly recommend that you log in to the Teamwork Cloud environment in the modeling tool with a dedicated user. This is the only way to ensure that all dedicated user permissions are checked and the properties file will be set correctly.

After the plugin is enabled in the **Project Usages** dialog you will see a new button: **Used Projects Auto Update Wizard** .

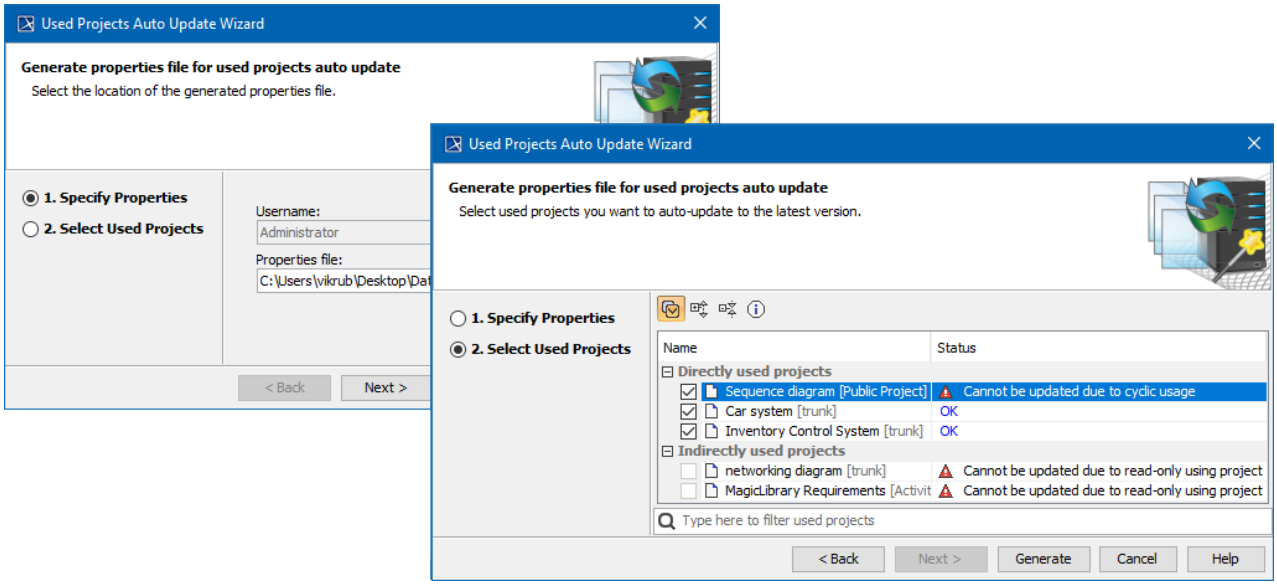
To open the Used Projects Auto Update Wizard

1. Make sure the plugin is enabled.
2. Login with the dedicated user, and only then open the project in order to continue using the Used Projects Auto Update Wizard.
3. To open a wizard, go to the **Project Usages** dialog.
4. Click on the button **Used Projects Auto Update Wizard**. The wizard dialog opens.

To configure a properties file with the Used Projects Auto Update Wizard

1. In the opened **Used Projects Auto Update Wizard** dialog, you can see the username you are logged in with. This user will be set up for the properties file to automatically update used projects.
2. In the **Properties file** field, you need to select a file and location to overwrite or save the new properties file that will be generated for the plugin.
3. After you have selected a properties file, click **Next**.
4. Select directly and indirectly used projects. Values listed in the Status column show whether or not a used project can be updated.
5. After you select the used projects, click **Generate** to create the properties file that will be used later by a command line tool.(see page 1478)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Used Projects Auto Update Wizard dialogs.

To see where the selected used project is used, click the **Project Usage Information** button. This will help to find update paths. An *update path* is a path of projects that must be updated in order to update a certain used project.

Possible errors and warnings with explanations

Warning: Causes version conflict with at least one used project.
This warning indicates that in this project, there is at least one other unselected used project that is using the same project. In order to achieve consistency on the model, we recommend updating all used projects that use this particular project.

ERROR: Cannot be updated due to insufficient permissions.
This error indicates that the user who is logged in does not have enough permissions to update this particular project. Please add required permissions for the user in order to update this project.

ERROR: Cannot be updated due to read-only using project
This error indicates that the used project is being used in a read-only project (e.g. public branch); thus, we can not commit a new version of a read-only project in order to update the selected project. Please use this project directly or in another project with a branch for which the user has read-write permissions.

ERROR: Cannot be updated due to cyclic usage.
This error indicates that the used project has cyclic dependencies between projects. Please remove any used projects from this used project that would cause a cycle between projects.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

ERROR: Cannot be updated due to missing update path.
This error can appear on indirectly used projects due to a missing update path. An *update path* is a path of projects that must be updated in order to update a certain used project. The following example explains this term:

An example
A Main Project uses project A. Project A uses Project B. In order to update Project B in the Main Project, we are required to update Project B in Project A first and then update Project A in the Main Project. Thus, the update path for Project B is: Main Project > Project A.

Manually configuring properties file

This section explains how to correctly configure a Used Projects Auto Update Plugin properties file.

To configure the properties file for updating used projects

- In the *<modeling tool installation directory>/plugins/com.nomagic.magicdraw.twcusageupdater* directory, open the *sample.properties* file (or any generated properties file) through a wizard.
- Specify the properties values listed below in the table (or leave the default values if they do not need to be changed).

Properties	Description
twc.server	TWCloud server hostname or IP (with port if it is not default)
twc.user.name	TWCloud username
twc.user.password	TWCloud user hashed password (the only way to get a hashed password is to generate it through Used Projects Auto Update Wizard(see page 1475) first)
twc.project	TWCloud main project URI*
twc.project.password	Project hashed password if there is password protection (the only way to get a hashed password is to generate it through Used Projects Auto Update Wizard(see page 1475) first)
twc.used.projects	List of used projects URI (with hashed passwords if project is password protected. Pattern <URI>,<URI>#<password>,<URI>. The only way to get a hashed password is to generate it through Used Projects Auto Update Wizard(see page 1475) first)
twc.use.ssl	Use SSL flag

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Updating a used project with a command line

This section outlines how to update used projects using a command line.

To update used projects from the command-line interface

1. Move the properties file to the plugin directory *<modeling tool installation directory>/plugins/com.nomagic.magicdraw.twcusageupdater*
2. Open the command-line interface.
3. Go to the *<modeling tool installation directory>/plugins/com.nomagic.magicdraw.twcusageupdater* directory containing the executive file for the used projects update.

> **Executive file**
> The executive file is the corresponding .sh or .bat file depending on your operating system.

To update used projects using a single properties file, type the following command in the command line

- On Windows:

```
update.bat -properties sample.properties
```

- On OS X or Linux

```
./update.sh -properties sample.properties
```

To update used projects using multiple properties files, type the following command in the command line

- On Windows:

```
update.bat -properties first.properties second.properties third.properties
```

- On OS X or Linux:

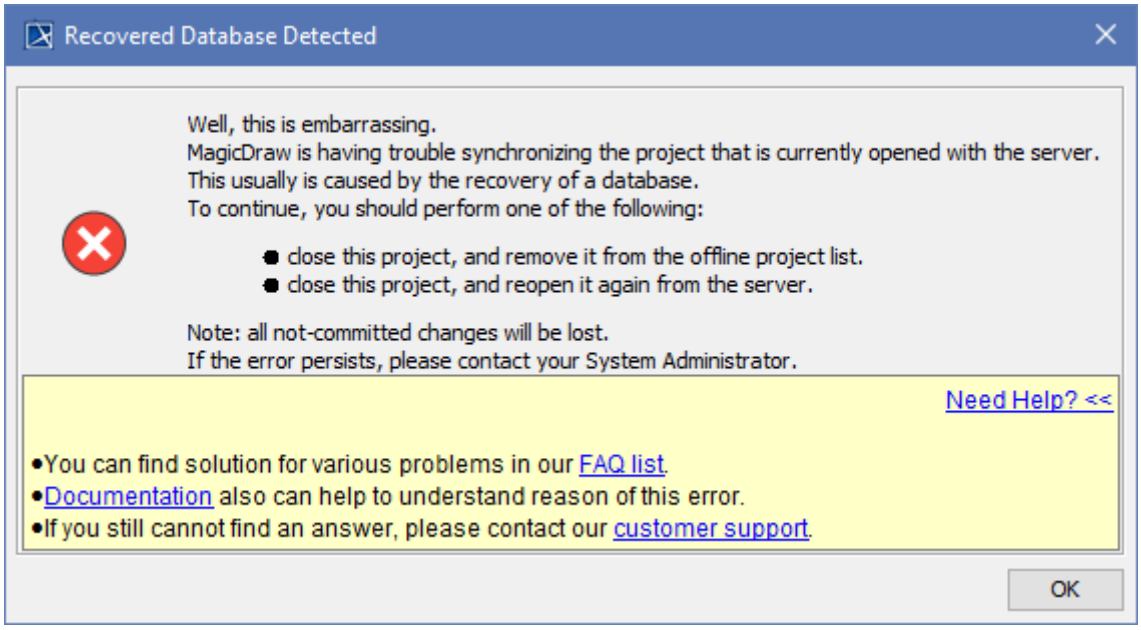
```
./update.sh -properties first.properties second.properties third.properties
```

Troubleshooting

Recovered Database Detected

If the current MagicDraw project version conflicts with the project version committed to the server earlier, and a recovered database is detected, then the following error message appears. Follow the instructions in the message or contact your system administrator to solve the issue.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The error message.

Using Teamwork Server

Teamwork Server makes it easy to exchange data directly in the context of your work. It provides a central repository for storing any kind of models. Using Teamwork Server, team members can access, review, or modify the same model at the same time. It supports importing, exporting, updating, branching or comparing in-server-stored models.

When you are connected to Teamwork Server, you must have certain permissions to work with server projects. The following pages explain the most important steps you need to take when working with server projects:

1. Create a server project or add a non-server project to the Teamwork Server. Open the server project.
2. Lock the elements you want to edit and edit them in the open project.
3. Select the model elements to unlock and commit the project.
4. Create categories for visual grouping of server projects.

If you want to lock an element for editing, the application will check for the permissions you have and the availability of the element.

Before committing elements, you can unlock the elements you have previously locked. Committing to Teamwork Server will create a new project version.

Adding projects to Teamwork Server

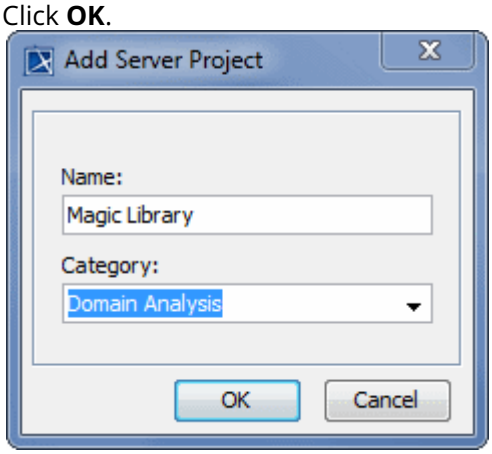
Projects can be added to Teamwork Server in one of the following ways:

- By creating a project in Teamwork Server from scratch(see page 1480)
- By adding a local project to Teamwork Server(see page 1480)

- By creating server projects in IDE integration(see page 1482)

To create a project in Teamwork Server from scratch

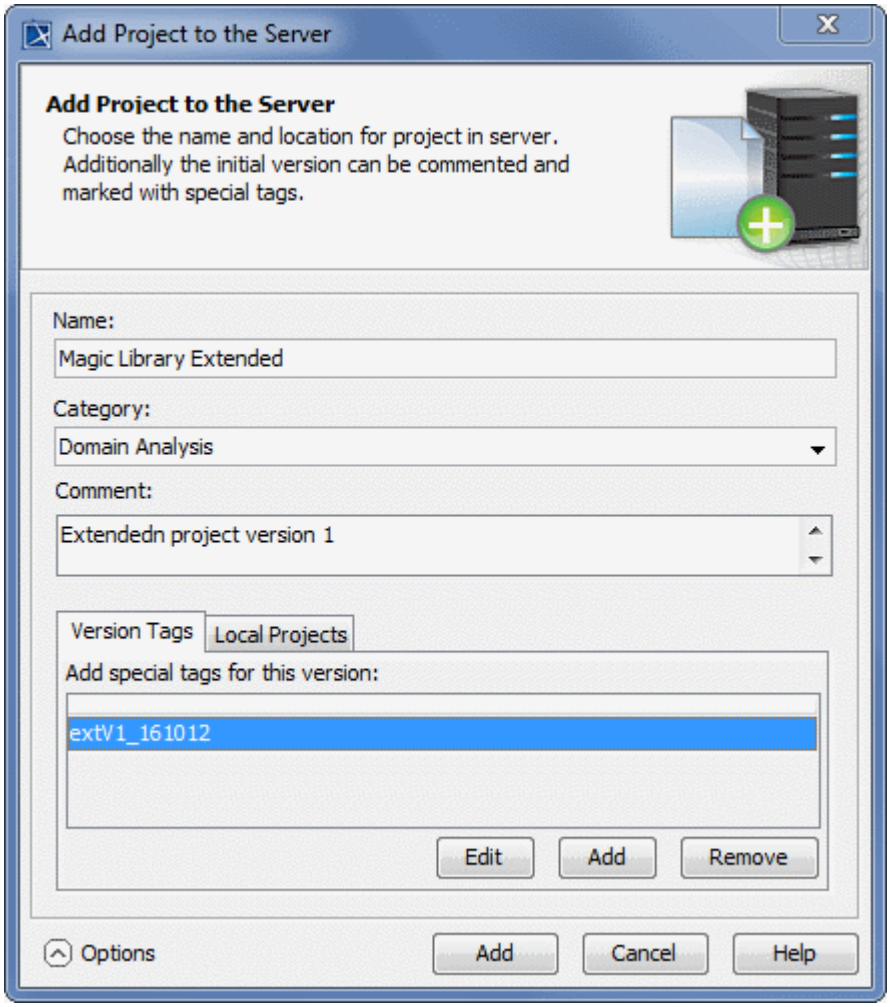
1. From the **Collaborate** menu, choose **Projects**.
2. Click the **Add** button. The **Add Server Project** dialog opens.
3. Type a new project name.
4. Either select an existing category or type a new category name.
5. Click **OK**.



To add a local project to Teamwork Server

1. Log in to Teamwork Server(see page 1363).
2. Open the project you want to add to Teamwork Server.
3. From the **Collaborate** menu, choose **Add Project to Server**. The **Add Project to the Server** dialog opens.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



4. Enter a different project name for Teamwork Server, if needed.
5. Either select an existing category or type a new category name.
6. Type your comments.
7. Click **Options** if you need to enter more details about the project. The dialog extends with the **Version Tags** and **Local Projects** tabs.
   - To add a tag, click **Add**. The **Enter Tag** dialog appears, where you can type the tag for your project version.
   - If your project uses local projects, the **Local Projects** tab lists all those projects used in the project you are adding to the server. You can change used projects' names on the Server. From the **Action** drop-down list, select what you want to do with local used projects when adding a project to Teamwork Server:
     - **Add to Server** adds the entire selected used project to Teamwork Server. Select a used project name on the server from the Name on Server drop-down list or type a new one.
     - **Leave as Local Project** leaves the selected used project on a local workstation.

       Standard profiles provided by the modeling tool can take only this action.
     - **Use Server Project** uses a local used project or its branch from Teamwork Server. You can select the entire used project (trunk) or a branch.
8. Click the **Add** button.

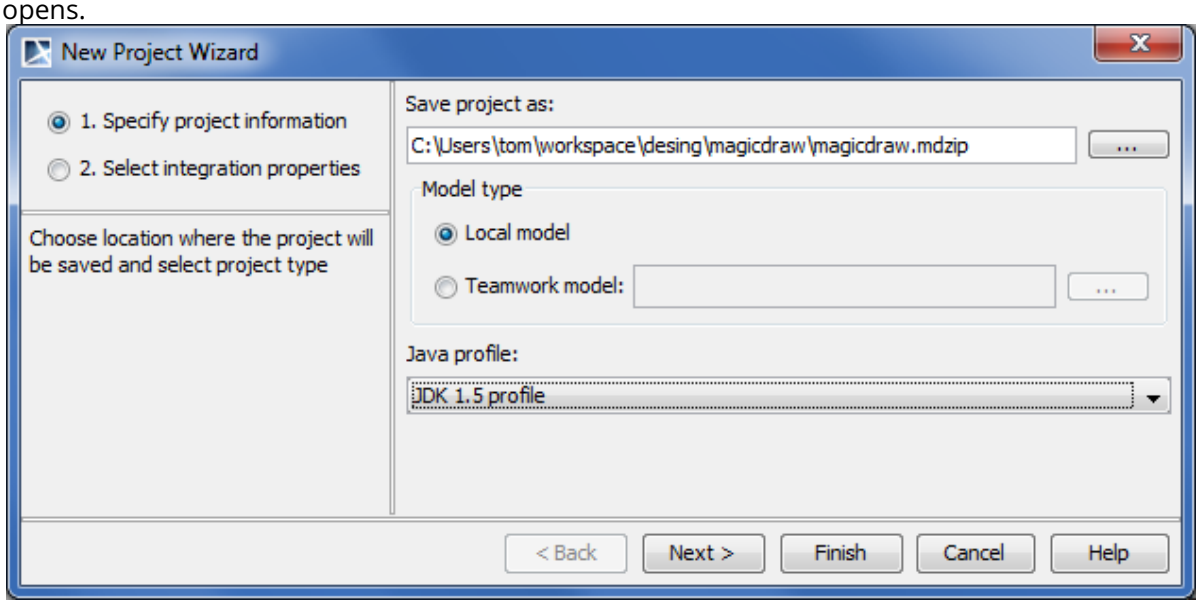
Watch this demo to see how to add projects to Teamwork Server.

Sorry, the widget is not supported in this export.
But you can reach it using the following URL:

https://www.youtube.com/watch?v=4Fw1ycgw_Xw

To create server projects in IDE integration

1. Start a new project in the IDE integration you are working on. The **New Project Wizard** dialog opens.



2. Select **Teamwork model** and click the ... button to login to Teamwork Server.
3. After log in, select an existing project or add a new one. The model from the IDE source will be updated in the selected project. Click **Next**.
4. Specify integration properties and click **Finish**. The server project opens in the modeling tool.

Locking a model for editing in Teamwork Server

On this page

- Representation of locked elements[326]
- Locking elements and diagrams[327]

[326] https://docs.nomagic.com/display/MD190SP4/Locking+model+for+edit+in+Teamwork+Server#LockingmodelforeditinTeamworkServer-Representationoflockedelements
[327] https://docs.nomagic.com/display/MD190SP4/Locking+model+for+edit+in+Teamwork+Server#LockingmodelforeditinTeamworkServer-Lockingelementsanddiagrams

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Locking symbol styles[328]
- Locking project usages[329]
- Locking for reverse engineering[330]
- Locking the project structure[331]

You must lock the part of the model you are going to edit so that other users cannot edit it at the same time. Locking is allowed if that part of the model is not locked by another user, because an item can be only be locked by one user at a time.

The following table explains what is editable after you lock a single element or diagram, diagram with elements, and so on.

When you lock...	You can...
Element	Edit element specification properties (in the Specification window(see page 913))
Diagram	Edit: • Diagram specification properties (in the Specification window(see page 913)) • Diagram properties (in the **Diagram Properties** dialog) • Symbol layout on the diagram pane • Symbol properties (in the **Symbol Properties** dialog)
Diagram with elements	Edit: • Diagram specification properties (in the Specification window(see page 913)) • Element specification properties (in the Specification window(see page 913) and on the diagram pane) • Diagram properties (in the **Symbol Properties** dialog) • Symbol properties (in the **Symbol Properties** dialog) • Symbol layout on the diagram pane

82 https://docs.nomagic.com/display/MD190SP4/Locking+model+for+edit+in+Teamwork+Server#LockingmodelforeditinTeamworkServer-Lockingsymbolstyles
92 https://docs.nomagic.com/display/MD190SP4/Locking+model+for+edit+in+Teamwork+Server#LockingmodelforeditinTeamworkServer-Lockingprojectusages
83 https://docs.nomagic.com/display/MD190SP4/Locking+model+for+edit+in+Teamwork+Server#LockingmodelforeditinTeamworkServer-Lockingforreverseengineering
33 https://docs.nomagic.com/display/MD190SP4/Locking+model+for+edit+in+Teamwork+Server#LockingmodelforeditinTeamworkServer-Lockingtheprojectstructure

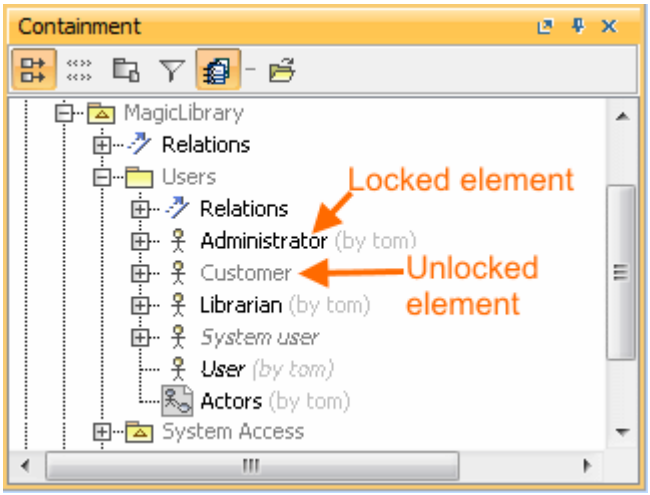
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

When you lock...	You can...
Symbol styles	Edit properties of symbol styles (in the **Project Options** dialog)
Project usages	• Import used projects as packages into the main project • Stop using projects • Change versions of used projects

Representation of locked elements

In the **Containment** tree, names of elements are presented in two colors:

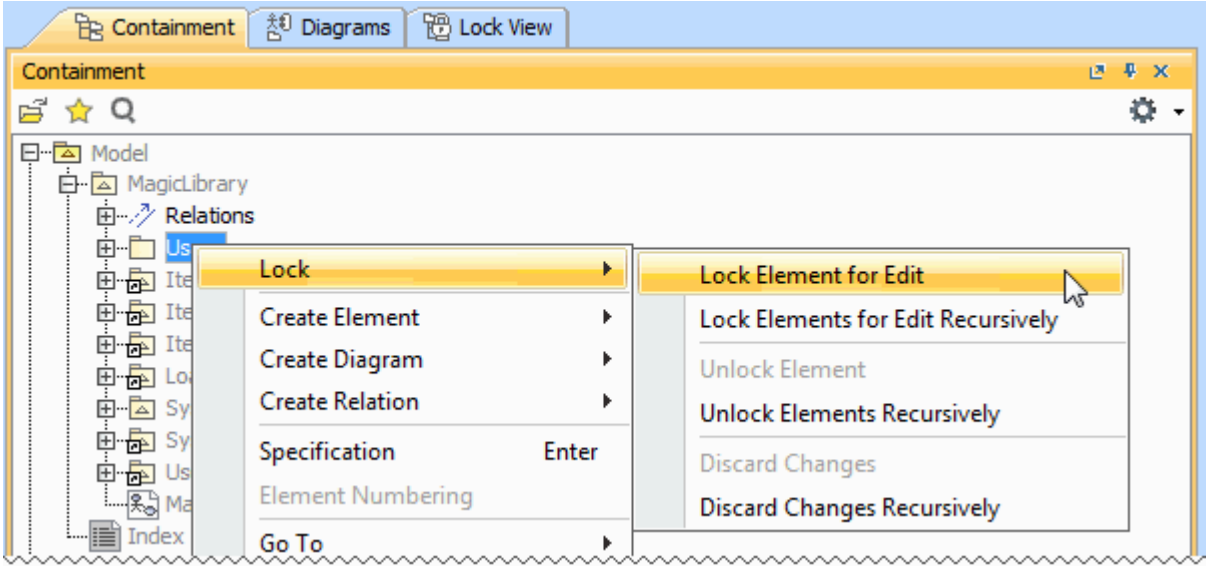
- Locked elements appear in black, meaning they are editable.
- Unlocked elements appear in grey, meaning they are not editable.



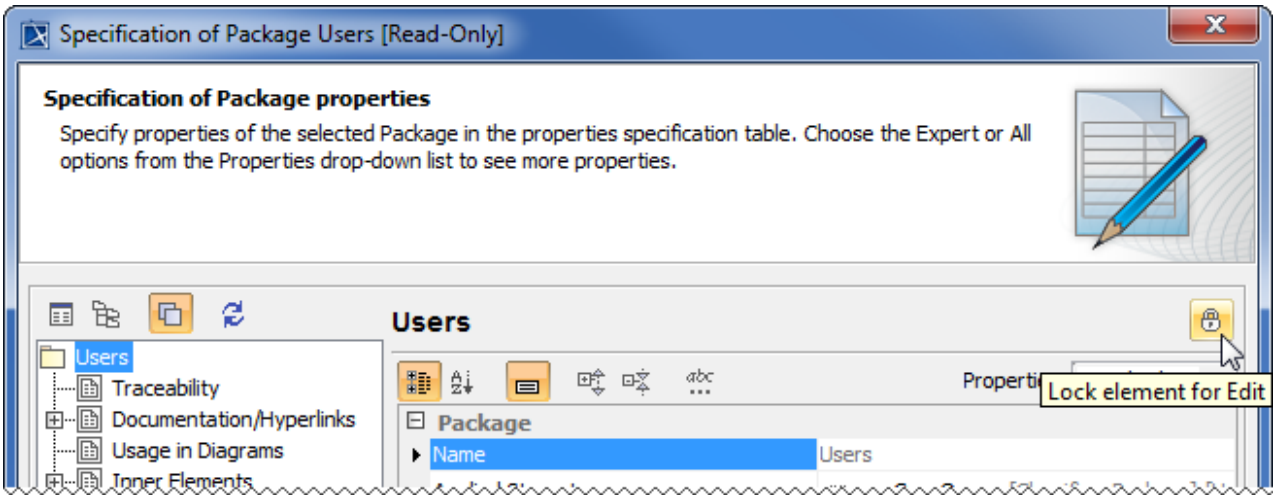
Locking elements and diagrams

Elements and diagrams can be locked by using the commands from their shortcut menu.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Elements can also be locked by clicking the Lock element for Edit button on their Specification window(see page 1482).



The following table provides suggestions on which locking commands to use for various purposes.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

If you need to edit...	Do the following
Specification properties of a single element	1. Right-click either: • This element in the Model Browser. • A symbol of this element on a diagram pane. 2. From the shortcut menu, select **Lock** > **Lock Element for Edit**. OR 1. Double-click either: • This element in the Model Browser. • A symbol of this element on a diagram pane. 2. In the Specification window, click the Lock element for Edit button .
Specification properties of both an element and the elements it contains (owns) Let's say a project has a Package *Analysis*, which contains two inner Packages: *Design* and *Implementation*. Each inner Package contains elements. Lock the Package *Analysis* recursively to edit all the elements in this Package.	1. Right-click either: • This element in the Model Browser. • A symbol of this element on a diagram pane. 2. From the shortcut menu, select **Lock** > **Lock Elements for Edit Recursively**
At least one of the following: • Diagram representation properties (including diagram layout) • Symbol properties of elements represented on that diagram pane	1. Right-click this diagram in the Model Browser. 2. From the shortcut menu, select **Lock** > **Lock Diagram for Edit**. OR 1. Right-click a free space on this diagram pane. 2. From the shortcut menu, select **Lock Diagram** > **Lock Diagram for Edit**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

If you need to edit...	Do the following
At least one of the following: • Diagram specification properties • Diagram representation properties (including diagram layout) • Properties of elements represented on that diagram pane • Symbol properties of elements represented on that diagram pane	1. Right-click this diagram in the Model Browser. 2. From the shortcut menu, select **Lock** > **Lock Diagram Content for Edit**. OR 1. Right-click a free space on this diagram pane. 2. From the shortcut menu, select **Lock Diagram** > **Lock Diagram Content for Edit**.
At least one of the following: • Symbol properties of a single element represented on a diagram pane • Position of the symbol on the diagram pane	1. Right-click the symbol of that element on a diagram pane. 2. From the shortcut menu, select **Lock** > **Lock Element for Edit in Diagram**.

Locking symbol styles

Symbol styles can be locked via the **Lock View** tab or the **Project Options** dialog.

To lock symbol styles via the **Lock View** tab

1. In the **Lock View** tab on the Model Browser, expand *Project Options*.

   Where is the Lock View tab?

   It is the last tab on the Model Browser. If the tab is closed, you can simply open it by selecting **Collaborate** > **View Locked Elements**.

2. Right-click *Symbol Styles* and from the shortcut menu select **Lock Symbol Styles for Edit**.

To lock symbol styles via the **Project Options** dialog

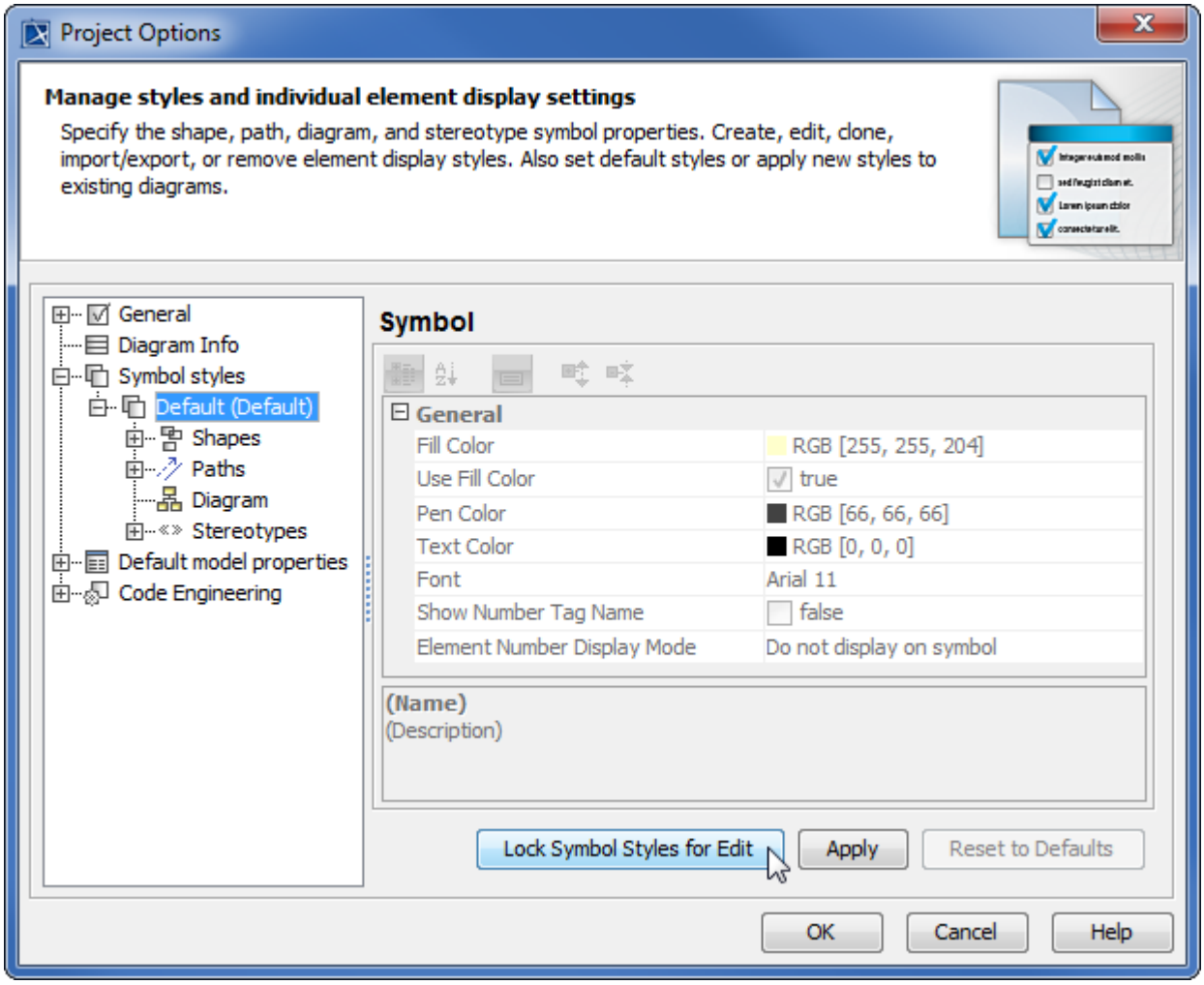
1. Do either:
   - Select **Options** > **Project**.
   - In the **Lock View** tab on the Model Browser, right-click *Project Options* and from the shortcut menu, select **Symbol Styles**.

     Where is the Lock View tab?

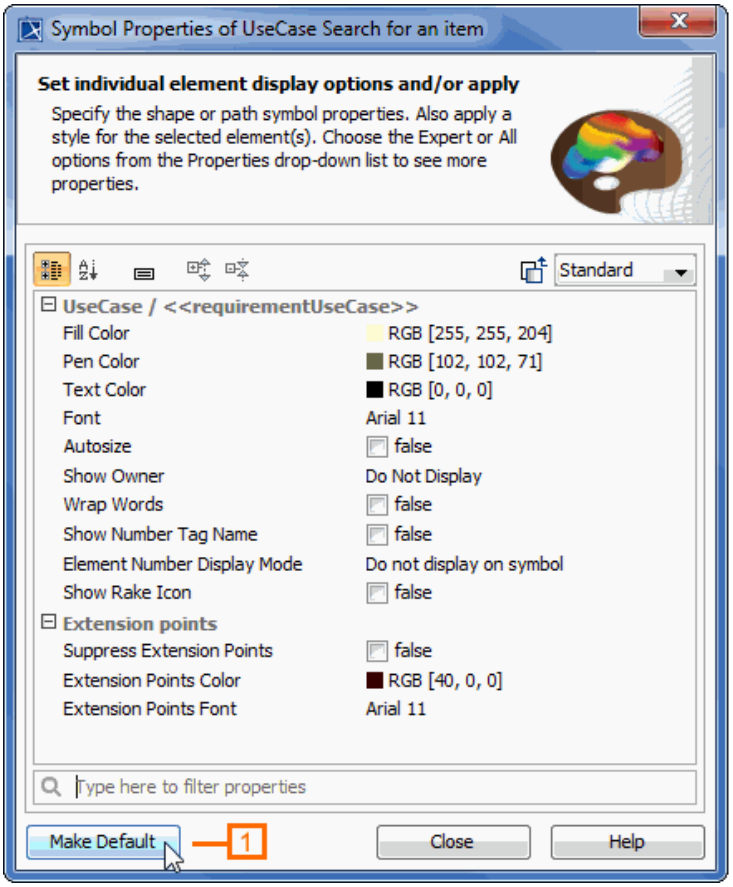
     It is the last tab on the Model Browser. If the tab is closed, you can simply open it by selecting **Collaborate** > **View Locked Elements**.

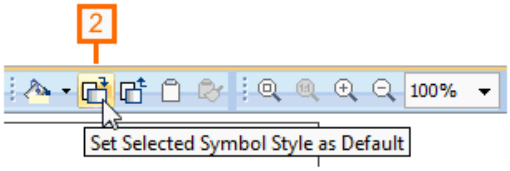
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

2. In the **Project Options** dialog, click the **Lock Symbol Styles for Edit** button.



As a result, symbol styles become editable. Additionally, the **Make Default** button (1) in all **Symbol Properties** dialogs and the Set Selected Symbol Style as Default button (2) on all the diagram toolbars become available.





Locking project usages

To lock a used project

---

1. Do either:
    - In the **Lock View** tab on the Model Browser, right-click *Project Structure* and select **Options**.
    - From the **Options** menu, select **Project Usages**.
2. In the opened **Used Projects** dialog, select a used project on the left side of the dialog and click the **Lock** button.

As a result, you can import this used project as a package to the main project, stop using it, or change its version.

Locking for reverse engineering

To reverse a code to the model that was already reversed, you need to lock elements that store code engineering information.

To lock elements for reverse engineering

1. In the **Code engineering sets** branch, select a code engineering set or code engineering element you want to reverse.
2. On the selected items shortcut menu, click **Reverse**. The **Reverse Options** dialog opens.
3. In the **Reverse Options** dialog, set the required options and click **OK**.
4. The **Lock Elements** dialog opens. Select elements you want to lock and click **OK**.
5. Depending on the options you have selected in the **Reverse Options** dialog, you may need to perform some more steps. Please follow the instructions provided on your screen.

Locking the project structure

There is additional model integrity protection for a collaborative environment. All project structure modifications can only be done by one person at a time. To prevent simultaneous structure modification, project structure locking is realized.

If you are in a collaborative environment and working simultaneously on the same project, the following operations cannot be performed in parallel:

- Modifying shared packages. Adding or removing shared packages is now controlled by the project structure lock.
- Importing new project features. Some plugins may install additional features, such as specific options or code engineering abilities. An opened project structure is locked automatically, and the project cannot be opened by other users.
- Upgrading project versions. The project structure is locked when the project is migrated from an earlier version to a later version (for example, from version 17.0 to 17.0.1). While one user is upgrading project version, the project is in read-only mode for other users until the first user commits the project.

The project structure is automatically locked to ensure correct model integrity when you perform one of these operations. While the project structure is locked, other users may continue getting error messages that may prevent their working with the model. The project structure is unlocked automatically when the project is committed. Alternatively, you can unlock the structure manually.

The project structure can be locked via the **Lock View** tab or the **Project Properties** dialog.

To lock a project structure via the **Lock View** tab

1. In the **Lock View** tab on the Model Browser, rigt-click *Project Structure* and select **Lock Project Structure**.

   Where is the Lock View tab?

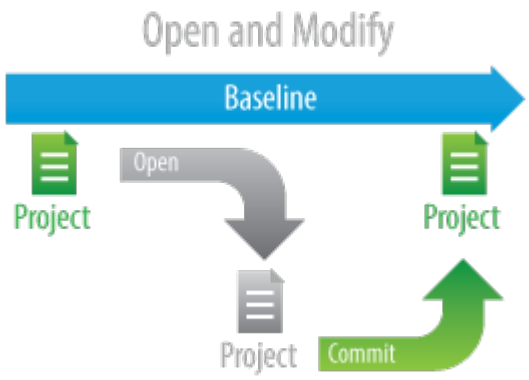
   It is the last tab on the Model Browser. If the tab is closed, you can simply open it by selecting **Collaborate** > **View Locked Elements**.

To lock a project structure via the **Project Properties** dialog

1. From the **File** menu, select **Project Properties**. The **Project Properties** dialog opens.
2. In the **General** tab, click the **Lock Project Structure** button.

Committing changes to Teamwork Server

After you finish editing the model, you must commit the changes to the server so that other team members can find the changes when editing other parts of the model. When you commit the changes, new data from your client application (for example, MagicDraw or a Cameo Suite product, such as Cameo Systems Modeler) is merged with the project on the server repository. As a result, a new version of the project is created. Other team members must update their projects from the server to see the committed changes.



You can choose to release or keep the locks on the model(see page 1482) when committing changes. If you are going to proceed with editing after committing, keep the locks.

Any new elements added to the parent scope are given a unique name when placed among elements of the same type.

Each new project version is created and saved to the server.

To commit a new project or changes in a project

---

1. From the **Collaborate** menu, select **Commit Changes To Server**, or press Ctrl+K. The **Commit Project to the Server** dialog opens.
2. If you need to describe the changes you are going to commit, type a comment in the open dialog.

   > You can select a comment from the recently used comments list. Simply click the Choose Comment button above the **Comment** box, and select any comment from the list in the **Choose Comment** dialog.

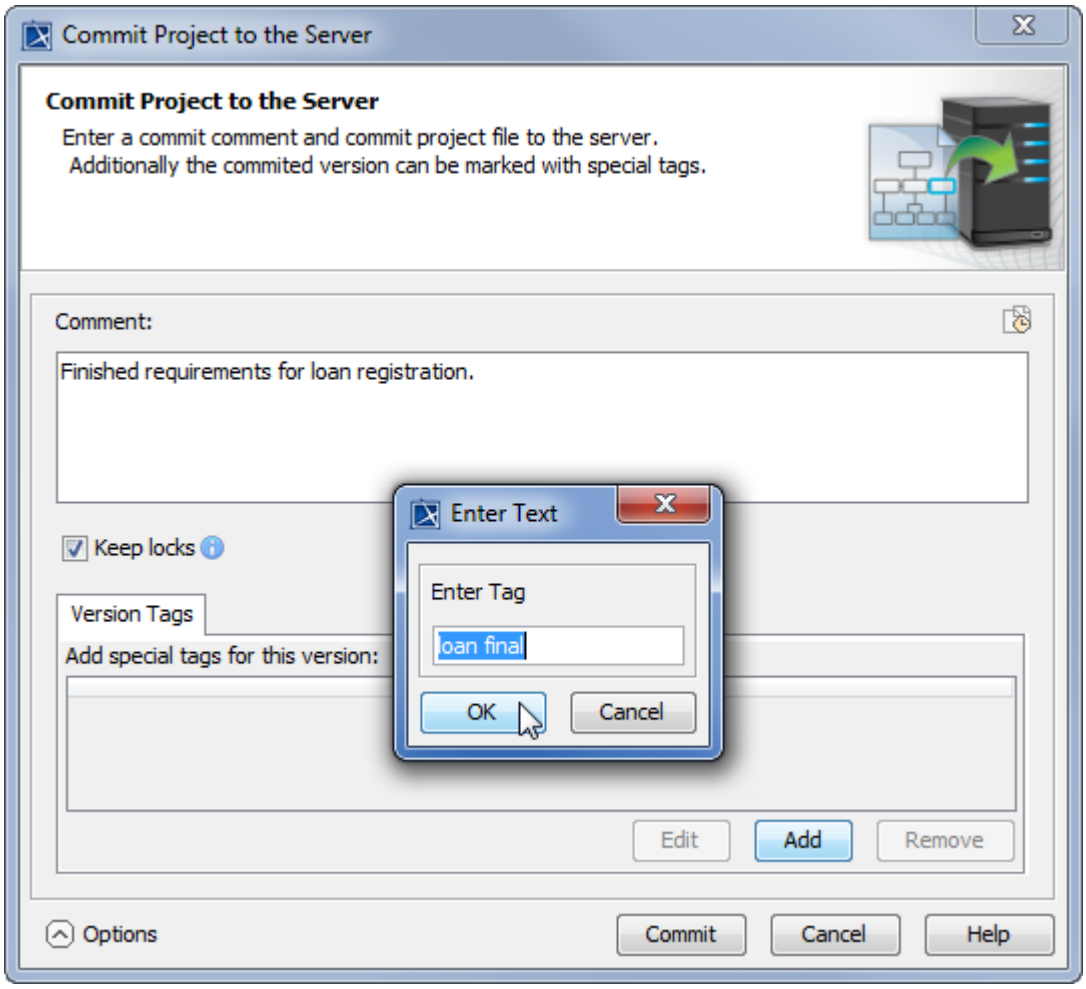
3. If you want to keep locks on the model and continue editing after committing, select the **Keep locks** check box.

   > The **Keep locks** check box remembers the last choice of the user. Thus, if you want to release the locks every time when committing changes, it is not necessary to unselect this check box each time you commit, once you make your choice during the first commit.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

4. If you want to tag the new project version (i.e., add a tag *approved*), do the following:

   a. Click **Options**. The **Version Tags** tab appears on the **Commit Project to the Server** dialog.
   b. Click the **Add** button.
   c. In the open dialog, type a text and click **OK**.



5. In the **Commit Project to the Server** dialog, click **Commit**

6. Wait while the new project version containing the committed changes is sent to the Teamwork Server repository.

Updating changes from Teamwork Server

When updating a project, the latest project version from the server is loaded. A new version of a project is created every time a user commits new data to the server.

> ⚠ Locked model elements cannot be updated.

You should update a project when you know that a new project version is available on the server.

To update the whole project

- From the **Collaborate** menu, select **Update Project**, or press CTRL+U.

A special case of the update feature is updating the lock information. Use it to refresh the list of elements that are locked by other users.

To update elements that are locked by other users

- From the **Collaborate** menu, select **Update Lock Information**, or press CTRL+SHIFT+U.

Branching projects in Teamwork Server

Watch this demo to get familiar with project branching in Teamwork server.

Sorry, the widget is not supported in this export.
But you can reach it using the following URL:

https://www.youtube.com/watch?v=hCxW2UssMns

Project Branching allows the user to derive one project version from another. It duplicates versions of a project and enables users to work simultaneously on them. Branching increases productivity for those who:

- Use MDA tools.
- Produce different products that are still very interrelated. This is common when a product has several versions and each of them is designed for different types of customers.
- Have several development teams working on a single product and have to merge several development branches before releasing the product.

MagicDraw supported Teamwork Server repository features:

- Teamwork Server stores a list of projects.
- Multiple hierarchical branches are supported for every single project version.
- Multiple versions of a single project and a branch are supported with one active (editable) version each.
- The user can open every version of a project or a branch as read-only and one active version as read-write.
- A project or branch can be derived from any previous version of that project or branch (either from the same branch or different branch).
- The user can create a branch from any version of a project or a branch.
- The user can compare two versions of the same or different projects or branches.
- Teamwork Server integrates with Subversion and ClearCase.

MagicDraw supports two branching scenarios:

- Branching a project causes branching of all or part of used modules.
- Branching a project does not cause branching of the modules it uses. The branch of a project uses the same versions of the modules used by the project it is derived from (it can be the module “latest” version or some explicit version specified by the user).

Related pages

Creating project branches

You can create project branches only if you are the project Administrator.

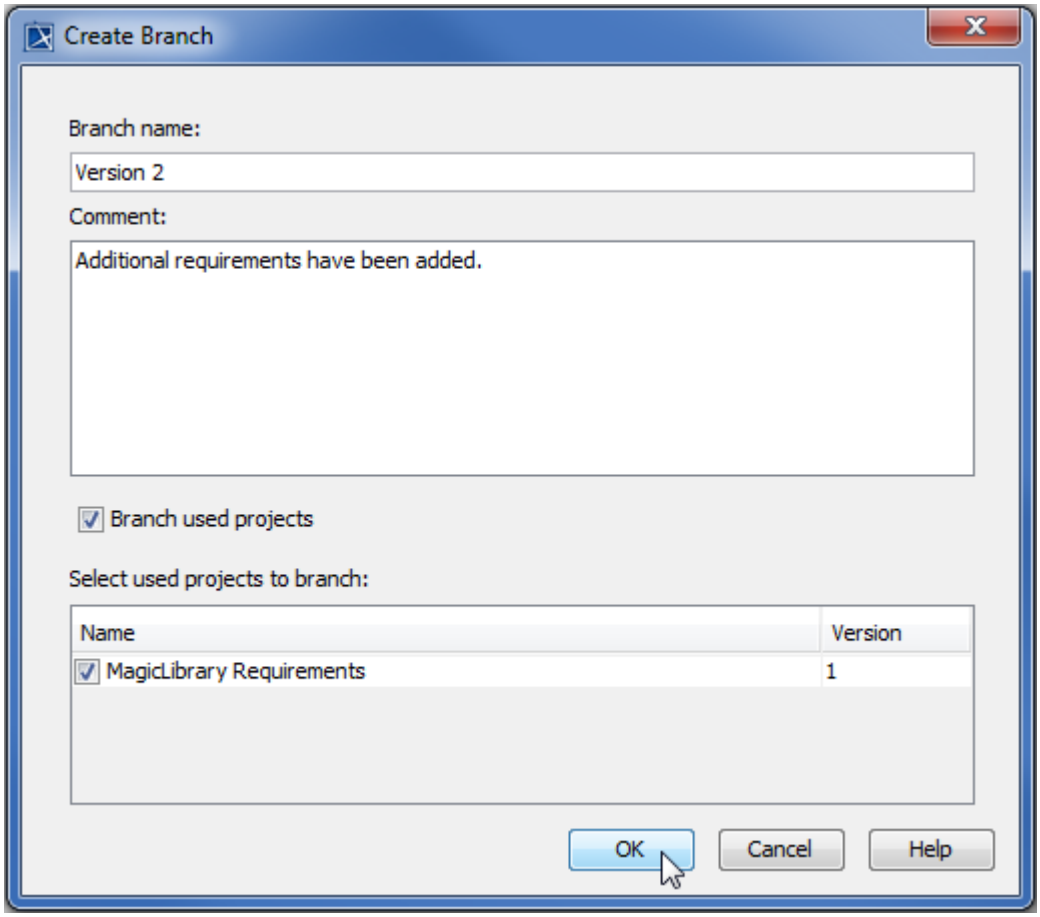
To create a project branch

1. From the **Collaborate** menu, select **Projects**. The **Manage Projects** dialog opens.
2. Select the project from which you want to create a branch.
3. Do one of the following:
   - Click the **Create Branch** button.
   - If you want to create a branch from the specific version of the project, click , in the open dialog select the desired project version, and click the **Create Project** button.

   The **Create Branch** dialog opens.
4. Enter the branch name and comment in the appropriate boxes.

   If you want to branch a project together with used projects, select the **Branch used projects** check box and select the used projects to branch.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



5. Click **OK**.

A new project branch is created.

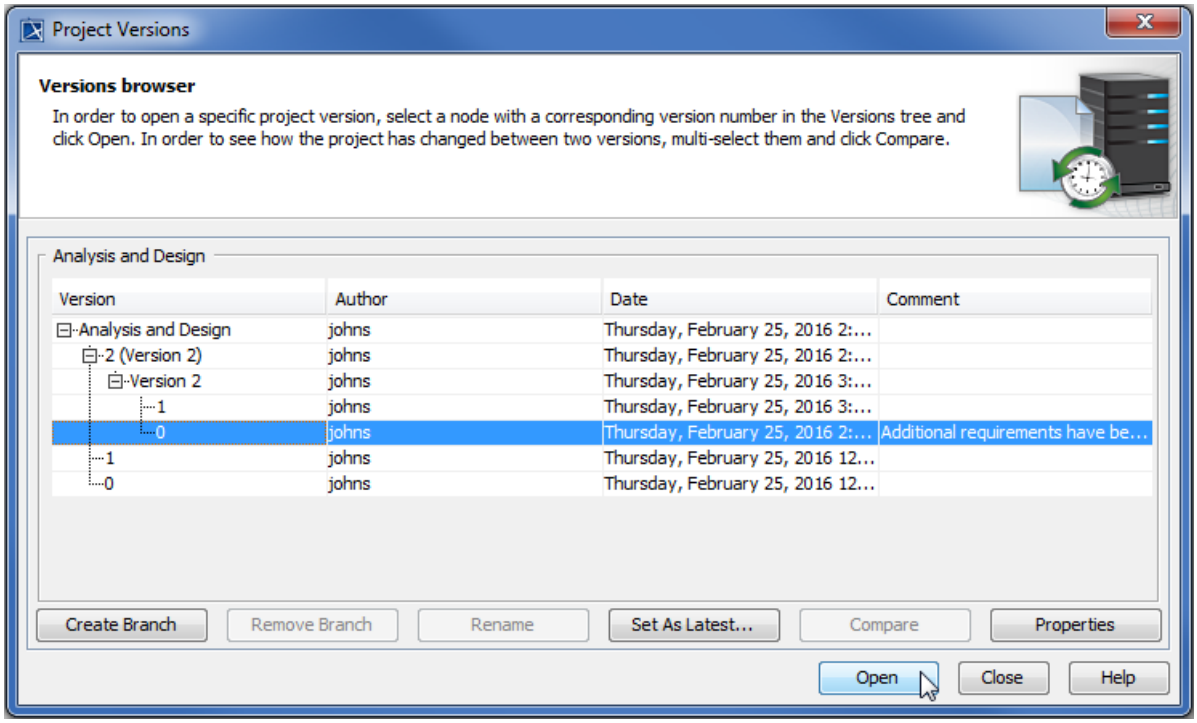
Related pages

- Opening project branches(see page 1495)

Opening project branches

To open a project branch

1. From the **Collaborate** menu, select **Projects**. The **Manage Projects** dialog opens.
2. Select the project that has the branch you want to open and click . The **Project Versions** dialog opens.



3. Select the branch you want to open and click the **Open** button.

The selected project branch opens.

To quickly open the latest version of a branch

---

1. From the **Collaborate** menu, select **Open Server Project**. The **Open Server Project** dialog(see page 1539) opens.
2. Select a project and click . The **Select Branch** dialog opens.
3. Select a project branch and click **OK**.
4. In the **Open Server Project** dialog, click the **Open** button.

The selected project branch opens.

Related pages

- Creating project branches(see page 1494)

Resaving server projects with latest version in Teamwork Server

As of MagicDraw version 17.0.1, the project file inner structure has been changed. Now the latest project file has additional auxiliary information about a project structure. After MagicDraw Teamwork Server has been upgraded to version 17.0.1, you should upgrade your projects used in earlier server versions to the latest file format. After that, you will have full project management features available. To migrate a project to the latest format, projects and used modules must be re-saved (re-committed) in the server.

For project migration, you can use the automatic server project migration feature or migrate each server project manually.

> **Important!**
> While the project is upgrading, other users are prevented from any modifications in that project. It is highly recommended to migrate all projects by the same person.

> **Tips!**
> You can migrate server projects to 17.0.5 or later version without upgrading the server. Starting with version 17.0.4, upgrading Teamwork Server is not required to access server projects with the upgraded version of the client application.

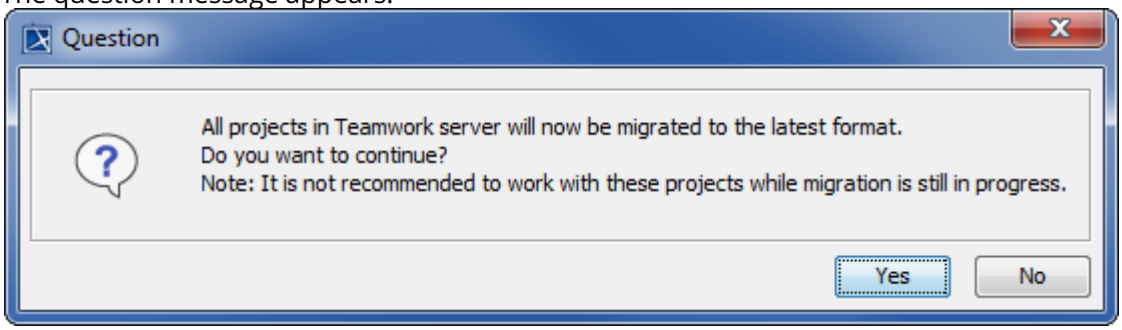
Automatic server project migration

The automatic server project migration feature allows for migrating all server projects at one time.

> **Permissions**
> You must have the Administer project permission for migrating projects.

To migrate all server projects automatically

1. From the **Collaborate** menu, select **Migrate Project to Version <version number>**.
2. The question message appears.



3. Do one of the following:
   - Click **Yes** to migrate all server projects.
   - Click **No** to cancel this action.

After the project migration is completed, you will get a message informing about successful project migration or containing the list of projects that were not migrated. You need to migrate those projects manually.

Manual server project migration

To migrate server projects manually

> **Permissions**
> You must have the Administer project permission for migrating projects.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

1. Open a server project in Teamwork Server. The message notifying that project has been upgraded and requires to be committed to a server of a new version appears.
2. Click **OK** and commit the project to the server(see page 1491).

Perform these actions for all projects and modules you want to migrate to Teamwork Server version 17.0.1 or later.

Troubleshooting

In particular cases, server projects are not upgraded to the latest file format. It may happen in the following cases:

- When the entire project or some modules are not committed to the server.
- When the project structure remains locked after the project has been committed to the server.

In these cases, you get the warning message that project modification is disabled because its structure is being upgraded by another user. You are not allowed to make any modifications in such opened project as projects are opened in the read-only mode.

To solve this problem, the following actions should be performed:

1. Close the project.
2. If any user is currently upgrading the project, he or she should commit this project to the server after the project has been upgraded.
   After the project has been committed, the user should check if the project structure is unlocked(see page 1482). If the project structure is locked, the user who locked the project or an administrator should unlock the project manually in the **Project Properties** dialog.
3. Open the project. If the project was upgraded and committed properly, the project structure is unlocked, and project is available for modifications.

On this page

- Automatic server project migration(see page 1497)
- Manual server project migration(see page 1497)
- Troubleshooting(see page 1498)

Related pages

- Migrating projects from Teamwork Server(see page 1372)
- Locking a model for editing in Teamwork Server(see page 1482)

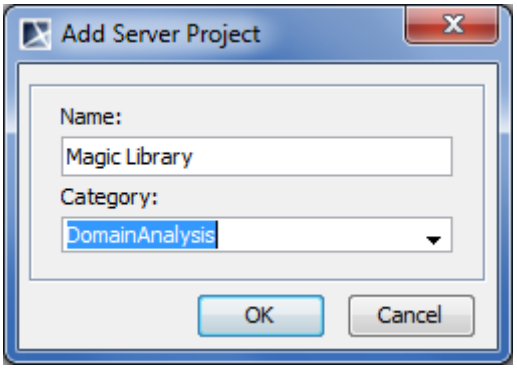
Creating Server Projects

This section provides the step-by-step instructions on how to create a new server project.

To create a new server project

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

1. From the **Collaborate** menu, choose **Projects**.
2. Click the **Add** button. The Add Server Project dialog opens.
3. Type a new project name.
4. Either select an existing category or type a new category name.
5. Click **OK**.



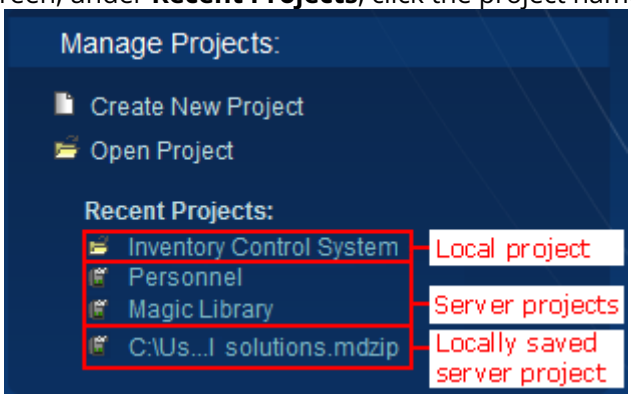
Example of Add Server Project dialog

Opening Recently Used Server Projects

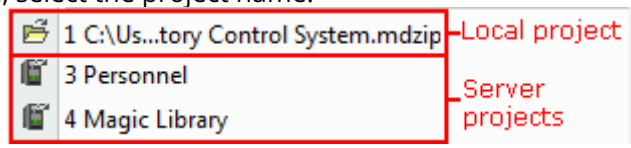
In order to enhance the usability of server projects, MagicDraw introduces a possibility to quickly open server projects, that were recently used. The latest project version will be opened. You can also define the list size of the recently used projects.

To open a recently used server project

- Do one of the following:
  - From the **Collaboration** menu, select **Recent Server Projects** and then select the projects you want to open.
  - On the **Welcome** screen, under **Recent Projects**, click the project name.



  - From the **File** menu, select the project name.



To manage the list of recent server projects

---

1. On the main menu, click **Options** and then select **Environment**. The Environment Options dialog opens.
2. In the **General** option list > **General** options group, set the **Recent Server Projects List Size** option to the wanted number of projects.

Migrating server projects to MagicDraw Teamwork Server 17.0.1 or later

As of MagicDraw version 17.0.1, the project file inner structure has been changed. Now the latest project file has additional auxiliary information about a project structure. After MagicDraw Teamwork Server has been upgraded to version 17.0.1, you should upgrade your projects used in earlier server versions to the latest file format. After that, you will have full project management features available. To migrate a project to the latest format, projects and used modules must be re-saved (re-committed) in the server.

For project migration, you can use the automatic server project migration feature or migrate each server project manually.

While the project is upgrading, other users are prevented from any modifications in that project. It is highly recommended to migrate all projects by the same person.

You can migrate server projects to 17.0.5 or later version without upgrading the server. Starting with version 17.0.4, upgrading Teamwork Server is not required to access server projects with the upgraded version of the client application.
For the instructions how to enable connections to the server from later client versions, see Activating Teamwork Server license after purchase[332] at nomagic.com[333].

Automatic server project migration

The automatic server project migration feature allows for migrating all server projects at one time.

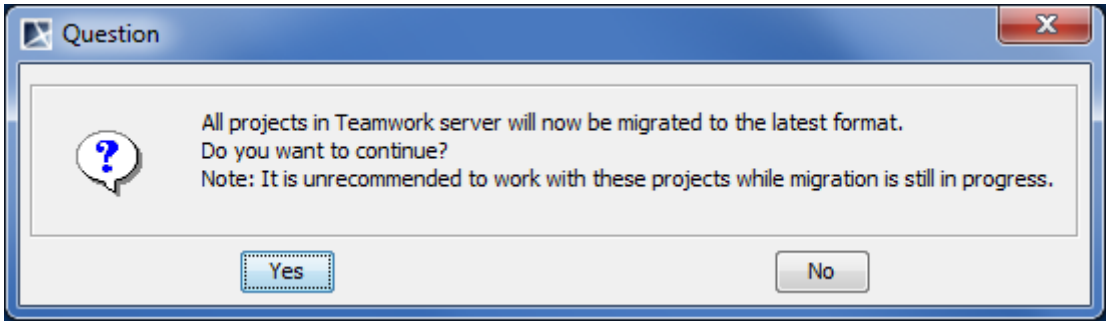
You must have the Administer project permission for migrating projects. For more information about the permissions, see “User permissions” in MagicDraw Teamwork UserGuide.pdf.

To migrate all server projects automatically

1. From the Collaborate menu, select Migrate Project to Version 17.0.1.
2. The question message appears.

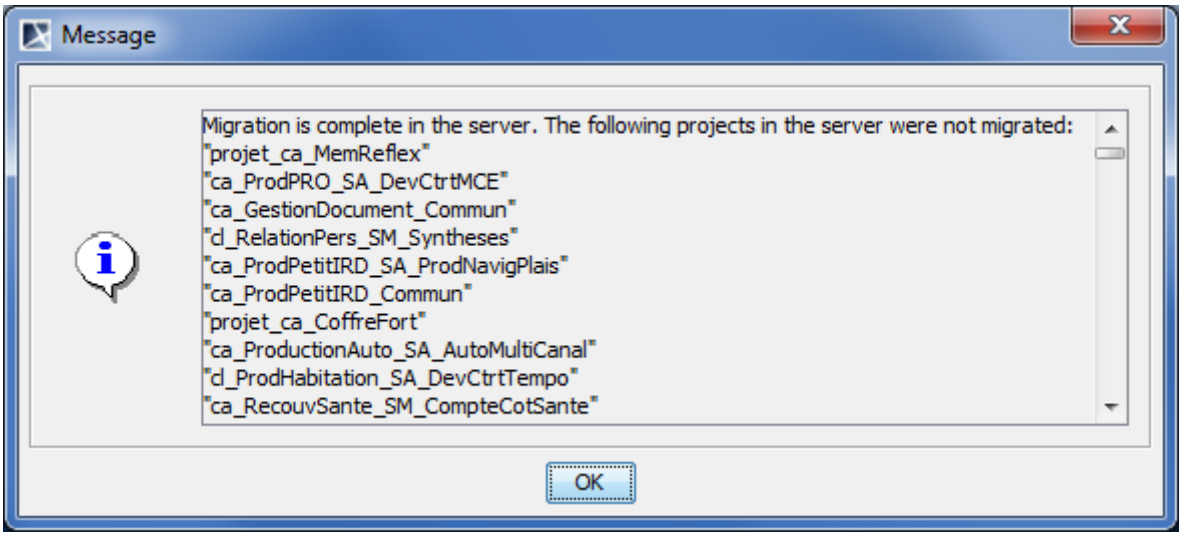
332 https://docs.nomagic.com/display/NMDOC/Activating+the+Teamwork+Server+commercial+license
333 http://nomagic.com

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



3. Do one of the following:
   - Click Yes to migrate all server projects.
   - Click No to cancel this action.

After the project migration is completed, you will get a message informing about successful project migration or containing the list of projects that were not migrated. You need to migrate those projects manually.



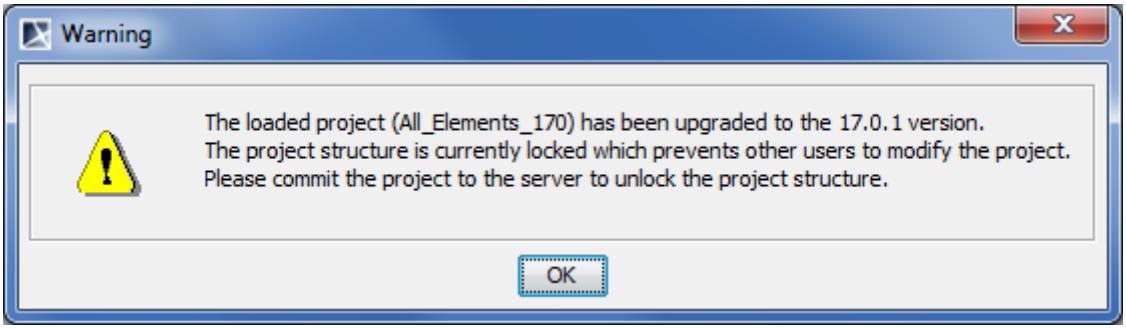
Example of message appearing after migrating server projects

Manual server project migration

To migrate server projects manually

You must have the Administer project permission for migrating projects. For more information about the permissions, see User permissions(see page 1500).

1. Open a server project in Teamwork Server. The message notifying that project has been upgraded and requires to be committed to a server of a new version appears. See the following example of the message.



2. Click **OK** and commit the project to the server.

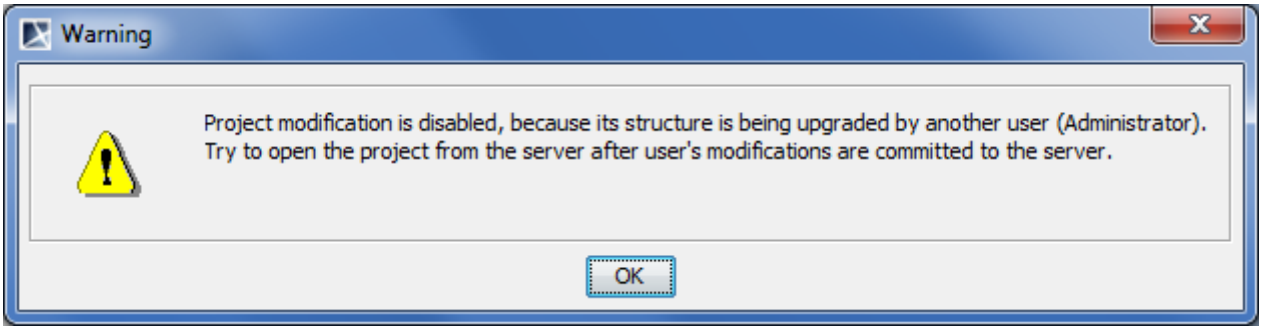
For more information about committing a project, see section Committing to Teamwork Server(see page 1491). Perform these actions for all projects and modules you want to migrate to Teamwork Server version 17.0.1 or later.

Troubleshooting

In particular cases, server projects are not upgraded to latest file format. It may happen in the following cases:

- When the entire project or some modules are not committed to the server.
- When the project structure remains locked after the project has been committed to the server.

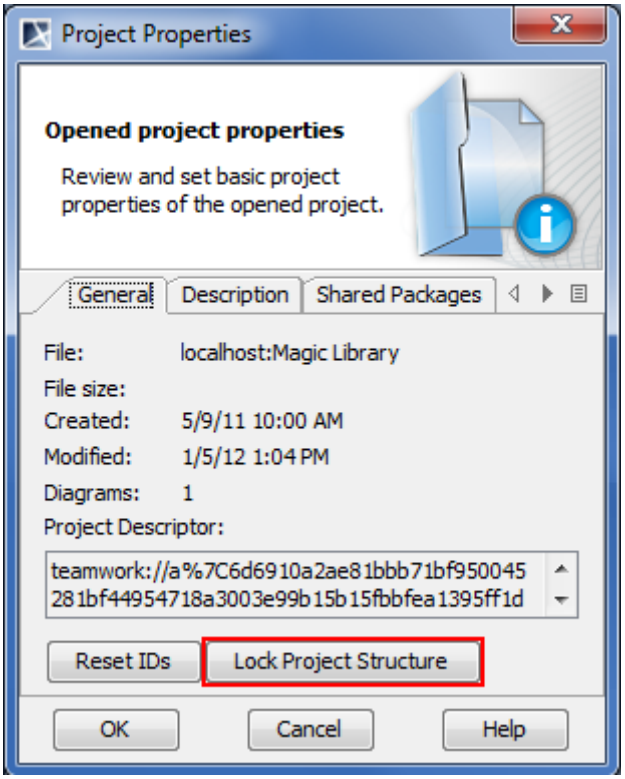
In these cases, you get the following message while loading the project:



You are not allowed to make any modifications in such opened project as projects are opened in the read-only mode.

To solve this problem, the following actions should be performed:

1. Close the project.
2. If any user is currently upgrading the project, he or she should commit this project to the server after the project has been upgraded.
   After the project has been committed, the user should check if the project structure is unlocked. If the project structure is locked, the user who locked the project or an administrator should unlock the project manually in the Project Properties dialog.
   For more information about project structure locking, see section Locking project structure.



View of Project Properties dialog when project structure is unlocked

3. Open the project. If the project was upgraded and committed properly, the project structure is unlocked, and project is available for modifications.

Related procedures:

Locking Project

Committing to Teamwork Server(see page 1491)

Related references:

Commit Project to the Server dialog

Working with Project Properties dialog(see page 108)

Unlocking Model Elements and Diagrams After Editing

After you have finished editing, you should unlock model elements and diagram.

To unlock elements and / or diagrams

1. Do one of the following:
    - Select a locked element in the Model Browser.
    - Select a locked diagram in the Model Browser.
    - Select a locked symbol on a diagram pane.
    - Right-click the diagram pane.
2. On the selected item's shortcut menu, point to Lock or Lock Diagram, and then click a wanted unlocking command. For more information about locking commands, see Unlocking commands(see page 1504).

> The list of commands may vary depending on the item you have selected.

To unlock several elements

1. Do one of the following:
    - Select several locked elements in the Model Browser.
    - Select several locked symbols on the diagram pane.
2. On the selected elements shortcut menu, click Unlock. This forbids editing element specification properties in the Specification window.

To unlock all element

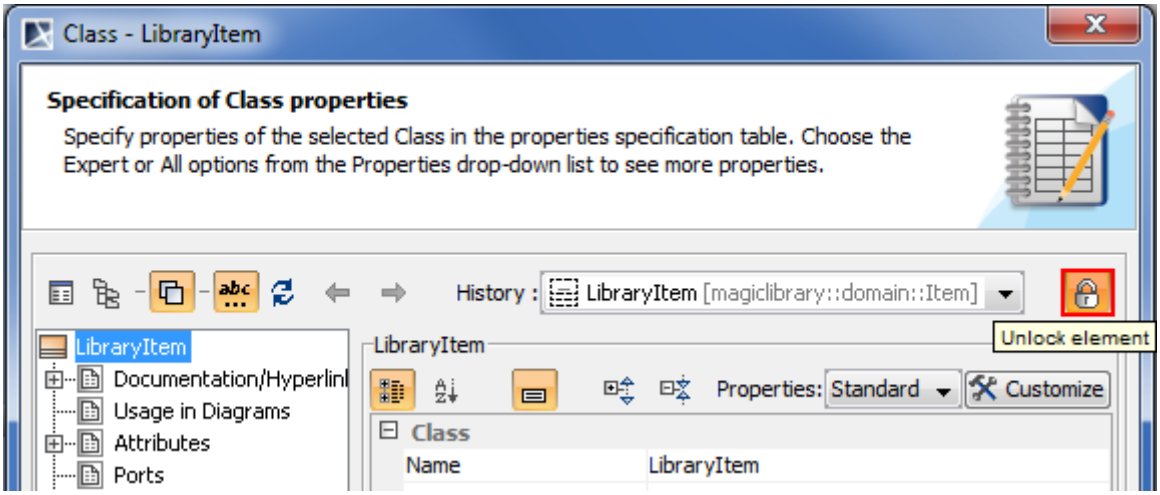
1. From the **Collaborate** menu, select **Unlock All**. The Commit Project to the Server dialog opens.
2. Click to clear the **Keep Locks** check box. By default, the check box is selected.
3. Click **Commit** when you are done.

To unlock an element in the Specification window

1. Open the Specification window of the selected locked element or locked symbol.
2. Click the Unlock element button (see the highlighted button in the following figure). This forbids editing element specification properties in the Specification window.

> If the element was modified, you will be requested to commit changes to the server.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Unlock element button (fragment of Specification window)

To unlock an element assigned as a property value

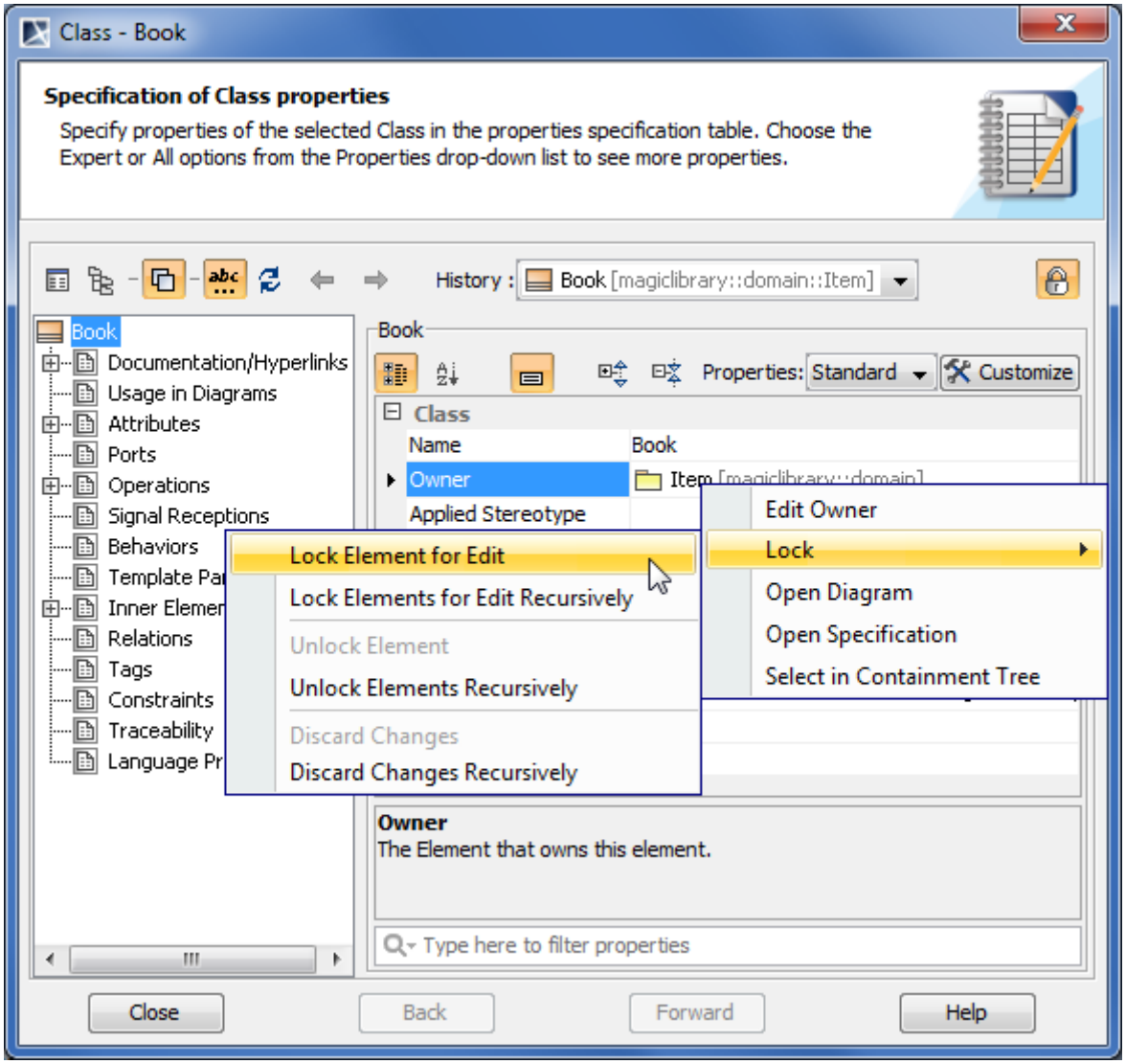
---

1. Open the Specification window of the selected element or symbol.
2. In the element property list, select the element assigned as a the property value.
3. On the selected value shortcut menu, point to **Lock**, and then click one of available commands:
   - **Unlock Element**.
   - **Unlock Elements Recursively**. This command is available if the selected value is an owner for some elements.

For more information about unlocking commands, see Unlocking commands[334].

---

[334] https://docs.nomagic.com/download/attachments/55842524/unlocking_commands_on_shortcut_menu.png?api=v2&modificationDate=1443087158411&version=1

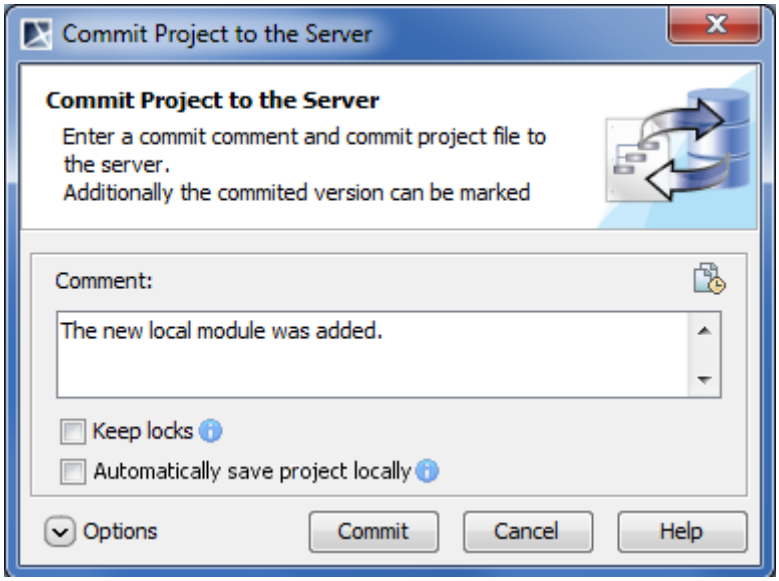


Unlocking property value in Specification window

To unlock on project committing

1. From the **Collaborate** menu, select **Commit Project** or press CTRL+K. The **Commit Project to the Server** dialog opens.
2. Click to clear the Keep Locks check box.
3. Click **Commit** when you are done.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Commit Project to the Server dialog

> Model elements that are not modified are unlocked without committing them to the server.

Unlocking with administrator permissions

The administrator (that is a user named "Administrator") can forcibly unlock elements locked by other users.

> The unlocking procedures for administrator are the same as described in section Unlocking model elements and diagrams after editing(see page 1504).

Forced unlocking is helpful when elements are left locked by the user who is currently unavailable.

> Use forced unlocking only in very special cases, as this may cause users working on this project to start their work again.

The user has a possibility to commit changes made to elements that were forcibly unlocked by an administrator. To commit changes, the following conditions should be satisfied:

- The user should have the latest version before locked elements have been lost.
- A new revision of the project has not been committed to the server.

The preceding scenario is very rare, and users should not rely on it. After all, if administrator forcibly unlocked elements, it means there was a need to edit these elements for some purpose.

Changes to elements that were locked and not forcibly unlocked by an administrator, can be committed to Teamwork normally.

To delete an element from the server project

1. Lock the element or elements you want to delete, see Locking model elements and diagrams for editing(see page 1482).

2. Delete the element.
3. Commit the project to the Teamwork Server.

> If the element contains inner elements which can be locked separately, these inner elements should not be locked by the other users to complete element removal. For this reason, we suggest to use command Lock Element for Edit Recursively to lock all inner elements so nobody could prevent you from removing selected elements.

Saving Server Projects Locally

You can save your server projects on a specific location in your computer. You can work with the local project and also be able to commit changes to the server. Working with the local copy of the server project helps to improve collaborative work productivity. Even if the connection with Teamwork Server is lost, you can still continue working with the project and commit changes later, when the connection is restored.

To save a server project locally

---

1. Do one of the following:
   - From the File menu, choose Save Changes Locally.
   - From the Collaborate menu, choose Save Changes Locally.
   - On the File toolbar, click .
   - Press CTRL+S.
2. Specify the name and location, if needed.

If you want to keep your local project copy up-to-date, you have to save changes locally every time committing to the server. For this purpose, select the Automatically save project locally check box on the Commit Project to the Server dialog.

Related Procedures:

Committing to Teamwork Server(see page 1491)

Converting Server Projects to Local

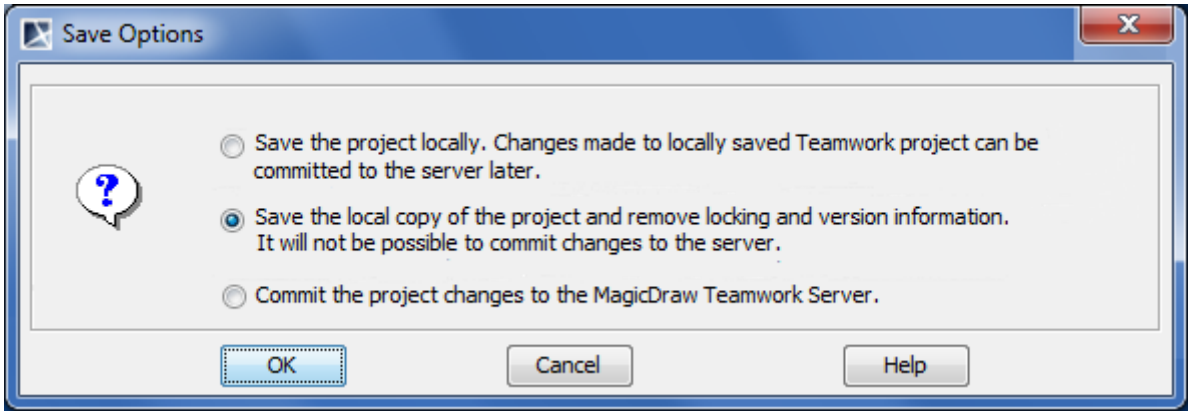
You can convert a server project back to the local copy. After converting, the project will no longer be related with a server and you will not be able to commit changes to the Teamwork Server anymore.

To convert a server project to a local file system

---

1. Open a server project.
2. On the **File** menu, click **Save Project As**. The Save Options dialog opens.
3. In the dialog, select **Save a local copy of the project and remove locking and version information**. The project will be saved on a disk and will not include any locking or version information. A local copy of the project cannot be committed to the Teamwork Server anymore.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



4. Click **OK** and specify a location to save the project.

If a server project uses one or more module versions created with an earlier version than MagicDraw 17.0.1, you need to save these modules with MagicDraw 17.0.1 manually. If you cannot change modules because your project uses fixed modules, you should perform the following steps:

1. Log on to Teamwork Server which stores the module.
2. Branch each module of the earlier version from the fixed version to a separate branch.
3. Open the branched version of each module and commit it to the server in order to migrate the module to the latest version.
4. In the project, change versions of each branched module to the branched version instead of original one.
5. Save the project locally again.

Updating Server Projects

When updating a project, the latest project version from the server is loaded. A new version of a project is created every time a user commits new data to the server.

Model elements that are locked for will not be updated.

When updating *lock information*, the list of elements that are locked by other users is refreshed. You should update a project when you know that a new project version is available on the server.

To update the whole project

- From the Collaborate menu, select Update Project, or press CTRL+U.

To update elements that are locked by other users

- From the Collaborate menu, select Update Lock Information, or press CTRL+SHIFT+U.

Versioning

Every time a project is committed, a new project version is created. You can undo committed changes by setting a previous project version as latest.

To view project versions

1. From the **Collaborate** menu, select **Projects** or **Open Server Project** commands.

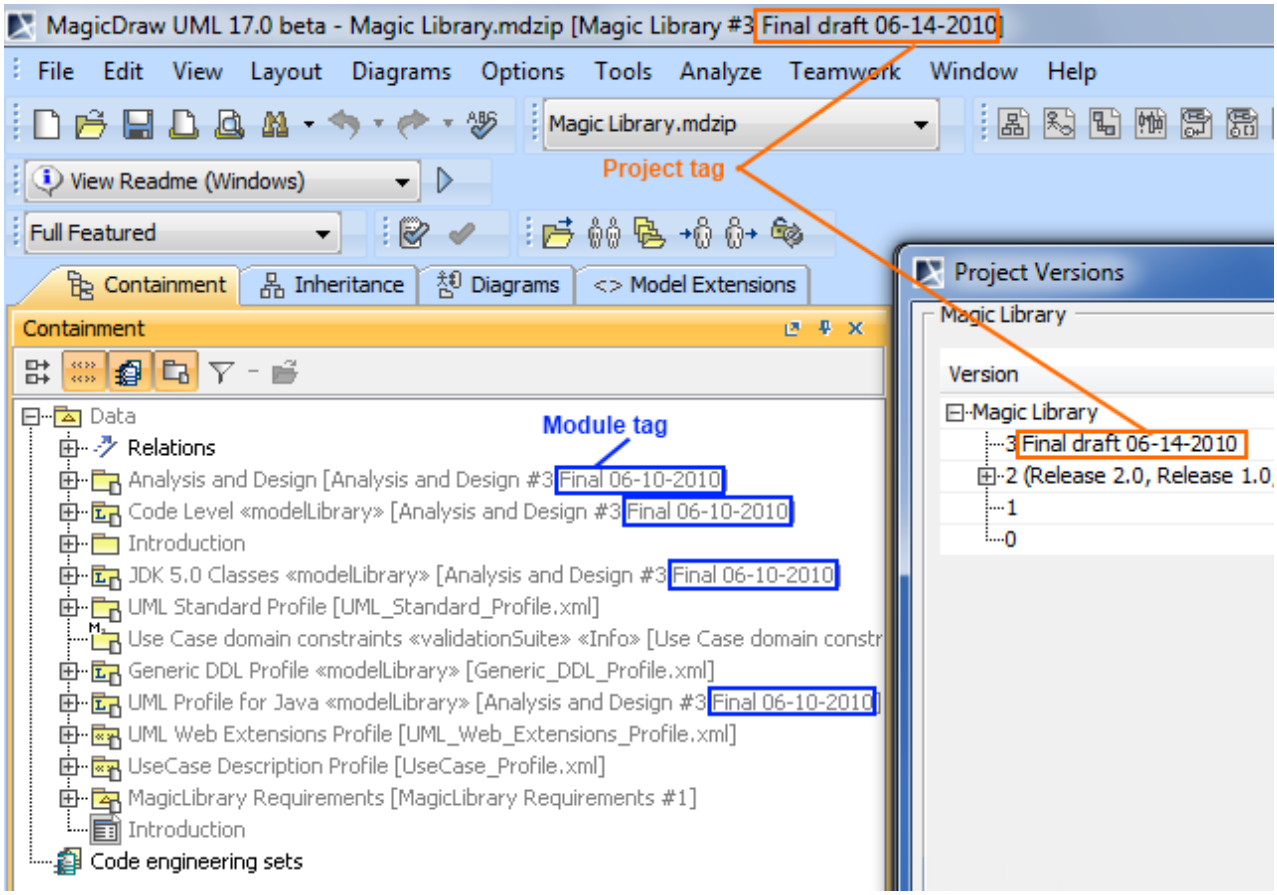
2. Click the Project History button. The **Project Versions** dialog opens.

The **Project Versions** dialog lists all versions of the selected project. You can set the selected project version as latest, edit comments, and review other project versions.

Visibility of Version Tags

Project version tags are visible in the following places:

- Version browser
- Branches browser
- Module/project name representation in the containment tree/application title.



Places where projects tags are displayed

Code Engineering in Server Projects

For more information about general code engineering functionality, see MagicDraw CodeEngineering UserGuide.pdf which can be found in <MagicDraw installation directory>\manual.

To generate code in the server project

1. From the Code engineering sets shortcut menu in the Browser, select **New** and then select a code engineering language. Type a name for new set.
2. Add elements from the Data branch to the newly created set. All elements (locked and unlocked) can be added to set for code generation. Drag & drop them from the Browser tree.
3. Select the code engineering set, containing elements in the Browser and from the shortcut menu, select **Generate**. The code is generated for all elements.

To reverse code in the server project

---

1. From the created code engineering set shortcut menu, select **Edit**.
2. Specify the Working Directory and Working Package, to where reversed elements will be placed in the model.

> Reversing the whole set to a defined package is allowed only if the package is locked.

1. In the **Add File** tab from the Round Trip Set dialog, add code files from the **All Files** list to the **Set** list. Click **OK**.
2. From the modified code engineering set shortcut menu, select **Reverse**. Define options in the Reverse Options dialog. Click **OK**.

When performing a round-trip code engineering, code cannot be reversed for unlocked elements - the Reverse command will be disabled in shortcut menu.

Project Usage Map

Additional Resources:

- http://www.nomagic.com/files/viewlets/Project_Usage_Map.html?ml=1
- http://www.nomagic.com/attachments/article/547/Four%20Key%20Improvements%20When%20Managing%20the%20Teamwork%20Server%20Repository.pdf

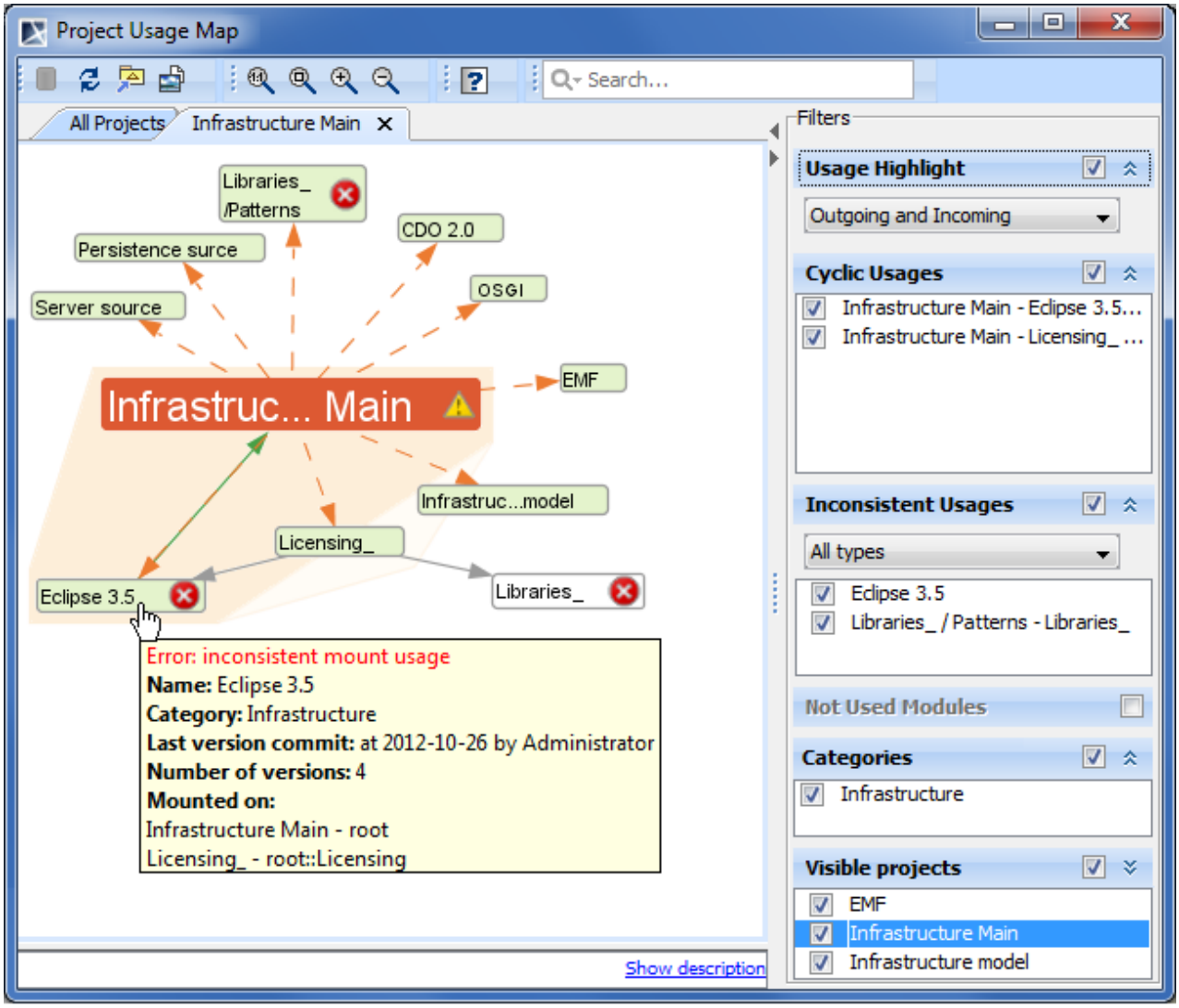
The Project Usage Map is a live visual graph that represents Teamwork Server project usages as well as identifies potential problem areas.

The Project Usage Map allows for representing projects and their dependencies in two views:

- All Projects view that shows all projects and all the dependencies among them.
- Individual project view that shows a particular project along with other directly and indirectly used modules.

Using the Project Usage Map you can easily do the following:

- Identify, analyze, and validate dependencies among projects (for example, you can find out easily all the projects, wherein a particular module is used).
- Identify cyclic dependencies among projects. Identify and fix inconsistent dependencies among projects.



Example of Project Usage Map.

In the following table, you can find definitions of some concepts that are used in descriptions of Project Usage Map:

Concept	Definition
Cyclic usages	Several parts of the project are directly or indirectly using the main project.
Inconsistent mount	The project is mounted on different packages of the main project.
Inconsistent version or branch usage	The main project uses different versions or branches of the same module.

Concept	Definition
Used project (module)	A project with shared parts.
Not converted project	A project whose usage information cannot be retrieved and analyzed by the Project Usage Map since it was created using MagicDraw version 17.0 or earlier. Open the project and add it back to the Teamwork Server so it will be converted into the newest format.
Non-sharing usage	A usage that makes a project invisible to other projects despite them using it. This usage allows using projects in a particular project while this used project is not visible in other projects related through sharing functionality. **Example** You have a project “A” that uses a project “B”. Additionally, a project “B” uses a project “C” (its used projects are not shared). If you open the project “B” as a standalone project, you will see that the project “C” is used as well; however, if you open the project “A”, you will not see that the project “C” is used because it is not shared through the project “B.”
Sharing usage	A usage that makes a project visible to other projects wherein this project is used as a module.
Unconfirmed usage	The usage has been created automatically and is not yet confirmed by the user.
Unused project (module)	The project that has shared parts that are not used in any other project.

Related Pages:

- Project Usage Map(see page 1511)
  - Working with Project Usage Map(see page 1513)
  - Project Usage Map Window(see page 1516)
  - Solving Project Usage Problems(see page 1519)

Working with Project Usage Map

MagicDraw provides Project Usage Map to identify, analyze, and validate dependencies between projects. It is enough to be connected to your Teamwork Server repository from MagicDraw to open Project Usage Map.

Using the Project Usage Map, you can:

- Review the content of the entire Teamwork Server repository in the All Projects view.
- Narrow the analysis scope by hiding in the view any types of conflicts and / or projects, in which you have no interest.
- Review usages of a particular project in the Individual project view.
- Open projects and resolve problems.

To open a current project usage map

Do one of the following:

- From the active project's **Collaborate** menu, select **Project Usage Map > Current Project Map.**
- In the **Edit Projects** dialog, select the project and then click the **Project Usage Map** button.

To open the all projects usage map

- From the **Collaborate** menu, select **Project Usage Map > All Projects Map**.

To identify project usage problems

Do one of the following:

- Select one or several problems in the filter area.
- Select a project with an error icon in the map area.

For more information about problem identification and solutions, see Solving Project Usage Problems(see page 1519).

In the Project Usage Map, you can easily identify project usages. This means, that you can see which projects uses or are used by the selected project.

To identify project usages

---

1. In the **Usage Highlight** filter, select one of the following:
    - **Outgoing and Incoming**
    - **Outgoing**
    - **Incoming**
2. Select any project in the map.

A green arrow represents the incoming usage. The incoming usage is drawn between the selected project and the project that is using it. In the description area, the project that is using the selected project is listed under Used by.

An orange arrow represents the outgoing usage.The outgoing usage is drawn between a selected projects and the project that is used by it. In the description area, pre project that is used by the selected project is listed under Using.

To narrow results visible in a project usage map

Do one of the following:

- Select a project in the map area and press Delete

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- From the project shortcut menu, select Hide
- In the Filter area, click to clear specific project, category, or problem check box
- In the Search box, type the project name.
- Open the specific project's usage map.

You can export a project usage map to the model. This allows you for saving the current server project usages representation in the class diagram and compare it with the latest representation (see the following figure).

For more information about comparing the projects, see Project Comparison(see page 1295).

To export a project usage map to the model

- In the **Project Usage Map** window toolbar, click the Export to Model button.

Any changes in the exported project usage map will not affect the Project Usage Map view.

Figure 674 -- Example of exported maps comparison

To save the project usage map as an image

---

1. In the **Project Usage Map** window, click the Copy as BMP Image button. The map view is copied to the clipboard.
2. Paste in any image editor.
3. Save to the selected location.

To select a particular project in the map

Do one of the following:

- Click a project in the map
- Select a project from the Visible projects filter
- In the Search box, type the project name.
- Select the project and then click the project name in the description area.

Search Criteria
You can select one of the search criteria:
- Match from start
- Match anywhere

Refresh the Project Usage Map to see the latest view of the Teamwork Server repository.

To refresh the project usage map

- In the **Project Usage Map** window toolbar, click the Refresh button.

Layouts and Filters
All the layouts and selected filters will be cleared after refreshing.

To open the individual project view

Do one of the following:

- Double-click a project
- Select Show Project Structure from the project shortcut menu

Solve the inconsistent usages and other problems by opening projects directly from the Project Usage Map.

To open as a project

---

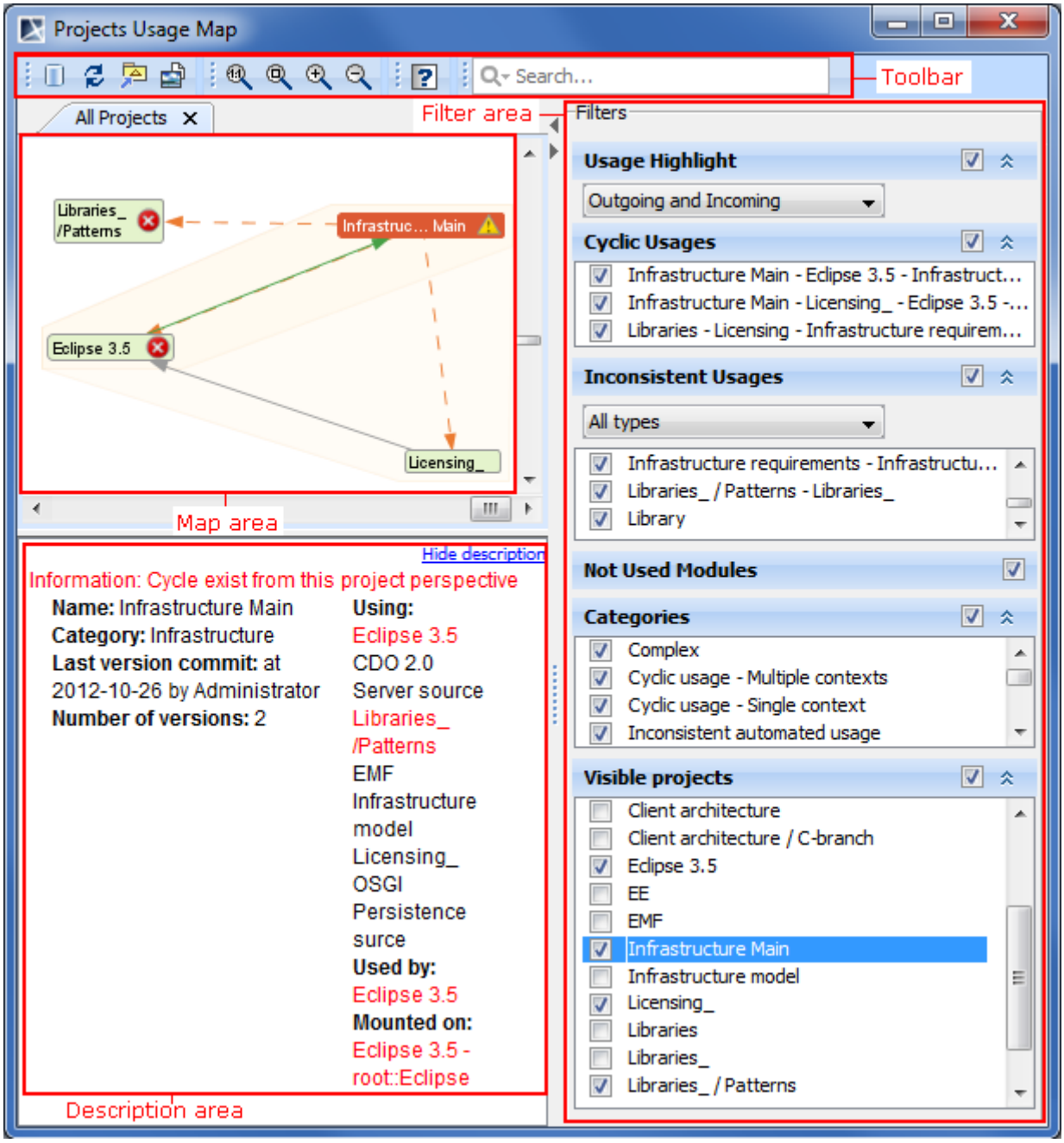
1. Select the project which usages you want to fix.
2. From the shortcut menu, select **Open Project.**

Related Pages:

- Project Usage Map(see page 1511)
  - Working with Project Usage Map(see page 1513)
  - Project Usage Map Window(see page 1516)
  - Solving Project Usage Problems(see page 1519)

Project Usage Map Window

In the Project Usage Map window, you can analyze the contents of the entire Teamwork Server repository, as well as individual projects. You can explore the displayed projects and analyze both project usages and identified problems.



Project Usage Map window.

Toolbar

Toolbar element	Description
Open All Projects Map	Click this button, if you closed the All Project view in the Project Usage Map. This button is unavailable if the All Project view is opened.
Refresh	Click this button if you want to see the latest version of the repository view. All the layouts and filters will be reset.

Toolbar element	Description
Export To Model	Click this button to export an active usage map view to the Class diagram. You can save the exported usage map for the later comparison of the latest version.
Copy as BMP Image	Click to copy the active usage map. The usage map view is copied to the clipboard. You can paste it in any image editor.

Filter Area

Window element	Description
Usage Highlight	Select which usages you want to highlight: incoming, outgoing, or both.
Cyclic Usages	Select which cyclic usages you want to be displayed in the map view.
Inconsistent Usages	Select which type of inconsistent usages you want to be displayed.
None	Select if no inconsistent usages should be displayed.
Version	Select if you want to see version inconsistencies.
Branch	Select if you want to see branch inconsistencies.
Mount Point	Select if you want to see mount point inconsistencies.
Unconfirmed	Select if you want to see unconfirmed usage inconsistencies.
All Types	Select if you want to see all type inconsistencies.
Not Used Modules	Select if you want to see only the projects with not used modules.
Categories	Specify the particular categories of projects.
Visible Projects	Select the projects you want to be displayed in the map. By default, all the projects are selected. Right-click in the Visible Projects list area and select Show all or Hide all. Press the Delete key to unselect a project.
Description Area	See the information of the selected project. All the problems are marked in red.

Map area: understanding the contents

The representation of the selected project, incoming and outgoing usages is depicted in the following figure.

An orange arrow represents outgoing usages. A green arrow represents incoming usages.

The representation of the Sharing usage is depicted in the following figure by the solid grey arrow.

The representation of the Non-sharing usage is depicted in the following figure by the dashed gray arrow.

The representation of a Not converted project is depicted in the following figure.

The representation of Cyclic usages is depicted in the following figure.

The representation of a branch or version inconsistencies is depicted in the following figure.

The representation of the mount point inconsistencies is depicted in the following figure.

The representation of Unconfirmed usage is depicted in the following figure.

The representation of Unused modules is depicted in the following figure.

Related Pages:

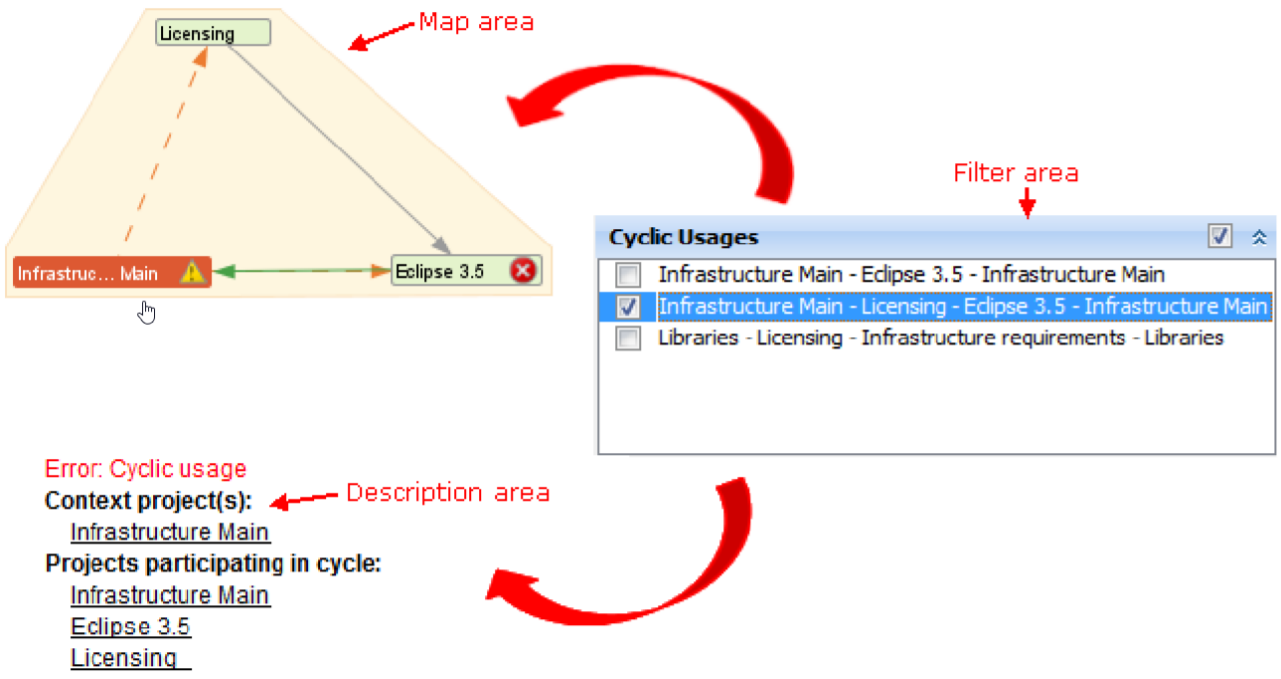
- Project Usage Map(see page 1511)
  - Working with Project Usage Map(see page 1513)
  - Project Usage Map Window(see page 1516)
  - Solving Project Usage Problems(see page 1519)

Solving Project Usage Problems

Using the Project Usage Map, you can identify usage problems. Then you can easily select and open projects that need to be fixed.

Resolving cyclic usages

Problem
When a project decomposition is used, the project is split into smaller projects. You can benefit from this decomposed project - you need to load only small part of the project instead of all of it. This reduces complexity and even improves performance. In addition to this, you would not expect and most likely not care to load the remaining parts of the main project each time you open a single part of it. This can happen if you have cyclic usages, i.e., project parts are using the main project. Typically, such usages increase complexity and reduce performance. Also, if the project takes part in a cycle, it cannot be reused as an independent part of another project. Other projects taking part in the cycle will be automatically used as well. Cycles are often a symptom of unintentional usages, created unknowingly by the user.



Identifying cyclic usages

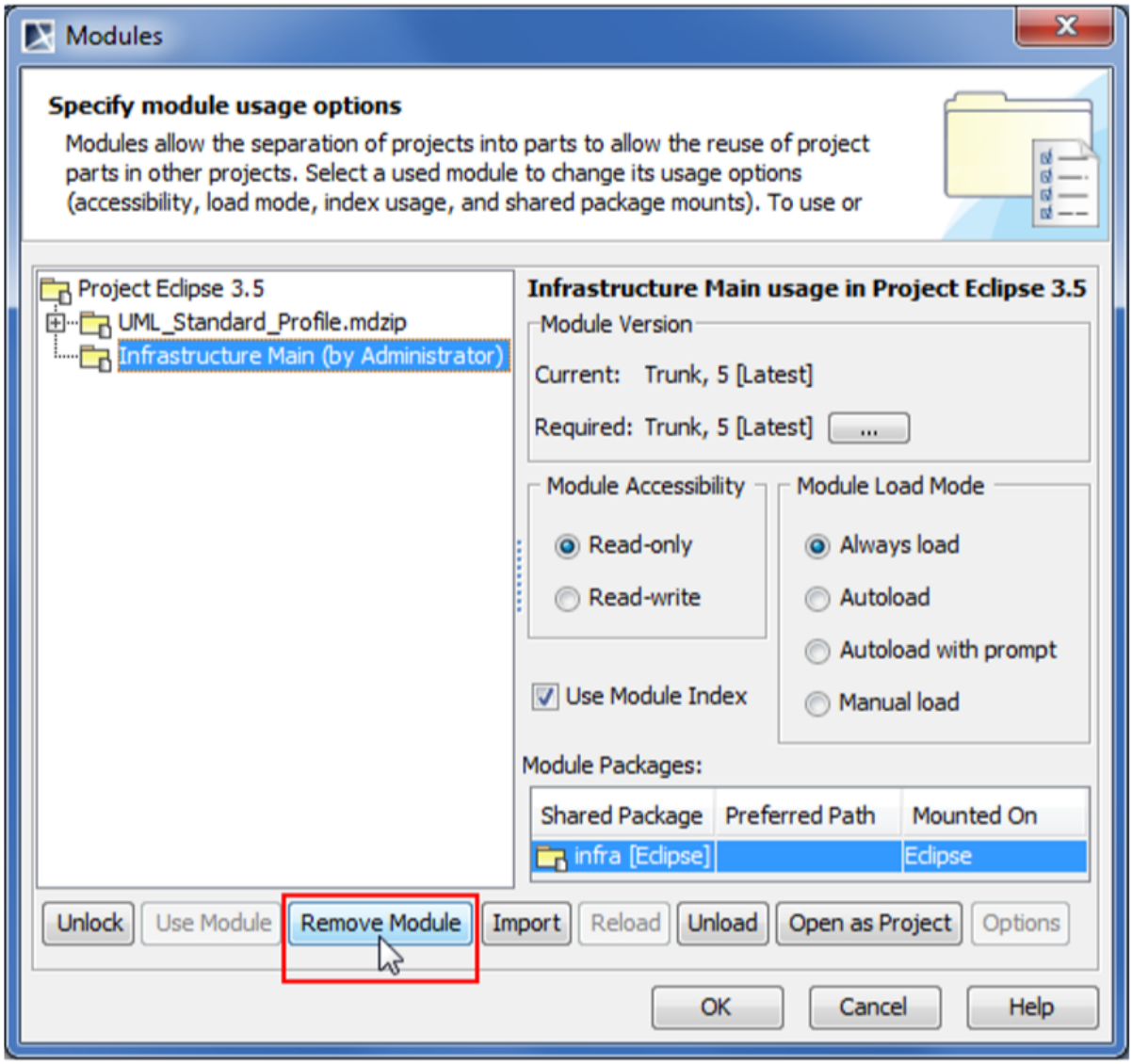
Solution

The Project Usage Map automatically identifies cycles and highlights them in the Repository View. You can then open a Project Usage Map for the suspected projects participating in a cycle to analyze it more closely and if necessary - break usages that cause cycles.

To resolve the cyclic usages

> **Model Level**
> Be sure to check and resolve the model level usages between the project and module.

1. Open the main project of the cycle.
2. From the **Options** menu, select **Modules**. The **Modules** dialog opens.
3. Select a module which should be removed, and click the **Lock** button.
4. Click the **RemoveModule** button.
5. Click **OK** when you are done.

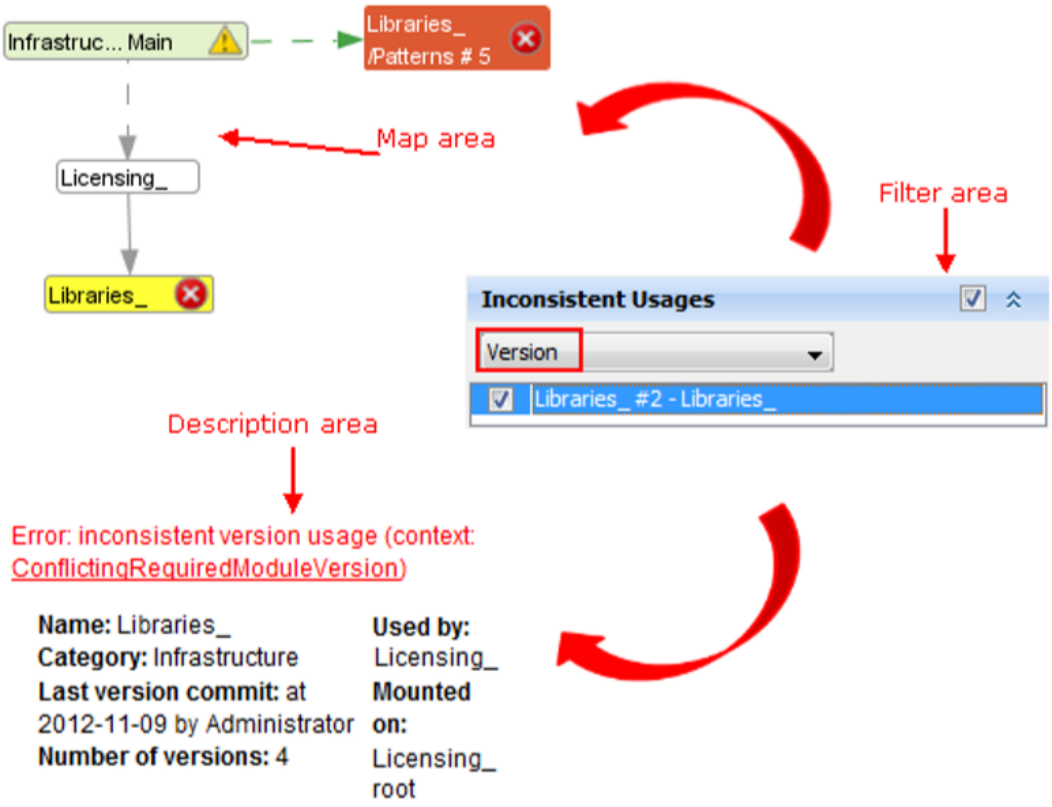


Removing module to break cyclic usage

Resolving version inconsistencies

Problem

You are using different versions of the same project in your main project.



Identifying version inconsistency

Solution

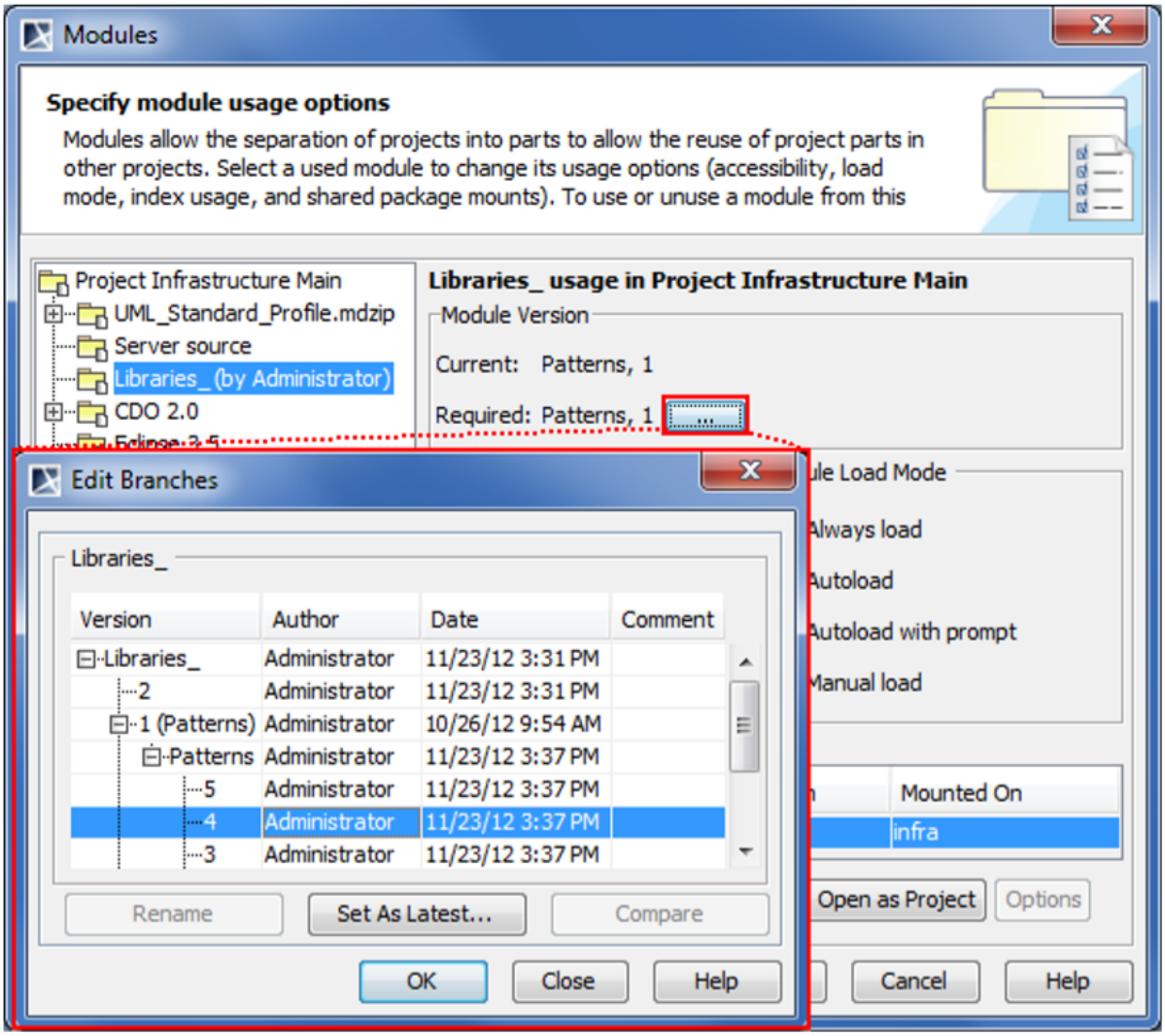
The Project Usage Map highlights these inconsistencies. You can then open projects with inconsistent usages and fix them by unifying the used project version.

To resolve a version inconsistency

> **Model Levels**
> Be sure to check and resolve the model level usages between the project and module.

1. Open a project using the projects of inconsistent version.
2. From the **Options** menu, select **Modules**. The **Modules** dialog opens.
3. Select a module which version you want to change, and click the **Lock** button.
4. In the **ModuleVersion** area, click the ... button. The **EditBranches** dialog opens.
5. Select the wanted version.
6. Click **OK** when you are done.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

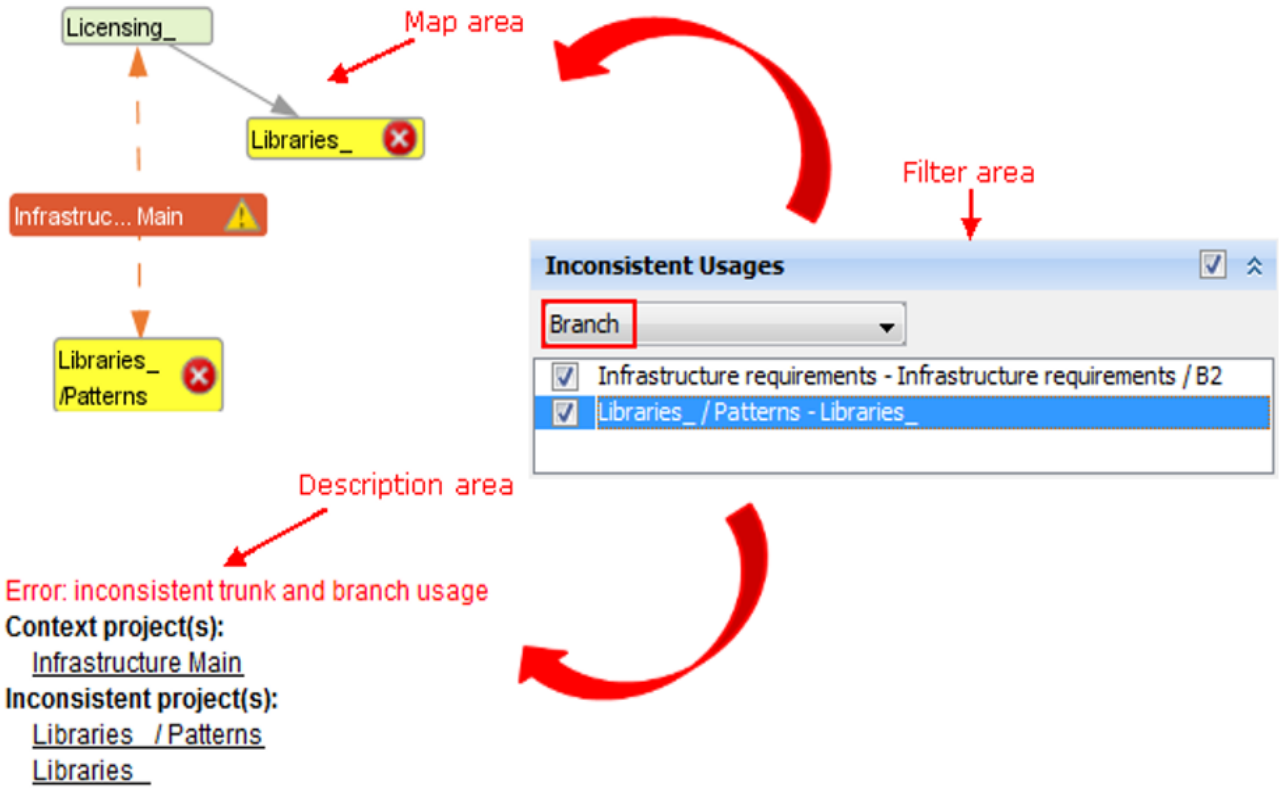


Changing module version

Resolving branch inconsistencies

Problem

You are using a version from the trunk and branch of the same project in your main project



Identifying branch inconsistency

Solution

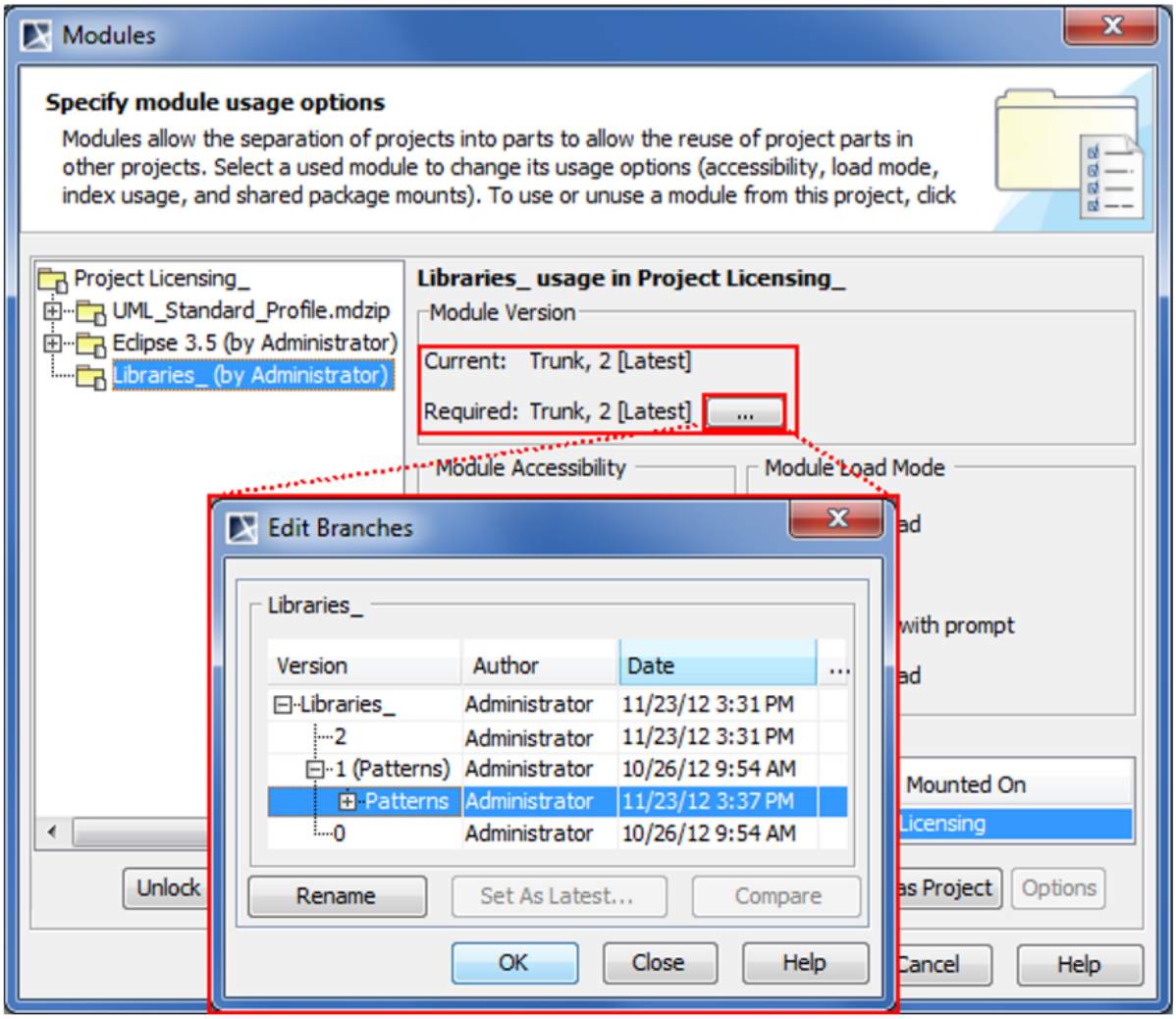
The Project Usage Map highlights these inconsistencies. You can then open projects with inconsistent usages and fix them by unifying the used project branch.

To resolve a branch inconsistency

> **Model Levels**
> Be sure to check and resolve the model level usages between the project and module.

1. Open a project using the projects of inconsistent version.
2. From the **Options** menu, select **Modules**. The **Modules** dialog opens.
3. Select a module which version you want to change, and click the **Lock** button.
4. In the **ModuleVersion** area, click the ... button. The **EditBranches** dialog opens.
5. Select the wanted branch or trunk.
6. Click **OK** when you are done.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

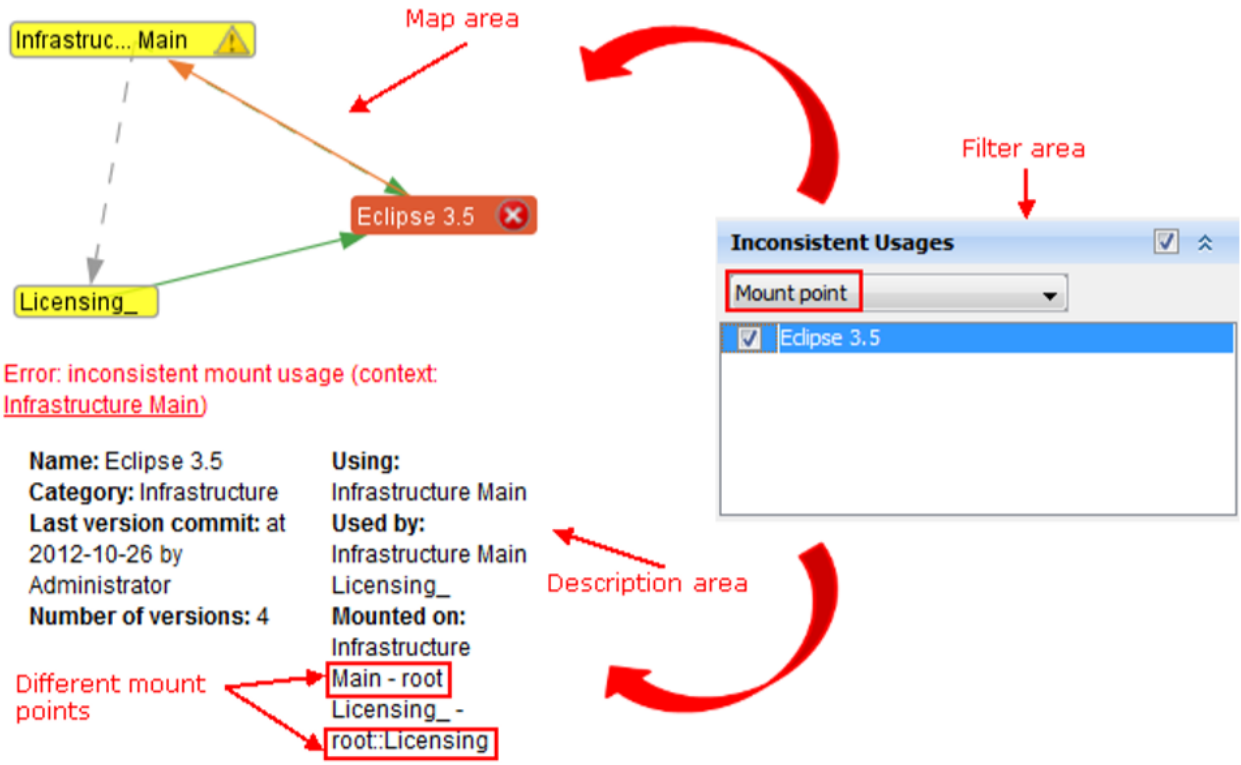


Changing module branch

Resolving mount point inconsistencies

Problem

You are mounting (mount - the other project usage in a particular package of the main project) the used project in different packages in your main project.



Identifying mount point inconsistency

Solution

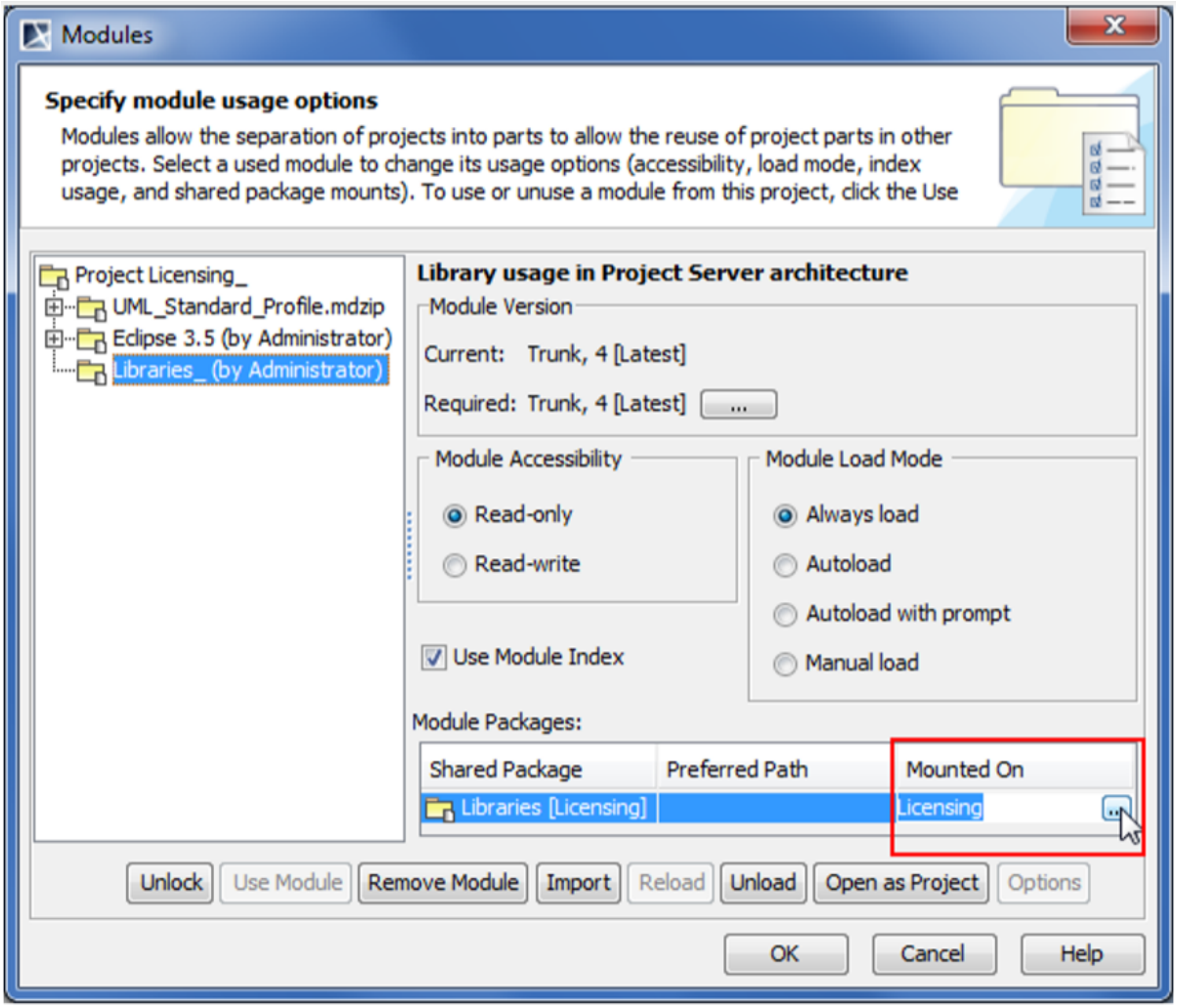
The Project Usage Map highlights these inconsistencies. You can then open projects with inconsistent usages and fix them by unifying the used project mounted package information.

To resolve a mount point inconsistency

> **Model Level**
> Be sure to check and resolve the model level usages between the project and module.

1. Open a project using the projects of inconsistent version.
2. From the **Options** menu, select **Modules**. The **Modules** dialog opens.
3. Select a module which version you want to change, and click the **Lock** button.
4. In the **Module Packages** table, select a wanted module.
5. In the **Mount On** cell, click the ... button. The **Select Package** dialog opens.
6. Select the wanted package and click **OK**.
7. Click **OK** when you are done.

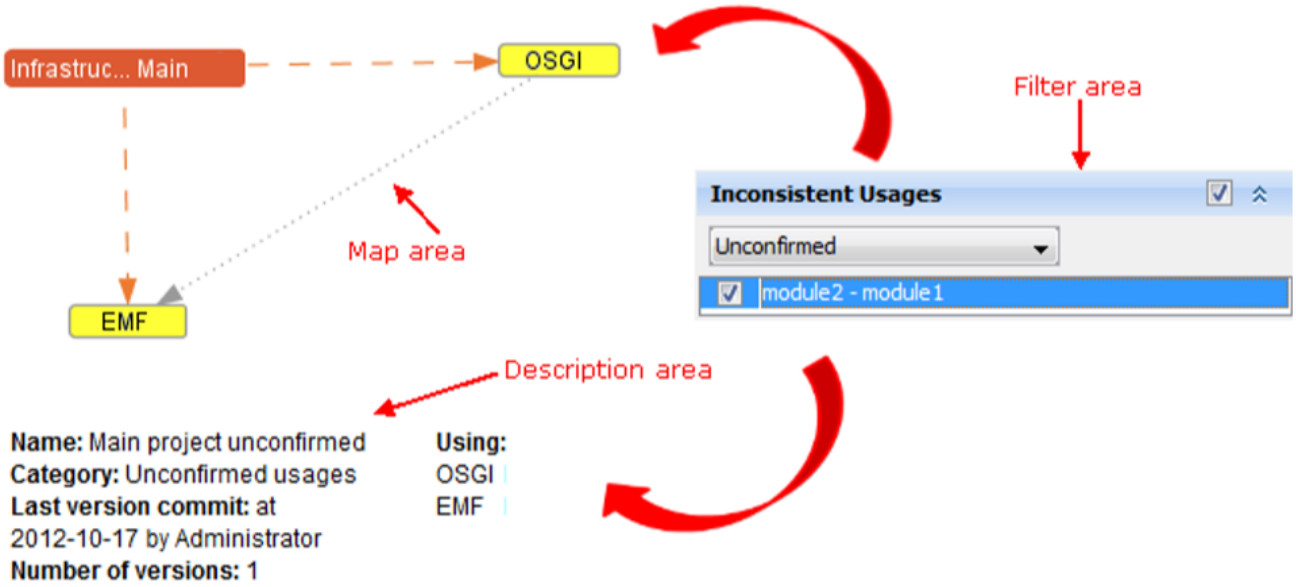
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Changing mount point

Resolving unconfirmed usages

This type of usages is created automatically.



Identifying unconfirmed usages

Solution

> **Model Level**
> Be sure to check and resolve the model level usages between the project and module.

There are two ways to solve unconfirmed module usage situation - either confirm it or reject it.

If the module usage A => B is good and necessary according to the end-user policy, it can be confirmed. I.e., the user-defined usage is to be created in place of the current unconfirmed automated module usage.

To do that, use the Confirm and use the module into <module_name> solver of the validation result. This solver opens the standard Use Module wizard and pre-selects the required module. When wizard is completed, the necessary user-defined module usage is created.

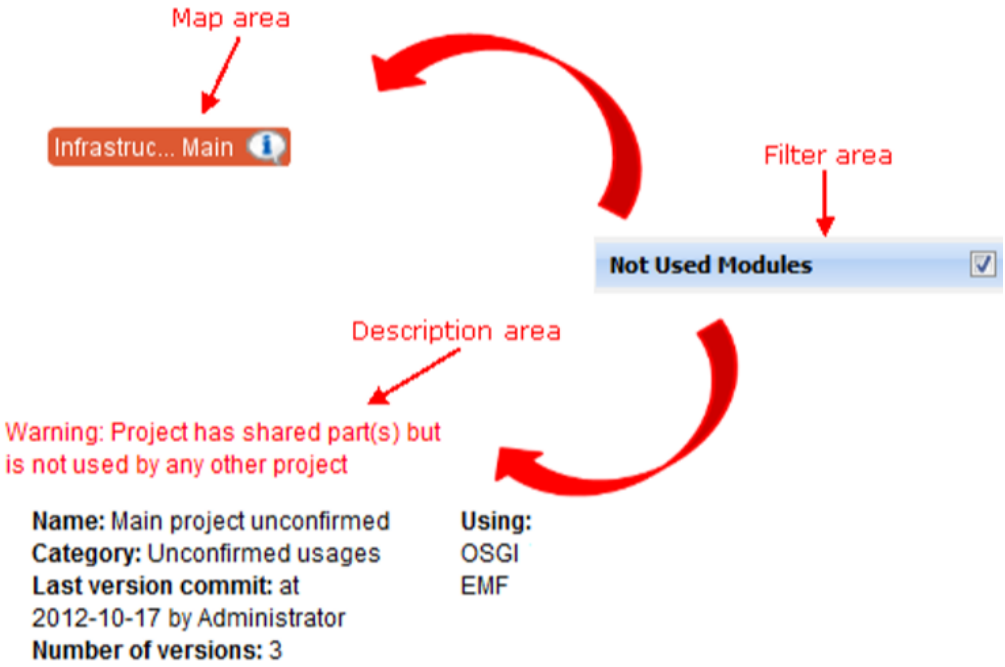
If the usage A => B is not good according to the end-user policy (for example – leads to module usage cycles or is incorrect because of semantically there should be no dependency between these modules), then it needs to be rejected and removed. To remove the usage, the model-level references, which are causing this automated module usage need to be changed – either removed or redirected to different elements.

> For more information about resolving the unconfirmed usages, see Resolving unconfirmed module usages.

Resolving not used modules problem

Problem

When the number of projects in the repository grows it is common for some of the projects to become outdated or not used anymore. You would prefer removing them BUT you are not sure if they are not used by some other, still-active project.



Identifying not used modules

Solution

> **Model Level**
> Be sure to check and resolve the model level usages between the project and module.

The Project Usage Map highlights unused modules. Based on this information, you can move all of the unused modules into the deprecated category or remove them entirely from the repository.

Related Pages:

- Project Usage Map(see page 1511)
  - Working with Project Usage Map(see page 1513)
  - Project Usage Map Window(see page 1516)
  - Solving Project Usage Problems(see page 1519)

Data partitioning on Teamwork Server

Data partitioning is the segmentation of a project into multiple independent parts or used projects. Since these used projects are saved using the same file format as regular projects, they can be opened and edited as separate projects. This approach allows easy reuse of the same basic model libraries in multiple projects.

Version of used projects
Please note that, in a server, project versions used on a project commitment (but not the latest used project version) are open.

Related pages

- Project Partitioning(see page 236)
- Creating and Managing Categories(see page 1380)
- MOF Support(see page 269)
- Use Server Module Wizard(see page 1544)

Creating used projects in Teamwork Server

Related pages

- Data partitioning on Teamwork Server(see page 1529)
  - Creating used projects in Teamwork Server(see page 1530)
  - Working with used server projects(see page 1531)

To create a used project using the Collaborate menu

1. Open a server project and lock it for editing(see page 1482).
2. From the **Collaborate** menu, select **Project Usages** > **Export Packages to New Server Project.** The **Export Packages to New Project** dialog(see page 1442) opens.

   You cannot export a package as a server project if another server project is mounted inside.

3. In the **All data** list, select one or more packages from which you want to create a server project and click the **Add** button. The packages will be added to the **Selected packages** list.
4. Click **OK.**

   In this step, you may be asked to confirm or cancel the dependency checking between the exported packages and the rest of the project (including shared packages that belong to the project and used projects). The dependency analysis step is described in Decomposing model(see page 237).

5. Enter the name of the project for Teamwork Server, if you need one.
6. Either select an existing category or type a new category name.
7. Click **OK.** The project will be created in Teamwork Server as a separate file.

To create a server project using the package shortcut menu

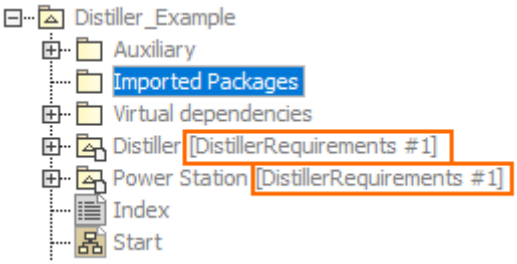
1. Open a server project and lock it for editing(see page 1482).
2. In the **Containment tree,** select one or more packages from which you want to create a server project.

3. From the package shortcut menu, select **Project Usages** and then click **Export Packages to New Server Project.** The **Export Packages to New Project** dialog(see page 1442) opens. The packages you have selected in the **Containment tree** are added to the **Selected packages** list.

   > You cannot export a package as a server project if another server project is mounted inside.

4. You may decide to add more packages. In this case, perform Step #3 described in the previous set of procedures.
5. Perform Steps #4 and step #5 described in the previous set of procedures.

Once the server project is created, all its packages and the elements they contain are read-only. A project name and version number are displayed in the brackets next to each exported package name in the Containment Tree.



Exported packages in a server project.

Working with used server projects

Related pages

- Data partitioning on Teamwork Server(see page 1529)
  - Creating used projects in Teamwork Server(see page 1530)
  - Working with used server projects(see page 1531)

In Teamwork Server, a project can be divided into several parts. You can perform several actions on each separate part of the model:

- Reuse in other server projects.
- Edit it separately.
- Export it as a server project to reuse it in other projects.

> You are not allowed to import a project into a server project if the imported project has cyclic dependencies or if the imported project is used by both the server project and another project that is used by the server project.

To use a server project into other server project

1. From the **Collaborate** menu, select **Project Usages** and then **Use Server Project**.
2. The **Use Server Project** dialog opens.
3. Select the project you want to use in your project. Click **Next.**
4. Specify usage options in the next step and click **OK.**

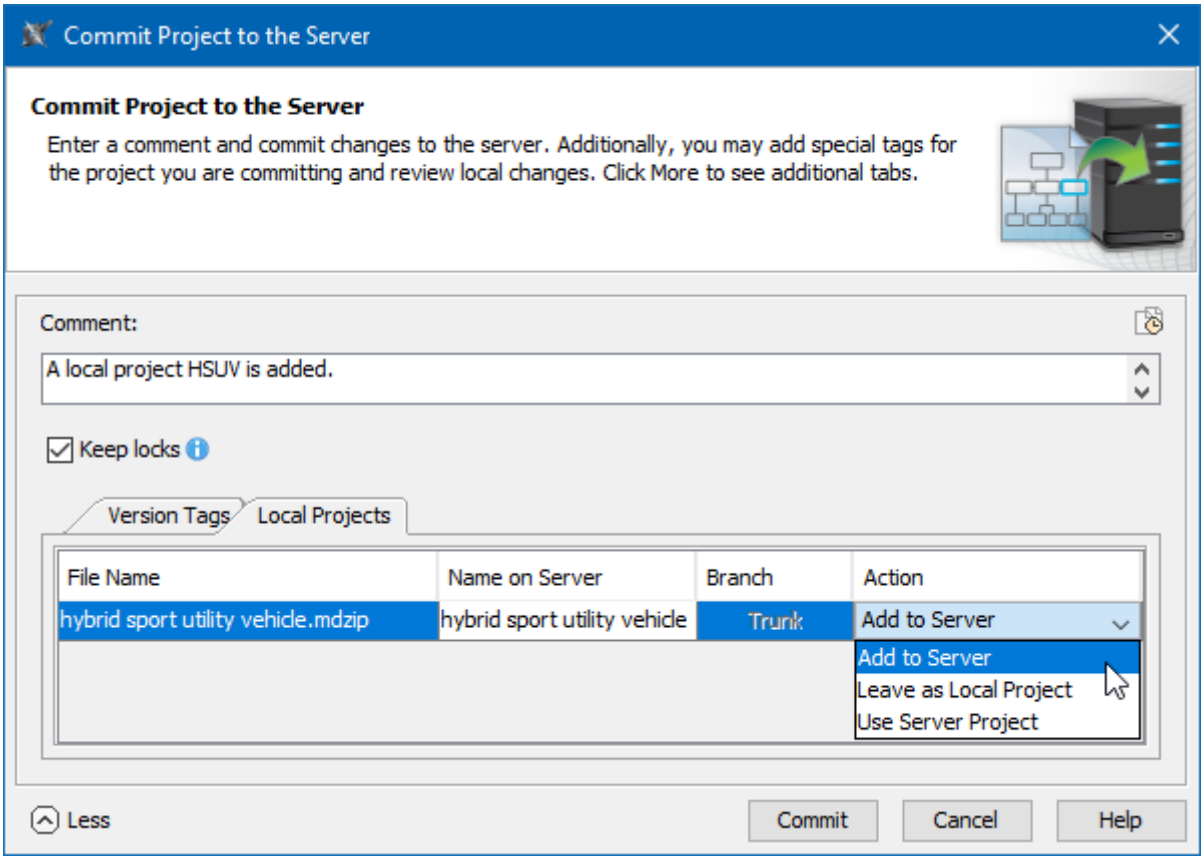
To commit a project containing local projects to Teamwork Server

---

1. From the **Collaborate** menu, select **Commit Changes to Server.** The **Commit Project to the Server** dialog opens.
2. If you want to save the local project as a separate server project, perform the following steps:
   a. Click **More** to extend the dialog.
   b. Click the **Local Projects** tab.
   c. In the **Action** list, select **Add to Server.**
   d. Perform this step for all used local projects you want to add to Teamwork Server.

   > Standard projects cannot be added to server as they already exist there.

3. Click **Commit.**



Local Projects tab in Commit Project to the Server dialog.

Now both the project and used local projects are committed to the Teamwork Server.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To edit a server project

1. In the **Containment tree,** select a shared package of a used project.
2. From the package shortcut menu, select **Project usages** > **Open.** The used project is opened as a new project.
3. Lock the project elements(see page 1482) you want to edit.
4. When unlocking an edited element, the **Commit Project to the Server** dialog opens. Add a comment and click **Commit.**

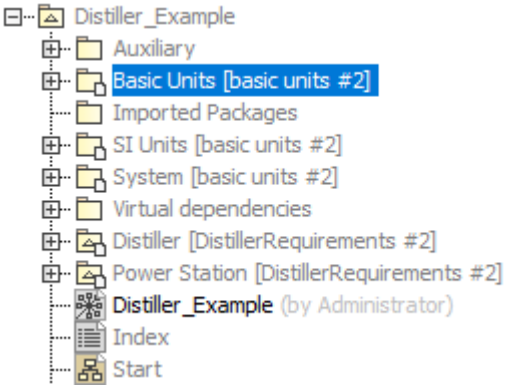
A new version of the used project on the Teamwork Server is created.

> Please note that a server project opens used project versions that were used on a project commitment (but not the latest version of the used project).

You must reload the used project to get the latest version to be used.

To reload a used server project

1. From the shared package of a used project shortcut menu, select **Project Usages** > **Reload.** The number of the latest version is shown next to the used project name.



This procedure allows you to use the latest version of a used project that was uploaded to the Teamwork Server.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Teamwork Server Dialogs

Here you will find descriptions about specific dialogs for Teamwork Server, which you will see only after connecting to the Teamwork server environment.

Related Pages:

- User management(see page 1534)
- Manage Projects Dialog(see page 1535)
- Open Server Project Dialog(see page 1539)
- Commit Project to the Server dialog(see page 1540)
- Project Versions dialog(see page 1541)
- Use Server Project wizard(see page 1544)

User management

Use the Add User dialog to create a new Teamwork user, and the Edit Users dialog to edit the Teamwork user. Both dialogs have the same structure; however, only the Login field in the Edit Users dialog is not editable. Teamwork Server users have their own user accounts (including login names and passwords assigned by the administrator) and various types of permissions. Depending on where the user accounts are stored, users can be either:

- **Native** - the user's account data is stored locally.
- **External** - the user's account data is stored in the external database (Subversion and/or LDAP). Only the login name of an external user is stored locally.

Add users

To open the Add User dialog

1. From the **Collaborate** menu, select **Users.**
2. Click the **Add** button.

Edit users

To open the Edit User dialog

1. From the **Collaborate** menu, select **Users.**
2. In the **Users** list, select the user and click the **Edit** button.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Element Name	Description
Login	The login name of the Teamwork user. Changing the user's login name is not allowed.
Name	The full name of the Teamwork user.
External User	The check box indicating whether the user is a native user or an external user. Do one of the following: • Leave the check box deselected for the native user. • Select the check box for the external user.
New Password	A new password. The field will be visible if you add or edit a native user (the External user check box is clear). To create or change a password for an external user, use an appropriate tool for managing the external database (Subversion, ClearCase or LDAP) wherein the user's account is stored.
Confirm New Password	The new password retyped. This field will be visible if you add or edit a native user (the External user check box is clear).

- Teamwork Server Dialogs(see page 1534)
  - User management(see page 1534)
  - Manage Projects Dialog(see page 1535)
  - Open Server Project Dialog(see page 1539)
  - Commit Project to the Server dialog(see page 1540)
  - Project Versions dialog(see page 1541)
  - Use Server Project wizard(see page 1544)

Manage Projects Dialog

You can use the **Manage Projects** dialog to manage projects, assign them to other Teamwork users, and set user permissions for the system and specific projects.

To open the Manage Project dialog

- From the **Collaborate** menu, select **Projects...**
- On the **Collaboration** toolbar, click the **Projects** button.

Element Name	Description
Buttons of the Category Toolbar	
Categorized View	Displays the categorized structure of the project list.
Alphabetical View	Displays the plain project list without any category.
Show Empty Categories	Shows the categories that have no projects.
Expand	Expands all categories. The button is available only in the categorized view.
Collapse	Collapses all categories. The button is available only in the categorized view.
Project Info	Opens the Versions dialog. In this dialog, you can see project's information, such as comment, tags, modules, and meta information.
Project History	Opens the Project Versions dialog(see page 1541). The button is available only when the project is selected.
Create Category	Opens the Create Category dialog for creating a new category. The button is available only in the categorized view.
Remove Category	Opens the dialog asking to confirm or cancel the removal of the selected category. The button is available only in the categorized view when the category or the project is selected.
Rename Category	Opens the Rename Category dialog for renaming the selected category. The button is available only in the categorized view when the category or the project is selected.
Move to Category	Opens the Move to Category dialog for choosing the category to move the selected project into. The button is available only in the categorized view when the project is selected.
Columns of the Projects List	

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Name	• The name of the project. • The name of the category.
Buttons of the Projects list	
Open	Opens the selected project.
Rename	Opens the Rename Dialog(see page 1538) for renaming the project. The button is available only when the project is selected.
Add	Opens the Add Server Project dialog for creating a new server project.
Remove	Opens the dialog to confirm or cancel the removal of the project from the Teamwork Server. The button is available only when the project is selected.
Create Branch	Opens the Create Branch dialog for creating a new branch for the selected project or its branch. The button is available only when the project is selected.
Project Usage Map	Opens the Project Usage Map(see page 1511) of the selected project. The button is available only when the project is selected.
Columns of the Assigned users list	
Login	The login name of the user assigned to the selected project.
Name	The full name of the user assigned to the selected project.
Permissions list	A list of permissions given to the selected user for a specific project.
Columns of the Available users list	
Login	The Teamwork user's login name.
Name	The Teamwork user's full name.
Button between the user lists	

<<	Moves the selected users from the Available Users list to the Assigned Users list.
>>	Move the selected users from the Assigned Users list to the Available Users list.

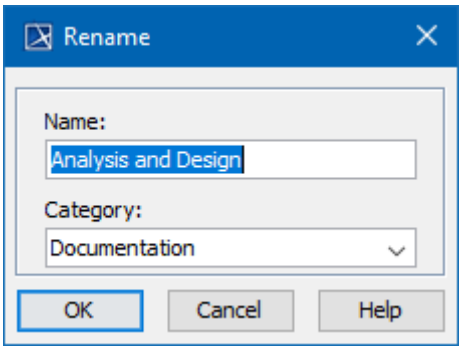
- Teamwork Server Dialogs(see page 1534)
  - User management(see page 1534)
  - Manage Projects Dialog(see page 1535)
  - Open Server Project Dialog(see page 1539)
  - Commit Project to the Server dialog(see page 1540)
  - Project Versions dialog(see page 1541)
  - Use Server Project wizard(see page 1544)

Rename Dialog

Use the Rename dialog to change the name of a project or module used in the project. In server projects, it is safe to change module names as modules are identified by a module ID in the Teamwork server, not by a module name.

To open the Rename dialog:

1. Open the **Manage Projects** dialog.
2. Select the project whose name you want to change.
3. Click the **Rename** button.



Rename dialog

Element	Function
Name	Type the new project name.
Category	Select the category for the project or create a new one.

-

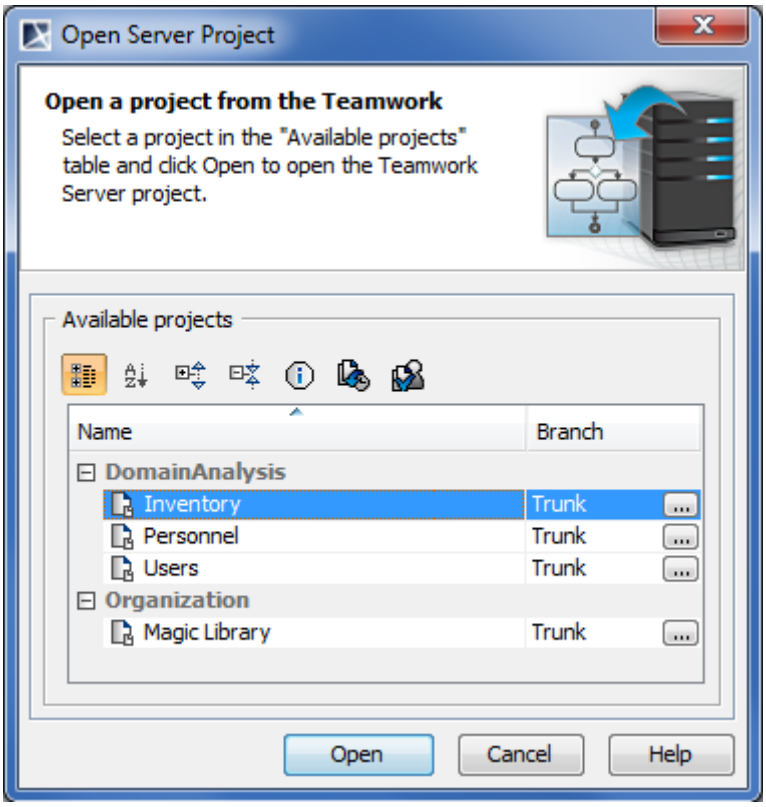
Teamwork Server Dialogs(see page 1534)
- User management(see page 1534)
- Manage Projects Dialog(see page 1535)
- Open Server Project Dialog(see page 1539)
- Commit Project to the Server dialog(see page 1540)
- Project Versions dialog(see page 1541)
- Use Server Project wizard(see page 1544)

Open Server Project Dialog

Use the Open Server Project dialog to open and manage projects.

To open the Open Server Project dialog

- From the **Collaborate** menu, select **Open Server Project**.
- Press CTRL+SHIFT+O.



Open Server Project dialog

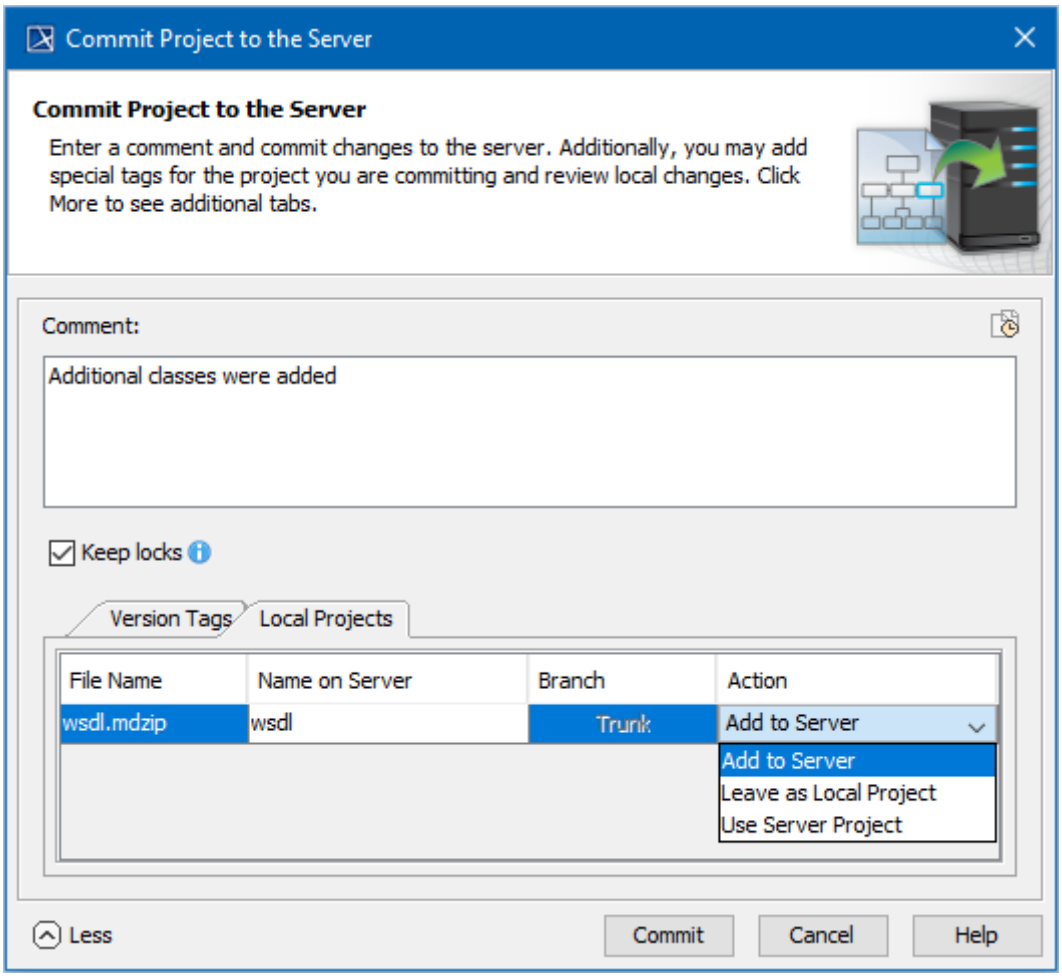
- Teamwork Server Dialogs(see page 1534)
  - User management(see page 1534)
  - Manage Projects Dialog(see page 1535)
  - Open Server Project Dialog(see page 1539)
  - Commit Project to the Server dialog(see page 1540)
  -

Project Versions dialog(see page 1541)
- Use Server Project wizard(see page 1544)

Commit Project to the Server dialog

To open the Commit Project to the Server dialog

- From the **Collaborate** menu, select **Commit Project.**
- Press **CTRL+K.**



Commit Project to the Server dialog

Element Name	Description
Choose Comment	Opens a Choose comment dialog with the recently added comments. You can select a comment from the dialog.
Comment	Add information about the project if needed.
Keep Locks	Select this option if you want to keep all the currently locked elements and modules locked.
Automatically Save Project Locally	Updates locally saved project with the latest committed version. Select this option, if you want to keep your locally saved project copy up-to-date. This option becomes available when you save a server project to your computer for the first time.
Version Tags	Add a version tag if needed.
Local Modules	List local modules that were used in the server project. Before committing, select an action you want to perform for modules.
Options	See more or fewer options.

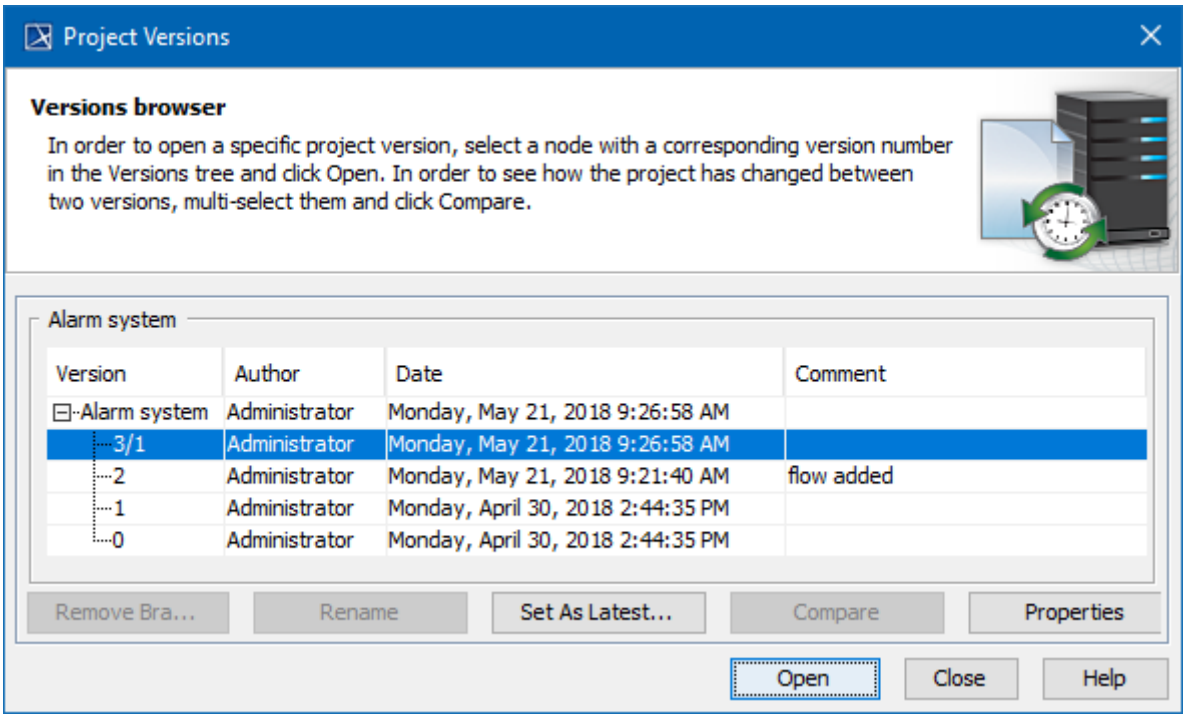
- Teamwork Server Dialogs(see page 1534)
  - User management(see page 1534)
  - Manage Projects Dialog(see page 1535)
  - Open Server Project Dialog(see page 1539)
  - Commit Project to the Server dialog(see page 1540)
  - Project Versions dialog(see page 1541)
  - Use Server Project wizard(see page 1544)

Project Versions dialog

In the Project Versions dialog, project versions are listed from oldest (on the top) to newest.

To open the Project Versions dialog

1. From the **Collaborate** menu, select **Open Server Project.**
2. In the **Open Server Project** dialog, select a project and click the **Project History** button.



Project Versions dialog

Element Name	Description
Version	The version number of a project. In the example above, the last project version number is denoted as “3/1”. This is the result of setting a previous project version as the newest one. In this particular example, “3” denotes the current project version number and “1” denotes the number of the version, which has been set as the newest one.
Author	The login name of the user who created the project version.
Date	The date and time the project version was created.
Comment	The description of changes in the version. You can make a version comment when committing a version.
Create Branch	Opens the Create Branch dialog for creating a new branch for the selected version or branch. The button is available only if a version or branch is selected.

Element Name	Description
Remove Branch	Opens the dialog to confirm or cancel the removal of the selected branch. The button is available only when a branch is selected. You will not be allowed to undo this action.
Rename	Indicates the branch name to rename. The button is available only when a branch is selected.
Set As Latest	Set the selected project version as the current version. The created version number consists of two parts: the current version number and the version number set as the current version. If a user does not have the rights to edit the project, this button is disabled.
Compare	Compares two selected branches. If there are differences between project branches, the result is presented in the Difference Viewer dialog. The button is available only when two project branches are selected (use the SHIFT or CTRL keys).
Properties	Display information about versions of a project.
Open	Open the selected version of the project. You can make changes only to the latest version of the project. Earlier versions are read-only.
Close	Exit the dialog.
Help	Documentation for the project version will be opened**.**

- Teamwork Server Dialogs(see page 1534)
  - User management(see page 1534)
  - Manage Projects Dialog(see page 1535)
  - Open Server Project Dialog(see page 1539)
  - Commit Project to the Server dialog(see page 1540)
  - Project Versions dialog(see page 1541)
  - Use Server Project wizard(see page 1544)

Version Information dialog

To open the Version Information dialog

1. Open the Project Versions dialog (see the procedure Project Versions dialog(see page 1541))
2. Select a project and click **Properties**.

Element Name	Description
Version	The version number of a project.
Author	The login name of the author who created the project.
Date	The date and time the selected project version was committed.
Comment	Allow users to type optional descriptions about changes in the committed version. You can enter or edit comments for any project version.
Tags	Displays modules used in the selected project version.
Modules	Displays modules used in the selected project version.
Meta Information	Displays the project's information, such as required plugins and used modules.

- Teamwork Server Dialogs(see page 1534)
  - User management(see page 1534)
  - Manage Projects Dialog(see page 1535)
  - Open Server Project Dialog(see page 1539)
  - Commit Project to the Server dialog(see page 1540)
  - Project Versions dialog(see page 1541)
  - Use Server Project wizard(see page 1544)

Use Server Project wizard

With the Use Server Project wizard, you can add a selected server project, such as a library, profile, or other user-defined project part.

To open the Use Server Project wizard

- On the **Collaborate** menu, click **Server projects** > **Use Server Project.**
- On the **Options** menu, click **Project Usages.** In the opened **Used Projects** dialog, click the **Use Project** button.

The wizard consists of two steps:

1. Selecting a project.
2. Specifying selected project settings.

Step #1: Selecting a project

After the wizard is opened, the list of available server projects appears.

Select the server project you need and do one of the following:

- Click **Next,** if you need to change default project settings. The next wizard step opens.
- Click **Finish.** The selected server project is added to your project.

Dialog Element Name	Element Type	Description
Name	Text Box	A project name.
Branch	Text Box	The number of the latest version.
Project Description	Text Box	A description of the last project version.

Step #2: Specifying usage settings

In this step, you can specify shared project accessibility and packages in which the corresponding shared project is placed.

Click **Finish** after you have specified project settings. The selected server project is added to your project

Dialog Element Name	Element Type	Description
Accessibility	Option Button	Specifies the project access properties: • Read-only: projects are loaded for reading only and cannot be edited. Elements of the project can be used and referenced in the project. Usually, libraries are used in the read-only accessibility mode in projects. • Read-write: projects can be directly edited in the project where the projects are used.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Dialog Element Name	Element Type	Description
Load Mode	Option Button	By selecting an appropriate load mode, you can control how memory is used by a program and project: • Always load (default). A selected project is always loaded when the project is opened. • Autoload. A selected project is not loaded when the project is loaded. MagicDraw monitors user activities in the project and loads a particular project on demand by the project. • Autoload with prompt. This mode is similar to an autoload mode. The difference is that MagicDraw asks the user for confirmation before loading it. • Manual load. A selected project is not loaded when a project which uses this project is loaded. The model integrity is not broken, as all required elements of the project exist, just simplified versions (that is, loaded as proxies) of the elements are used in the project. This load mode is recommended for all projects that are stable or rarely modified. **Load Mode** Reload the project after changing the project load mode.
Use Index	Check Box	Select this check box to load indexed elements from the not loaded project. This is valid only when the project is not loaded. Using indexed elements in the project increases the performance, as only simplified versions (that is, proxies) of elements are loaded.
Packages	Table	This table lists all shared packages of the project and paths where these packages are mounted in the project.
Shared Package	Text Box	The name of a shared package.
Preferred Path	Text Box	Suggests where to mount the package in the project. This location is used by default if not otherwise specified.
Mounted On	Text Box	Displays the real location where the package is currently mounted in the project. Click the ... button to change a package location.

- Teamwork Server Dialogs(see page 1534)
  - User management(see page 1534)
  -

Manage Projects Dialog(see page 1535)
- Open Server Project Dialog(see page 1539)
- Commit Project to the Server dialog(see page 1540)
- Project Versions dialog(see page 1541)
- Use Server Project wizard(see page 1544)

Report Wizard

Report Wizard is a report engine for MagicDraw version 14.0 and greater. It is designed to solve several problems of our legacy report engines (*XSL/XSLT* and *JPython*).
The Report Wizard report engine is built on top of Velocity Engine (Open Source Templating engine) and is integrated with the MagicDraw application. To make the best of Report Wizard, you need to understand the Report Wizard UI, the Velocity Template Language (*VTL*), the application's Open API, and the helper modules.
Report Wizard supports text-based templates to generate reports. Each report file format depends on the type of the templates. The type of template files that Report Wizard supports includes *plain text, RTF, HTML, Office Open XML (ISO/IEC 29500:2008)*, *OpenDocument format (ISO/IEC 26300)*, and *XML template (DocBook or FO)*. It also supports the MS Word template in VBA Editor on Windows 8.

All commercial MagicDraw editions will have full use of all features within Report Wizard. The MagicDraw Community edition allows you to generate reports with watermarks.

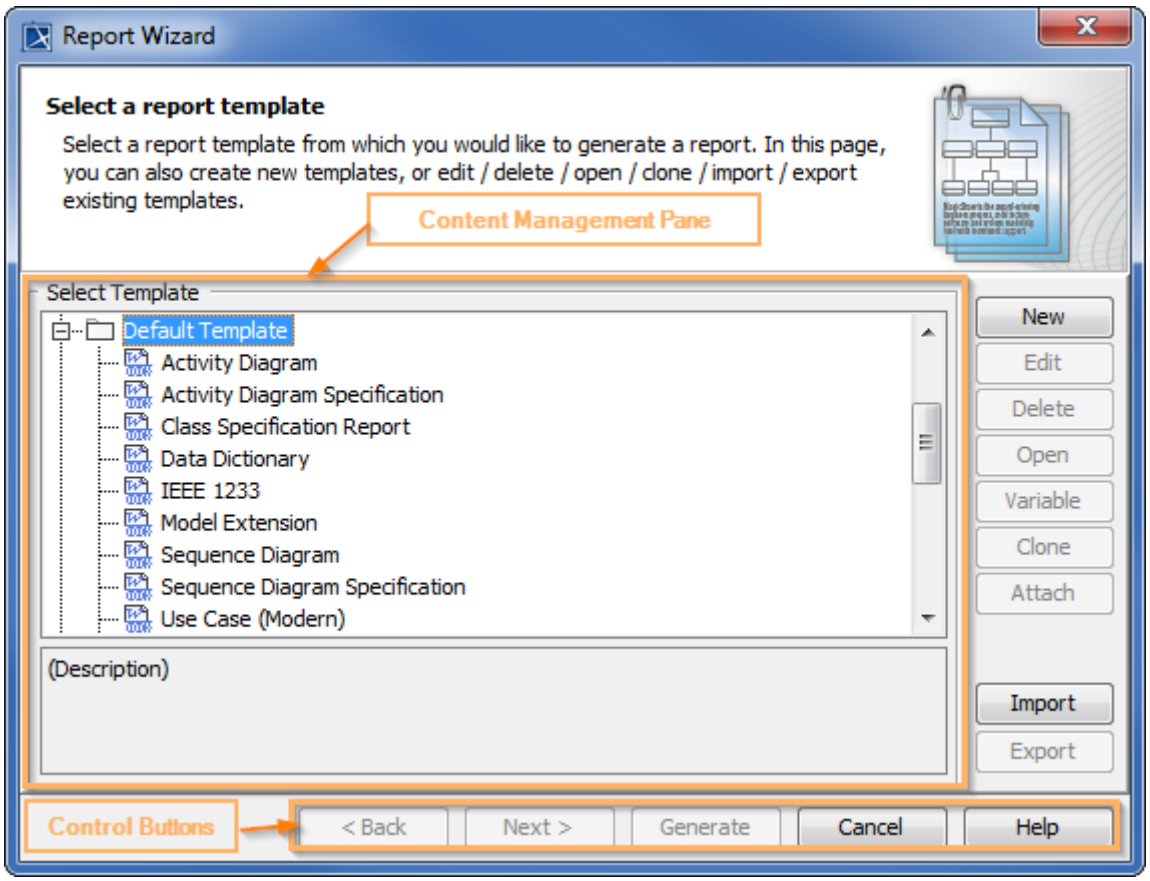
Before generating a report, you need to open Report Wizard.

How to open Report Wizard

- On the **Tools** menu, click **Report Wizard**. The **Report Wizard** dialog will appear.

Report Wizard dialog

The Report Wizard dialog, see figure below, consists of 2 panes: **Control buttons** and **Content Management pane**.



The Report Wizard Dialog.

Control buttons

There are 4 control buttons

- The < Back button is used for proceeding to the previous content management pane.
- The Next > button is used for proceeding to the next content management pane.
- The Generate button is used for generating a report.
- The Cancel button is used for cancelling the report generation process.

Content Management pane

This pane is used for managing the template content and includes the following sub-panes:

- Template Management Pane(see page 1549)
- Report Data Management Pane(see page 1566)
- Select Element Scope pane(see page 1585)
- Generate Output pane(see page 1587)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Click the < Back or Next > button to go to a specific pane.

Related pages

Template Management Pane

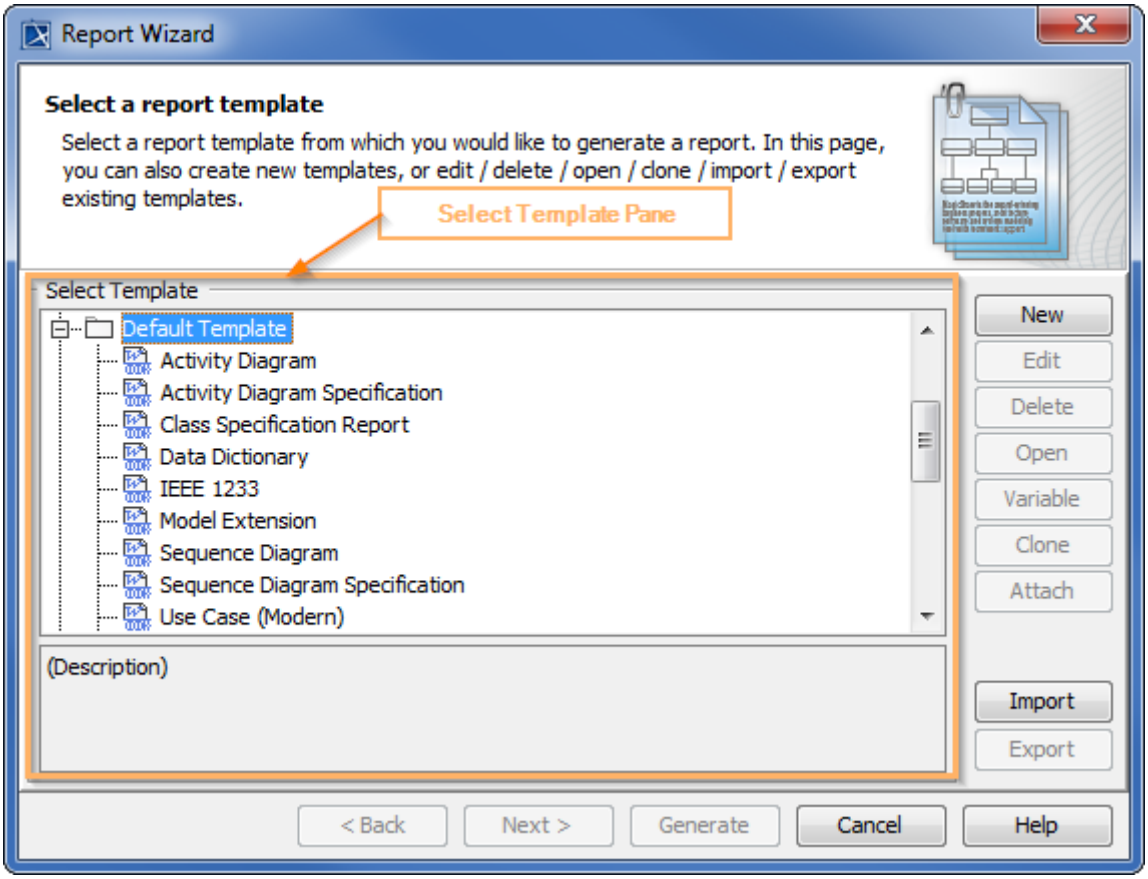
On this page

- Using the Template Management Pane(see page 1550)
- Working with a Template(see page 1551)
- Working with a template variable(see page 1561)
- Glossary(see page 1566)
- More information about working with templates in Report Wizard(see page 1566)

Report Wizard uses templates to generate reports. Each report file format depends on the type of template files. The type of template files that Report Wizard supports includes *plain text, RTF, HTML, Office Open XML (ISO/IEC 29500:2008)*, *OpenDocument format (ISO/IEC 26300)*, and *XML template (DocBook or FO)*.

Report Wizard provides predefined templates such as Use Case, Model Extension, Data Dictionary, IEEE 1233, Class Specification Diagram, Business Process Diagram, and Web Publisher templates. The report template management pane in the **Report Wizard** dialog lists all of the report templates from which you can select to manage or generate your report.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Template options, which are highlighted in orange, are available to choose from in the Report Wizard dialog.

Using the Template Management Pane

In this pane, you can create a new template, or use an existing one. The following table displays the buttons from which you can choose while working with your template. Additionally, the control buttons in the lower part of the dialog < Back, Next >, and Generate help you cancel an action, go to the next step, and generate a report.

Button	Function
New	Opens the **New Template** dialog through which you can create a new template.
Edit	Edits a template and saves it in Report Wizard.
Delete	Deletes a template from Report Wizard.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

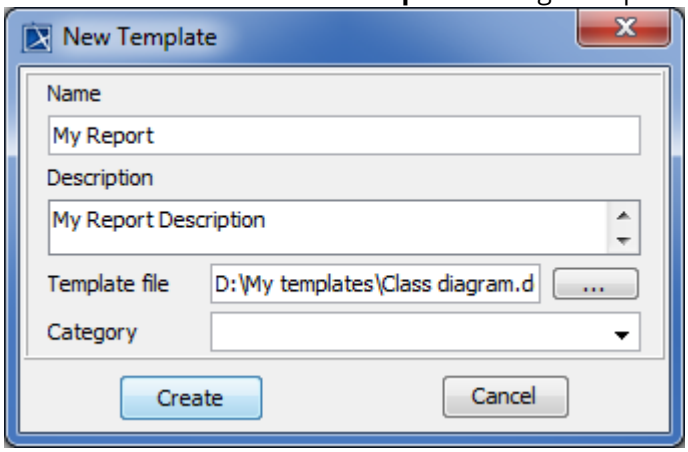
Button	Function
Open	Opens a template file in the default editor.
Variable	Creates a new template variable and modify or delete an existing template variable in the **Template Variable** dialog.
Clone	Clones a selected template.
Attach	Copies a selected template and adds it to the project file.
Extract	Extracts an attached template to a local report templates folder.
Import	Imports a template.
Export	Exports a template.

Working with a Template

The following instructions illustrate how to use each button in the Report Wizard to create, select, and manage a template.

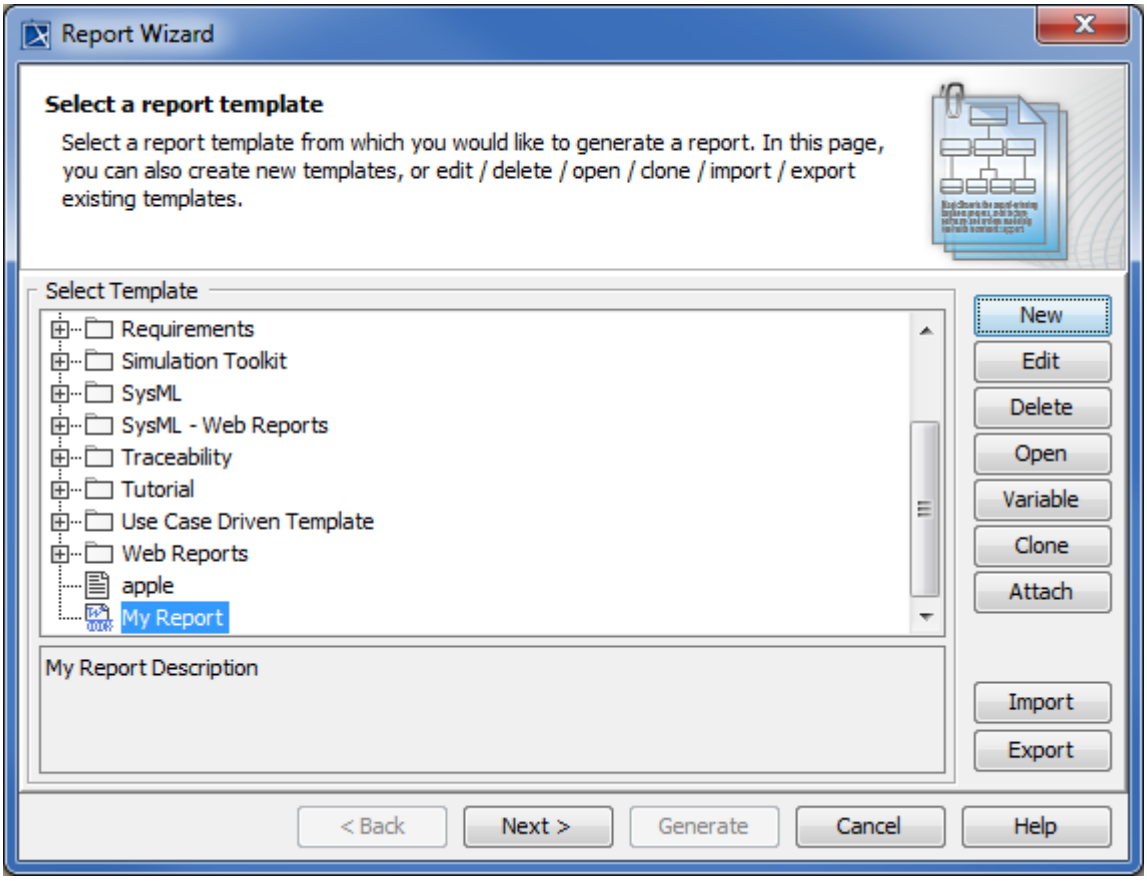
To create a template

1. On the MagicDraw main menu, click **Tools** > **Report Wizard**. The **Report Wizard** dialog will open.
2. Click New. The **New Template** dialog will open.



Creating a new template in the New Template dialog.

3. Enter the template name and description.
4. Click ... to select the location of the new template.
5. Select the template file location and click Select.
6. Click Create. The template file will be created in the **Select Template** pane.

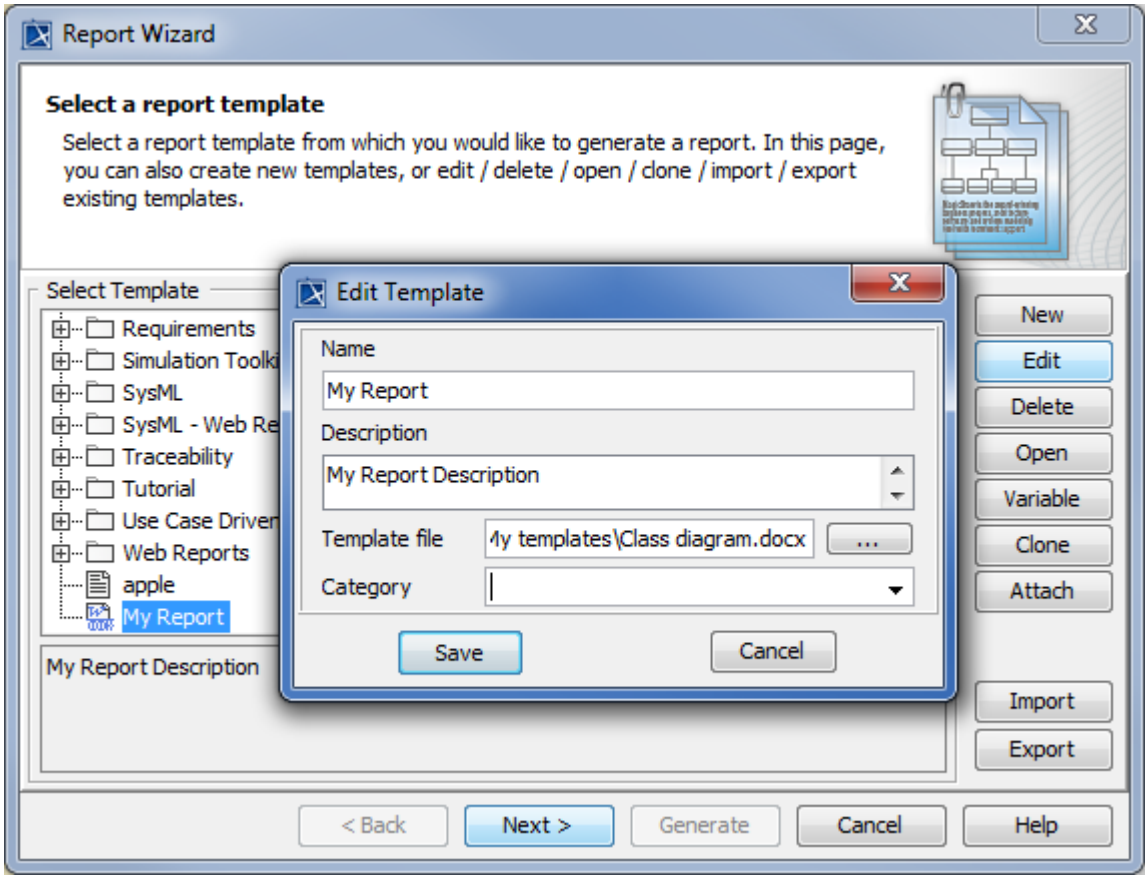


The template file is created in the Select Template pane.

To edit a template

---

1. In the **Report Wizard** dialog, select a template and click Edit. The **Edit Template** dialog will appear.



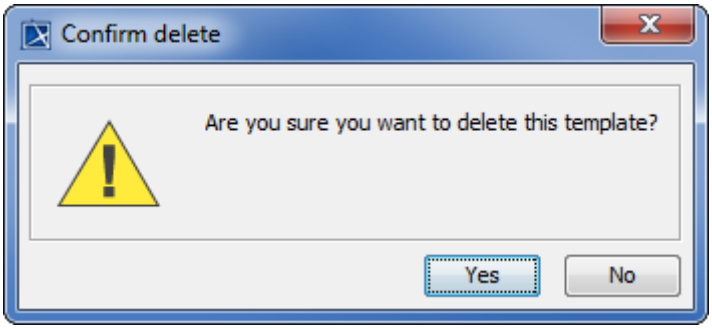
Editing a template in the Edit Template dialog.

2. Edit the template name, description, category, and locate the template file's location.
3. Click Save.

To delete a template

---

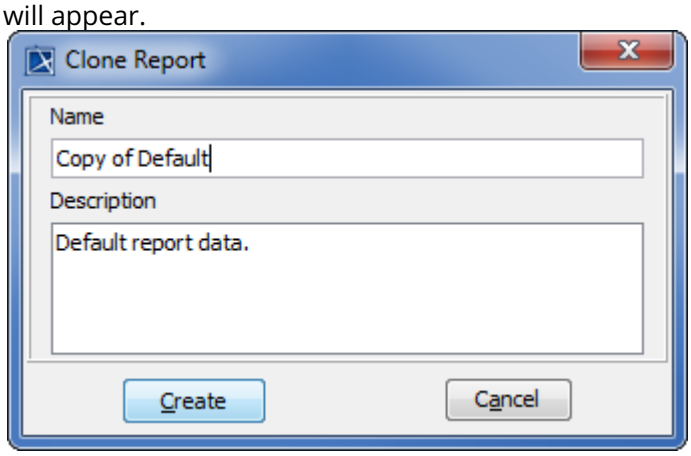
1. In the **Report Wizard** dialog, select a template and click Delete. The **Confirm delete** dialog will appear.
2. Click Yes to delete the selected template from the template list, or click No to cancel the operation.



The Confirm delete dialog.

To clone a template

1. In the **Report Wizard** dialog, select a template and click Clone. The **Clone Template** dialog will appear.



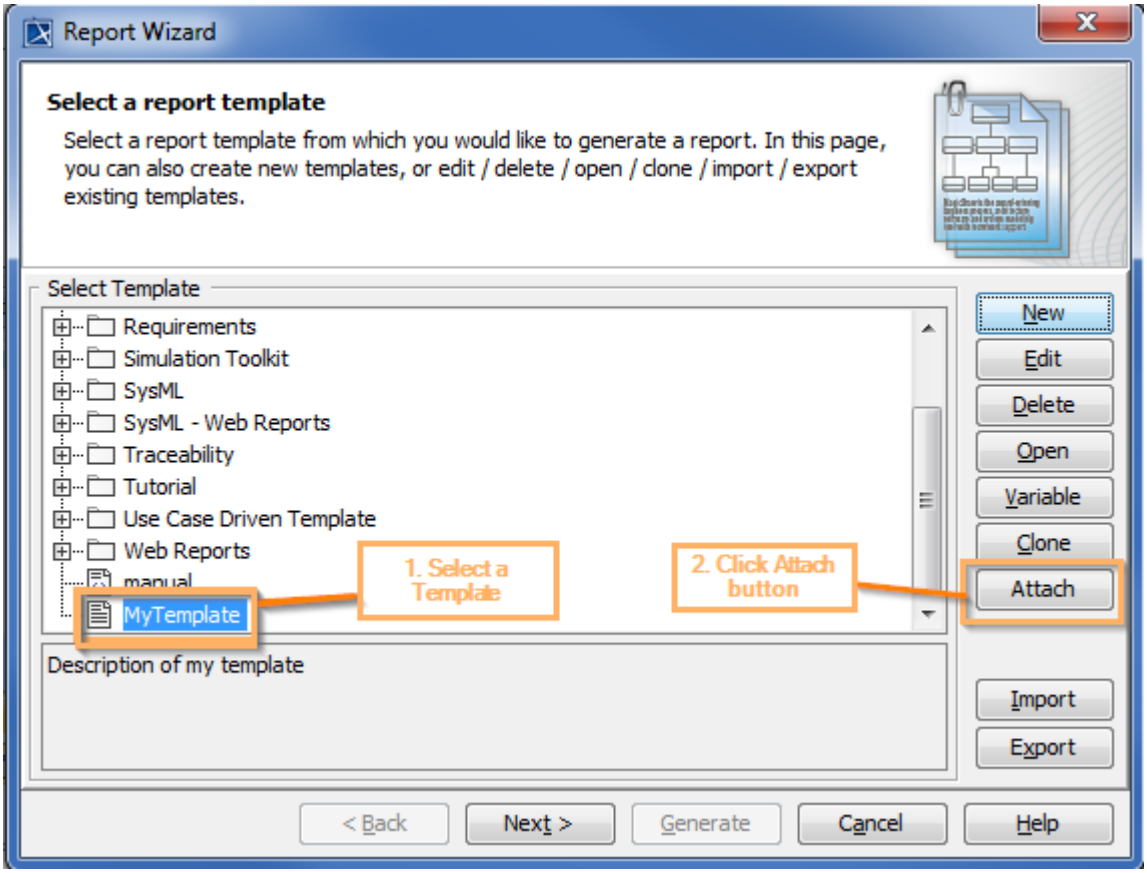
Cloning a template in the Clone Template dialog.

2. Enter the name and description. The name of the cloned template should begin with **Copy of** (name of template).
3. Click Create to clone the template.

To copy a template and add it to the project file

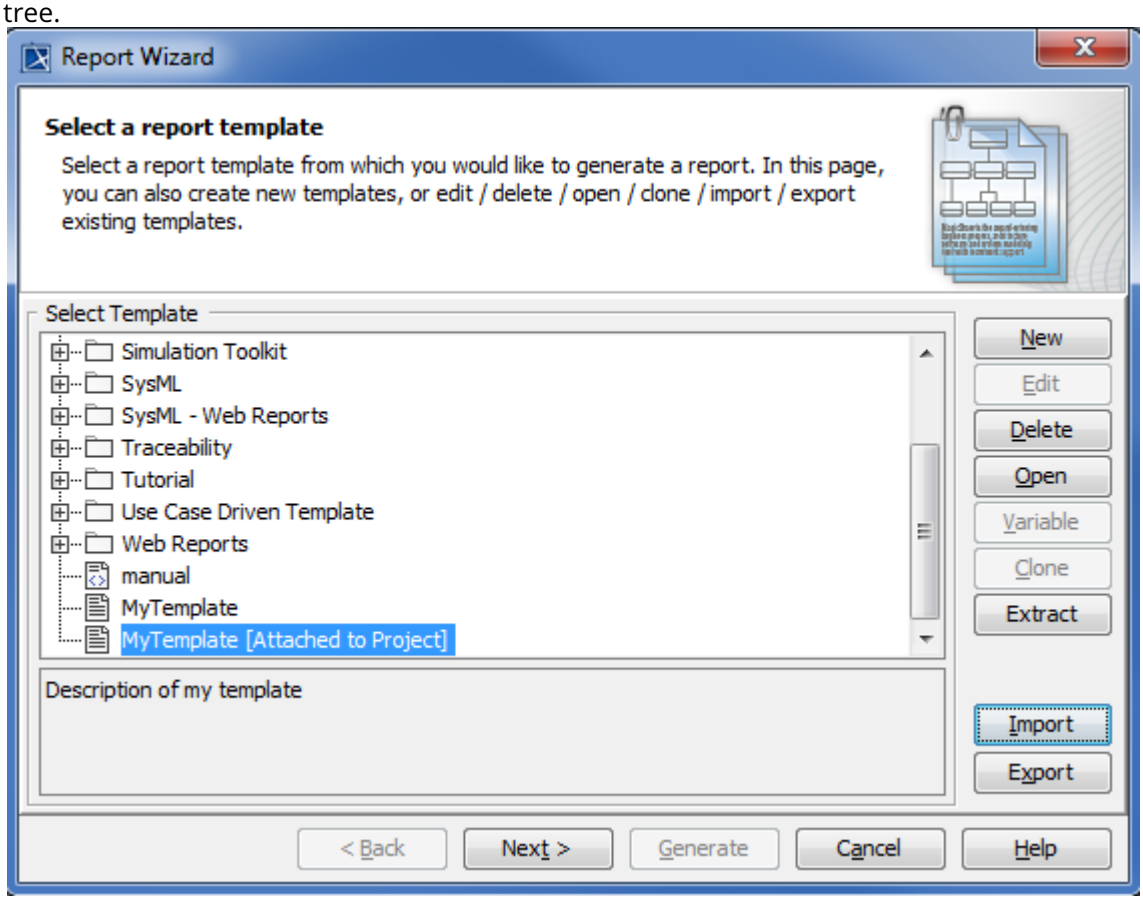
1. Select a template in the **Report Wizard** dialog.
2. Click Attach to make a copy of the selected template and add it to the project's files, so anyone with access to the project can generate a report.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

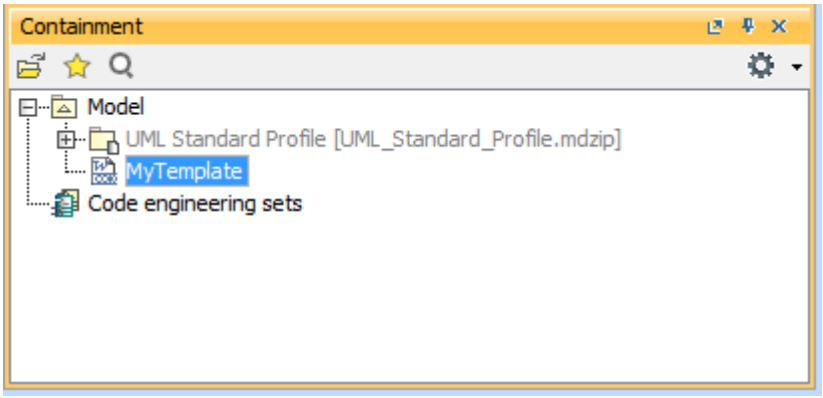

Report Wizard
Select a report template
Select a report template from which you would like to generate a report. In this page, you can also create new templates, or edit / delete / open / clone / import / export existing templates.
Select Template
Requirements
Simulation Toolkit
SysML
SysML - Web Reports
Traceability
Tutorial
Use Case Driven Template
Web Reports
manual
MyTemplate
1. Select a Template
2. Click Attach button
New
Edit
Delete
Open
Variable
Clone
Attach
Description of my template
Import
Export
< Back
Next >
Generate
Cancel
Help

The steps to attach a template to a project file.

The new template will be created and displayed in the **Report Wizard** dialog and the Containment tree.



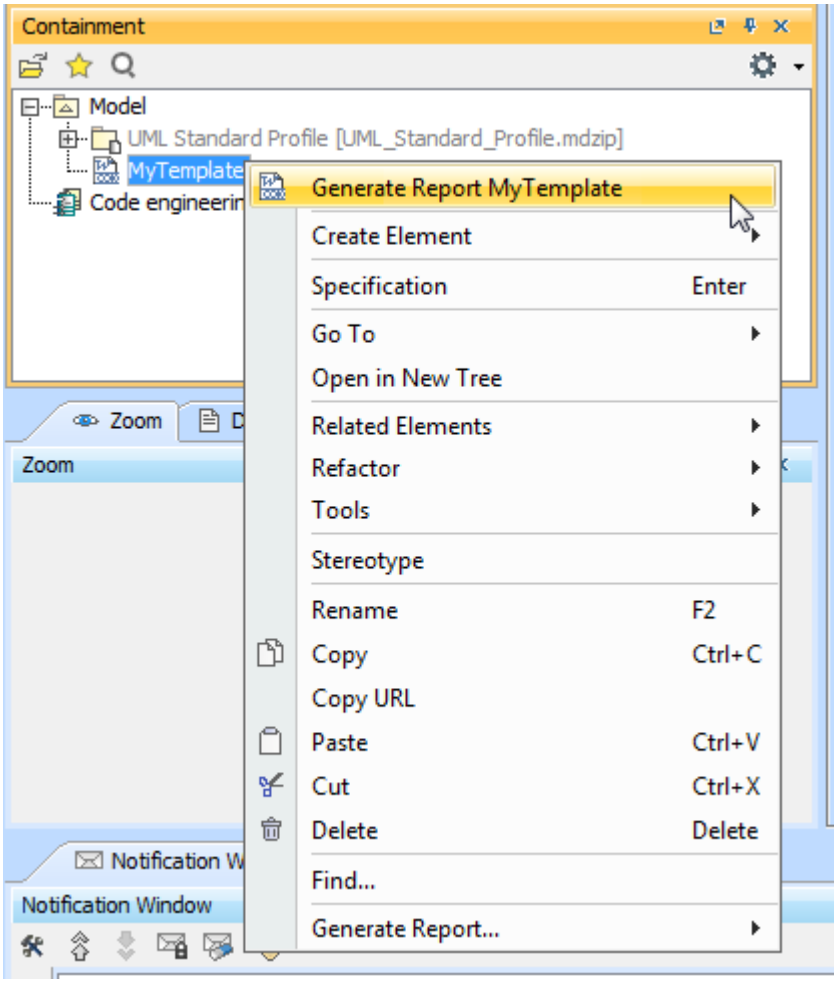
Result of attaching a template in the Report Wizard dialog.



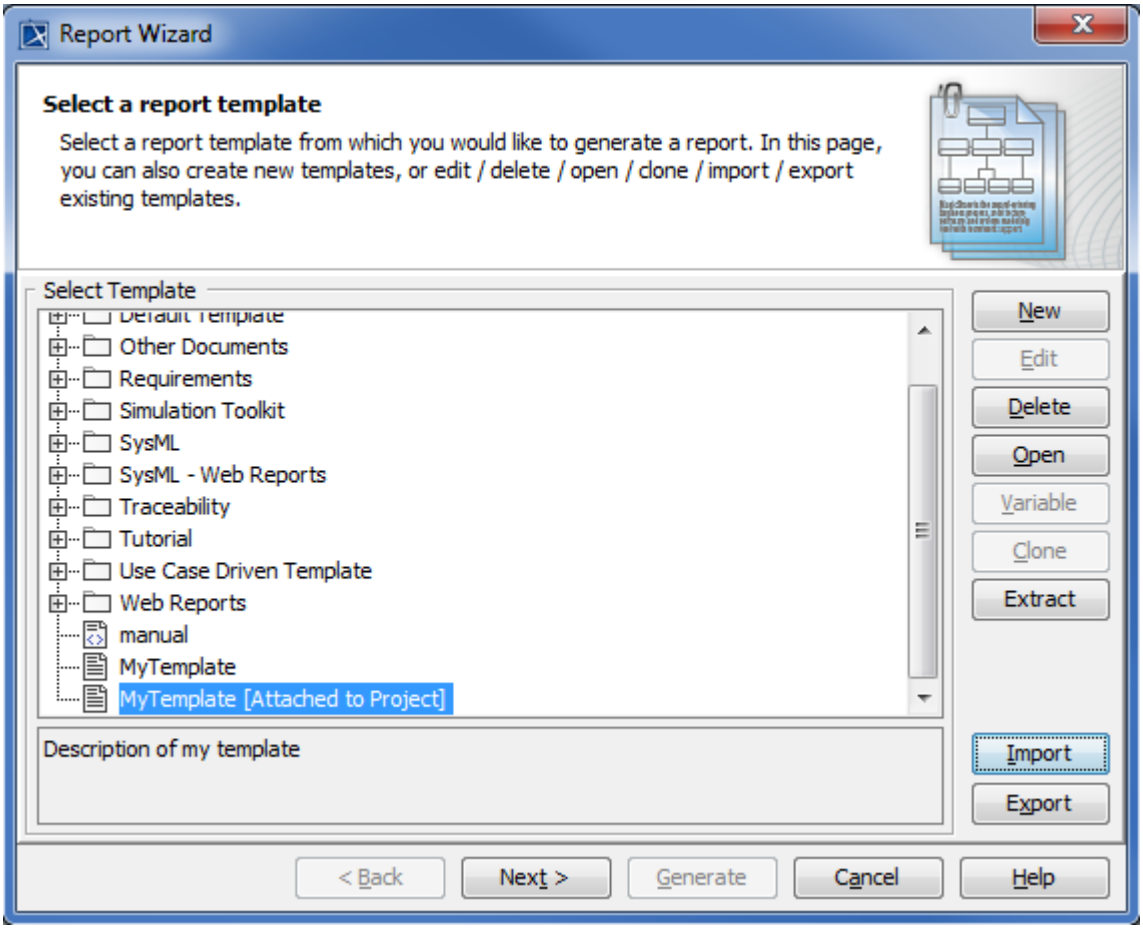
Result of attaching a template in the Containment tree.

You can right-click the attached template in the Containment tree, and select **Generate Report{attached template name}** to open the **Report Wizard** dialog.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.


Containment
Model
UML Standard Profile [UML_Standard_Profile.mdzip]
MyTemplate
Code engineerin
Generate Report MyTemplate
Create Element
Specification Enter
Go To
Open in New Tree
Related Elements
Refactor
Tools
Stereotype
Rename F2
Copy Ctrl+C
Copy URL
Paste Ctrl+V
Cut Ctrl+X
Delete Delete
Find...
Generate Report...
Zoom
Zoom
Notification W
Notification Window

Opening the Report Wizard dialog via attached template in the Containment tree.



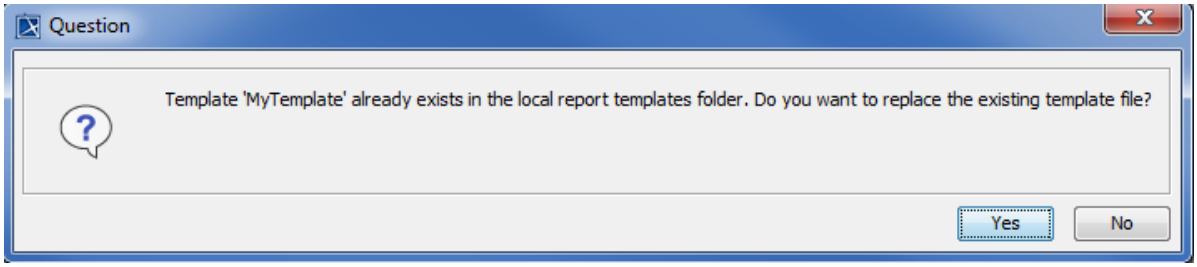
The Report Wizard template selection dialog.

Note

- In the **Report Wizard** dialog, when you select the attached template, the Attach button will be changed to Extract button.
- You can use the Extract button to extract the attached template to the local report template folder. After you finish editing, you can use the Attach button to re-attach the updated report template to your project.

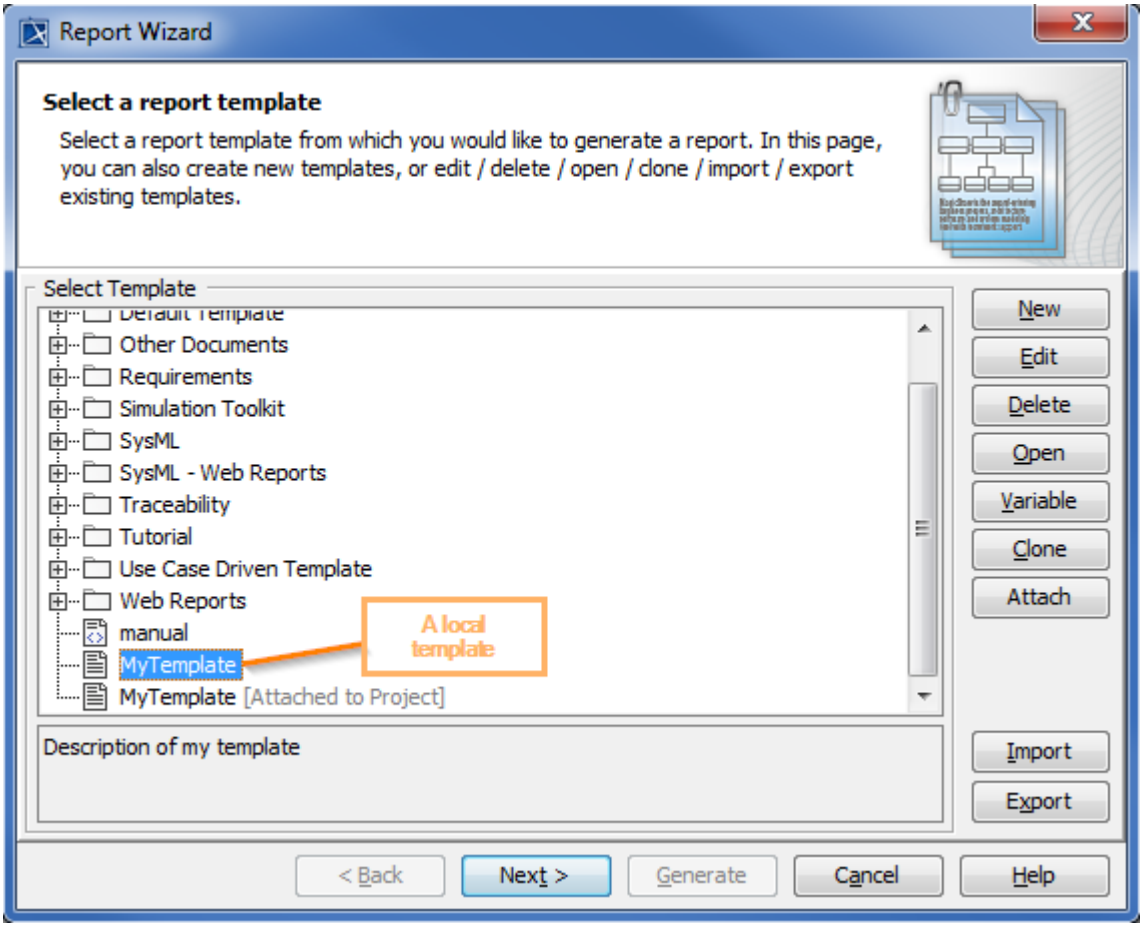
To extract a template

1. Select an attached template in the **Report Wizard** dialog.
2. Click Extract to extract the attached template to the local report template folder.
3. If the template already exists in the local report template folder, a confirmation dialog will be displayed.



The confirmation dialog when you extract a template.

4. Click Yes to extract the template. The attached template will be extracted to a local template.

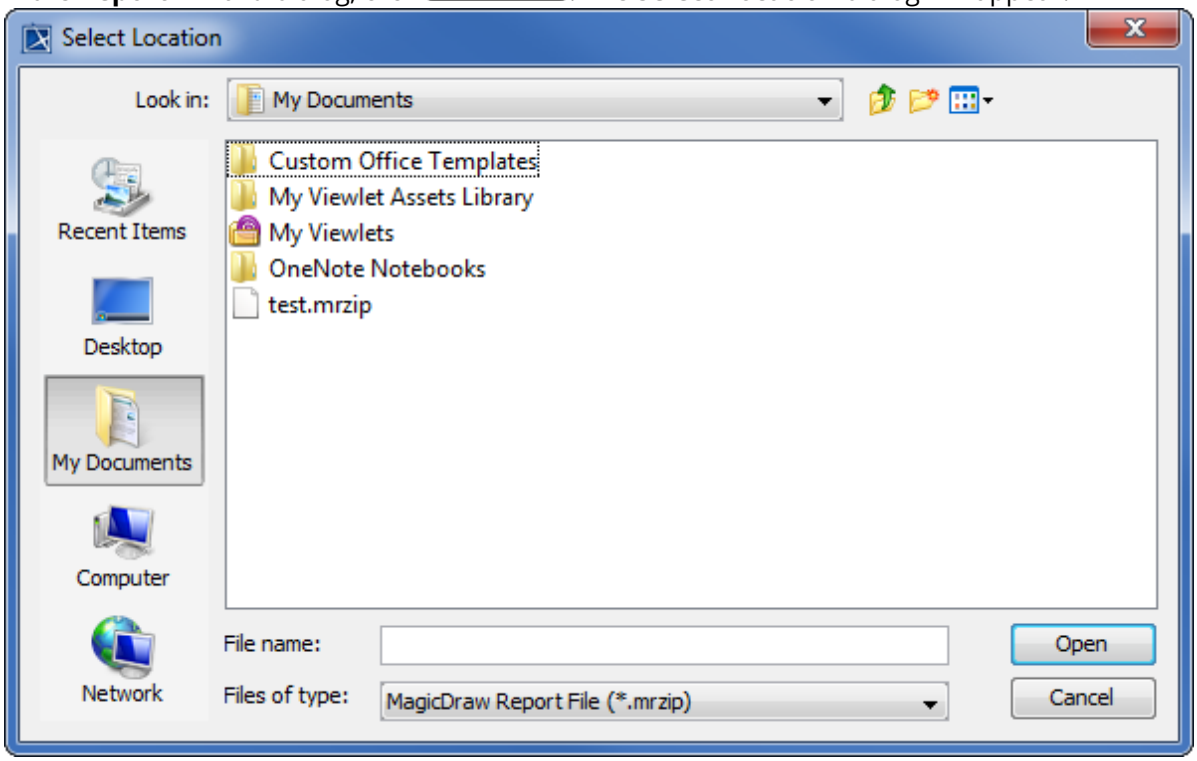


A local template in the Report wizard's template selection pane.

- After you finish editing, you can use the Attach button to re-attach the updated report template to your project.
- You can remove an attached template from a project by selecting it and clicking Delete.

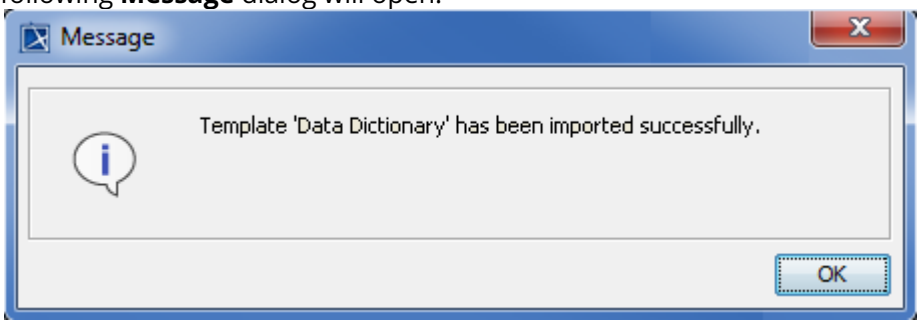
To import a template

1. In the **Report Wizard** dialog, click Import. The **Select Location** dialog will appear.



Selecting the location to import a template.

2. Select a Report Wizard template with the filename extension *.mrzip* and click Open. The following **Message** dialog will open.

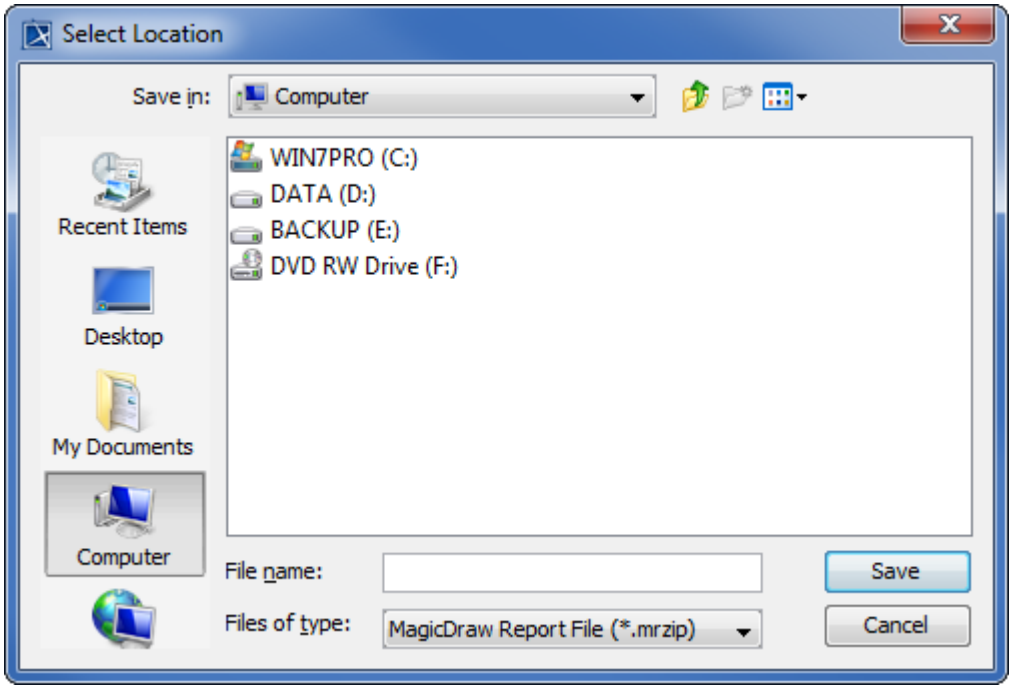


The Message dialog of a successful import.

To export a template

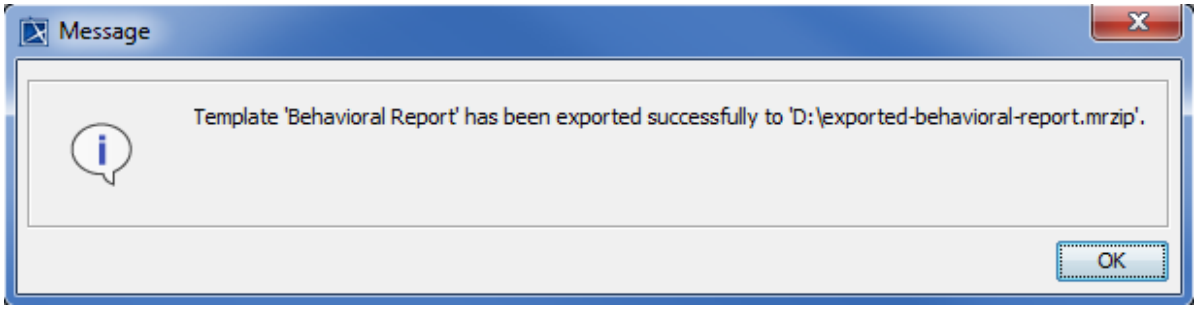
---

1. In the **Report Wizard** dialog, select a template and click Export. The **Select Location** dialog will open.



Selecting a location to export the template.

2. Select a directory or folder location in which you want to export the template.
3. Type the filename and click Save. The **Message** dialog will open.
4. Click OK.



The Message dialog of successful export.

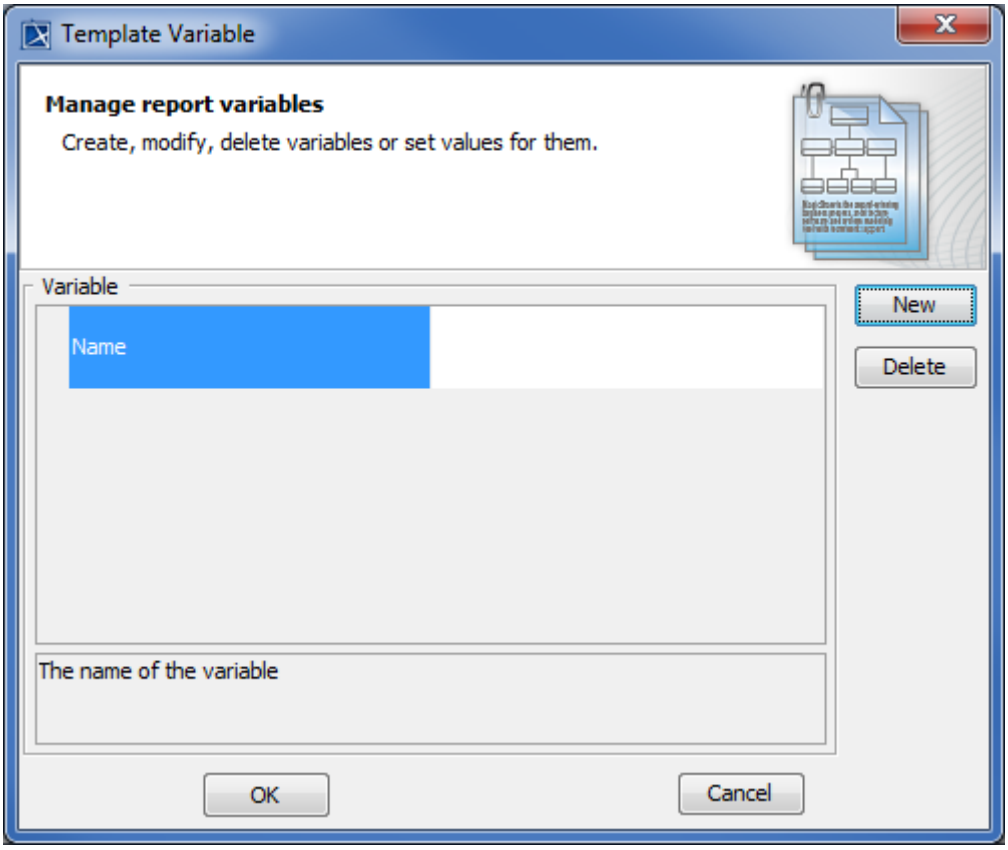
To open a template file in the default editor

- In the **Report Wizard** dialog, select a template and click Open. The template file will open in the default editor.

Working with a template variable

A template contains variables and their values. You can create a new template variable, and modify or delete an existing template variable in the **Template Variable** dialog. You cannot modify the variables of a built-in template. You can only create or modify a template variable of a user-created template.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The Template Variable dialog.

The **Template Variable** dialog, contains the following:

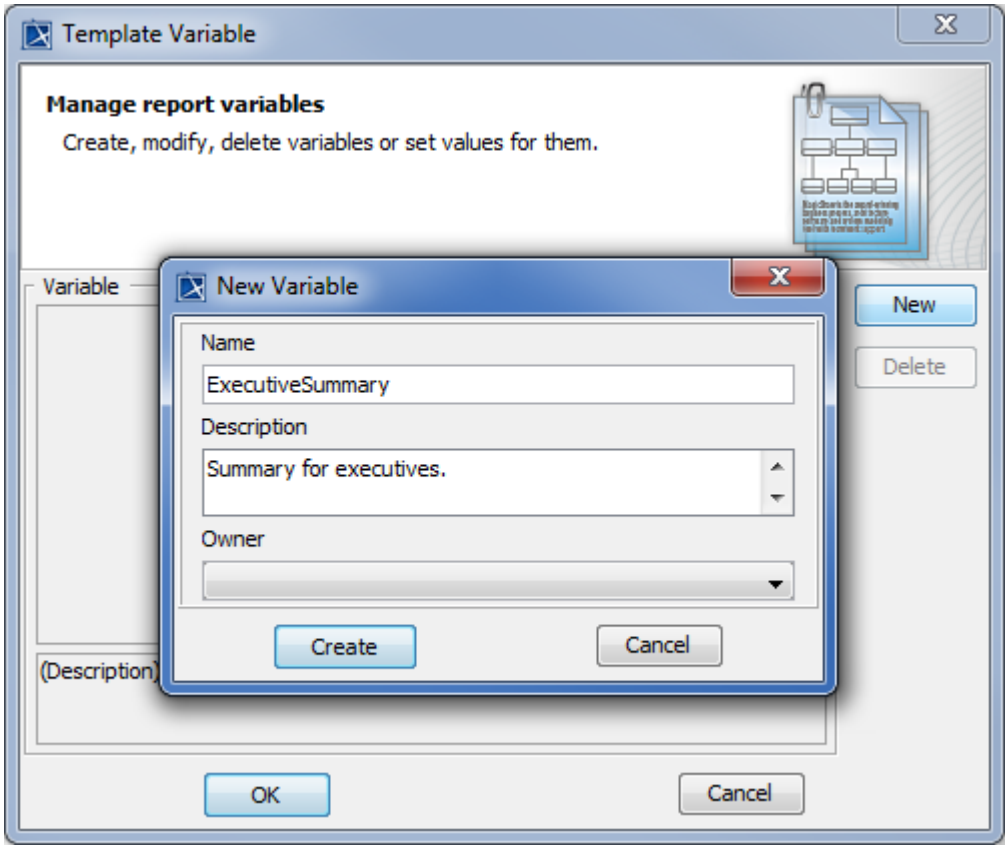
- **Variable** pane: This pane consists of a table in which the first column is the variable name, the second column is the variable value, and also includes a report description field. You can view and directly edit the value of a selected variable in the second column.
- Control buttons: OK and Cancel.

The following instructions explain how to manage the variables of a template.

To create a template variable

---

1. Select a template in the **Report Wizard** dialog and click Variable. The **Template Variable** dialog will open.
2. Click New. The **New Variable** dialog will appear.



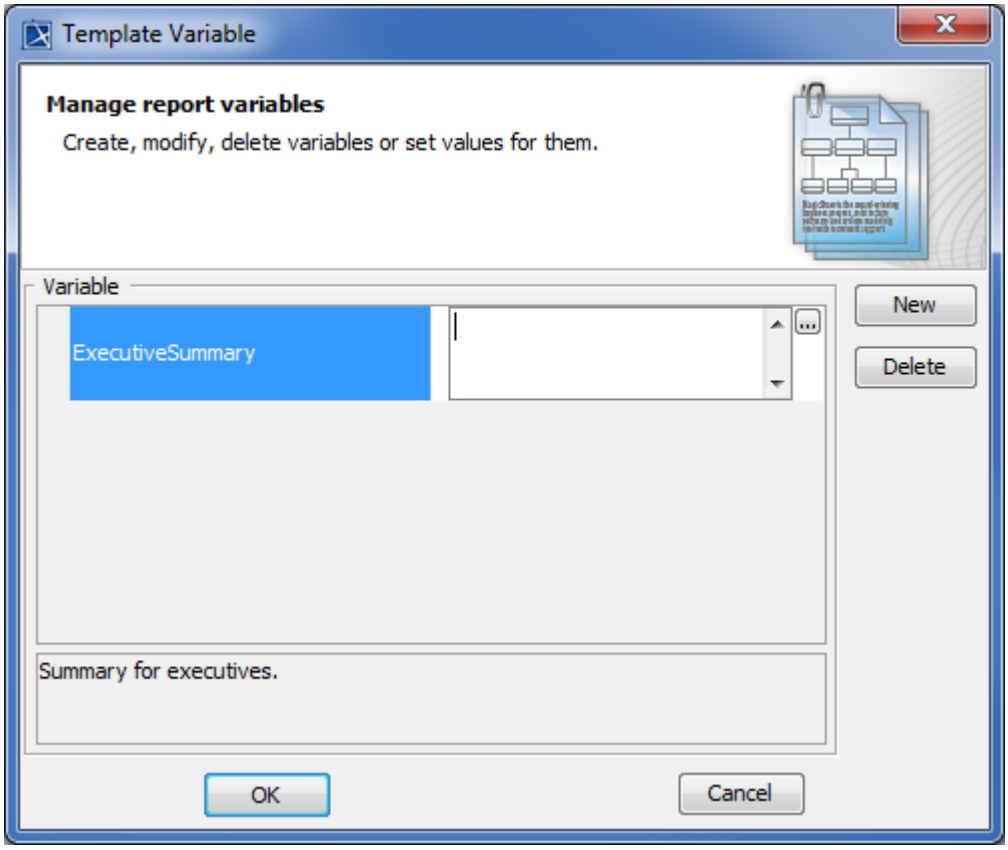
Creating a new variable in the New Variable dialog.

3. Enter the variable name (no space between characters) and description.
4. Select an owner for this new variable from the **Owner** drop-down list (optional).
5. Click Create. You will see the newly-created variable's name and value in the **Variable** pane.
6. In the **Template Variables** dialog, click OK.

To modify a variable value in the Template Variable dialog

---

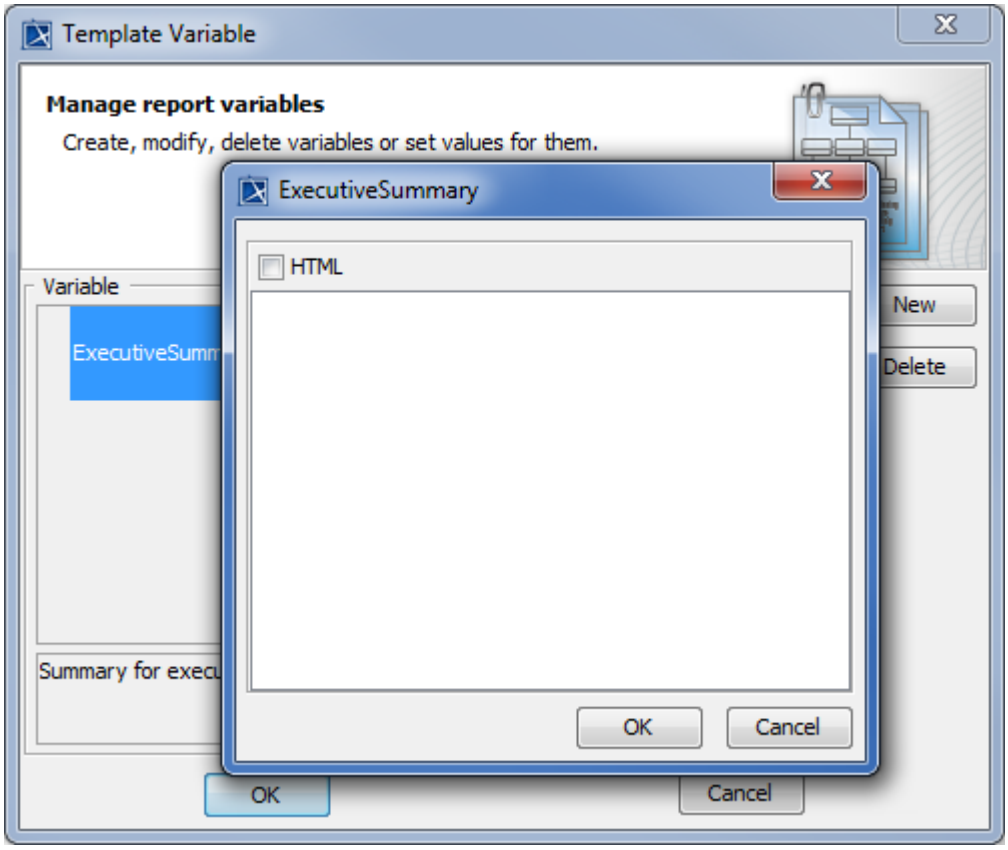
1. Select a template in the **Report Wizard** dialog, and click **Variable** to open the **Template Variable** dialog.
2. Double-click the **Variable** value column and modify the value in the column.



Editing a variable value in the Template Variable dialog.

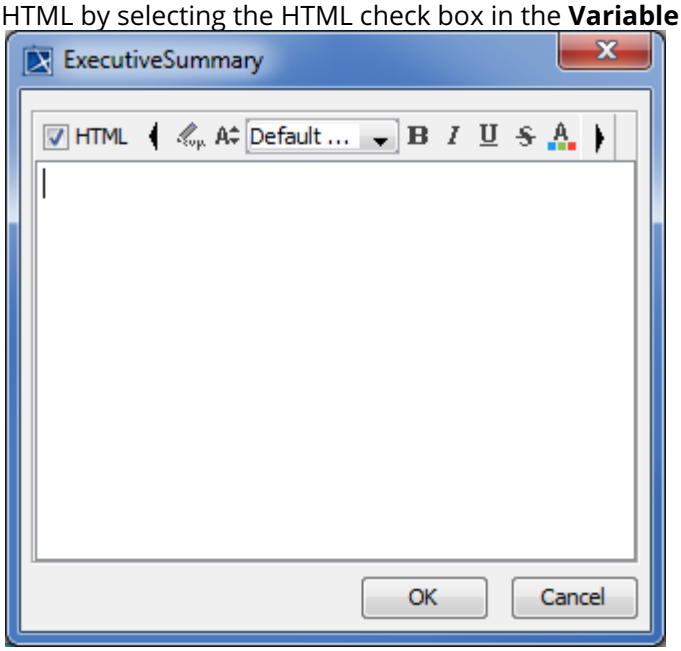
To modify a value in the Template Variable value dialog

1. Double-click the **Variable** value column in the **Variable** pane in the **Template Variable** dialog, and click the ... button. The **Variable** value dialog will open.



Editing a variable value in the Variable value dialog.

2. To modify the variable value, you can enter the value of a report variable either in plain text or HTML by selecting the HTML check box in the **Variable Value** dialog.



Entering a report variable value in HTML.

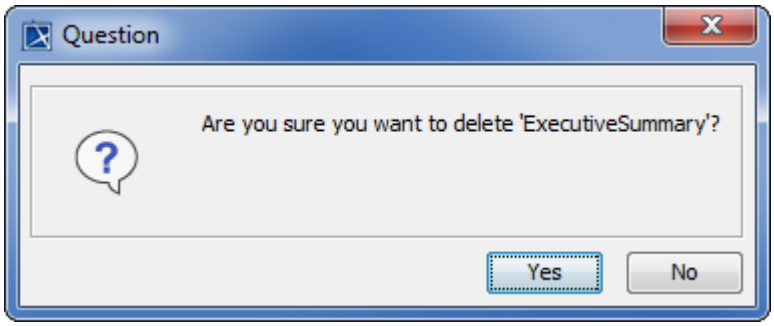
3. Click OK.

4. When you finish modifying variables and their values in the **Template Variable** dialog, click either OK to confirm the changes or Cancel to discard them.

To delete a variable

---

1. Select a variable in the **Template Variable** dialog and click Delete. The **Question** dialog will appear prompting you to confirm before deleting the selected variable.
2. Click Yes to delete the selected variable.



Deleting a variable.

Glossary

- **Use case**
  A a kind of behavior classifier that specifies a complete unit of useful functionality performed by one or more subjects to which the use case applies in collaboration with one or more actors. It also, for complete use cases, yields an observable result that is of some value to those actors or each subject. Use cases allow you to capture the requirements of systems under design or consideration, describe functionality provided by those systems, and determine the requirements those systems pose on their environment.
- **Business process diagram**
  A road map for implementation. It outlines the expected outcome, and provides something concrete from which to build.

More information about working with templates in Report Wizard

To learn more about templates and template variables, you can visit the following pages:

- Report Wizard Template Language(see page 1599)
- Report Wizard Environment Options(see page 1811)
- Template Variables(see page 1607)
- Report Wizard Template Editor(see page 1727)

Report Data Management Pane

On this page

- Report Data and Variables(see page 1567)

- Report Data Management Pane(see page 1568)
- Working with Report Data(see page 1570)
- Working with Report Data Variables(see page 1576)
- Glossary(see page 1584)
- More information about working with reports in Report Wizard(see page 1584)
- Sample model(see page 1585)

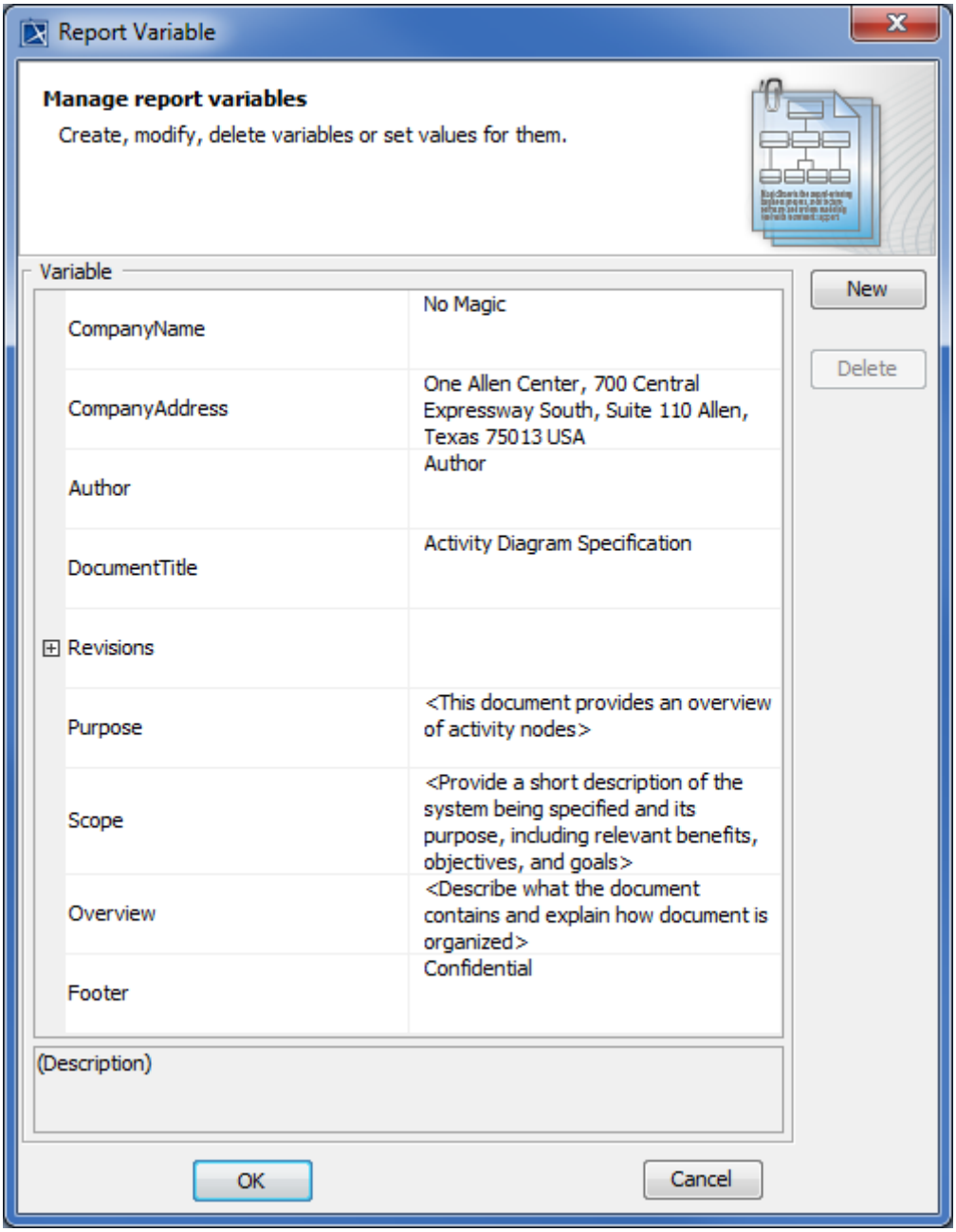
Report Wizard provides predefined report data templates that you can use to generate a report. The built-in report data template consists of several report data variables such as title, author, scope, and purpose that you can modify and adjust to meet your needs. You can also create a new report data template as needed, and manage the variables using the report data management pane.

This section defines report data, and how to use it in your report to represent variables that you want to include. Click the following links to navigate to the content of this page.

- Report Data and Variables(see page 1567)
- Report Data Management pane(see page 1568)
- Working with Report Data(see page 1570)
- Working with Report Data Variables(see page 1576)

Report Data and Variables

A report in Report Wizard has a collection of variables which are called report data. Report Wizard provides a built-in report data for every predefined template. The built-in report data comes with predefined variables and values which you can edit or delete. You can also add more variables to the built-in report data.



The built-in report data's predefined variables and values in Report Wizard.

You can create your own report data, and add variables and child variables under it if you do not want to use the built-in one. Report Wizard also allows you to create a report data as an element in the Containment tree inside a MagicDraw project by using the Report Profile which allows you to commit the Report Data to Teamwork Server, and share it with other users.

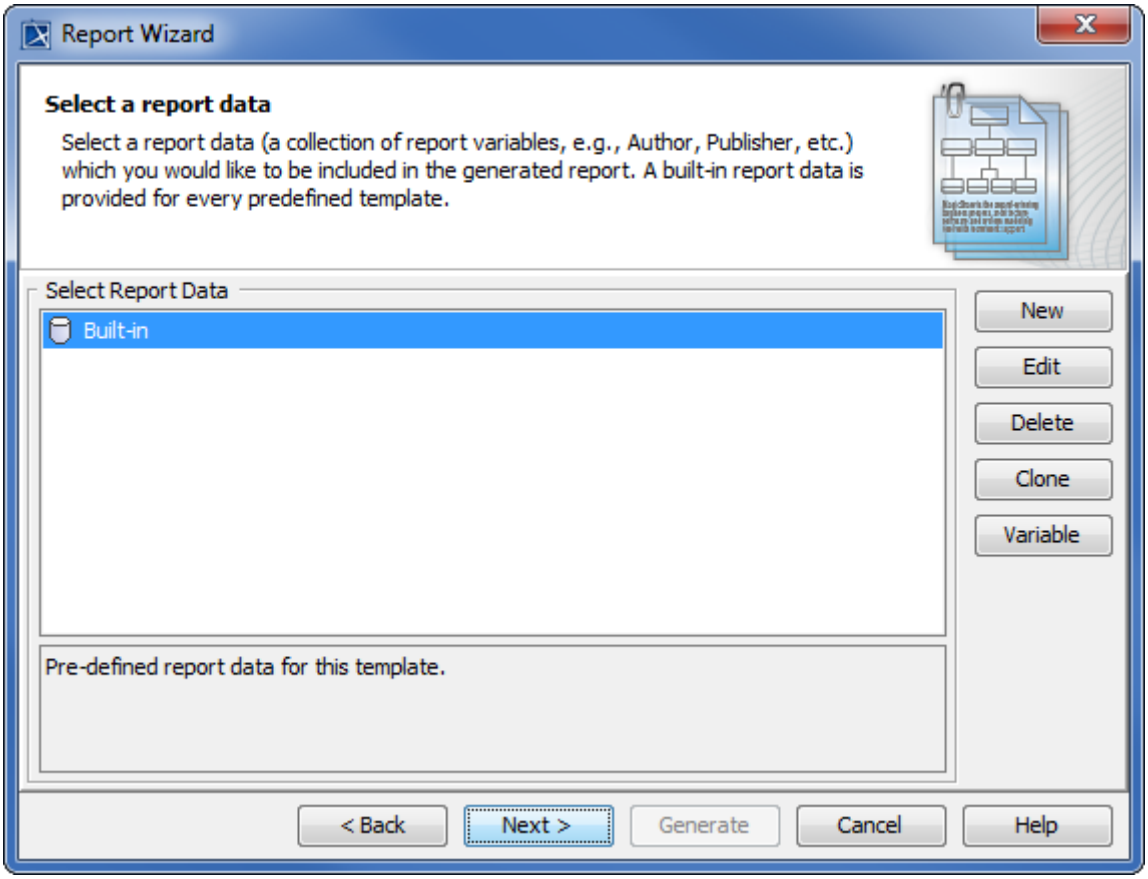
Report Data Management Pane

Report Wizard stores the built-in report data in the **Report Data** management pane which helps you to organize information into groups, and keep revision history created in a MagicDraw project.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

This pane also allows you to:

- Create, edit, delete, or clone a Report Data.
- Create or organize the variables in a Report Data.



Selecting the built-in report data in the Report Data management pane.

The following table lists the buttons and their function in the **Report Data** management pane.

Button	Function
New	Creates report data.
Edit	Edits report data.
Delete	Deletes report data.
Clone	Clones report data.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Button	Function
Variable	Creates a report variable.

Working with Report Data

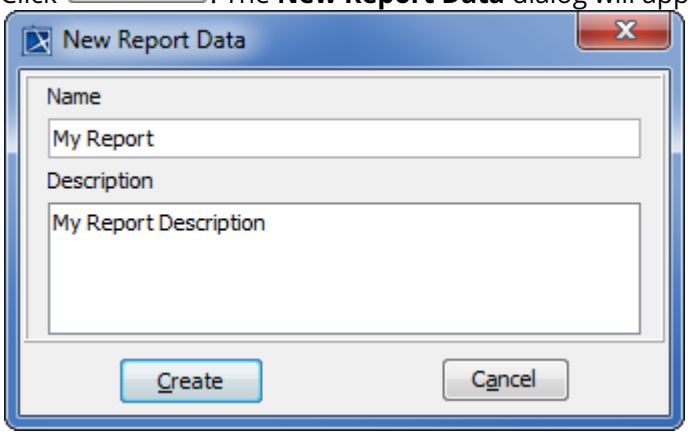
You can create report data either in:

- Report Wizard in which all data will be saved into an XML file.
- MagicDraw project, in which case the report data will be saved with the project itself. Saving report data in a project will enable you to commit the report to Teamwork Server.

The following instructions show you how to create and manage your report data in Report Wizard and in a MagicDraw project.

To create report data in Report Wizard

1. Click **Tools** > **Report Wizard**.
2. Select a template and click Next >.
3. Click New. The **New Report Data** dialog will appear.



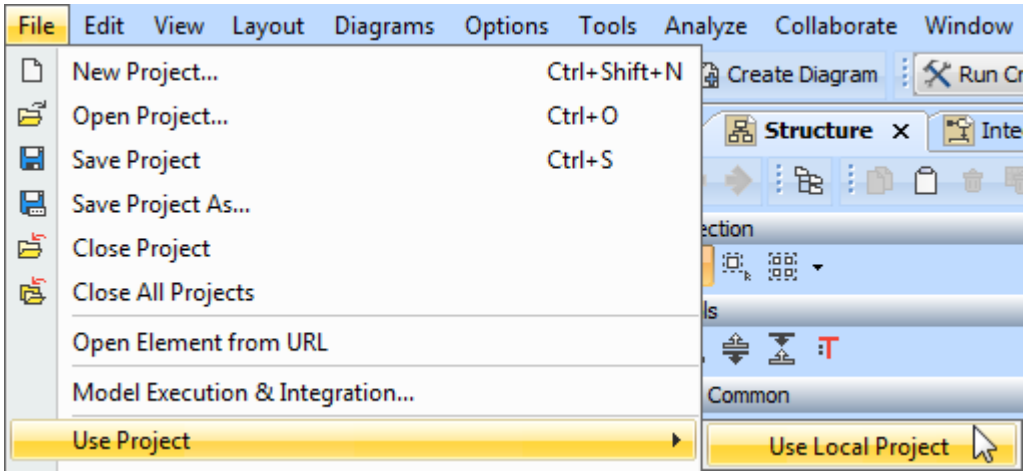
Creating a Report Data in the New Report Data dialog.

4. Enter a new report name and description, and then click Create.

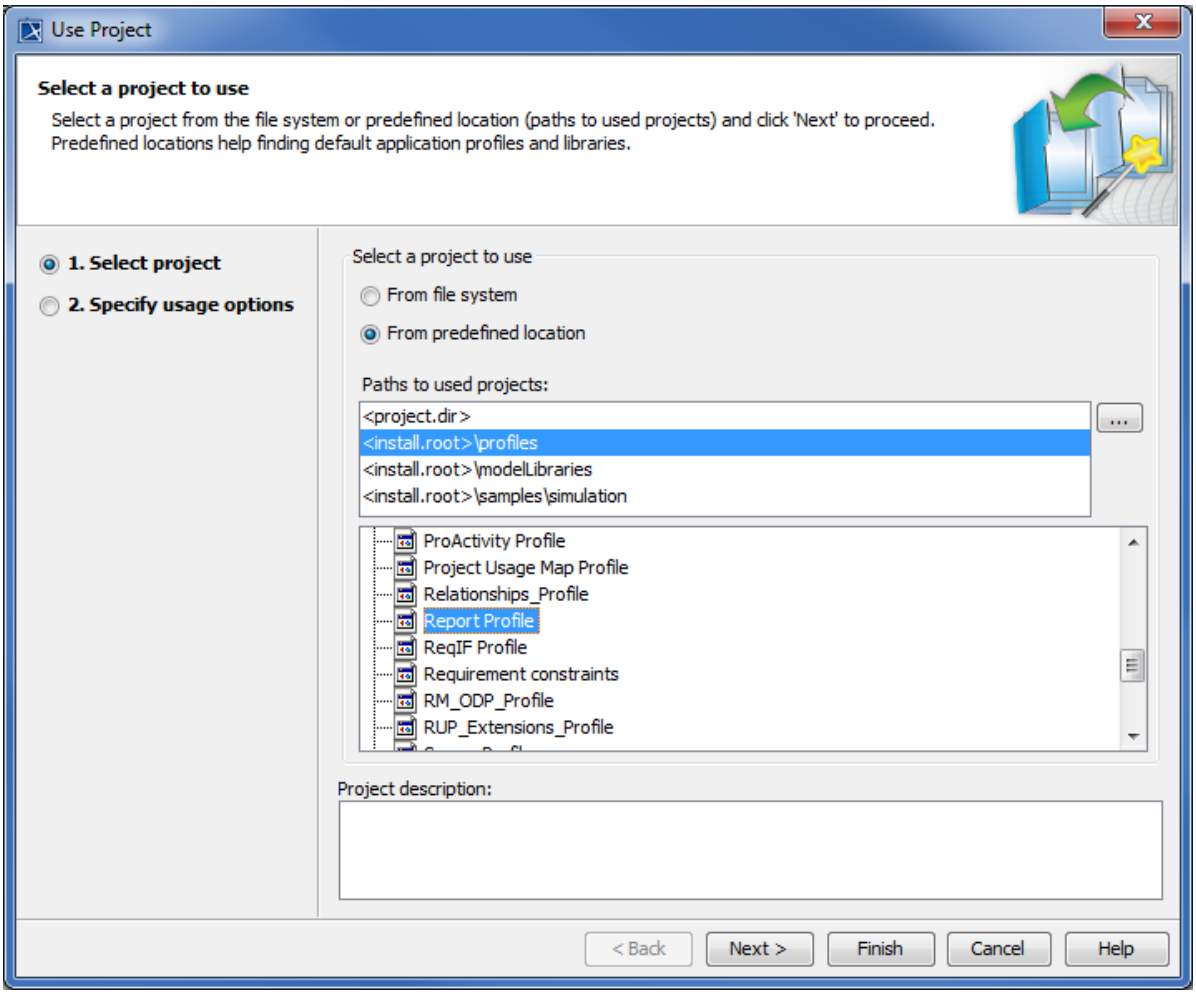
Before creating report data in a MagicDraw project(see page 0), you need to use a report profile, *Report Profile.mdzip*, which is located in the *<install.root>\profiles* folder.

To use a report profile (Report Profile.mdzip)

1. On the MagicDraw main menu, click **File** > **Use Project** > **Use Local Project**. The **Use Project** dialog will open, and the project module path *<install.root>\profiles* will be selected by default.

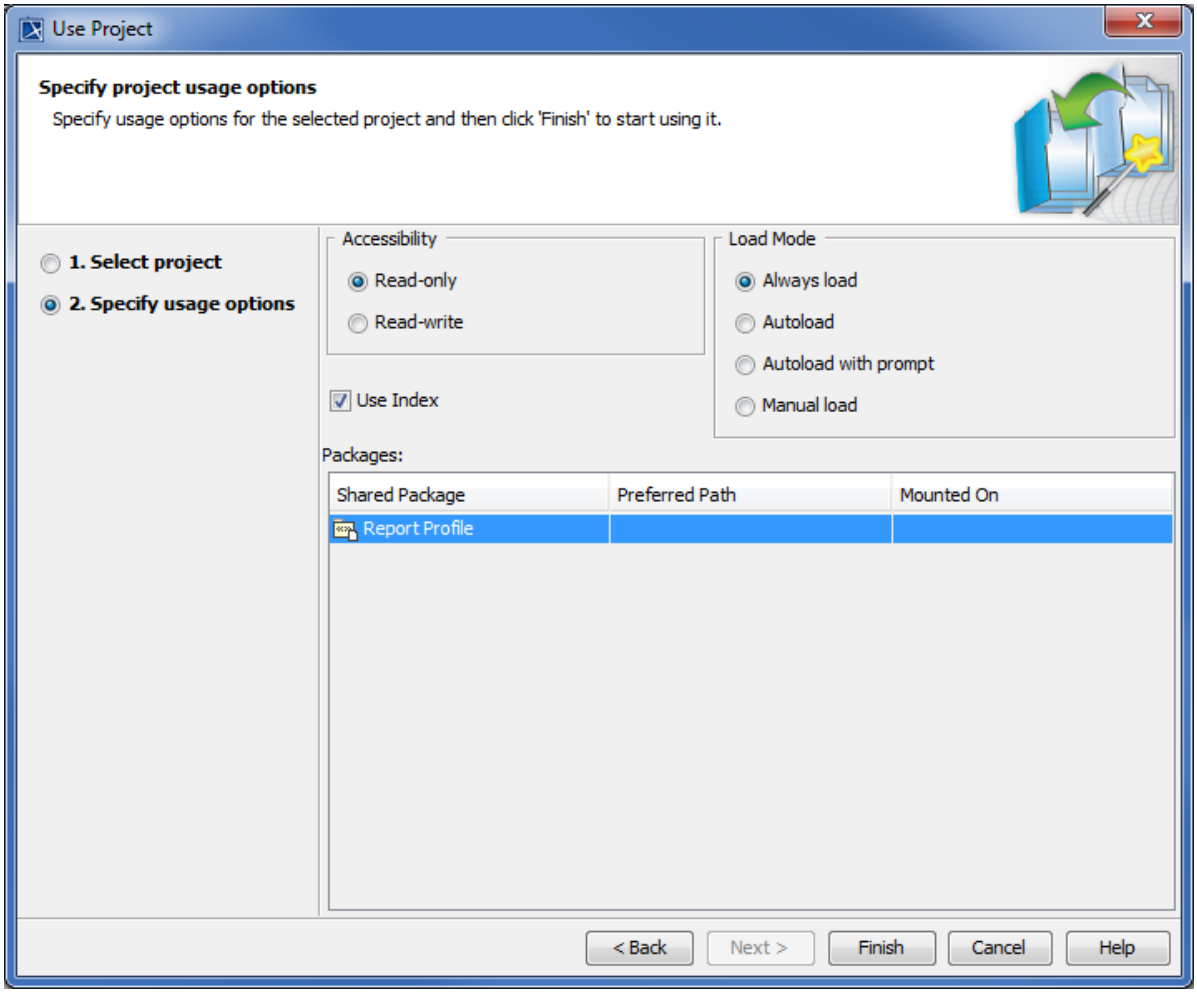


The Use Local Project menu opens the Use Project dialog that contains a report profile you need before creating a report data in a MagicDraw project.



Selecting a project module path for the report profile and Report Profile.mdzip in the Use Project dialog.

2. Select *Report Profile* and click Next >.
3. Specify the usage options.



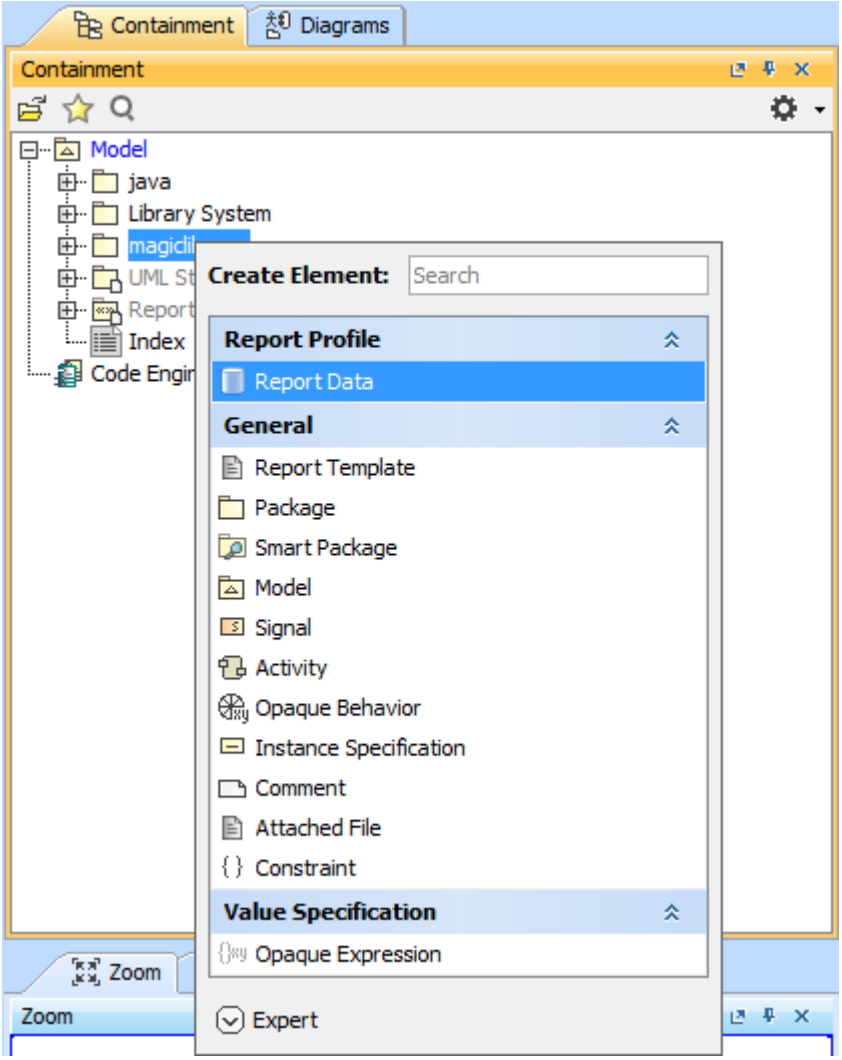
Usage Options.

4. Click Finish. The “Report Profile” profile will open in the Containment tree as a read-only profile. You can now use it in your project.

To create a Report Data in a MagicDraw project

---

1. Use *Report Profile.mdzip* in your MagicDraw project.
2. Right-click a data model in the Containment tree and select **Create Element**.



Creating Report Data in the MagicDraw project

3. Select **Report Data** from the menu.
4. Type the name of the **Report Data** element in the Containment tree.

You can right-click a data model, a package, or a profile in the Containment tree to create a Report Data.

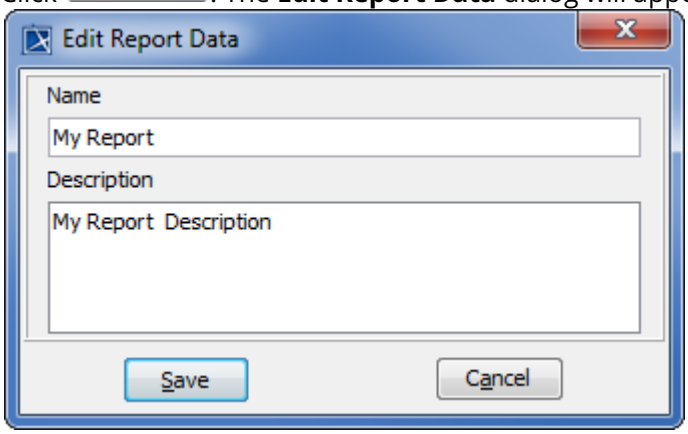
Tag Values	Function
autoImageSize	Change the image size.
imageFormat	Select an image format, either JPEG or PNG.
emptyText	Store a string value that will be replaced with an empty variable.
data	Contain elements that will be published by Report Wizard.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Tag Values	Function
template	Determine which template to use in a particular Report Data.
generateRecursively	Determine whether or not a report will be generated recursively.

To edit report data

1. Click **Tools** > **Report Wizard**.
2. Select a template and click Next >.
3. Click Edit. The **Edit Report Data** dialog will appear.

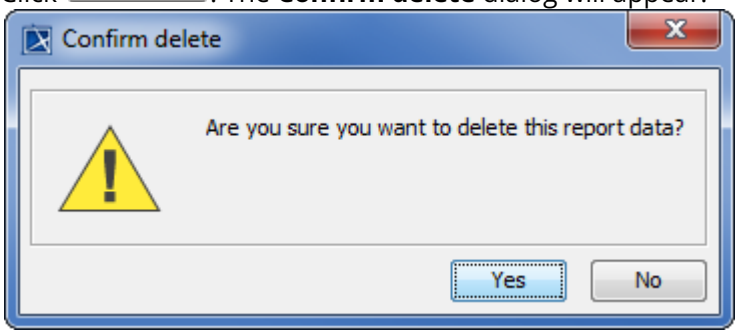


Editing a report data in the Edit Report Data dialog

4. Edit the report's name and description, and then click Save.

To delete report data

1. Click **Tools** > **Report Wizard**.
2. Select a template and click Next >.
3. Click Delete. The **Confirm delete** dialog will appear.

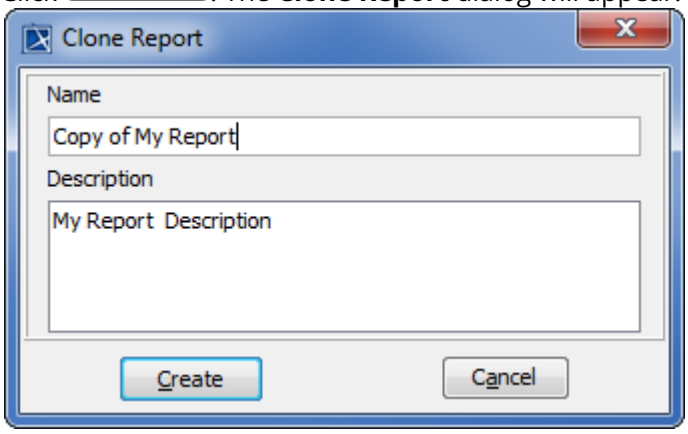


The Confirm delete dialog.

4. Click either Yes to delete the selected data or No to cancel the operation.

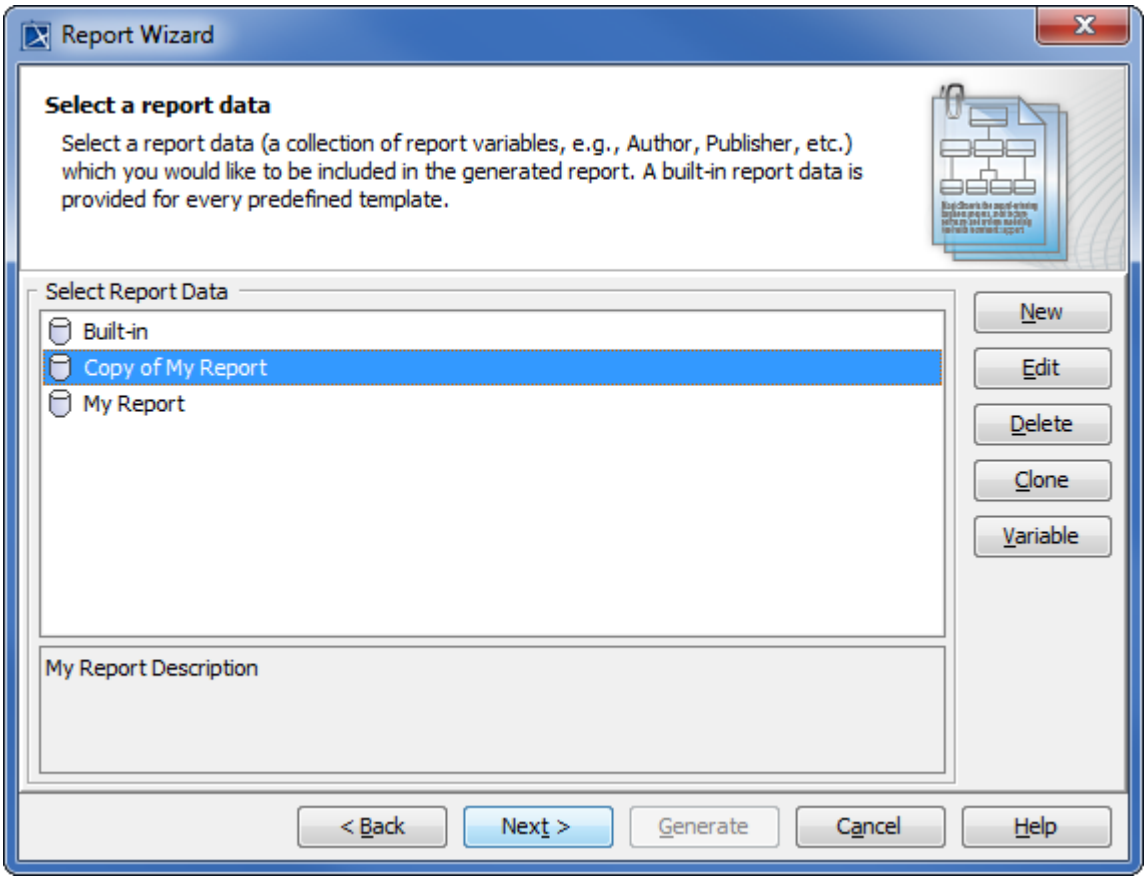
To clone report data

1. Click **Tools** > **Report Wizard**.
2. Select a template and click Next >.
3. Click Clone. The **Clone Report** dialog will appear.



Cloning a report data in the Clone Report dialog.

4. Enter the name and description of the report. The name should begin with **Copy of** (cloned report name).
5. Click Create. The cloned Report Data will then be created.



The Copy of My Report created in the Report Data Management pane.

When a cloned Report Data is created, a copy of the Report Data (variable) will also be created.

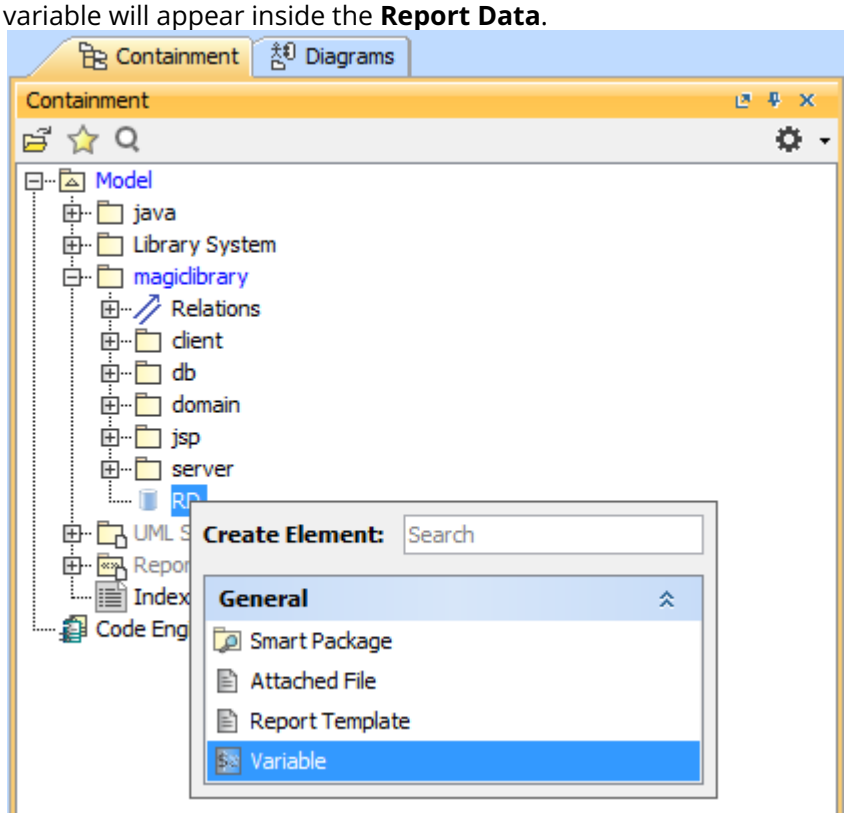
Working with Report Data Variables

Variables are created because some information is not “naturally” contained in a model, e.g. the company's name, author's name, revisions. Although you can put all of the information in a model's specification (Document/Hyperlinks in the **Specification** dialog), it is difficult and tedious to get data from the model's specification, as this results in more lines of codes and scripts in the report templates. Therefore, instead of writing lines of codes, you can simply create a variable, give it a name, and call it directly from the template. For example, you can create a variable called “Author” in Report Wizard and write **$Author** in the template. When you generate the report, you will see the value inside “Author.” Essentially, a variable is used to represent data that you want to include in the report, not in the model.

Variables contain information that you want to store in a project, such as names and dates. You can create a report variable in a MagicDraw project or the Report Wizard dialog.

To create a report variable in a MagicDraw project

1. Right-click a **Report Data** in the Containment tree and select **Create Element** > **Variable**. The variable will appear inside the **Report Data**.



Creating a variable for Report Data in the Containment tree.

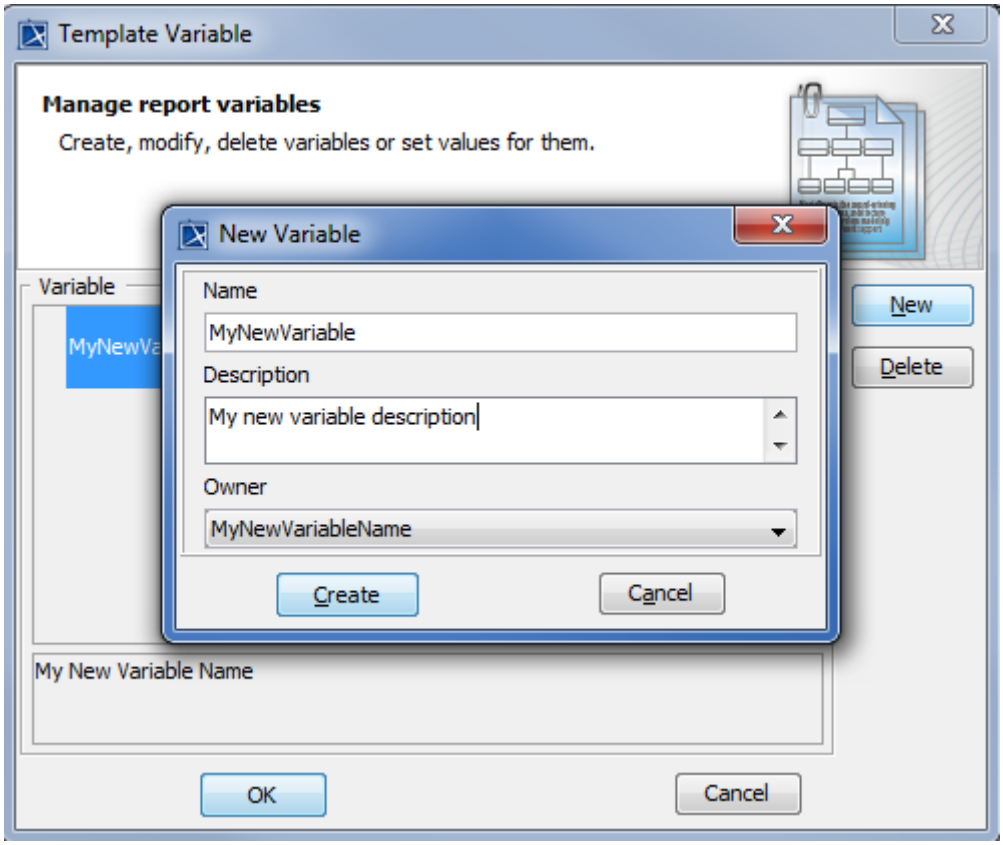
You can also create, edit, and delete variables through the **Report Variable** dialog in Report Wizard.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To create a report variable in the Report Wizard dialog

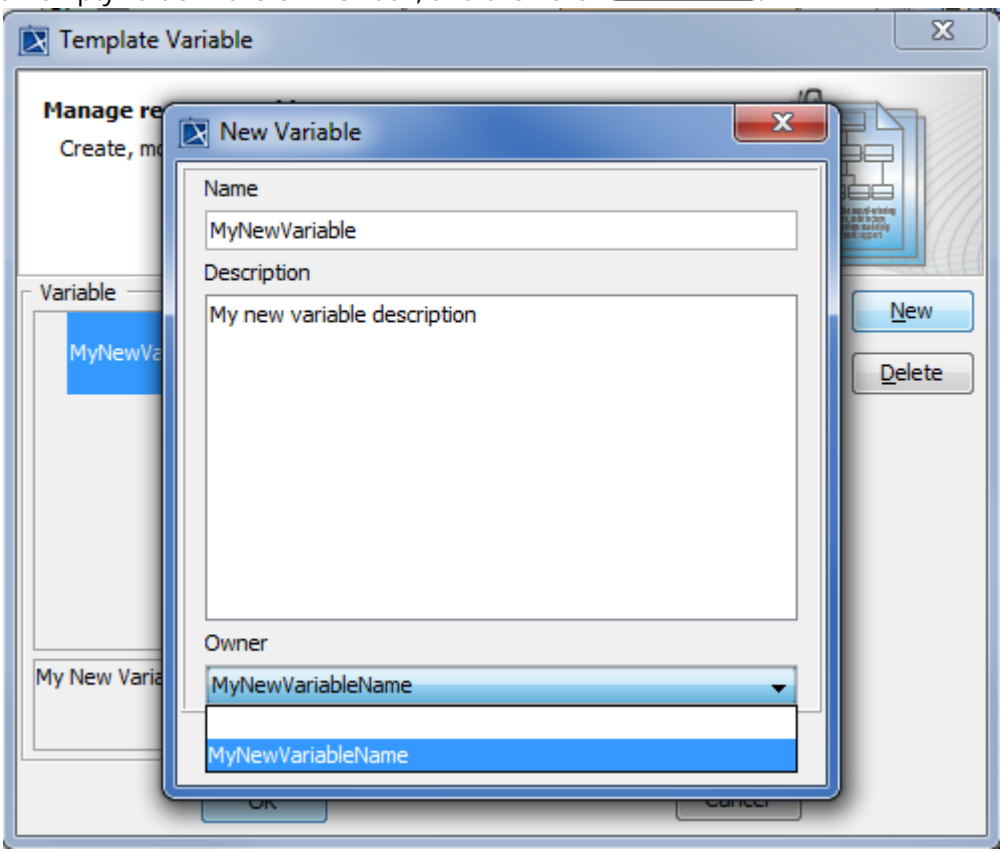
1. Open the **Report Variable** dialog.
2. Click New. The **New Variable** dialog will open. Enter the variable name and description. You can create either a parent variable or a child variable in the **Owner** box.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



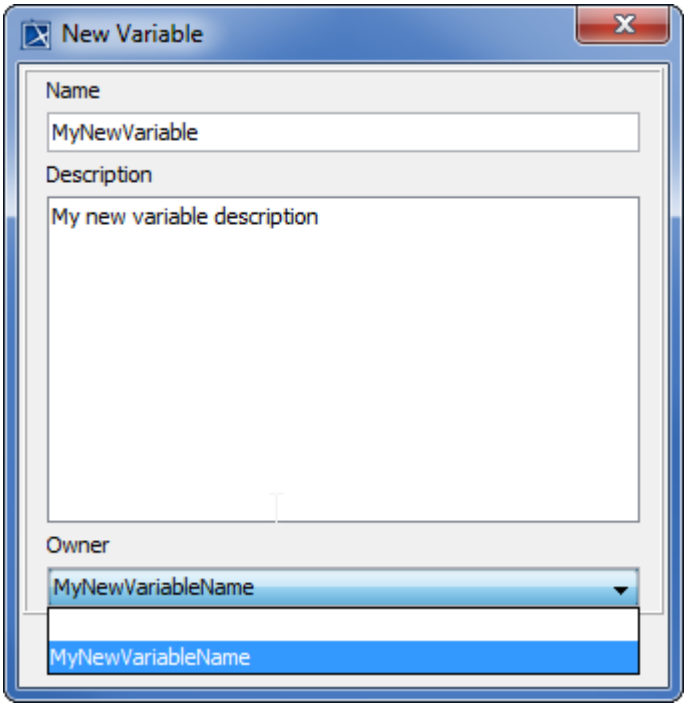
Creating a new variable in the New Variable dialog.

To create a parent variable, type the variable name in the **Name** box, enter the description, select an empty value in the **Owner** box, and then click Create.



Creating MyNewVariableName as a parent variable.

To create a child variable, type the variable name in the **Name** box, enter the description, select **Parent** in the **Owner** box, and then click Create.



Creating a child variable of MyNewVariableName.

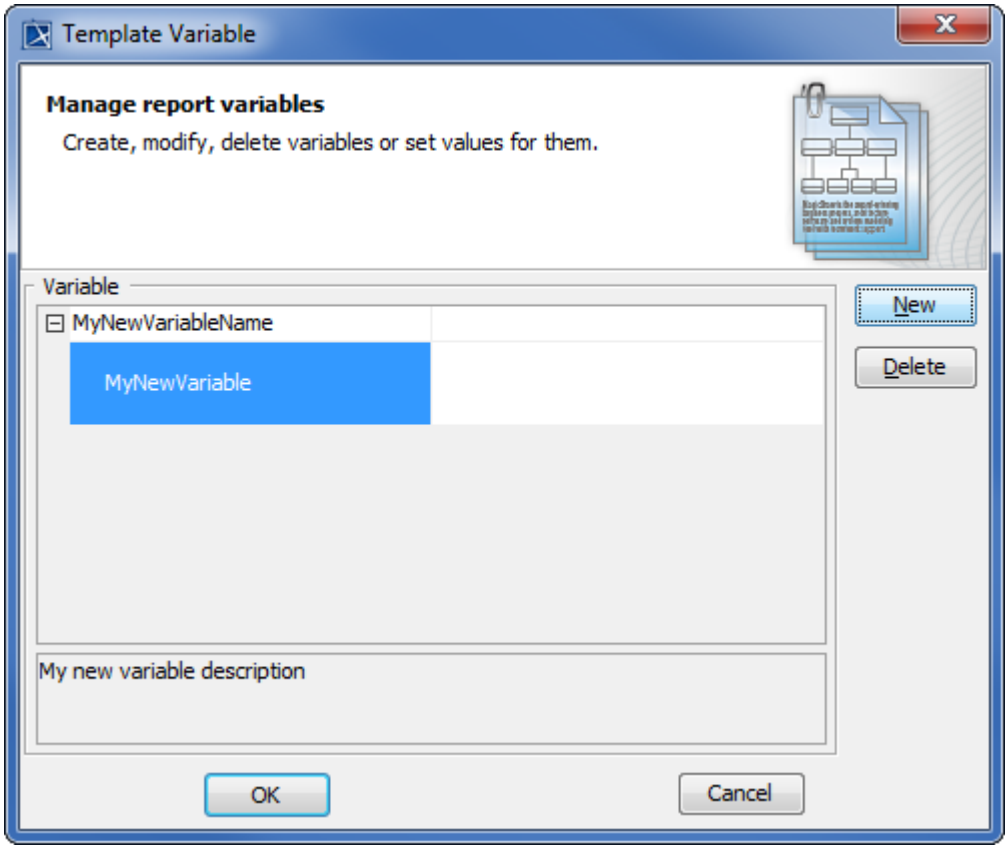
3. The new variable will appear in the table in the **Report Variable** dialog.

4. Click OK to save the variables in the Report Data.

To edit a report variable

---

1. Open the **Report Variable** dialog.
2. Double-click on the column next to the variable name column and click .... A dialog will open in which you can edit the variable value.

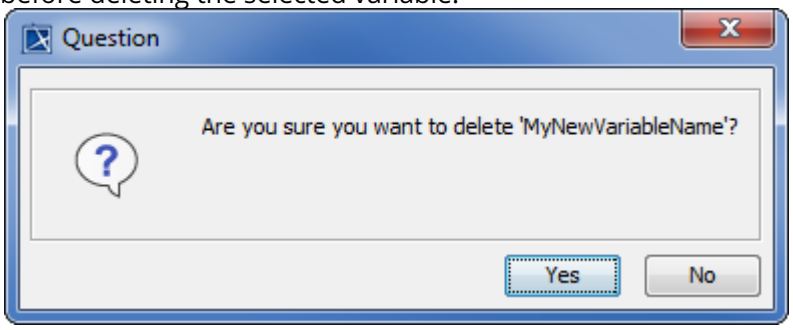


Editing a variable in the Template Variable dialog

3. Click OK. The new variable value will appear in the column next to the variable name column.

To delete a report variable

1. Open the **Report Variable** dialog.
2. Click a variable in the table, and then click **Delete**. A dialog will open prompting you to confirm before deleting the selected variable.

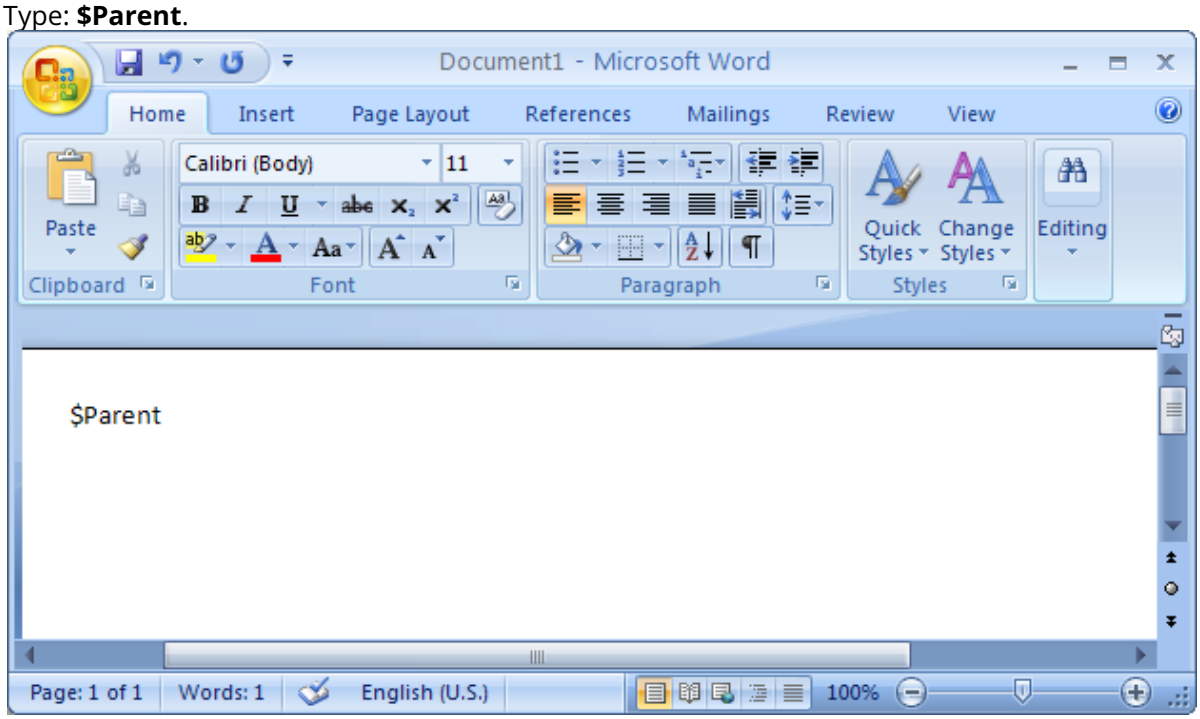


Dialog to confirm the deletion of a report variable

3. Click Yes and the variable will be deleted.

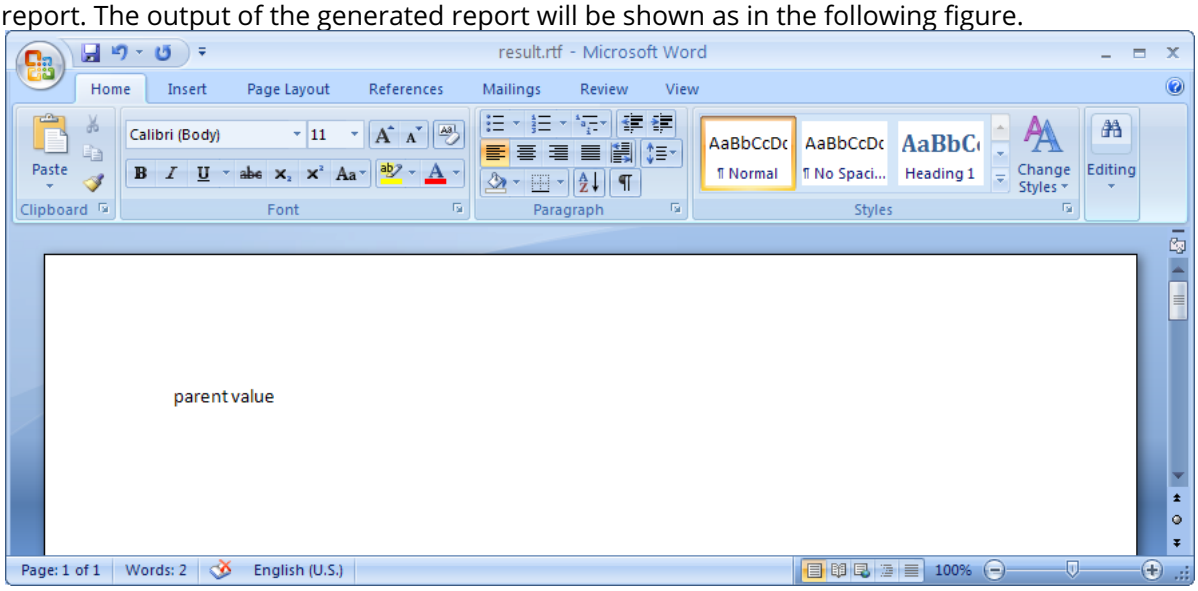
To get the value of a top-level variable

1. Open a blank document in Microsoft Word.
2. Type: **$Parent**.



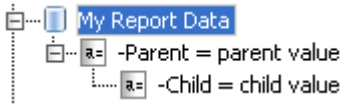
Referencing to a parent variable in the template.

3. Save the file as "*sampletemplate.rtf*". Choose Rich Text Format (**.rtf*) as the file type.
4. Open the **Report Wizard** dialog and click New to create a new template.
   The **New Template** dialog will open.
5. Type the name and description of the new template. Click ... to select "*sampletemplate.rtf*" as the template file.
6. Click Create > Next >. The **Select Report Data** pane will open.
7. Select a report data, in this example, **My Report Data** and click Generate to generate a report. The output of the generated report will be shown as in the following figure.



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Note
When you include the variables you have created in a template, each variable value will be included in the generated report. This section will use the following **My Report Data** as an example.

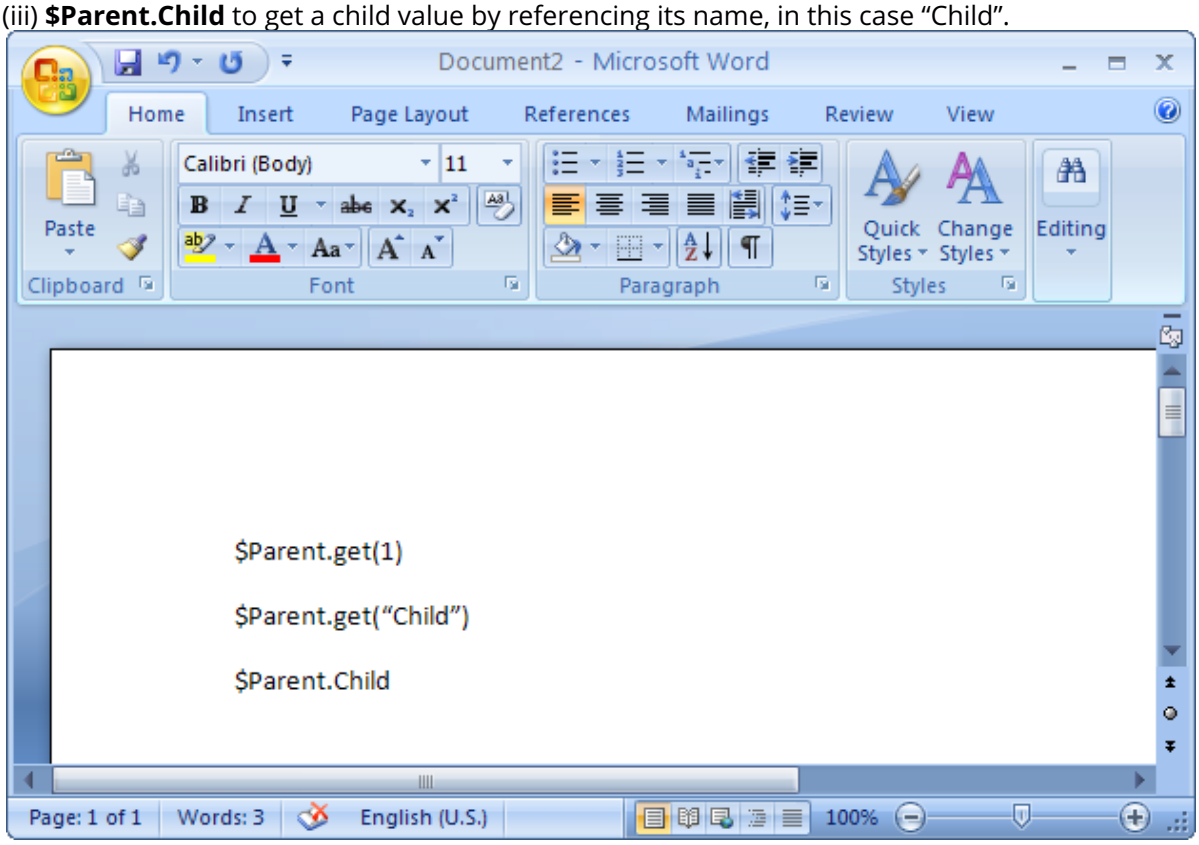


My Report Data as an example.

To get the value of the child of a variable

---

1. Open a blank document in Microsoft Word.
2. Type any of the following to print a child variable.
   (i) **$Parent.get(1)** to get a child value by index.
   (ii) **$Parent.get("Child")** to get a child value by name comparison.
   (iii) **$Parent.Child** to get a child value by referencing its name, in this case "Child".

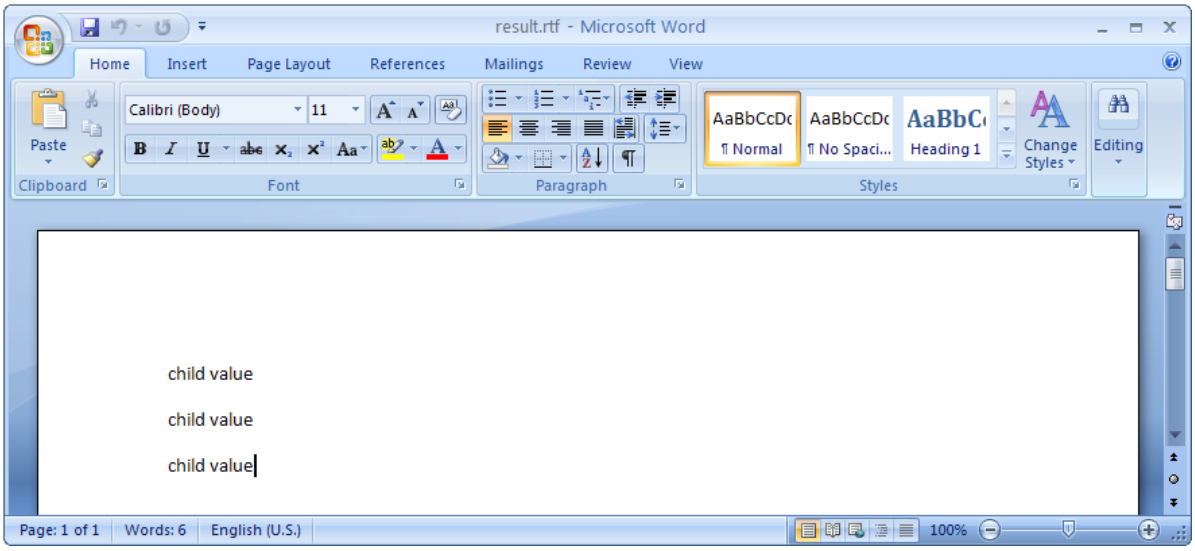


Referencing to the child variable in the template

3. Save the file as "*sampletemplate.rtf*". Choose **Rich Text Format** (**.rtf*) as the file type.
4. Open the **Report Wizard** dialog and click New to create a new template. The **New Template** dialog will open.
5. Type the name and description of the new template. Click ... to select "*sampletemplate.rtf*" as the template file.
6. Click Create > Next >. The **Select Report Data** pane will open.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

7. Select a report data, in this example, **My Report Data** and click Generate to generate a report. The output of the generated report will be shown as in the following figure.



The resulting RTF file.

Glossary

- **Clone**
  To clone is to create an exact copy of report data.
- **RTF**
  Rich Text Format (RTF) is a file format developed by Microsoft that can be used with different word processors in different operating systems.

More information about working with reports in Report Wizard

To learn more about reports in Report Wizard, you can visit the following pages:

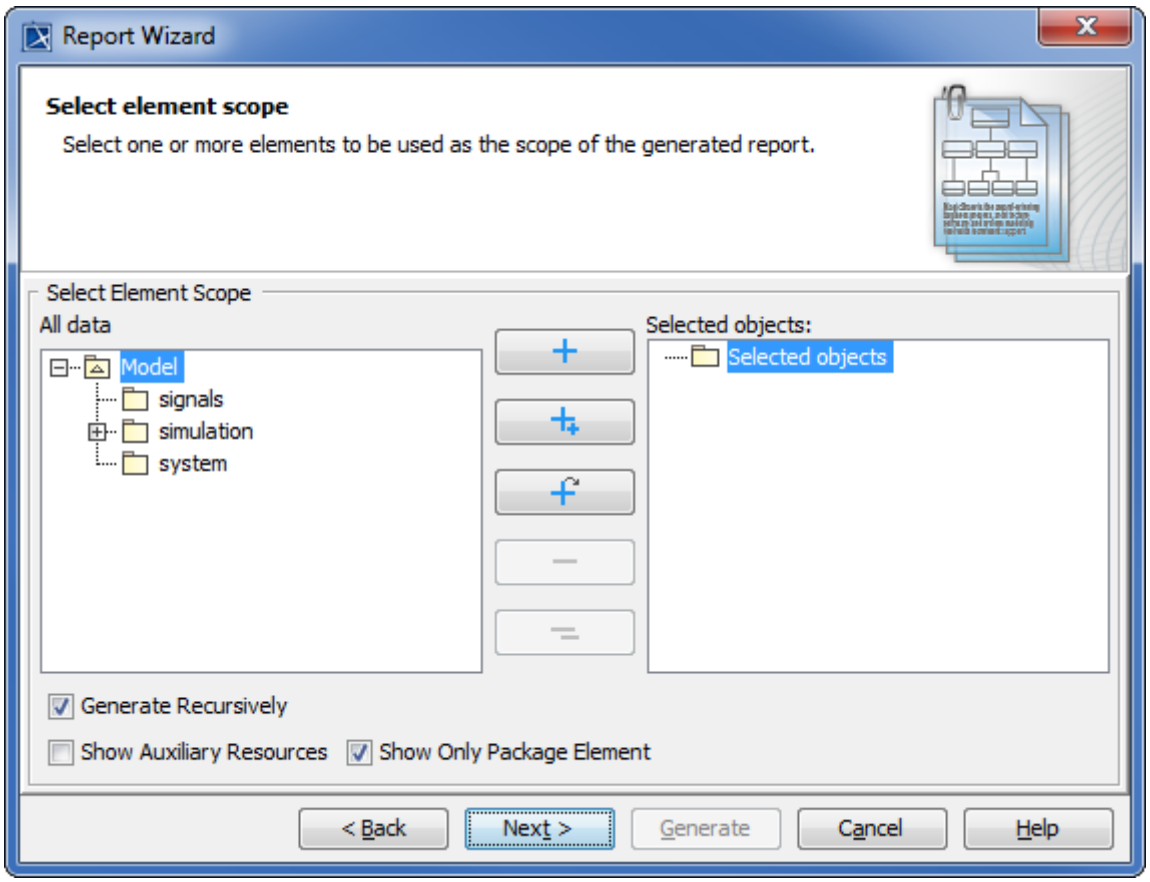
- Generating Reports from Report Wizard(see page 1734)
- Generating Reports from the Containment Tree(see page 1590)
- Generating Reports from the Command Line(see page 1793)

Sample model

Magic Library.mdzip

Select Element Scope pane

The **Select Element Scope** pane (see the figure below) allows you to select the scope of MagicDraw data to generate a report.



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

The Select Element Scope pane.

The table below describes the detail of each component in the **Select Element Scope** pane.

Component	Name description
All data tree	Select the desired packages from the **All data** tree then add them to the **Selected objects** tree.
Selected objects tree	Select packages and click the **Add**, **Add All**, or **Add Recursively** button. The selected packages will be added to the **Selected objects** tree.
Add button	Select packages and click **Add**. The selected packages will be added to the **Selected objects** tree.
Add All button	Select packages and click **Add All**. The selected packages and the elements directly owned by those packages will be added to the **Selected objects** tree.
Add Recursive button	Select packages and click **Add Recursively**. The selected packages and its recursive packages will be added to the **Selected objects** tree.
Remove button	Select packages and click **Remove**. The selected packages will be removed from the **Selected objects** tree.
Remove All button	Click **Remove All** and all packages in the **Selected objects** tree will be removed.

You can perform the following operations in the **Select Element Scope** pane:

- Add packages into the **Selected objects** tree.
- Remove a selected package from the **Selected objects** tree.
- Select or clear the **Generate Recursively** option.
- Show auxiliary resources.
- Show only package elements.

To add packages

- In the **Select Element Scope** pane, see figure above, select the packages from the **All data** tree and click **Add**, **Add All**, or **Add Recursively** to add them into the **Selected objects** tree.

To remove packages

- In the **Select Element Scope** pane, see figure above, select the packages from the **Selected objects** tree and click **Remove** or **Remove All** to remove them from the **Selected objects** tree.

To select or clear the **Generate Recursively** option

- Select or clear the **Generate Recursively** check box (see the figure above).

To include auxiliary resources

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Select the **Show Auxiliary Resources** check box and add auxiliary resources into the **Selected objects** tree (see the figure above).

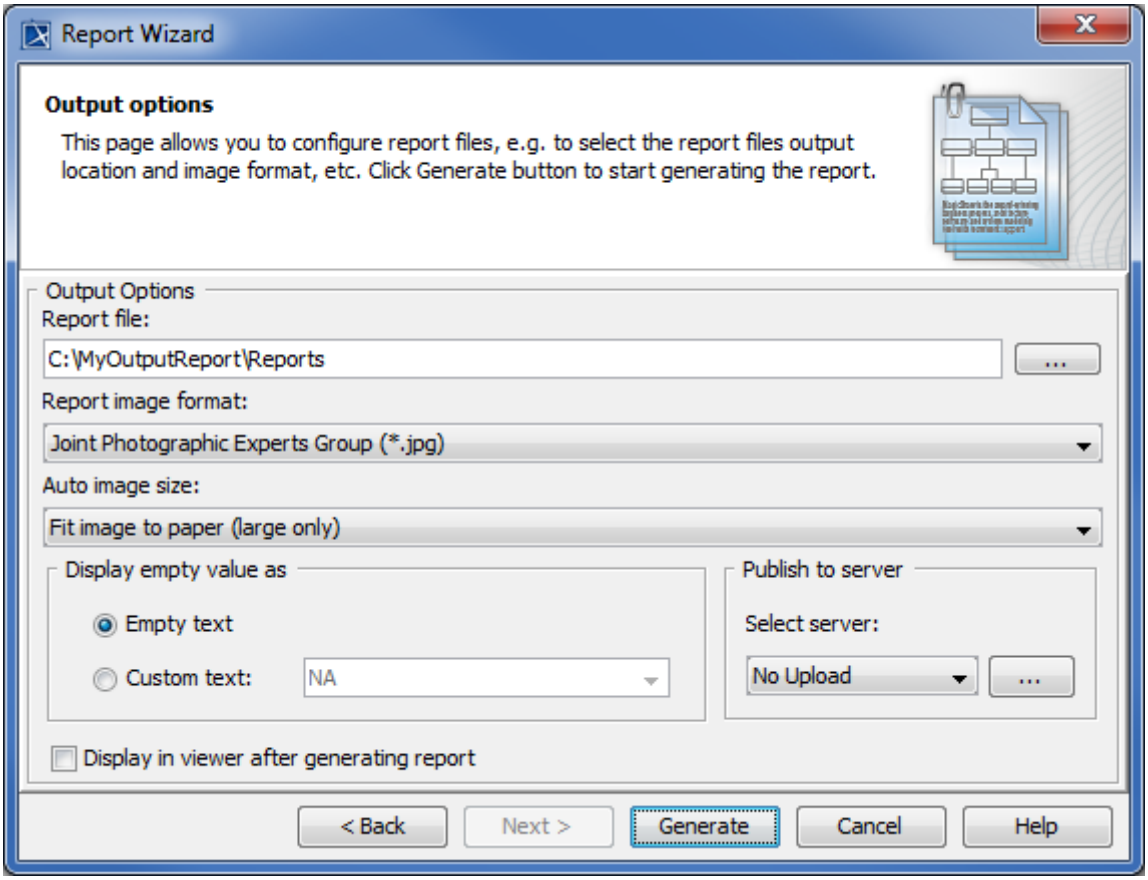
To show only package elements

- Select or clear the **Show Only Package Element** check box (see the figure above).

> The figure above shows the **UML 2 Elements** package and the **Generate Recursively** option were selected. It means that the **UML 2 Elements** package and its subpackages will be generated in the report.

Generate Output pane

The **Generate Output** pane in Report Wizard allows you to configure report files by selecting the report files' output location and image format, and displaying the report in the viewer.



The Generate Output pane.

The table below describes the function of each option in the **Generate Output** pane of Report Wizard.

Component Name	Description
Report file	Shows the report file's location and name. The default report location will be *\data\template_folder\reports*. The default report name will be the same as the report name defined by the user.
...	Opens the **Select Location** dialog in order to locate the report file.
Report image format	Selects an image format for your report: *JPG, PNG, SVG, EMF*, or *WMF*. Note: • Use **.JPG* and **.PNG* for any template format. • Use **.SVG* for text and *HTML* templates. • Use **.EMF* and **.WMF* for text and Microsoft Office templates (*RTF, DOCX, XLSX,* and *PPTX*).
Auto image size	Changes the size and orientation of an image before inserting it into a document. There are four options available: a. **No Resize-**Image will not be resized or rotated. b. **Fit image to paper (large only)-**Large image will be fit to within the paper size. c. **Fit and rotate (clockwise) image to paper (large only)-**Large image will be fit to within the paper size and rotated clockwise. d. **Fit and rotate (counter-clockwise) image to paper (large only)-** Large image will be fit to within the paper size and rotated counter-clockwise.
Display empty value as	• **Empty text-**Leave an empty value of the report output blank. • **Custom text as-**Enter a custom value for an empty value. The default value is **NA**.
Display in viewer after generating report	Displays a complete report in the viewer. Otherwise, the report will be created and kept in a selected location.
Generate	Generates a report.
Cancel	Cancels the report generation process and closes the **Report Wizard** dialog.
Help	Provides the Help content.

Displaying an empty value with a text feature was deprecated. Or, you can select one of the following two options:

- Use a template code to replace empty value with text. For example:

Using normal code	Using macro code

`#if (!$var || $var == "")`
`Empty`
`#end`
	`#macro (replaceNullValueWithText $var)`
	`#if (!var || $var == "")`
	`EMPTY`
	`#end`
	`#end`

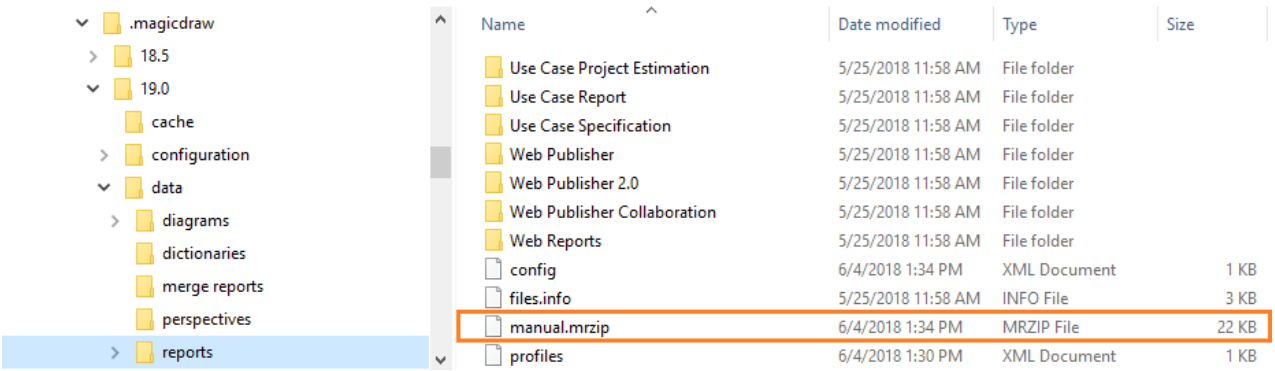
- Use report wizard environment option to configure empty.

Add the property "template.output.on.blank.field" and value in the Report Wizard Environment option to enable this feature. The value of this property will be inserted in the generated output when the variable is empty (see Report Wizard environment options[335]).

MRZIP file automatic deployment

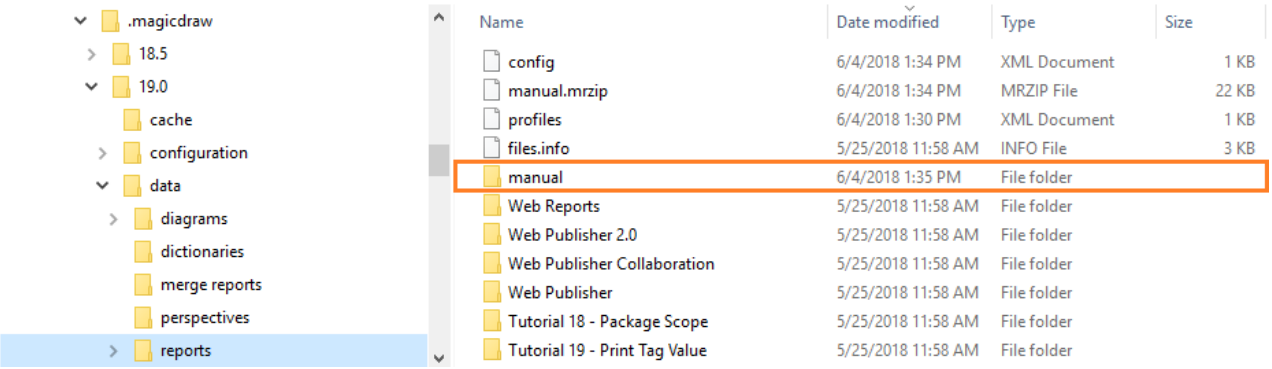
An MRZIP file is a report template package file. You can place an MRZIP file into your template folder *<user folder>/AppData/Local/.magicdraw/19.0/data/reports*, and MagicDraw will automatically deploy the template into the **Report Wizard** dialog.

The following three figures show you how Report Wizard can automatically deploy an MRZIP file.

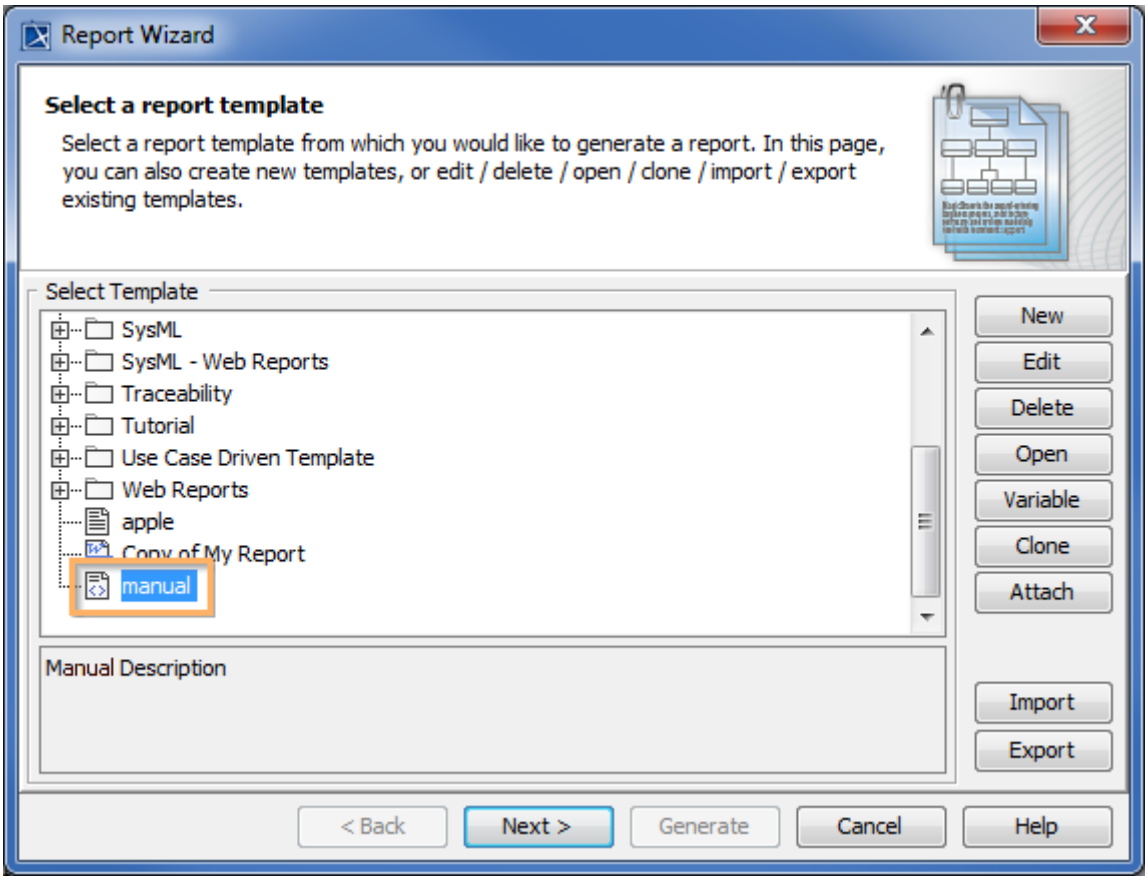


Copying the MRZIP file into a template folder.

335 https://docs.nomagic.com/display/MD190SP1/Report+Wizard+environment+options



Automatic deployment of the MRZIP file.



Report Template installed in the Report Wizard dialog.

Generating reports from the Containment tree

After creating a Report Data in your model, you can generate a report directly from the Containment tree. You can right-click either Model or Report Data to generate it.

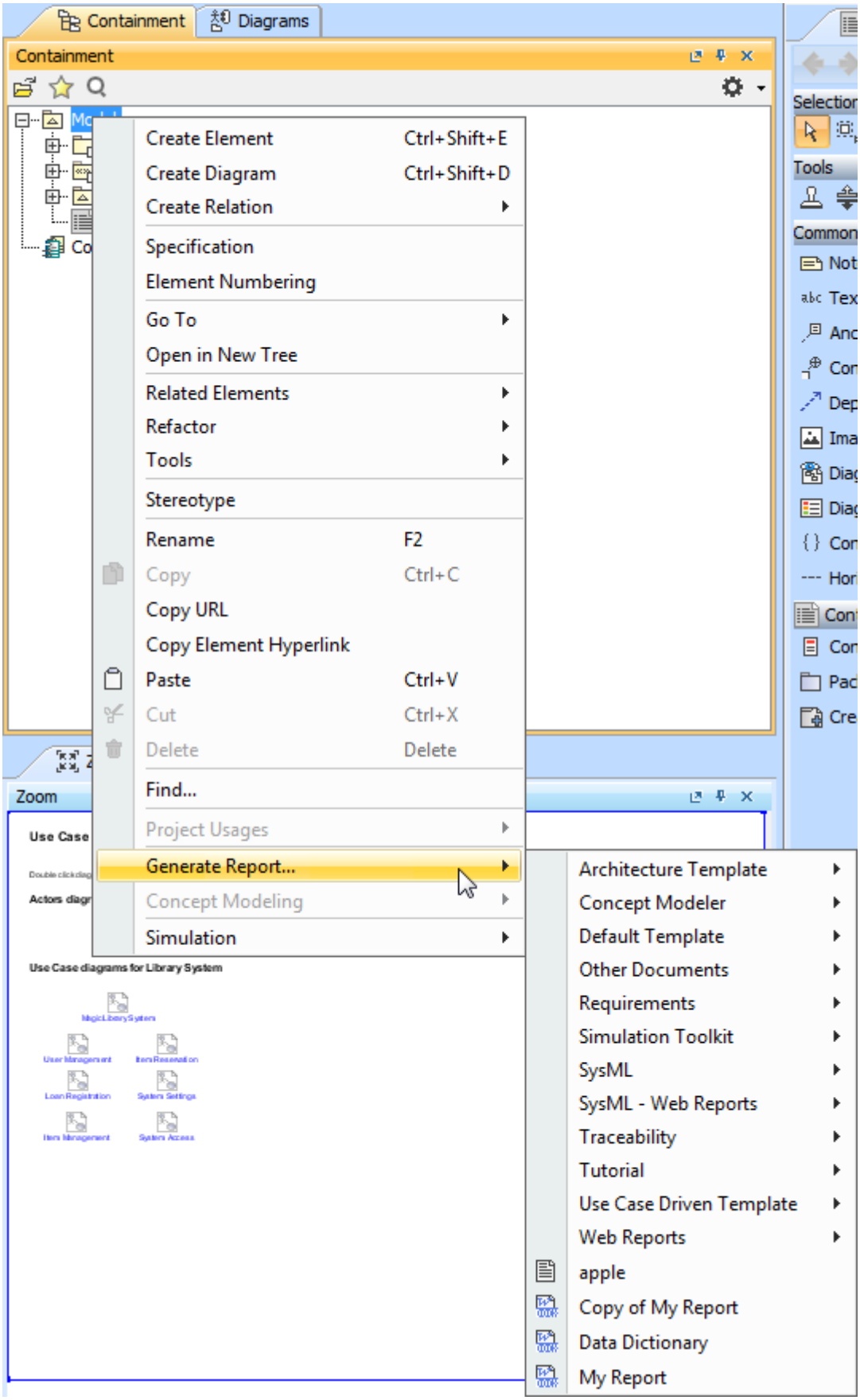
- Generating a report from Model in the Containment tree(see page 1591).
- Generating a report from Report Data in the Containment tree(see page 1593).

You must specify the template and data in the report data to generate a report from the **Report Data** in the Containment tree.

To generate a report from Model in the Containment tree

1. Right-click a package in the Containment tree, select **Generate Report**, and then select a template.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Generating a report from Model in the Containment tree.

2. Select the save location and type the filename. A dialog will open asking if you want to open the report in the default viewer after the report generating process is complete.

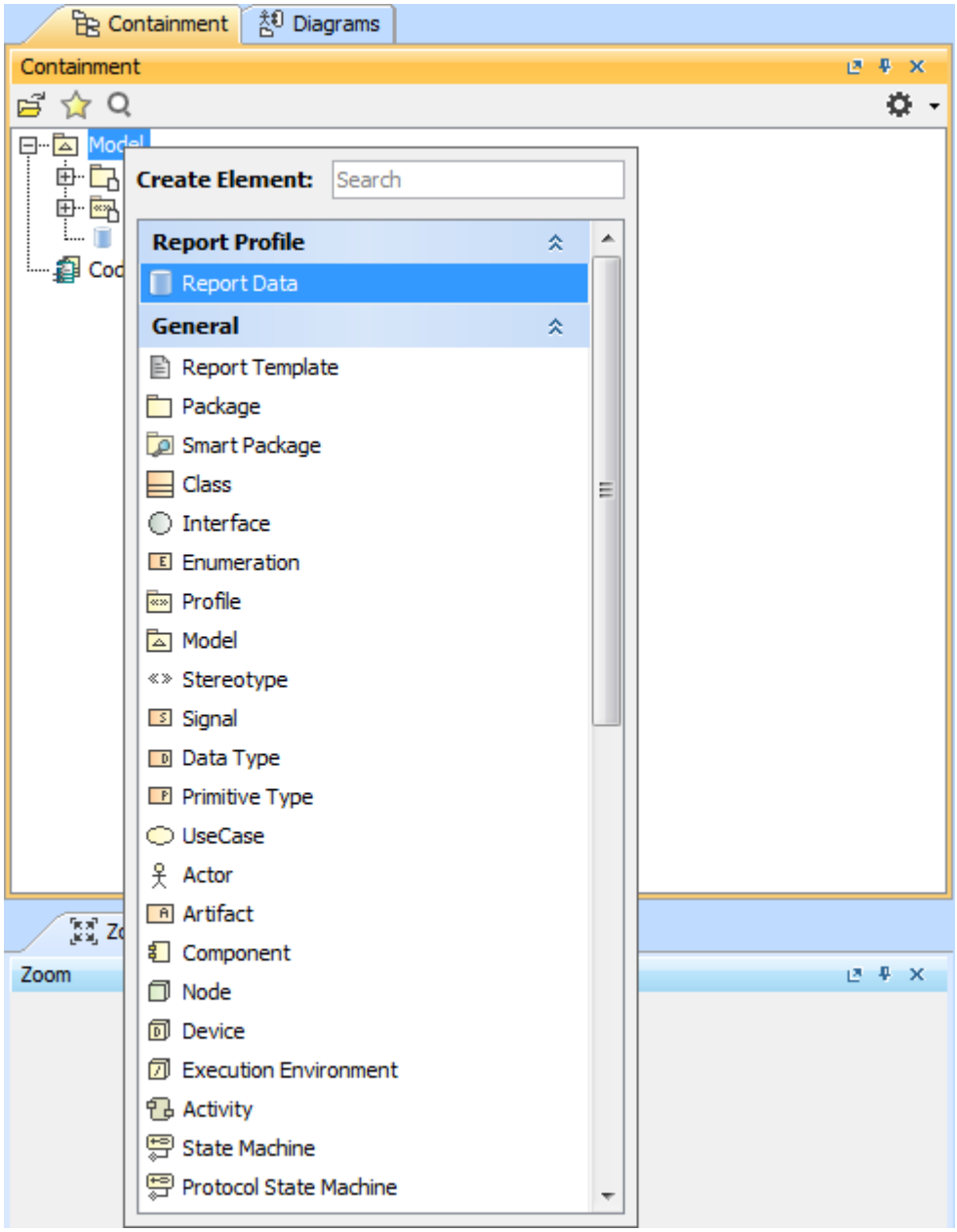
3. Click either Yes or No.

To generate a report from Report Data in the Containment tree

---

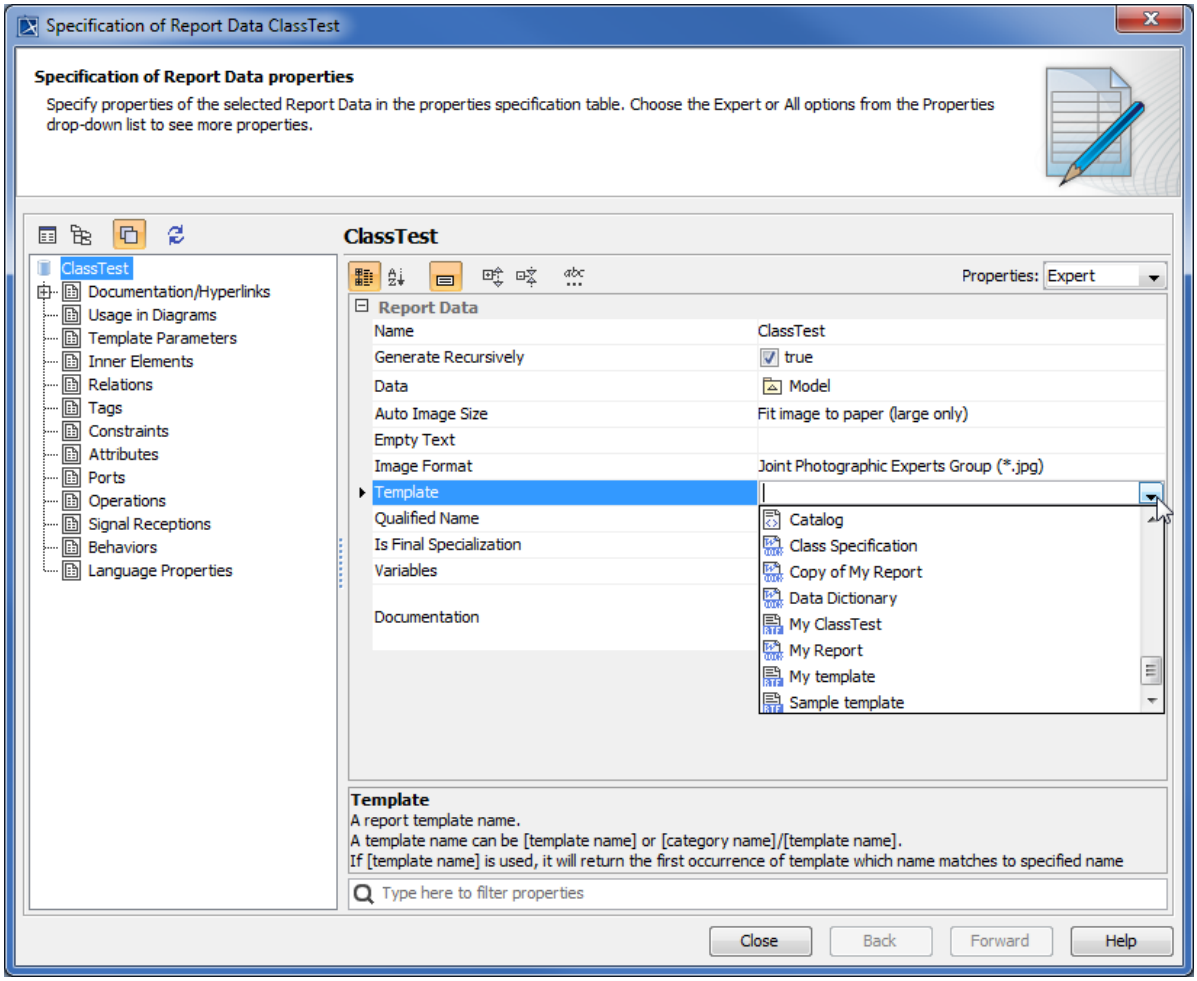
1. Create a project in MagicDraw.
2. Click **File** > **Use Project** > **Use Local Project**. The **Use Project** dialog will open.
3. Select the predefined path: **<install.root>\profiles** and the predefined profile: **Report Profile**
4. Click Finish. The Report Profile will be added to your project in read-only mode.
5. Right-click Model and select **Create Element**.
6. Select **Report Data**. **Report Data** will be created under Model.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



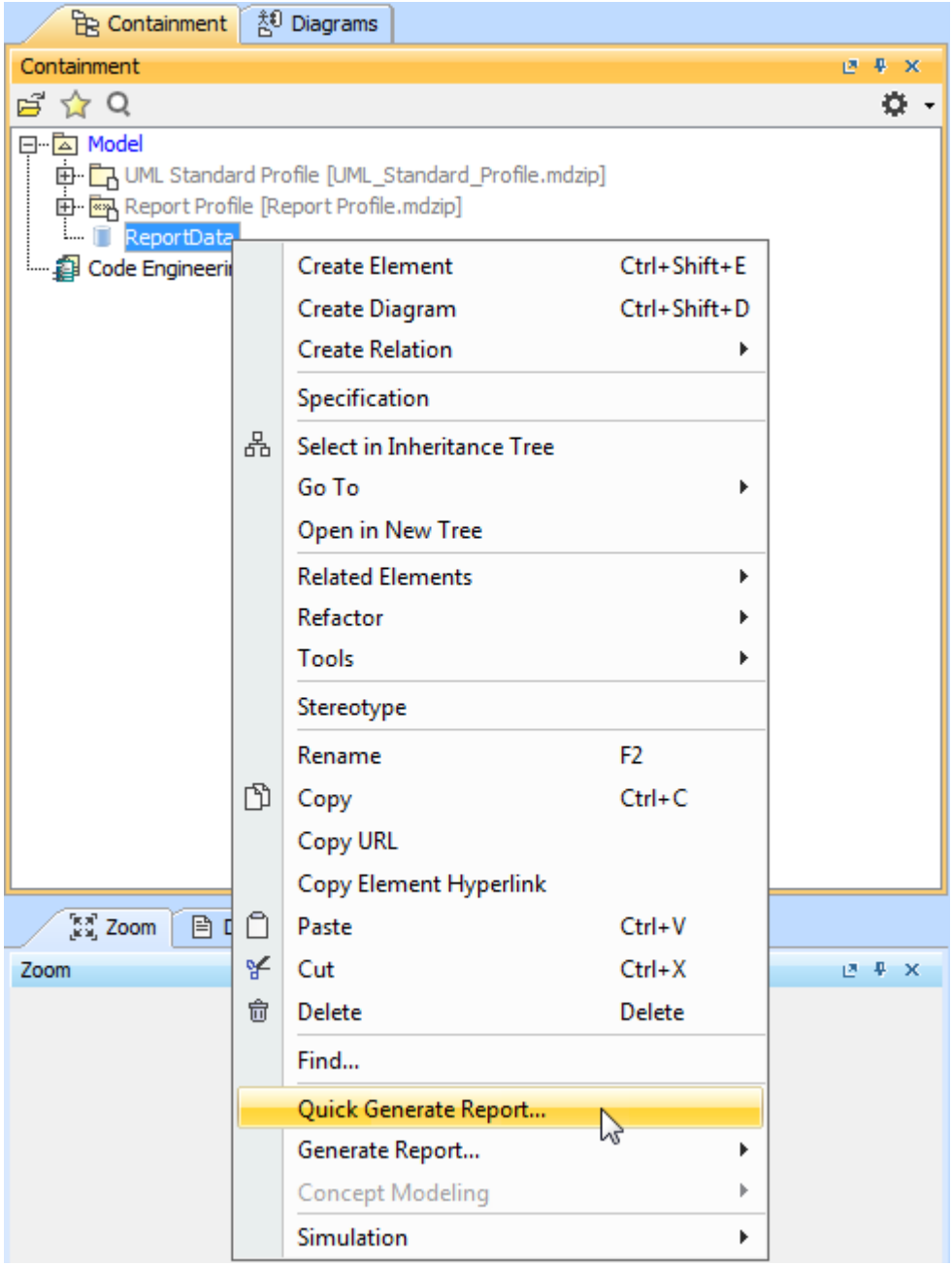
Creating report data in the Model in the Containment tree.

7. Right-click the Report Data in the Containment tree, and select Specification to open its **Specification** dialog.
8. Select a template file, and click Close.



Selecting a template file for the Report Data using the Specification dialog.

9. Right-click the Class element in the Containment tree, and select **Quick Generate Report**. The **Select Location** dialog will open.



Generating a report from Report Data in the Containment tree.

10. Either create a new file, or select an existing one (the content of the generated report will override that of the selected file).

- To open the **Report Wizard** dialog and modify the data or output properties, click the **Report Wizard** on the shortcut menu.
- By default, the **Quick Generate Report** command will select the default settings from the last changes saved to the **Report Data**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Glossary

- **Model**
  The package that is placed at the highest level of the Containment tree in MagicDraw.
- **Report data**
  A collection of variables in a report in Report Wizard.
- **Report template**
  A file that serves as a starting point for a new report.

More information about generating reports in Report Wizard

To learn more about generating reports using Report Wizard, you can visit Generating Reports from Report Wizard(see page 1734).

Report Wizard Quick Print

Report Wizard provides five keyboard shortcuts to generate a report, with last used options, from a recently-used template. The table below lists the predefined keyboard shortcuts:

Keyboard Shortcut	Quick Print Template
Alt + 1	Template name A
Alt + 2	Template name B
Alt + 3	Template name C
Alt + 4	Template name D
Alt + 5	Template name E

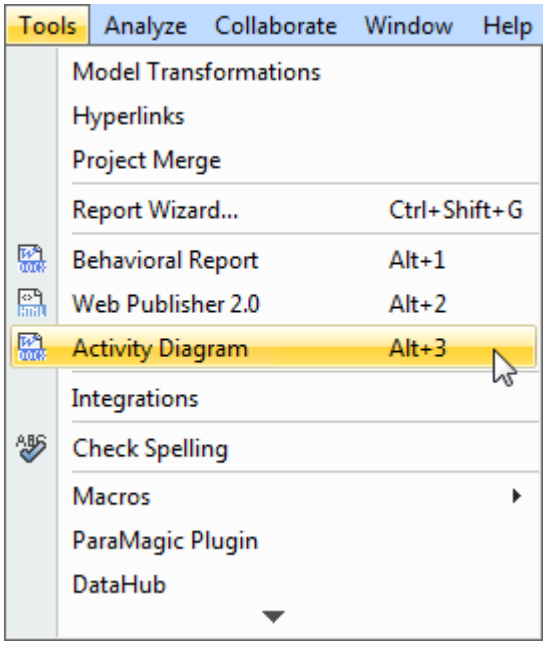
To change a keyboard shortcut

1. On the main menu, click **Options** > **Environment** to open the **Environment Options** dialog.
2. Select **Keyboard** from the tree list.
3. Select **Tools** from the **Category** drop-down menu.
4. Press the new shortcut keys on your keyboard, and click Assign.

To use Report Wizard Quick Print

1. After a report has been generated, a new menu will appear below the **Report Wizard** menu. The name of the new menu is the name of the recently generated template.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Newly generated reports are listed on the Tools main menu.

2. Either click the template name, or press **Alt+1**, **Alt+2**, or **Alt+3** to generate this template with the last used options.
3. Select the saved location and filename.
4. A message dialog will open, asking if you want to open the report in the default viewer once the generating process has completed.

> **Info**
> - The next time you generate a report, a new menu will create and map the keyboard shortcuts until the keyboard shortcuts list is fully filled. The number of keyboard shortcuts is limited to five.
> - If the keyboard shortcuts list is full, and the new template has been generated, the keyboard shortcuts will be reused.

The table below lists some of the keyboard shortcuts provided by Report Wizard Quick Print:

Shortcut Key	Template Name
Alt + 1	Tutorial 01 – Print Element
Alt + 2	Tutorial 02 – Print Diagram
Alt + 3	Tutorial 03 – Print Project
Alt + 4	Tutorial 04 – Print Report Data
Alt + 5	Tutorial 05 – Macro

The table above shows that the Tutorial 01 – Print Element keyboard shortcut is the most outdated template, and the following report has been completed using the template. Report Wizard Quick Print will reuse the keyboard shortcut of the most outdated template.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

The table below lists commonly re-used keyboard shortcuts:

Shortcut Key	Template Name
Alt + 1	Business Process Diagram
Alt + 2	Tutorial 02 – Print Diagram
Alt + 3	Tutorial 03 – Print Project
Alt + 4	Tutorial 04 – Print Report Data
Alt + 5	Tutorial 05 – Macro

If the next template to be generated has already been in the Quick Print list, the list will not change.

Glossary

- **Template**
  A file that serves as a starting point for a new report.

More information about Report Wizard options

To learn more about Report Wizard environment options, you can visit Customizing environment options(see page 65).

Report Wizard template language

Report Wizard is built on Velocity Engine. Knowledge of the Velocity Template Language and the Report Wizard Custom Language used within the Report Wizard template is necessary for understanding, editing, or creating templates.

Related pages

Velocity template language

The user guide for Velocity (VTL) can be found at: http://velocity.apache.org/engine/devel/user-guide.html.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Important

- The formal reference such as ${hello} is not supported in RTF templates.
- A macro cannot be used as macro parameter in any RTF report template. This is a limitation of VTL itself: Velocity treats a macro like copy-and-paste content defined in the macro (between #macro and #end) at the inserted position.
- In an RTF report template, the style and formatting defined on a directive will have no effect on the report output. In addition, the paragraph symbol at the end of a line will also be removed.

Report Wizard custom language

On this page

- #forrow directive(see page 1600)
- #forpage directive(see page 1601)
- #forpage directive for OpenDocument Presentation and Spreadsheet(see page 1601)
- #forcol directive for OpenDocument Spreadsheet, Excel Workbook, Word Document, and PowerPoint Presentation(see page 1602)
- #include and #parse directives(see page 1604)
- #includeSection directive(see page 1605)
- #include, #parse, and #includeSection directives: A comparison(see page 1605)

#forrow directive

The Velocity template language(see page 1599) does not support loops inside a table structure. However, the Report Wizard engine introduces a new custom syntax that allows looping inside the table structure in order to clone the table rows.

```
The syntax is : #forrow <query data> #endrow
```

For example:

Name	Owner
#forrow ($uc in $UseCase) $uc.name	$uc.owner.humanName #endrow

#forrow ($uc in $UseCase) $uc.name is used for printing a use case that one row will be printed for one use case. However, there are two use cases in this example, thus there are two rows printed out. The output will be as follows:

Name	Owner
UC1	Package Requirement
UC2	Package Requirement

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Tip

- Statements preceding the **#forrow** directive, but in the same row or line will execute after the **#forrow** directive because **#forrow** is a first priority directive, while statements in the preceding rows will execute in normal sequence.
- Other statements like importing a library, or creating an array variable, have to be declared in the rows preceding the **#forrow** directive.

#forpage directive

The **#forpage** directive is used to provide a loop over the codes within a page. This directive provides implementation like **#forrow**, but it creates a loop over a page instead of a row. For example:

```
#forpage($uc in $UseCase)
$uc.name
#endpage
```

The report will contain a UseCase name for each document page.

Tip

- Statements preceding the **#forpage** directive, but in the same page will execute after the **#forpage** directive because **#forpage** is a first priority directive, while statements in the preceding pages will execute in normal sequence.
- Other statements like importing a library, or creating an array variable, have to be declared in the pages preceding the **#forpage** directive.

#forpage directive for OpenDocument Presentation and Spreadsheet

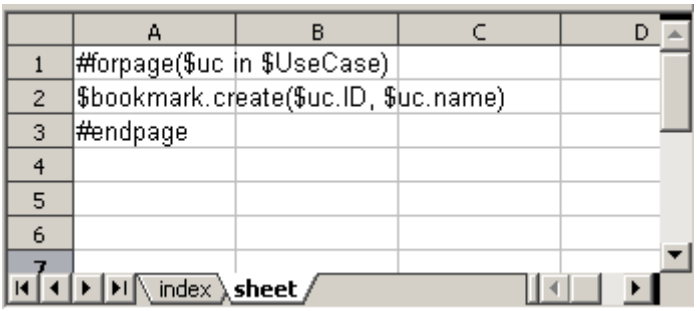
When this directive appears in the OpenDocument Presentation template(see page 1939), it will create a loop over all directives that are present on the current page. All directives on this page will be included inside **#forpage** as shown in the figures below.

An example of a **#forpage** directive.

```
#forpage ($uc in $UseCase)
$uc.name
#endpage
```

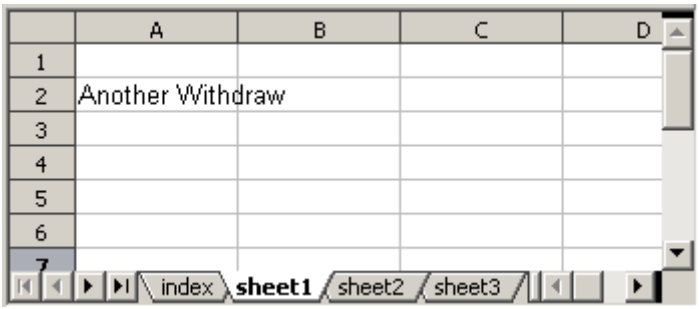
In OpenDocument Spreadsheet template(see page 1939), you can also use the directives as a template. The figure below displays an example of the **#forpage** directive in an OpenDocument Spreadsheet (ODS) file. When this directive is used in an ODS template, it will create a separate spreadsheet representing each name of the use cases in the model.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The sample of #forpage directive to create a template in an OpenDocument Spreadsheet file.

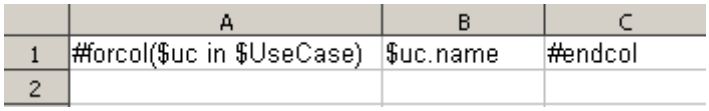
The figure below shows the output after including the directive code shown in the figure above. **Another Withdraw** is the name of one of the use cases in the model.



The output after including the #forpage directive in an OpenDocument Spreadsheet file.

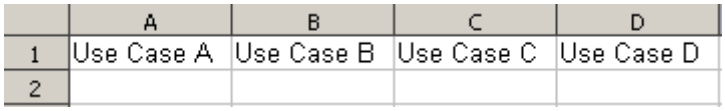
#forcol directive for OpenDocument Spreadsheet, Excel Workbook, Word Document, and PowerPoint Presentation

This directive is designed only for the spreadsheet templates, which are OpenDocument Spreadsheet (ODS), Excel Workbook (XLSX), Word Document (DOCX), and PowerPoint Presentation (PPTX). This directive provides looping over the columns. In the figure below, the engine will generate a report with different columns for each Use Case name.



The sample of #forcol directive to create a template in an OpenDocument Spreadsheet file.

The output from this sample will be as shown in the figure below.



The output after including the #forcol directive in an OpenDocument Spreadsheet file.

You can combine both **#forrow** and **#forcol** directives and produce a more complex report output as shown below.

	A	B	C
1	#forrow($p in $Package)	#forcol($e in $p.ownedElement)$e.name#endcol	#endrow

The sample of #forrow and #forcol directives to create a template in an OpenDocument Spreadsheet file.

The figure below shows an output generated from the Magic Library.mdzip[336] sample project.

	A	B	C	D	E	F
1	Domain Analysis	Diagram Loan Collaboration	Diagram Loa	Diagram Obje	Dependency	Dependency
2	conceptual perspective					
3	Implementation	Node Remote Web Client	Node Remot	Artifact Intern	Artifact Magi	Artifact Magi
4	High Level Domain Analysis	Diagram conceptualPerspective1	Diagram con	Class Penalt	Association i	Association i
5	collaboration	Collaboration Loan Library Item				
6	Analysis and Design	Diagram Analysis and Design	Package Hig	Package Dor	Package Dor	Package Tech
7	object diagram	Instance Specification	Instance Spe	Instance Spe	Instance Spe	Instance Spe
8	Technical Design	Diagram domain.user	Diagram dom	Diagram dom	Diagram Mac	Package Magi
9	MagicLibrary	Instance Specification	Class Library	Class Books	Class VideoP	Class AudioP
10	Domain Design	Diagram Package Dependencies	Diagram Imp	Dependency	Dependency	Package Impl
11	composite structure	Collaboration Loan	Class Partici	Collaboration	ReadingItem	Loan
12						

The output after including the #forrow and #forcol directives in an OpenDocument Spreadsheet file.

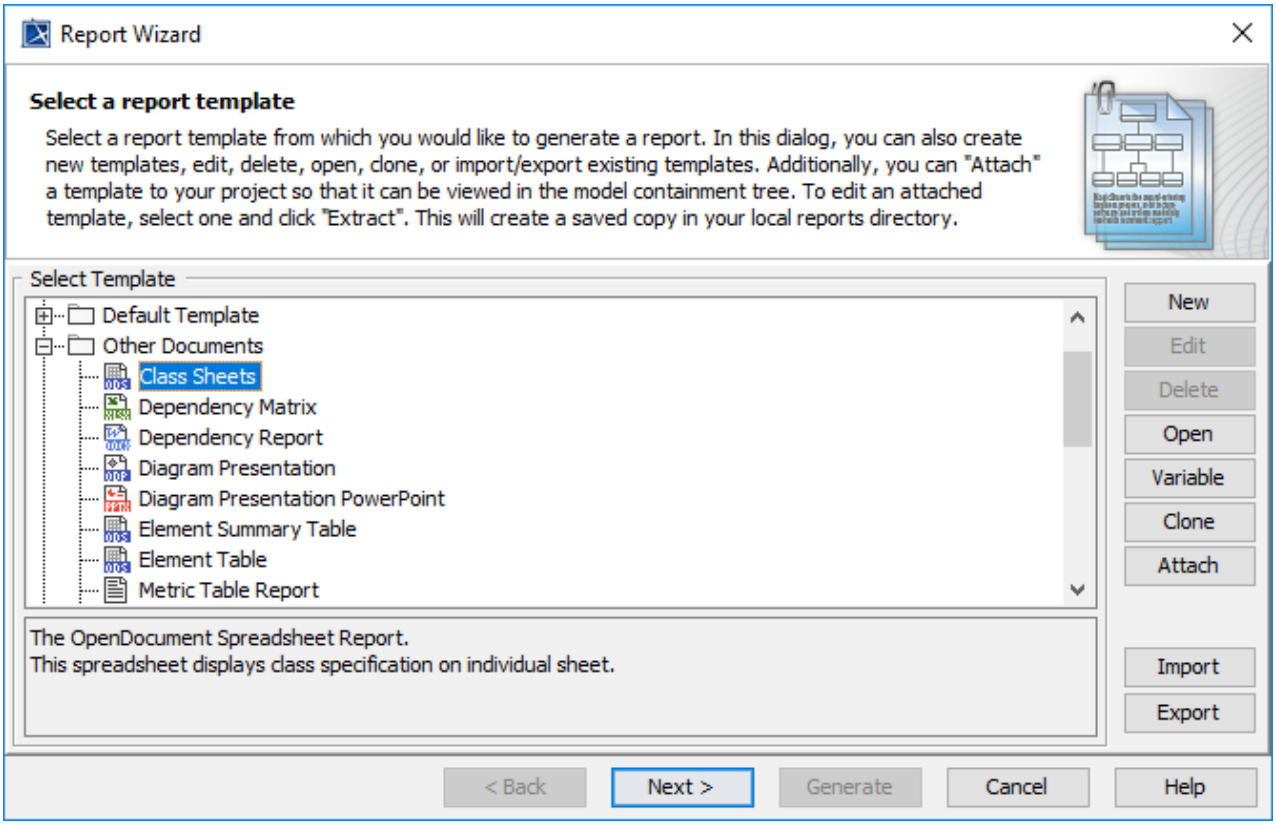
Since the **#forrow** syntax is similar to the **#foreach** syntax, the **#foreach** syntax can then be used in **#forrow**.

You can find more examples of using the **#forrow** and **#forcol** directives in the **Other Documents** templates in the Report Wizard dialog(see page 1547).

To open an **Other Documents** template in the **Report Wizard** dialog(see page 1547)

1. On the main menu, click **Tools** > **Report Wizard**. The **Report Wizard** dialog opens.
2. In the **Select Template** tree, expand **Other Documents**.
3. Select any template and click **Open**.

[336] https://docs.nomagic.com/download/attachments/55844608/Magic%20Library.mdzip?api=v2&modificationDate=1510562160640&version=1



#include and #parse directives

The original Velocity Engine[337] provides two include directives: **#include** and **#parse**.

#include allows you to import another template. The contents of the file will not be rendered through the template engine.

To include a filename using **#include($fileName)**, for example, type either:

```
#set($fileName = $attachedFile.file)
#include($fileName)
```

or

```
#include("document.docx")
```

Information

- **$fileName** can be either an absolute or relative path to a file in the file system.
- **$fileName** can be only static or dynamic String.
- If the file is not found in the file system, **$fileName** would be searched as an Attached File Element in a MagicDraw project in the File property.

337 http://velocity.apache.org/engine/1.7/user-guide.html#include

#parse allows you to import another template. The contents of the file will be rendered through the template engine. However, the file being included will be inserted with all contents.

#includeSection directive

Report Wizard introduces a statement, which allows a template to include any section of a document from another template. This statement requires the template to define the beginning and the end of the section.

The logical concept of the **#includeSection** and **#parse** directives is similar. Both directives allow a template to include another template and render it through the template engine. However, **#includeSection** can be used to specify only the section that you would like to include.

To declare a section, for example, type:

```
#sectionBegin ('sectionA')
...
#sectionEnd
```

To include a section, for example, type:

```
#includeSection('document.docx', 'sectionA')
```

Information

- **fileName** and **sectionName** must be static String only, e.g., **#includeSection('document.docx', 'sectionA')**.
- If the file is not found in the file system, **fileName** would be searched as an Attached File Element in a MagicDraw project in the File property.
- **#sectionBegin** and **#sectionEnd** must be in the template body. Headers and footers are not allowed.

#include, #parse, and #includeSection directives: A comparison

The **#include** and **#parse** directives are built-in directives provided by Velocity. The **#includeSection** directive is a custom directive implemented by MagicDraw. The table below shows the differences among these three directives.

	#include	#parse	#includeSection
Execution Time	Executed at runtime.	Executed as a separate template at runtime.	Executed at translation time.

	#include	#parse	#includeSection
Variable Scope	Variables declared in the parent template are not accessible in the included page.	Variables declared in the parent template can be accessed in the included page.	Variables declared in the parent template can be accessed in the included page.
Rendering	The Include template is not rendered through the template engine.	The Include template is rendered through the template engine as a separate process.	The Include template is rendered through the template engine as a single process.
Include only required section	No	No	Yes
Size of parent template	The size of the parent template remains unchanged.	The size of the parent template remains unchanged.	The size of the parent template is increased by the included section.
Processing overhead	The #include directive increases the processing overhead with the necessity of an additional call to the template engine.	The #parse directive increases the processing overhead with the necessity of an additional call to the template engine.	The #includeSection directive does not increase the processing overhead.
Support RTF template	No	No	Yes
Support ODF template	No	No	Yes
Support DOCX template	No	No	Yes
Support XLSX template	No	No	No
Support PPTX template	No	No	No

Related pages

- Creating a template for an OpenDocument Presentation (ODP) or Microsoft PowerPoint presentation (PPTX)(see page 1939)
- Creating a template for an OpenDocument spreadsheet or Microsoft Excel spreadsheet(see page 1939)
- Velocity template language(see page 1599)
- Velocity engine: #include and #parse directives[338]

338 http://velocity.apache.org/engine/1.7/user-guide.html#include

Formatting template code

Report Wizard supports the **Tab** key to help you format template code at ease. The **Tab** key is useful to indent velocity code. You can press the **Tab** key after the symbol "#" or "$" followed by the velocity code. The **Tab** key can help you format the template code, but it does not affect the output of a generated report. See figure below for example.

```
#foreach($c in $Class)¶
#  →  if($c.name == "Target Class")¶
$  →     →  c.name¶
#  →  end¶
#end¶
```

The velocity code in the figure above uses the **Tab** key to indent code. However, the generated report does not include those indents. The **Tab** key support is available in *RTF*, *DOCX*, and *ODT* format files.

Unparsed code

Velocity allows the template designer to easily use a large amount of uninterpreted and unparsed content in *VTL* code. This is especially useful if you want to avoid multiple directives or sections whose content would otherwise be invalid (and thus unparseable) *VTL*.

The syntax is as follows:

```
#[[ Unparsed Code ]]#
```

For example, the following code:

```
#[[
#foreach($c in $Class)
$c.name
#end
]]#
```

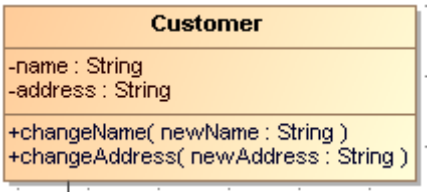
Will be rendered as:

```
#foreach($c in $Class)
$c.name
#end
```

ⓘ The Unparsed Code support is available in *HTML*, *XML*, and *text* files only.

Template variables

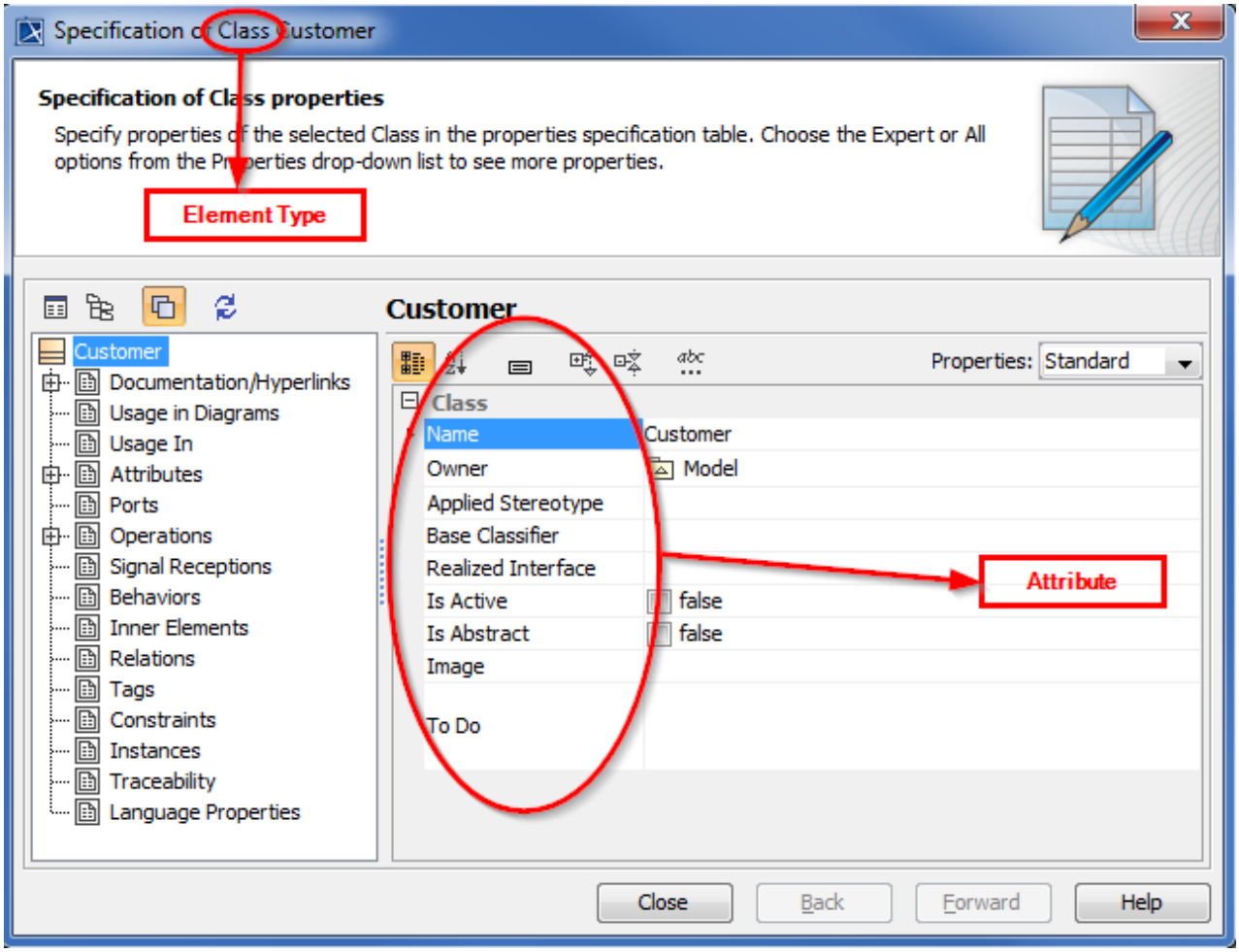
The variables imported to the template are collected by the type of element selected in the package scope. Use the fourth step in the **Report Wizard** dialog to select the package scope. In this example, we take a class diagram with the class name Customer, see figure below, to print it in a report.



Class Diagram: Customer.

To print attributes of the Customer class in a report

- Right-click the **Customer** class diagram and open the **Specification** dialog. The element type will appear on the dialog title, and the attribute names will appear on the right-hand side of the dialog box (see the following figure).



The Specification dialog.

With the following VTL code, you can print the attributes of the Class element:

```
#foreach ($class in $Class)
   Name: $class.name
   Owner:  $class.owner.name
   Visibility: $class.visibility
   Is Abstract: $class.isAbstract
#end
```

The output will be as follows:

```
Name: Customer
Owner:  com
Visibility: public
Is Abstract: false
```

The table below gives and explains the additional properties which are not part of the UML specifications, but retrievable by Report Wizard.

Element Owner	Property Name	Function
Diagram	image	To return the diagram's image.
Diagram	diagramType	To return the diagram's type.
Element	image	To generate the element's image or to print an empty text if the element does not refer to any diagram.
Element	elementType	To return a name of an element's type (metaclass / stereotype) in lowercase letters and without spaces. For example: • return "usecase" for a Use Case • return "class" for a Class • return "callbehavioraction" for a Call Behavior Action • return "flowport" for a Flow Port (a stereotype defined in SysML)
Element	humanName	To return an element's human readable name, which is a string concatenation between the human element type and the element's name.
Element	humanType	To return an element's type in a human readable format.
Element	appliedStereotype	To return an applied stereotype of an element.
Element	activeHyperlink	To return an active hyperlink of an element.
Element	hyperlinks	To return all hyperlinks attached to an element.
Element	toDo	To return an element's ToDo attribute in text format.
Element	elementID	To return an element's ID.
Element	documentation	To return an element's comment/documentation.
Element	presentation	Element To return a presentation element.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Element Owner	Property Name	Function
Element	typeModifier	To return an element's type modifier.
Element	tags	To return a list of element tags.
Element	text	To return a text representation of an element. In general, a 'text' property returns a string which "textually represents" the element. The string may vary depending on the symbol properties and environment options. This property returns the result from OpenAPI. `RepresentationTextCreator.getRepresentedText(Element element)`
Element	slots	To return a list of element's slots.
Element	elementURL	To return an element's URL.
Classifier	baseClassifier	To return a base classifier of an element.
BehavioredClassifier	realizedInterface	To return a Classifier's Realized Interface.
Package	appliedProfile	To return a list of profiles applied to a package.
Stereotype	metaClass	To return a Meta Class of a stereotype.
DurationConstraint	min	To return a minimum value of the Duration Constraint.
DurationConstraint	max	To return a maximum value of the Duration Constraint.
TimeConstraint	min	To return a minimum value of the Time Constraint.
TimeConstraint	max	To return a maximum value of the Time Constraint.
MultiplicityElement	multiplicity	To return a multiplicity value.
Property	navigable	To return a navigable value.
Property	ownedBy	To return a property's owner.
Lifeline	type	To return a lifeline type.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Element Owner	Property Name	Function
Message	event	To return a message event.
Message	operation	To return a message operation.
Message	signal	To return a message signal.
Message	number	To return a message number.
Trigger	eventType	To trigger an event type.
RequirementsUseCase	number	To return a use case number.

Some predefined variables, which are not part of the UML specifications, are usable in the templates:

- ***$elements***
  contains a list of all the elements selected from the element scope.
- ***$packageScope***
  contains a selected package from the element scope.
- ***$elementScope***
  contains selected elements from the element scope.
- ***$empty***
  contains a String for empty value, which is specified from the Output Options pane.

humanName and ***humanType*** are locale-specific text. If you plan to generate a report in different languages, try to avoid ***humanType*** and ***humanName***. Instead, use ***elementType*** for a non-locale-specific report.

Related pages

Getting property value

You can directly get the property value of the element by calling property name:

```
$element.propertyName
```

propertyName should be typed in camel case style. The name and returned type of property can be found in MagicDraw UMLMetaModel UserGuide. It lists all the properties which can be called through each element type.

Please be aware of the type of the returned value, as some properties may return an array. You can get each value by using a **#foreach** directive or by specifying array index number. For example:

```
#foreach($c in $class.classifier)
$c.name
#end
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

```
#set($classifier = $class.classifier)
$classifier.get(0)
$classifier.get(1)
```

Moreover, the customized elements are able to get their property value immediately by calling property name:

```
$RepresentationText.propertyName
```

You can see the representation text of the element by opening the Specification dialog.

The dialog name pattern is

```
Specification of <RepresentationText> <ElementName>
```

While the property name is shown in the same dialog.

If the property value is not printed out to your report, you can assume the property that you call is not the property of the customized element. It may be the property of the applied stereotype. You would rather check the stereotype owner of that property by going to the Applied Stereotype property of the element, then right click on the right column, and select the stereotype in the containment tree.

If you navigate to the stereotype element on the containment tree, then you can expand it to see its owned properties. If you can see your desired property in the stereotype, you can get its value by:

```
$report.getStereotypePropertyString($element, "stereotypeName", "propertyName")
```

We also provide helper module in section Helper tools(see page 1621). These modules are utility modules for getting property values. More details about openAPI of each element can be found at:

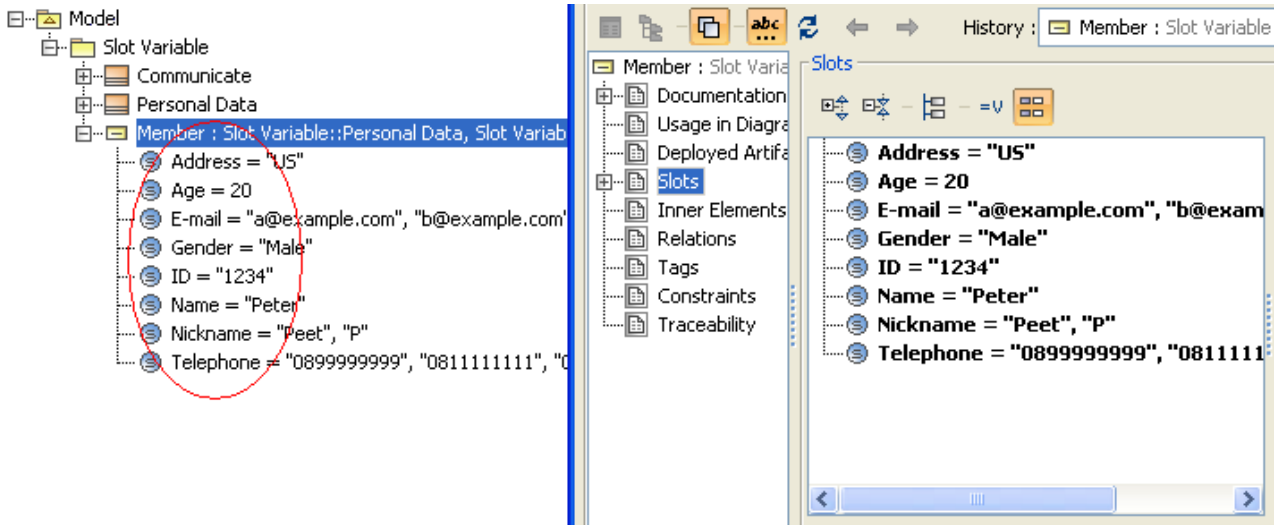
{md_install}\plugins\com.nomagic.magicdraw.reportwizard\api\javadoc.zip

Retrieving Slot Values

On this page

- Using normal UML Specification to Retrieve Slot Values(see page 1614)
- Using Slot Property to Retrieve Slot Values(see page 1614)
  - Retrieving slot information from an element(see page 1614)
  - Code examples to retrieve slot values(see page 1616)

Instead of using Stereotype and Tag, some projects may use Instance Specification and Slot to define new attributes of an element.



Instance Specification and Slot defining element attributes.

You can retrieve a slot value by either using a normal UML Specification structure, or by using a slot property.

Using normal UML Specification to Retrieve Slot Values

For example, to retrieve a Name slot value type:

```
#foreach($instance in $InstanceSpecification)
    #foreach($slot in $instance.slot)
        #if($slot.definingFeature.name == 'Name')
            #if($slot.value.size() > 0)
                #set($v = $slot.value.get(0).text)
            #else
                #set($v = "")
            #end
            Slot value is $v
        #end
    #end
#end
```

Using Slot Property to Retrieve Slot Values

You can use a slot property as a shortcut to get the following information from a slot.

Retrieving slot information from an element

The following code retrieves slot information from an element.

```
$element.slots
```

In which:

- $element is an element.
- slots is a property for getting slot variables.

For example, if you want to obtain a collection of slot variables of an element, type:

```
#foreach($slot in $instanceSpecification.slots)
    $slot.name
#end
```

Retrieving slot information from an element and a classifier's name
The following code retrieves slot information from an element and a classifier's name.

```
$element.slots.ClassifierName
or
$element.slots.get("ClassifierName")
```

Where:

- $element is an element.
- slots is a property for getting the slots variable.
- ClassifierName is the classifier's name.

The code returns:

- A list of slot variables whose classifier names are ClassifierName.
- If there are multiple classifiers that match the ClassifierName, the code returns the slot property of the first classifier.
- If there is no classifier that matches the ClassifierName, the code returns null.

For example, if you want to retrieve all slot values from the "Communicate" classifier, type:

```
$instanceSpecification.slots.Communicate
$instanceSpecification.slots.get("Communicate")
```

Retrieving slot value from an element, a classifier's Name, and a defining feature name

```
$element.slots.ClassifierName.DefiningFeatureName
or
$element.slots.ClassifierName.get("DefiningFeatureName")
or
$element.slots.get("ClassifierName").DefiningFeatureName
or
$element.slots.get("ClassifierName").get("DefiningFeatureName")
```

Where:

- ***$element***
  is an element.
- ***slots***
  is a property for getting slots variable.
- ***ClassifierName***
  is the classifier's name.
- ***DefiningFeatureName***
  is the name of a slot.

The code returns:

- A slot value whose name is DefiningFeatureName and a classifier whose name is ClassifierName.
- If the slot multiplicity is < =1 (less than or equal to 1), it will return the value in ValueSpecification.
- If the slot multiplicity is > 1, it will return the value in array of ValueSpecification.

For example, if you want to retrieve the values of a slot whose name is "Address" and those of a classifier whose name is "Communicate", type:

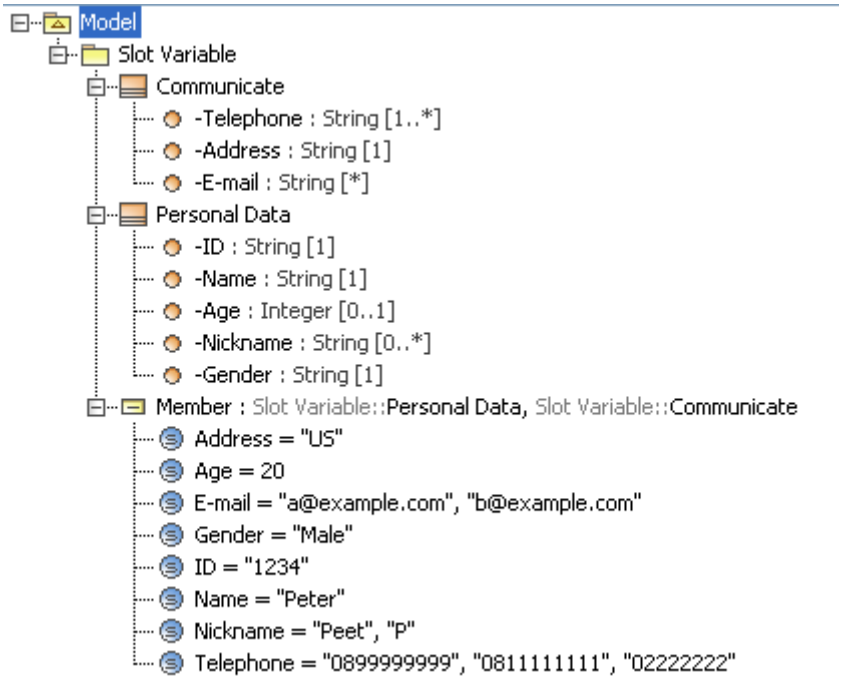
```
$instanceSpecification.slots.Communicate.Address
$instanceSpecification.slots.Communicate.get("Address")
$instanceSpecification.slots.get("Communicate").Address
$instanceSpecification.slots.get("Communicate").get("Address")
```

- ClassifierName and DefiningFeatureName are case-sensitive properties.
- You can use:
  `$report.getSlotValue($element, ClassifierName, DefiningFeatureName)`

  to retrieve the slot value. See $report below for further details.

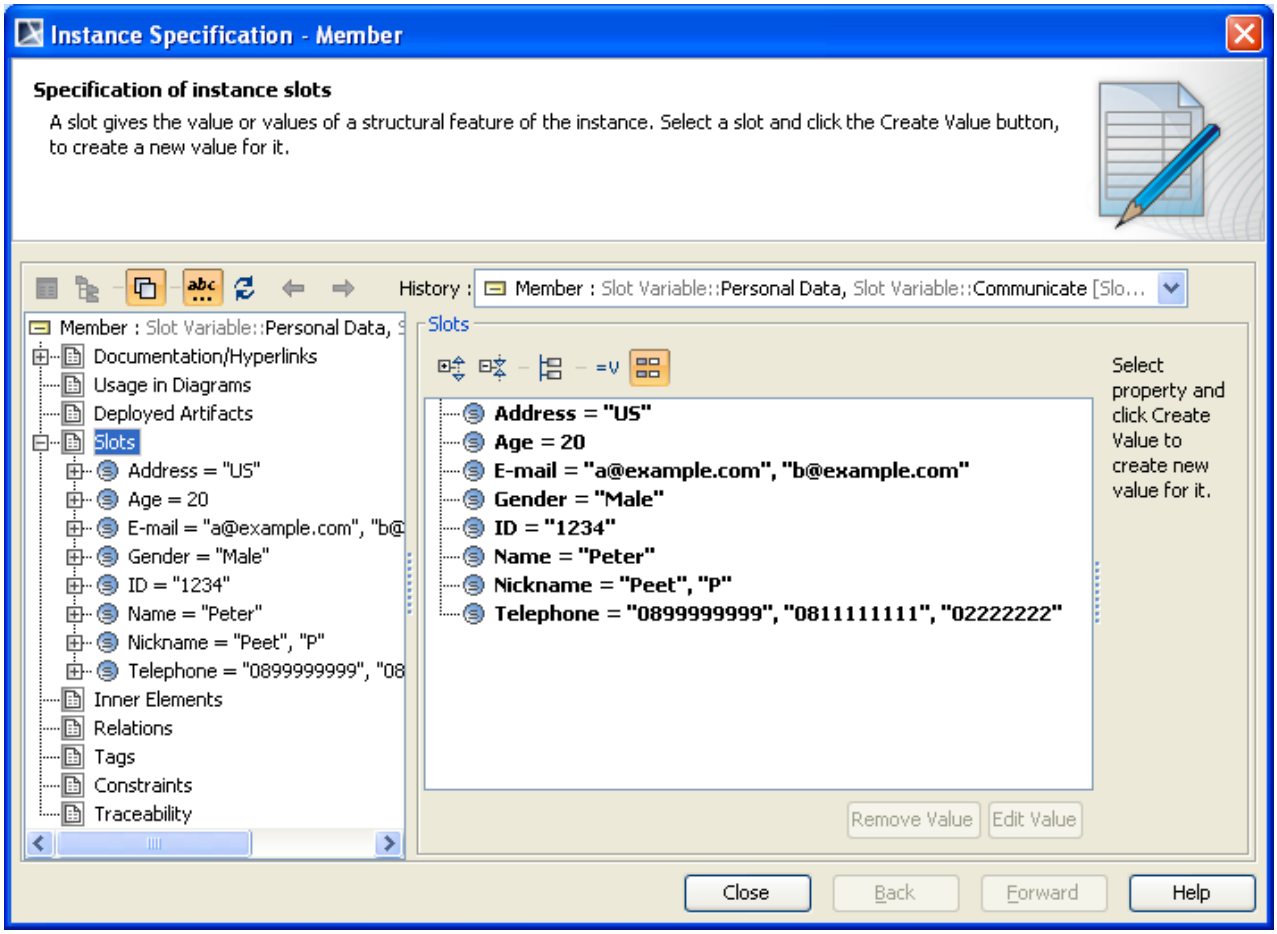
Code examples to retrieve slot values

This section explains how to get the value of an instance slot from a classifier. The figure below shows the elements of a slot variable in the Containment tree in MagicDraw. You can see the values of the element **Member** in the tree, or you can right-click on it and open the Specification dialog to see the values in the dialog.



An example of elements in the Containment tree.

The following figure shows the Instance Specification dialog of the element Member of the slot variable. As you can see in the dialog, the element Member of the slot variable contains slot names and values.



All slots of Instance Specification.

The following are examples of using the code to retrieve a value of a slot.

Example 1.

If you want to print the value of a slot **Address** of the element **Member** from the classifier **Communicate,** you can use the following template code as a shortcut. Note that the slot multiplicity = 1.

```
#foreach($instance in $InstanceSpecification)
    Slot value is $instance.slots.Communicate.Address
#end
```

The output of the code shown above is :

```
Slot value is US
```

Example 2.

If you want to print the value of a slot **Telephone** of the element **Member** from the classifier **Communicate**, you can use the following template code as a shortcut. Note that the slot multiplicity is > 1, [1..*].

```
#foreach($instance in $InstanceSpecification)
    Slot value is $instance.slots.Communicate.Telephone
#end
```

The output of the code shown above is:

```
Slot value is [0899999999, 0811111111, 02222222]
```

Example 3.

If you want to print the value of a slot Name of the element Member from the classifier Personal Data, you can use the following template code as a shortcut. Note that the slot multiplicity = 1.

```
#foreach($instance in $InstanceSpecification)
    Slot value is $instance.slots.get("Personal Data").get("Name")
#end
```

The output of the code shown above is:

```
    Slot value is Peter
```

Example 4.

If you want to print the value of a slot **Nickname** of the element **Member** from the classifier **Personal Data**, you can use the following template code as a shortcut. Note that the slot multiplicity is > 1, [0..*].

```
#foreach($instance in $InstanceSpecification)
    Slot value is $instance.slots.get("Personal Data").get("Nickname")
#end
```

The output of the code shown above is:

```
Slot value is [Peet, P]
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example 5.

If you want to use $Report method to print the value of a slot **Name** of the element **Member** from the classifier **Personal Data**, you can use the following template code.

```
#foreach($instance in $InstanceSpecification)
    Slot value is $report.getSlotValue($instance, "Personal Data", "Name")
#end
```

The output of the code shown above is:

```
Slot value is [Peter]
```

Example 6.

If you want to use $Report method to print the value of a slot **Nickname** of the element **Member** from the classifier **Personal Data**, you can use the following template code.

```
#foreach($instance in $InstanceSpecification)
    Slot value is $report.getSlotValue($instance, "Personal Data", "Name")
#end
```

The output of the code shown above is:

```
 Slot value is [Peet, P]
```

Example 7.

If you want to print the names and values of all classifiers and the slot properties of all instance elements, you can use the following template code as a shortcut.

```
#foreach($instance in $InstanceSpecification)
  #foreach($classifier in $instance.slots)
    Classifier name: $classifier.name
    Slots:
      #foreach($slot in $classifier)
       $slot.name : $slot.value
      #end

  #end
#end
```

The output of the code shown above is:

```
Classifier name: Personal Data
    Slots:
       Name : Peter
       Age : 20
       Nickname : [Peet, P]
       Gender : Male
       ID : 1234

    Classifier name: Communicate
    Slots:
       Telephone : [0899999999, 0811111111, 02222222]
       E-mail : [a@example.com, b@example.com]
       Address : US
```

Helper tools

For more details on element types, see OpenAPI(see page 2156).

Report tool

On this page

- $report.createValueSpecificationText(specification)(see page 1623)
- $report.filterDiagram(diagramList, diagramTypes)(see page 1623)
- $report.filterElement(elementList, humanTypes)(see page 1623)
- $report.filterElementType(elementList, elementType)(see page 1624)
- $report.filter(elementList, propertyName, propertyValue)(see page 1624)
- $report.findElementInCollection(elementList, name)(see page 1625)
- $report.findRelationship(modelPackage)(see page 1625)
- $report.findRelationship(modelPackage, recursive)(see page 1626)
- $report.getAppliedStereotypeByName(element, stereotypeName)(see page 1626)
- $report.getBaseClassAssociations(classifier)(see page 1626)
- $report.getBaseClassInheritableAttributes(classifier)(see page 1627)
- $report.getBaseClassInheritableOperations(classifier)(see page 1627)
- $report.getBaseClassPorts(classifier)(see page 1627)
- $report.getBaseRealizedInterfaces(behavioredClassifier)(see page 1627)
- $report.getBaseRelations(classifier)(see page 1628)
- $report.getBaseClassifiers(child)(see page 1628)
- $report.getClientElement(element)(see page 1628)
- $report.getComment(element)(see page 1628)
- $report.getDerivedClassifiers(parent)(see page 1629)
- $report.getDiagramElements(diagram)(see page 1630)
- $report.getDiagramType(diagram)(see page 1630)
- $report.getElementProperties(element)(see page 1630)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- $report.getDSLProperty(element, propertyName)(see page 1631)
- $report.getElementComment(element)(see page 1631)
- $report.getElementName(element)(see page 1631)
- $report.getIconFor(element)(see page 1632)
- $report.getIconFor(element, prefix, suffix, hashCode)(see page 1632)
- $report.getIconFor(type)(see page 1633)
- $report.getIconFor(type, prefix, suffix, hashCode)(see page 1633)
- $report.getIncludeUseCase(useCase)(see page 1634)
- $report.getInnerElement(element)(see page 1635)
- $report.getInteractionMessageType(message)(see page 1635)
- $report.getMetaClass(stereotype)(see page 1635)
- $report.getPresentationDiagramElements(diagram)(see page 1635)
- $report.getPresentationDiagramElements(diagram, includeNonManipulator)(see page 1636)
- $report.getPresentationElementBounds(diagram, element)(see page 1636)
- $report.getPresentationElementBounds(element)(see page 1637)
- $report.getPresentationElementRectangle(diagram, element)(see page 1637)
- $report.getQualifiedName(namedElement, separator)(see page 1637)
- $report.getPackageQualifiedName(namedElement, separator)(see page 1638)
- $report.getReceivingOperationalNode(element)(see page 1638)
- $report.getRelationship(element)(see page 1639)
- $report.getRelationship(element, recursive)(see page 1639)
- $report.getRelativeActor(element)(see page 1639)
- $report.getSendingOperationalNode(element)(see page 1640)
- $report.getSlotValue(element, ClassifierName, DefiningFeatureName)(see page 1640)
- $report.getStereotypeProperty(element, stereotypeName, propertyName)(see page 1641)
- $report.getStereotypeProperty(element, profileName, stereotypeName, propertyName)(see page 1641)
- $report.getStereotypePropertyString(element, stereotypeName, propertyName)(see page 1641)
- $report.getStereotypePropertyString(element, profileName, stereotypeName, propertyName)(see page 1642)
- $report.getStereotypes(element)(see page 1642)
- $report.getSupplierElement(element)(see page 1643)
- $report.getUsageElements(usagesMap, element)(see page 1643)
- $report.getUsages(selectedObjects)(see page 1643)
- $report.hasStereotype(element)(see page 1644)
- $report.containsStereotype(element, stereotypeName)(see page 1644)
- $report.containsStereotype(element, stereotypeName, includeDerived)(see page 1644)
- $report.isDerivedClassifier(parent, child)(see page 1645)
- $report.isNamedElement(element)(see page 1645)
- $report.isNull(obj)(see page 1645)
- $report.isRelationship(element)(see page 1646)
- $report.serialize(hyperlink)(see page 1646)
- $report.getUsedBy(element)(see page 1646)
- $report.hasProperty(element, propertyName)(see page 1647)
- $report.getProperty(element, propertyName)(see page 1647)
- $report.findElementByName(source, regex)(see page 1648)
- $report.getPresentationElements(diagram)(see page 1648)
- $report.getUsageRepresentationText(baseElement, bool)(see page 1649)
- $report.getUseCaseNumber(element)(see page 1650)
- $report.getElementURL(element)(see page 1650)
- $report.isEmpty(obj)(see page 1650)
- $report.getBasicFlows(usecase:UseCase) : List(see page 1651)
- $report.getAlternativeFlows(usecase:UseCase) : List(see page 1652)
- $report.getExceptionalFlows(usecase:UseCase) : List(see page 1653)
- $report.getOwnedElementsIncludingAdditional(element, includePureOwned) : List(see page 1653)

$report is a utility tool enabling a template to get MagicDraw data.

$report.createValueSpecificationText(specification)

Create texts representing the ValueSpecification element.

	Name	Type	Description
Parameter(s)	specification	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.ValueSpecification	A given Value Specification.
Return	-	java.lang.String	A String value for ValueSpecification.

$report.filterDiagram(diagramList, diagramTypes)

Return a collection of diagrams from a list of diagrams filtered by types.

	Name	Type	Description
Parameter(s)	diagramList	java.util.Collection	A collection of diagrams.
	diagramTypes	java.util.Collection	A collection of diagram types used as filters.
Return	-	java.util.Collection	A collection of filtered diagrams.

For example:

Sample of $report.filterDiagram(diagramList, diagramTypes)

```
#foreach($diagram in $report.filterDiagram($Diagram,["Class Diagram","Communication
Diagram"]))
$diagram.name : $diagram.diagramType
#end
```

$report.filterElement(elementList, humanTypes)

Return a collection of elements from a list of elements filtered by human types.

	Name	Type	Description
Parameter(s)	elementList	java.util.Collection	A collection of elements.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

	Name	Type	Description
	humanTypes	java.util.Collection	A collection of human types used as filters.
Return	-	java.util.Collection	A collection of filtered elements.

> **Note**
> You can change the human type depending on the language used in MagicDraw. For example, "Class" is human type of the classes in English MagicDraw while "Classe" is humanu type for the classes in French MagicDraw. The specified humanTypes needs to be changed when you change the language used in MagicDraw.

For example:

$report.filterElement(elementList, humanTypes)

```
#foreach($element in $report.filterElement($elements, ["Use Case", "Actor"]))
$element.name : $element.humanType
#end
```

$report.filterElementType(elementList, elementType)

Return a collection of elements from a list of elements filtered by element types.

	Name	Type	Description
Parameter(s)	elementList	java.util.Collection	A collection of elements.
	elementTypes	java.util.Collection	A collection of element types used as filters.
Return	-	java.util.Collection	A collection of filtered elements.

$report.filter(elementList, propertyName, propertyValue)

Return a collection of elements filtered by a specified property name.

	Name	Type	Description
Parameter(s)	elementList	java.util.Collection	A collection of elements.

	Name	Type	Description
	propertyName	java.lang.String	A property name.
	propertyValue	java.util.Collection	A collection of property names used as filters.
Return		java.util.Collection	A collection of filtered elements.

For example:

$report.filter(elementList, propertyName, propertyValue)

```
#foreach ($e in $report.filter($elements, "name", ["foo", "bar"]))
   $e.name
#end
```

$report.findElementInCollection(elementList, name)

Find an element from a collection of elements by name.

	Name	Type	Description
Parameter(s)	elementList	java.util.Collection	A collection of elements.
	name	java.lang.String	An element name.
Return	-	com.nomagic.uml2.ext.magicdraw .classes.mdkernel.Element	An element instance. If it cannot find the element, it will return a null value.

$report.findRelationship(modelPackage)

Search and return a collection of relationship elements inside a package.

	Name	Type	Description
Parameter(s)	modelPackage	com.nomagic.uml2.ext.magicdraw .classes.mdkernel.Package	A package element.
Return	-	java.util.Collection	A collection of relationships in a package.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

$report.findRelationship(modelPackage, recursive)

Search and return a collection of relationship elements inside a package.

	Name	Type	Description
Parameter(s)	modelPackage	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Package	A package element.
	recursive	boolean	If true, performs recursively.
Return	-	java.util.Collection	A collection of relationships in a package.

$report.getAppliedStereotypeByName(element, stereotypeName)

Return a stereotype assigned to an element.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.
	stereotypeName	java.lang.String	A stereotype name.
Return	-	com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype	An assigned stereotype with a specified name.

$report.getBaseClassAssociations(classifier)

Obtain a collection of associations of a classifier.

	Name	Type	Description
Parameter(s)	classifier	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Classifier	A classifier.
Return	-	java.util.Collection	A collection of associations.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

$report.getBaseClassInheritableAttributes(classifier)

Get a collection of inheritable attributes of a classifier.

	Name	Type	Description
Parameter(s)	classifier	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Classifier	A classifier.
Return	-	java.util.Collection	A collection of attributes.

$report.getBaseClassInheritableOperations(classifier)

Obtain a collection of inheritable operations of a classifier.

	Name	Type	Description
Parameter(s)	classifier	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Classifier	A classifier.
Return	-	java.util.Collection	A collection of operations.

$report.getBaseClassPorts(classifier)

Find a collection of ports of a classifier.

	Name	Type	Description
Parameter(s)	classifier	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Classifier	A classifier.
Return	-	java.util.Collection	Collection of ports.

$report.getBaseRealizedInterfaces(behavioredClassifier)

Find a collection of realized interfaces of a behaviored classifier.

	Name	Type	Description
Parameter(s)	behavioredClassifier	com.nomagic.uml2.ext.magicdraw.commonbehaviors.mdbasicbehaviors.BehavioredClassifier	A BehavioredClassifier.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

	Name	Type	Description
Return	-	java.util.Collection	A collection of RealizedInterfaces.

$report.getBaseRelations(classifier)

Find a collection of relations of a classifier.

	Name	Type	Description
Parameter(s)	classifier	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Classifier	A classifier.
Return	-	java.util.Collection	A collection of relations.

$report.getBaseClassifiers(child)

Return a collection of base elements of a classifier.

	Name	Type	Description
Parameter(s)	child	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Classifier	A child class.
Return	-	java.util.Collection	A collection of base elements (classifiers).

$report.getClientElement(element)

Return a client of a relationship.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	A relationship model element.
Return	-	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	The relationship's client.

$report.getComment(element)

Return a documentation of a given element.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.
Return	-	java.lang.String	The documentation of an element.

$report.getCustomizationProperty(element, stereotypeName, propertyName)

Get a customization property from the specified applied stereotype of the element.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.
	stereotypeName	java.lang.String	A stereotype name.
	propertyName	java.lang.String	A property name.
Return	-	java.lang.Object	A property value.

$report.getCustomizationPropertyString(element, stereotypeName, propertyName)

Get a customization property as a String value from the specified applied stereotype of the element.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.
	stereotypeName	java.lang.String	A stereotype name.
	propertyName	java.lang.String	A property name.
Return	-	java.lang.String	A property value as String.

$report.getDerivedClassifiers(parent)

Return a collection of derived elements of a classifier.

	Name	Type	Description
Parameter(s)	parent	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Classifier	A parent class.
Return	-	java.util.Collection	A collection of derived elements (classifiers).

$report.getDiagramElements(diagram)

Get a collection of elements from a diagram.

	Name	Type	Description
Parameter(s)	diagram	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram	A diagram instance.
Return	-	java.util.Collection	A collection of elements in a diagram.

$report.getDiagramType(diagram)

Return a diagram type.

	Name	Type	Description
Parameter(s)	diagram	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram	A diagram instance.
Return	-	java.lang.String	A diagram type.

$report.getElementProperties(element)

Return a list of property names, including UML properties and properties that are customized through DSL customization.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.
Return	-	java.util.Collection	A collection of property names.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

$report.getDSLProperty(element, propertyName)

Return a DSL Property.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.
	propertyName	java.lang.String	A property name.
Return	-	java.lang.Object	A property value.

$report.getElementComment(element)

Return a comment attached to an element. If no comment is attached, it will return a null value.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.
Return	-	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Comment	A comment instance of the element.

$report.getElementName(element)

Get an element's name. If the name is empty, it will return "<unnamed>". This method performs the following procedures:

- If the element is generalized from the NamedElement metaclass, it will return *$element.name*
- If the element is generalized from the Slot metaclass, it will return *$element.definingFeature.name*
- If the element is a UML element, it will return *$element.humanName*
- Else returns *$element.toString()*

	Name	Type	Description
Parameter(s)	element	java.lang.Object	An element.
Return	-	java.lang.String	An element name.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

$report.getIconFor(element)

Return an image icon of an element with a default icon filename (“icon_”+element type).

	Name	Type	Description
Parameter(s)	element	com.nomagic.magicdraw.uml.BaseElement	A MagicDraw element.
Return	-	com.nomagic.magicreport.Image	An image object of the element's icon.

For example:

$report.getIconFor(element)

```
$report.getIconFor($classObject)
```

This results in an Image object of a class whose name is “icon_class”.

$report.getIconFor(element, prefix, suffix, hashCode)

Return an image icon of an element. The icon name depends on the ***hashCode*** value if ***hashCode*** equals true, the icon’s name is set as ***”icon_”+hash*** code value of the icon. If the ***hashCode*** value equals false, the icon name is set as

	Name	Type	Description
Parameter(s)	element	com.nomagic.magicdraw.uml.BaseElement	A MagicDraw element.
	prefix	java.lang.String	A prefix of an icon name.
	suffix	java.lang.String	A suffix of an icon name.
	hashCode	boolean	If true, use hash code of an icon as the icon name and exclude a prefix or suffix from the name.
Return	-	com.nomagic.magicreport.Image	An image object of an element’s icon.

For example:

To get an icon and set a name for hash code

```
$report.getIconFor($classObject, “prefix_”, “_suffix”, true)
```

This results in an Image object of a class whose name “icon_24676200”. The number in the icon name is hash code, which is regenerated whenever MagicDraw starts.

To get an icon and set a name for prefix and suffix

```
$report.getIconFor($classObject, “prefix_”, “_suffix”, false)
```

This results in an Image object of a class whose name is “prefix_class_suffix”.

$report.getIconFor(type)

Return an image icon for an element type with a default icon’s filename ((“icon_”+element type).

	Name	Type	Description
Parameter(s)	type	java.lang.String	A type name.
Return	-	com.nomagic.magicreport.Image	An image object for the element’s icon.

For example:

$report.getIconFor(type)

```
$report.getIconFor("class")
```

This results in an Image object of a class whose name is “icon_class”.

$report.getIconFor(type, prefix, suffix, hashCode)

Return an image icon for an element type. The icon’s name depends on the ***hashCode*** value if ***hashCode*** equals true, the icon name is set as ***“icon_”+hash*** code value of the icon. If the ***hashCode*** value equals false, the icon name is set as ***prefix + element type + suffix***.

	Name	Type	Description
Parameter(s)	type	java.lang.String	A type name.
	prefix	java.lang.String	A prefix for an icon name.
	suffix	java.lang.String	A suffix for an icon name.
	hashCode	boolean	If true, use hash code of an icon as the icon's name and exclude a prefix or suffix from the name.
Return	-	com.nomagic.magicreport.Image	An image object for an element's icon.

For example:

To get an icon and set a name for hash code

```
$report.getIconFor("class", "prefix_", "_suffix", true)
```

This results in an Image object of a class whose name "icon_24676200". The number in the icon name is hash code, which is regenerated whenever MagicDraw starts.

To get an icon and set a name for prefix and suffix

```
$report.getIconFor("class", "prefix_", "_suffix", false)
```

This results in an Image object of a class whose name is "prefix_class_suffix".

$report.getIncludeUseCase(useCase)

Return a collection of included elements of a UseCase.

	Name	Type	Description
Parameter(s)	useCase	com.nomagic.uml2.ext.magicdraw.mdusecases.UseCase	A UseCase instance.
Return	-	java.util.Collection	A collection of included UseCases.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

$report.getInnerElement(element)

Return a collection of inner elements.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.
Return	-	java.util.Collection	A collection of inner elements.

$report.getInteractionMessageType(message)

Return an interaction message type.

	Name	Type	Description
Parameter(s)	message	com.nomagic.uml2.ext.magicdraw.interactions.mdbasicinteractions.Message	A message instance.
Return	-	java.lang.String	A message type.

$report.getMetaClass(stereotype)

Return a stereotype meta class.

	Name	Type	Description
Parameter(s)	stereotype	com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype	A stereotype instance.
Return	-	java.util.Collection	A collection of Meta Classes.

$report.getPresentationDiagramElements(diagram)

Return presentation elements in a diagram. This method will return all presentation elements from the diagram, except non-manipulator symbols.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

	Name	Type	Description
Parameter(s)	diagram	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram	A diagram instance.
Return	-	java.util.Collection	A collection of elements.

$report.getPresentationDiagramElements(diagram, includeNonManipulator)

Return presentation elements in a diagram.

	Name	Type	Description
Parameter(s)	diagram	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram	A diagram.
	include NonManipulator	boolean	True to include all non-manipulator elements.
Return	-	java.util.Collection	A collection of elements.

$report.getPresentationElementBounds(diagram, element)

Return the polygonal bounds of an element. The bounds specify the component coordinates to its diagram.

	Name	Type	Description
Parameter(s)	diagram	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram	A target diagram.
	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	A presentation element on a given diagram.
Return	-	java.util.List	Returns the polygonal bounds of an element. If the element is not found in the diagram, it will return null.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

$report.getPresentationElementBounds(element)

Return the polygonal bounds of an element. The bounds specify this component's coordinate to its diagram.

	Name	Type	Description
Parameter(s)	element	com.nomagic.magicdraw.uml.symbols.PresentationElement	A presentation element on a diagram.
Return	-	com.nomagic.magicdraw.magicreport.helper.Polygon	Returns the polygonal bounds of an element.

$report.getPresentationElementRectangle(diagram, element)

Return the rectangular bounds of an element. The bounds specify the component coordinates to its diagram.

	Name	Type	Description
Parameter(s)	diagram	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram	A target diagram.
	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	A presentation element on a given diagram.
Return	-	java.util.List	Returns the rectangular bounds of an element. If the element is not found in the diagram, it will return null.

$report.getQualifiedName(namedElement, separator)

Get a qualified name. A qualified name is the name that allows a ***NamedElement*** to be identified within a hierarchy of nested ***Namespaces***. It is constructed from the names of the containing namespaces starting at the root of the hierarchy and ending with the name of the ***NamedElement*** itself.

	Name	Type	Description
Parameter(s)	namedElement	NamedElement	A NamedElement.
	separator	java.lang.String	A separator symbol. If the value is null or an empty string, the '::' will be used.
Return	-	java.lang.String	A qualified name.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

$report.getPackageQualifiedName(namedElement, separator)

Get a qualified name by considering only **Packages** and a given element. **Models** and **Profiles** will not be included in the qualified name.

For example, given the element hierarchy:

Sample of element hierarchy

```
Design : Model -> com : Package -> nomagic : Package -> ui -> Package -> BaseDialog :
Class
```

Sample of a template code of the element hierarchy

```
$report.getPackageQualifiedName($class, ".")
```

If $class is a “BaseDialog” element, the result from the above template code will be as follows.

The result of the template code

```
com.nomagic.ui.BaseDialog
```

	Name	Type	Description
Parameter(s)	namedElement	NamedElement	A NamedElement.
	separator	java.lang.String	A separator symbol. If the value is null or an empty string, the '::' will be used.
Return	-	java.lang.String	A qualified name.

$report.getReceivingOperationalNode(element)

Get the needline association ends.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

	Name	Type	Description
Return	-	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	A needline association instance.

$report.getRelationship(element)

Return an element's relationship.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.
Return	-	java.util.Collection	A collection of relationships.

$report.getRelationship(element, recursive)

Return a collection of element relationships.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.element	An element.
	recursive	boolean	If true, performs recursively.
Return	-	java.util.Collection	A collection of relationships.

$report.getRelativeActor(element)

Return an element's relative actor.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.
Return	-	java.util.Collection	A collection of actors.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

$report.getSendingOperationalNode(element)

Return the needline's association ends.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.
Return	-	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	A needline association instance.

$report.getSlotValue(element, ClassifierName, DefiningFeatureName)

Find and retrieve the values of a slot whose name and classifier's name are ***DefiningFeatureName*** and ***ClassifierName*** respectively. The code enables case-sensitivity queries such that only those case-sensitive matches will return.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element whose slot value you want to find.
	ClassifierName	java.lang.String	A classifier name.
	DefiningFeature Name	java.lang.String	A slot name.
Return	-	java.util.Collection	A collection of ValueSpecifications of a slot.

For example:

Use the code to retrieve the value of a slot whose name and classifier's name are Name and Personal Data respectively.

```
#foreach($instance in $InstanceSpecification)
Slot value is $report.getSlotValue($instance, "Personal Data", "Name")
#end
```

- "Personal Data" is the name of the classifier.
- Name" is the name of the slot.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

$report.getStereotypeProperty(element, stereotypeName, propertyName)

Get a stereotype property.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.
	stereotypeName	java.lang.String	A stereotype name.
	propertyName	java.lang.String	A property name.
Return	-	java.lang.Object	A property value.

Warning: Duplicate stereotype name
stereotypeName is String. Please be aware of duplicate name of stereotype that may cause to random returned result. $report.getStereotypeProperty(element, profileName, stereotypeName, propertyName) may be an option.

$report.getStereotypeProperty(element, profileName, stereotypeName, propertyName)

Get a stereotype property.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element instance.
	profileName	java.lang.String	A profile name.
	stereotypeName	java.lang.String	A stereotype name.
	propertyName	java.lang.String	A property name.
Return	-	java.lang.Object	A property value.

$report.getStereotypePropertyString(element, stereotypeName, propertyName)

Return a stereotype property as a String value.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element instance.
	stereotypeName	java.lang.String	A stereotype name.
	propertyName	java.lang.String	A property name.
Return		java.lang.String	A property value as String.

Warning: Duplicate stereotype name
stereotypeName is String. Please be aware of duplicate name of stereotype that may cause to random returned result. $report.getStereotypePropertyString(element, profileName, stereotypeName, propertyName) may be an option.

$report.getStereotypePropertyString(element, profileName, stereotypeName, propertyName)

Get a stereotype property as a String value.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element instance.
	profileName	java.lang.String	A profile name.
	stereotypeName	java.lang.String	A stereotype name.
	propertyName	java.lang.String	A property name.
Return	-	java.lang.String	A property value as String.

$report.getStereotypes(element)

Return all stereotypes applied to an element. This method is replaced by ***$element.appliedStereotype***.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

	Name	Type	Description
Return	-	java.util.List	A list of stereotype instances.

$report.getSupplierElement(element)

Return the supplier of a relationship.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	A relationship model element.
Return	-	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	The relationship's supplier.

$report.getUsageElements(usagesMap, element)

Return the usage of a specified element.

	Name	Type	Description
Parameter(s)	usagesMap	java.util.Map	The Usage Map of MagicDraw.
	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.
Return	-	java.util.Collection	A collection of element usages.

$report.getUsages(selectedObjects)

Return the Usage Map of MagicDraw. It uses the getUsageElements method to return the usage of a specified element.

	Name	Type	Description
Parameter(s)	selectedObjects	java.lang.Object	An element.
Return	-	java.util.Map	A Map instance of element usages.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

$report.hasStereotype(element)

Check if an element has stereotypes.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element to check.
Return	-	boolean	True if it has a stereotype.

$report.containsStereotype(element, stereotypeName)

Return true if an element contains a stereotype (including all derived stereotypes) for a specified stereotype name.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element to check.
	stereotypeName	java.lang.String	A stereotype name to be tested.
Return	-	boolean	True if the element contains a stereotype for the specified stereotype name.

$report.containsStereotype(element, stereotypeName, includeDerived)

Return true if an element contains a stereotype for a specified stereotype name.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element to test.
	stereotypeName	java.lang.String	A stereotype name to be tested.
	includeDerived	boolean	True if the searched target includes all derived stereotypes; otherwise, false.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

	Name	Type	Description
Return	-	boolean	True if the element contains a stereotype for the specified stereotype name.

$report.isDerivedClassifier(parent, child)

Check if a child is derived from the parent by generalization.

	Name	Type	Description
Parameter(s)	parent	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Classifier	A parent.
	child	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Classifier	A possible child.
Return	-	boolean	True if it is derived from the parent by generalization.

$report.isNamedElement(element)

Return whether an element is a NamedElement.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element to test.
Return	-	boolean	True if the given element is a NamedElement; otherwise, false

$report.isNull(obj)

Test and return true if an object is null.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

	Name	Type	Description
Parameter(s)	obj	java.lang.Object	An object being tested.
Return	-	boolean	True if the object is null.

$report.isRelationship(element)

Test and return true if an element is a relationship.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element being tested.
Return	-	boolean	True if the object is a relationship.

$report.serialize(hyperlink)

Convert *com.nomagic.magicdraw.hyperlinks.Hyperlink* to *com.nomagic.magicdraw.plugins.impl.magicreport.helper.Hyperlink*. Report Wizard needs the *com.nomagic.magicdraw.plugins.impl.magicreport.helper.Hyperlink* class wrapper to return *com.nomagic.magicdraw.hyperlinks.Hyperlink* data.

	Name	Type	Description
Parameter(s)	hyperlink	com.nomagic.magicdraw.hyperlinks.Hyperlink	A MagicDraw hyperlink.
Return	-	com.nomagic.magicdraw.plugins.impl.magicreport.helper.Hyperlink	A Report Wizard hyperlink instance.

$report.getUsedBy(element)

Return a list of element(s) used by this element (except diagrams).

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element of which to determine the usage.
Return		java.util.Collection	A collection of elements used by an input element.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

$report.hasProperty(element, propertyName)

Return true when a property with a given name is specified in this element, otherwise returns false.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.
	propertyName	java.lang.String	A property name.
Return	-	java.lang.Boolean	Return true when a property with a given name is specified in this element, otherwise returns false.

For example:

```
#foreach($element in $elements)
$report.hasProperty($element, "name")
#end
```

- $element is the element.
- data is the name of the property.

$report.getProperty(element, propertyName)

Get element property.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.
	propertyName	java.lang.String	A property name.
Return	-	java.lang.Object	A property value.

For example:

Sample of $report.getProperty(element, propertyName)

```
#foreach($element in $elements)
$report.getProperty($element, "name")
#end
```

- $element is the element.
- name is the name of the property.

$report.findElementByName(source, regex)

Search and return elements that match the name using a regular expression.

	Name	Type	Description
Parameter(s)	source	java.util.Collection	A collection of elements.
	regex	java.lang.String	A regular expression with which the name is to be matched.
Return	-	java.util.Collection	A collection of matching elements.

For example:

$report.findElementByName(source, regex)

```
#foreach($ele in $report.findElementByName($elements, "[A]+.*"))
$ele.name
#end
```

- $elements is a collection of elements to be found.
- "[A]+.*" is a regular expression to find which element matches the name. In this example, the following are the names that match, such as Auxiliary, AppServer, and Alternative Fragment.

$report.getPresentationElements(diagram)

Get a presentation element in a diagram. This method is equivalent to $report.getPresentationDiagramElements(diagram).

	Name	Type	Description
Parameter(s)	diagram	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram	A diagram instance.

	Name	Type	Description
Return	-	java.util.Collection	A collection of elements.

For example:

$report.getPresentationElements(diagram)

```
#foreach($diagram in $Diagram)
$foreach($d in $report.getPresentationElements($diagram))
$d.name
#end
#end
```

- **$diagram** is the diagram instance.

$report.getUsageRepresentationText(baseElement, bool)

Format the usage subject. The output string is the same as the Result column of Used by table in Magic Draw.

	Name	Type	Description
Parameter(s)	baseElement	com.nomagic.magicdraw.uml.BaseElement	A model element to be formatted.
	bool	java.lang.Boolean	True if a full path is used, otherwise false.
Return	-	java.lang.String	A formatted element.

For example:

$report.getUsageRepresentationText(baseElement, bool)

```
#foreach($baseElement in $Elements)
$report.getUsageRepresentationText($baseElement, false)
#end
```

- $baseElement is the model element to be formatted.
- false if a full path is not used. In this example, a full path is not used.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

$report.getUseCaseNumber(element)

Return a use case number of an element.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	A model element.
Return	-	java.lang.String	A use case number.

For example:

Sample of $report.getUseCaseNumber(element)

```
#foreach ($uc in $RequirementsUseCase)
$report.getUseCaseNumber($uc) $uc.name
#end
```

$report.getElementURL(element)

Return a MagicDraw's element URL.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.
Return	elementURL	java.lang.String	An element URL.

For example:

Sample of $report.getElementURL(element)

```
#foreach ($element in $elements)
$report.getElementURL($element)
#end
```

$report.isEmpty(obj)

Test and return true if an object is null, empty string, or empty collection.

	Name	Type	Description
Parameter(s)	obj	java.lang.Object	An object being tested.
Return	-	boolean	True if the object is null, empty string, or empty collection.

For example:

Sample of $report.isEmpty(obj)

```
#if ($report.isEmpty($var))
Object is null, empty string or empty collection
#end
```

$report.getBasicFlows(usecase:UseCase) : List<FlowStep>

Find and return a list of basic flows from a given use case. If a use case scenario is not being used, it will retrieve the value from "requirementUseCase" stereotype tags.

	Name	Type	Description
Parameter(s)	usecase	com.nomagic.uml2.ext.magicdraw.mdusecases.UseCase	A targeted Use Case.
Return	-	List<com.nomagic.magicdraw.usecasescenarios.scenarios.FlowStep>	A list of flow steps. A Flow Step is an object representing each step of a Use Case flow.

The <FlowStep> is a MagicDraw "com.nomagic.magicdraw.usecasescenarios.scenarios.FlowStep" class.

The important properties of <FlowStep> are as follows:

- name prints a name of a current step, for example: $flowStep.name.
- element returns an element associates with a current step, for example: #set ($e = $flowStep.element).
- alternativeConditions returns a list of alternative steps <AlternativeCondition> from a current step, for example: #foreach ($alter in $flowStep.alternativeConditions) #end.
- exceptionTypes returns a list of exception steps <ExceptionType> from a current step, for example: #foreach ($alter in $flowStep.exceptionTypes) #end.

For example:

Sample of $report.getBasicFlows(usecase:UseCase) : List

```
#set ($basicFlowList = $report.getBasicFlows($useCase))
#foreach ($flowStep in $basicFlowList)
   $flowStep.name
   #foreach ($alter in $flowStep.alternativeConditions)
      - $alter.name
   #end
#end
```

$report.getAlternativeFlows(usecase:UseCase) : List<FlowStep>

Find and return all alternative flows from a given use case. The alternative flows will be listed from all possible branches of basic flows. If a use case scenario is not being used, it will retrieve the value from "requirementUseCase" stereotype tags.

	Name	Type	Description
Parameter(s)	usecase	com.nomagic.uml2.ext.magicdraw.mdusecases.UseCase	A targeted Use Case.
Return	-	List<com.nomagic.magicdraw.usecasescenarios.scenarios.FlowStep>	A list of flow steps. A Flow Step is an object representing each step of a Use Case flow.

For example:

Sample of $report.getAlternativeFlows(usecase:UseCase) : List

```
#set ($alternativeFlowList = $report.getAlternativeFlows($useCase))
$useCase.name
 #foreach ($flowStep in $alternativeFlowList)
   $flowStep.name
   #foreach ($alternative in $flowStep.alternativeConditions)
      $alternative.name
         #foreach ($alternativeFlow in $alternative.alternativeFlowSteps)
         $alternativeFlow.name
         #end
   #end
 #end
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

$report.getExceptionalFlows(usecase:UseCase) : List<FlowStep>

Find and return all exceptional flows from a given use case. The alternative flows will be listed from all possible branches of basic flows. If the use case scenario is not being used, it will retrieve the value from "requirementUseCase" stereotype tags.

	Name	Type	Description
Parameter(s)	usecase	com.nomagic.uml2.ext.magicdraw.mdusecases.UseCase	A targeted Use Case.
Return	-	List<com.nomagic.magicdraw.usecasescenarios.scenarios.FlowStep>	A list of flow steps. A flow step is an object representing each steps of a Use Case flow.

For example:

Sample of $report.getExceptionalFlows(usecase:UseCase) : List

```
#set ($exceptionFlowList = $report.getExceptionalFlows($useCase))
     #foreach ($flowStep in $exceptionFlowList)
         $flowStep.name
         #foreach (#exception in $flowStep.exceptionTypes)
             $exception.name
             #foreach ($exceptionalFlow in $exception.exceptionalFlowSteps)
                  $exceptionalFlow.name
             #end
         #end
     #end
```

$report.getOwnedElementsIncludingAdditional(element, includePureOwned) : List<Element>

This code returns owned elements of a given element including additional owned elements defined in the DSL specification by additionalContentProperty.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	A model element.
	includePureOwned	java.lang.Boolean	True if includes an element owned directly in the ownedElement UML metaproperty. Otherwise False.

	Name	Type	Description
Return	-	java.util.List	A list of owned elements of a given element.

The following are the examples of using the code.

Sample 1: $report.getOwnedElementsIncludingAdditional(element, includePureOwned) : List

```
#foreach($e in $elements)
$e.name : $e.humanType
  #foreach($child in $report.getOwnedElementsIncludingAdditional($e, true))
    $child.name : $child.humanType
  #end
#end
```

Sample 2: $report.getOwnedElementsIncludingAdditional(element, includePureOwned) : List

```
#foreach($e in $SmartPackage)
$e.name : $e.humanType
    #foreach($child in $report.getOwnedElementsIncludingAdditional($e, true))
       $child.name : $child.humanType
    #end
#end
```

Sample 3: $report.getOwnedElementsIncludingAdditional(element, includePureOwned) : List

```
#set($packages = $array.createArray())
#set($void = $array.addCollection($packages, $Package))
#set($void = $array.addCollection($packages, $SmartPackage))
#foreach($e in $packages)
$e.name : $e.humanType
    #foreach($child in $report.getOwnedElementsIncludingAdditional($e, true))
        $child.name : $child.humanType
    #end
#end
```

Project tool

On this page

- $project.getName()(see page 1655)
- $project.getTitle()(see page 1655)
- $project.getFileName()(see page 1655)

- $project.getExtension()(see page 1656)
- $project.getDirectory()(see page 1656)
- $project.getVersionList()(see page 1656)
- $project.getDiagrams()(see page 1657)
- $project.getDiagrams(type)(see page 1657)
- $project.getPresentationDiagrams()(see page 1657)
- $project.getPresentationDiagrams(type)(see page 1657)
- $project.isRemote()(see page 1658)
- $project.isDirty()(see page 1658)
- $project.getElementByID(id)(see page 1658)
- $project.getAllElementId()(see page 1659)
- $project.getXmiVersion()(see page 1659)
- $project.getVersion()(see page 1659)
- $project.getModel()(see page 1659)
- $project.getPrimaryModel()(see page 1660)
- $project.getAllModels()(see page 1660)
- $project.getModuleList()(see page 1660)
- $project.getModuleList(includeAuxiliary)(see page 1660)
- $project.getSharedModule()(see page 1661)
- $project.getSharedModule(projectModule)(see page 1661)
- $project.getServerModule()(see page 1661)
- $project.getServerModule(includeAuxiliary)(see page 1661)
- $project.getLocalModule()(see page 1662)
- $project.getLocalModule(includeAuxiliary)(see page 1662)
- $project.getAuxiliaryResource()(see page 1662)
- Project Version(see page 1662)
- Project Module(see page 1663)

$project is a project reference enabling a template to return the project information.

$project.getName()

Returns a project name.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	java.lang.String	A project name.

$project.getTitle()

Gets a project title.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	java.lang.String	A project title.

$project.getFileName()

Gets a project filename.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	java.lang.String	A project filename.

$project.getExtension()

Gets a project filename extension.

> **Note**
> This function is only for local projects.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	java.lang.String	A project filename extension.

$project.getDirectory()

Get a project directory name.

> **Note**
> This function is only for local projects.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	java.lang.String	A project directory name.

$project.getVersionList()

Returns a list of version information from an open Server project.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	java.util.List	A list of com.nomagic.magicdraw.magicreport.tools.ProjectVersion(see page 1662).

$project.getDiagrams()

Returns all existing diagrams stored in a particular project.

	Name	Type	Description
Return	-	java.util.Collection	A collection of diagram instances.

$project.getDiagrams(type)

Returns existing diagrams of a given type stored in a particular project.

	Name	Type	Description
Parameter(s)	type	java.lang.String	A diagram type.
Return	-	java.util.Collection	A collection of diagram instances.

Sample code

```
#set($classDiagram = $project.getDiagrams("Class Diagram"))
#foreach($cl in $classDiagram)
$cl.name
#end
```

- “Class Diagram” is a diagram type.

$project.getPresentationDiagrams()

Returns all existing presentation diagrams stored in a particular project.

	Name	Type	Description
Return	-	java.util.Collection	A collection of diagram views.

$project.getPresentationDiagrams(type)

Returns all existing presentation diagrams of a given type stored in a particular project.

	Name	Type	Description
Parameter(s)	type	java.lang.String	A diagram type.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

	Name	Type	Description
Return	-	java.util.Collection	A collection of diagram views.

Sample code

```
#set($classDiagram = $project.getPresentationDiagrams("Class Diagram"))
#foreach($cl in $classDiagram)
$cl.name
#end
```

- “Class Diagram” is a diagram type.

$project.isRemote()

Returns the remote or non-remote state of a project.

	Name	Type	Description
Return	-	java.lang.Boolean	Return true if a project is a remote project, otherwise false.

$project.isDirty()

Returns true if that particular project was modified after it had been saved or loaded.

	Name	Type	Description
Return	-	java.lang.Boolean	Return true if a project was modified after it had been saved/loaded, otherwise false.

$project.getElementByID(id)

Returns an element with a given ID.

	Name	Type	Description
Parameter(s)	ID	java.lang.String	An element ID.

	Name	Type	Description
Return	-	com.nomagic.magicdraw.uml.BaseElement	An element with a given ID or null if the element with such ID is not registered in the project.

Sample code

```
#set($ele = $project.getElementByID("_9_0_62a020a_1105704887361_983947_8206"))
$ele.name
$ele.humanType
```

- "_9_0_62a020a_1105704887361_983947_8206" is the element's ID number.

$project.getAllElementId()

Returns a collection of all element IDs in a project.

	Name	Type	Description
Return	-	java.util.Collection	A collection of all element IDs in a project.

$project.getXmiVersion()

Returns the XMI version of a project.

	Name	Type	Description
Return	-	int	An XMI version.

$project.getVersion()

Return a project version number.

	Name	Type	Description
Return	-	long	A project version number.

$project.getModel()

Returns a model (the root container of all model structures).

	Name	Type	Description
Return	-	com.nomagic.uml2.ext.magicdraw.auxiliaryconstructs.mdmodels.Model	A model.

$project.getModel() is deprecated since MagicDraw 18.2, please use **$project.getPrimaryModel()** instead

$project.getPrimaryModel()

Returns the primary (main) model of the project (not including the ones from modules (attached projects)).

	Name	Type	Description
Return	-	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Package	The primary model of the project.

$project.getAllModels()

Returns a list of models in the project (the primary model of the project and all the models of the modules(attached projects)).

	Name	Type	Description
Return	-	Java.util.List	A list of models.

$project.getModuleList()

Returns a list of used server and local modules from a current project.

	Name	Type	Description
Return	-	java.util.Collection	A collection of com.nomagic.magicdraw.magicreport.tools.ProjectModule(see page 1663).

$project.getModuleList(includeAuxiliary)

Returns a list of used server and local modules as well as auxiliary resources from a current project.

	Name	Type	Description
Parameter(s)	includeAuxiliary	boolean	Returns modules including auxiliary resources if this value is true (false by default).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

	Name	Type	Description
Return	-	java.util.Collection	A collection of com.nomagic.magicdraw.magicreport.tools.ProjectModule(see page 1663).

$project.getSharedModule()

Returns a list of shared modules from a current project.

	Name	Type	Description
Return	-	java.util.Collection	A collection of com.nomagic.magicdraw.magicreport.tools.ProjectModule(see page 1663).

$project.getSharedModule(projectModule)

Returns a list of shared modules from a specified project module.

	Name	Type	Description
Parameter(s)	projectModule	com.nomagic.magicdraw.magicreport.tools.ProjectModule(see page 1663)	A project module.
Return	-	java.util.Collection	A collection of com.nomagic.magicdraw.magicreport.tools.ProjectModule(see page 1663).

$project.getServerModule()

Returns a list of used server modules.

	Name	Type	Description
Return	-	java.util.Collection	A collection of com.nomagic.magicdraw.magicreport.tools.ProjectModule(see page 1663).

$project.getServerModule(includeAuxiliary)

Returns a list of used server modules as well as auxiliary resources.

	Name	Type	Description
Parameter(s)	includeAuxiliary	boolean	Returns server modules including auxiliary resources if this value is true (false by default).

	Name	Type	Description
Return	-	java.util.Collection	A collection of com.nomagic.magicdraw.magicreport.tools.ProjectModule(see page 1663).

$project.getLocalModule()

Returns a list of used local modules.

	Name	Type	Description
Return	-	java.util.Collection	A collection of com.nomagic.magicdraw.magicreport.tools.ProjectModule(see page 1663).

$project.getLocalModule(includeAuxiliary)

Returns a list of used local modules as well as auxiliary resources.

	Name	Type	Description
Parameter(s)	includeAuxiliary	boolean	Returns local modules including auxiliary resources if this value is true (false by default).
Return	-	java.util.Collection	A collection of com.nomagic.magicdraw.magicreport.tools.ProjectModule(see page 1663).

$project.getAuxiliaryResource()

Returns a list of auxiliary resources.

	Name	Type	Description
Return	-	java.util.Collection	A collection of com.nomagic.magicdraw.magicreport.tools.ProjectModule(see page 1663).

Project Version

The full class name is com.nomagic.magicdraw.magicreport.tools.ProjectVersion.
ProjectVersion is a class that represents some version of the project or module. It contains date, user, number, etc.

ProjectVersion contains the following properties:

Property	Return type	Description
comment	java.lang.String	Returns a version committed comment.
date	java.util.Date	Returns a committed date.
dateAsString	java.lang.String	Returns a committed date as text.
number	long	Returns a committed version.
numberAsString	java.lang.String	Returns a committed version as text.
user	java.lang.String	Returns a committer's name.
version	java.lang.Object	Returns a version object that can be com.nomagic.teamwork.common.projects.Version for a Teamwork project or com.nomagic.magicdraw.esi.project.project.IEsiVersionDescriptor for a Teamwork Cloud project.
tags	java.util.List<java.lang.String>	Returns all tags.

Sample code

```
Current version : $project.version
All version:
#foreach ($version in $project.getVersionList())
    Date :              $version.date
    Number :            $version.number
    Number As String :  $version.numberAsString
    User :              $version.user
    Comment :           $version.comment
    Tags:
#foreach($tag in $version.tags)
        - tag
#end
---
#end
```

Project Module

The full class name is com.nomagic.magicdraw.magicreport.tools.ProjectModule. ProjectModule is a wrapper Class of an attached project (com.nomagic.ci.persistence.IAttachedProject) that represents a special kind of the project that is used as a component in the current project.

The following are some properties and methods of ProjectModule:

Property / Method	Return type	Description
representationString	java.lang.String	Returns user-friendly representation string about the module.
description	java.lang.String	Returns project description of the module.
version	long	Returns a version number of the module.
requiredVersion	java.lang.String	Returns a required version of the module. This property works for Teamwork Server projects. It will return -1 when the module is the latest version, and the required version is not set.
remoteID	java.lang.String	Returns a remote ID for the module.
URI	java.net.URI	Converts EMF URI to java.net.URI.
versionList	java.util.List< ProjectVersion(see page 1662)>	Returns a list of version descriptors of the module.
isRemote()	boolean	Checks if the module is remote - from Teamwork server or Teamwork Cloud.

Sample code

```
#foreach($module in $project.getModuleList())
    Name : $module.representationString
    Description : $module.description
    Version : $module.version
    RemoteID : $module.remoteID
    URI : $module.URI
#if($module.isRemote())
#set($version = $module.projectVersion)
    Current module version :
        date : $version.date
        number : $version.number
        numberAsString : $version.numberAsString
        user : $version.user
        comment : $version.comment
        tags:
#foreach($tag in $version.tags)
- $tag
#end
    All module versions :
#foreach($version in $module.versionList)
        date : $version.date
        number : $version.number
        numberAsString : $version.numberAsString
        user : $version.usercomment : $version.comment
        tags:
#foreach($tag in $version.tags)
- $tag
#end
----------
#end
#end
    Shared module:
#foreach ($child in $project.getSharedModule($module))
        - $child.representationString
#end
#end
```

Iterator tool

$iterator is used with the **#foreach** loops. It wraps a list to let you specify a condition to terminate the loop and reuse the same list in a different loop. The following example shows how to use **$iterator**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

```
#set ($list = [1, 2, 3, 5, 8, 13])
#set ($numbers = $iterator.wrap($list))
#foreach ($item in $numbers)
#if ($item < 8) $numbers.more()#end
#end
$numbers.more()
Output
------
1 2 3 5
8
```

$iterator.wrap(list)

Wrap a list with the tool.

	Name	Type	Description
Parameter(s)	list	java.util.List	An array or a list instance.
Return	-	IteratorTool	Return an IteratorTool instance.

$<IteratorTool instance>.hasMore()

Check if the iteration has more elements.

	Name	Type	Description
Return	-	boolean	Return true if there are more elements in the wrapped list.

$<IteratorTool instance>.more()

Ask for the next element in the list.

	Name	Type	Description
Return	-	java.lang.Object	An element instance.

$<IteratorTool instance>.remove()

Remove the current element from the list.

$<IteratorTool instance>.reset()

Reset the wrapper so that it will start over at the beginning of the list.

$<IteratorTool instance>.stop()

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Put a condition to break off a loop.

$<IteratorTool instance>.toString()

Return an object as a string.

List tool

$list is a list tool used to work with lists and arrays in a template. It provides a method to get and manage specified elements. You can also use it as a means to perform some actions on a list or array such as:

- Check if it is empty.
- Check if it contains certain elements.

For example:

$primes $list.size($primes) $list.get($primes, 2) $list.set($primes, 2, 1) $list.get($primes, 2)For example: $list.isEmpty($primes) $list.contains($primes, 7)	-> new int[] {2, 3, 5, 7} -> 4 -> 5 -> (primes [2] becomes 1) -> 1 -> false -> true

$list.contains(list, element)

Check if a list or array contains certain elements.

	Name	Type	Description
Parameter(s)	list	-	A list or array instance.
	element	-	An element instance.
Return	-	boolean	Return true if a list or array contains certain elements. Otherwise, return false.

$list.get(list, index)

Return a specified element of a list or array.

	Name	Type	Description
Parameter(s)	list	-	A list or array instance.

	Name	Type	Description
	index	-	An index of an object in a list.
Return	-	object	Return an object in a list.

$list.isArray(object)

Check if an object is an array.

	Name	Type	Description
Parameter(s)	object	-	An object
Return	-	boolean	Return true if the object is an array. Otherwise, returns false.

$list.isEmpty(list)

Check if a list or array is empty.

	Name	Type	Description
Parameter(s)	list	-	A list of objects
Return	-	boolean	Return true if the list or array is empty. Otherwise, returns false.

$list.isList(object)

Check if an object is a list.

	Name	Type	Description
Parameter(s)	object	-	An object
Return	-	boolean	Return true, if the object is a List. Otherwise, return false.

$list.set(list, index, value)

Set a specified element of a List/array.

	Name	Type	Description
Parameter(s)	list	-	A list of objects.
	index	-	An index of an object in the list.
	value	-	An object to be set to the list.
Return	-	object	An object at the previously-specified position with the set value.

$list.size(list)

Return the size of a List or array.

	Name	Type	Description
Parameter(s)	list	-	A list of objects.
Return	-	Integer	Return the size of the list as an Integer.

Bookmark tool

$bookmark contains utility functions for bookmarking. The functions of this module are accessible from templates through **$bookmark**.

> **Information**
> The default style of bookmarks depends on the editor used. For example, Microsoft Word 2003 displays hyperlinks in blue.

$bookmark.openURL(url, content)

Create a hyperlink to open a URL. For example:

```
$bookmark.openURL("http://www.nomagicasia.com","NoMagic(see page 1669) Asia")
```

	Name	Type	Description
Parameter(s)	url	java.lang.String	A URL text.
	content	java.lang.Object	The text content.

	Name	Type	Description
Return	-	com.nomagic.magicreport.Link	The text content with a hyperlink.

$bookmark.openURL(url, content)

Create a hyperlink to open a URL. For example:

```
$bookmark.openURL($url,"NoMagicAsia")
```

	Name	Type	Description
Parameter(s)	url	java.net.URL	A URL instance.
	content	java.lang.Object	The text content.

	Name	Type	Description
Return	-	com.nomagic.magicreport.Link	The text content with a hyperlink.

$bookmark.openURL(uri, content)

Create a hyperlink to open a URI. For example:

```
$bookmark.openURL($uri,"NoMagicAsia")
```

	Name	Type	Description
Parameter(s)	uri	java.net.URI	A URI instance.
	content	java.lang.Object	The text content.

	Name	Type	Description
Return	-	com.nomagic.magicreport.Link	The content with a hyperlink.

$bookmark.open(content)

Create a hyperlink for a bookmark. The bookmark ID will be automatically generated from the content.

	Name	Type	Description
Parameter(s)	content	java.lang.Object	The text content.
Return	-	com.nomagic.magicreport.Link	The text content with a hyperlink.

$bookmark.open(bookmarkId, content)

Create a hyperlink for a bookmark. The bookmark ID value must match the parameter bookmark ID that passes onto the link.

	Name	Type	Description
Parameter(s)	bookmarkId	java.lang.String	The bookmark ID.
	content	java.lang.Object	The text content.
Return	-	com.nomagic.magicreport.Link	The text content with a hyperlink.

$bookmark.create(bookmarkObject)

Create a bookmark. The bookmark ID will be automatically generated from the bookmark object.

	Name	Type	Description
Parameter(s)	bookmarkObject	java.lang.Object	A bookmark object or content.
Return	-	com.nomagic.magicreport.Bookmark	The content with a bookmark.

$bookmark.create(bookmarkId, bookmarkObject)

Create a bookmark. The default element type for this function is "label".

	Name	Type	Description
Parameter(s)	bookmarkId	java.lang.String	The bookmark ID.
	bookmarkObject	java.lang.Object	A bookmark object or content.

	Name	Type	Description
Return	-	com.nomagic.magicreport.Bookmark	The content with a bookmark.

$bookmark.create(bookmarkId, bookmarkObject, elementType)

Create a bookmark by specifying an element type.

	Name	Type	Description
Parameter(s)	bookmarkId	java.lang.String	The bookmark ID.
	bookmarkObject	java.lang.Object	A bookmark object or content.
	elementType	java.lang.String	An element type name, e.g., html tag.
Return	-	com.nomagic.magicreport.Bookmark	The content with a bookmark.

$bookmark.getBookmarkId(id)

Return a bookmark ID from a given string.

	Name	Type	Description
Parameter(s)	id	java.lang.String	An original string value.
Return	-	com.nomagic.magicreport.Bookmark	The bookmark ID.

$bookmark.createRef(referenceID)

Create a reference bookmark of any Microsoft Word reference numbered items.

	Name	Type	Description
Parameter(s)	referenceID	java.lang.String	A reference ID.
Return	-	com.nomagic.magicreport.TextReference	The content with a bookmark in the Microsoft Word format.

$bookmark.openRef(referenceID)

Create a hyperlink for a reference bookmark of any Microsoft Word reference numbered items.

	Name	Type	Description
Parameter(s)	referenceID	java.lang.String	A reference ID.
Return	-	com.nomagic.magicreport.TextReference	The content with a bookmark in the Microsoft Word format.

> **Note**
> You can press F9 to replace the text content shown as **Refer** with the updated numbered item that it refers to.

$bookmark.openRef(referenceID, referenceText)

Create a hyperlink for a reference bookmark of any Microsoft Word reference numbered items.

	Name	Type	Description
Parameter(s)	referenceID	java.lang.String	A reference ID.
	referenceText	java.lang.String	The text content.
Return	-	com.nomagic.magicreport.TextReference	The content with a bookmark in the Microsoft Word format.

> **Note**
> You can press F9 to replace the text content shown as **Refer referenceText** with the updated numbered item that it refers to.

Sorter tool

$sorter is used to sort a collection of report templates.

$sorter.sort(Collection, fieldName)

The sort function for report templates. The context name of this class is "sorter". Use $sorter to access public functions of this class through templates.

	Name	Type	Description
Parameter(s)	collection	java.util.Collection	A collection to be sorted

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

	Name	Type	Description
	fieldname	java.lang.String	A fieldName to be sorted and the sort direction.
Return	-	java.util.Collection	A sorted collection.

For example:

```
#foreach ($rel in $sorter.sort($package, "name"))
$rel.name
#end
```

- $package is the collection to be sorted.
- "name:desc" is separated by ":" in two parts:
  - The first part is to identify fieldName to be sorted.
  - The second part is the option to identify the sorting direction. Sometimes, the direction is not identified. It is ascending by default.

$sorter.sort(Collection)

This is the sort function for report templates. The context name of this class is "sorter". Use $sorter to access public functions of this class through the templates.

	Name	Type	Description
Parameter(s)	collection	java.util.Collection	A collection to be sorted.
Return	-	java.util.Collection	A sorted collection.

For example:

```
#foreach ($rel in $sorter.sort($package))
$rel.name
#end
```

- $package is a collection to be sorted.

The collection will be sorted by name, in ascending order, by default.

$sorter.sortByFirstNumber(Collection, fieldName)

The sortByFirstNumber function is for report templates. The context name of this class is "sorter". Use $sorter to access public functions of this class through templates.

	Name	Type	Description
Parameter(s)	collection	java.util.Collection	A collection to be sorted.
	fieldName	java.lang.String	A fieldName to be sorted and the sorting direction.
Return	-	java.util.Collection	A sorted collection.

For example:

```
#foreach ($rel in $sorter.sortByFirstNumber($package, "name:desc"))
$rel.name
#end
```

- $package is a collection to be sorted by **FirstNumber**.
- "name:desc" is separated by ":" in two parts:
  - The first part is to identify fieldName to be sorted.
  - The second part is the option to identify the sorting direction. Sometimes, the direction is not identified. It is ascending by default.

$sorter.sortByFirstNumber(Collection)

The sortByFirstNumber function for report templates. The context name of this class is "sorter". Use $sorter to access public functions of this class through templates.

	Name	Type	Description
Parameter(s)	collection	java.util.Collection	A collection to be sorted.
Return	-	java.util.Collection	A sorted collection.

For example:

```
#foreach ($rel in $sorter.sortByFirstNumber($package))
$rel.name
#end
```

- $package is a collection to be sorted by FirstNumber.

$sorter.sortByLocale(Collection, String)

This is a function to sort for report templates. The context name of this class is "sorter". Use $sorter to access public functions of this class through templates. To sort a given collection by a particular language, identify the country code to specify the language.

	Name	Type	Description
Parameter(s)	collection	java.util.Collection	A collection to be sorted.
	countryCode	java.lang.String	The country code to specify a language to perform sorting.
Return	-	java.util.Collection	A sorted collection.

For example:

```
#foreach ($p in $sorter.sortByLocale($package, "DE"))
$p.name
#end
```

- $package is a collection to be sorted by country code.
- "DE" is the country code for GERMANY (ISO country code).

This method sorts a collection by “name” attribute of each element by default.

$sorter.sortByLocale(Collection, String, String)

This is a function to sort for report templates. The context name of this class is "sorter". Use $sorter to access public functions of this class through templates. To sort a given collection by a particular language, specify the language by identifying the country code and field name.

	Name	Type	Description
Parameter(s)	collection	java.util.Collection	A collection to be sorted.
	fieldName	java.lang.String	A fieldName to b 0e sorted.
	countryCode	java.lang.String	The country code to specify a language to perform sorting.
Return	-	java.util.Collection	A sorted collection.

For example:

```
#foreach ($p in $sorter.sortByLocale($package, "name", "DE"))
$p.name
#end
```

- $package is the collection to be sorted.
- “name” is the field name to be sorted.
- "DE" is the country code for GERMANY (ISO country code).

$sorter.humanSort(collection, fieldName)

This is a special function to sort text in a human-like order. It splits text into numeric and non-numeric chunks and sorts them in numerical order. For example, "foo10" is ordered after "foo2".

	Name	Type	Description
Parameter(s)	collection	java.util.Collection	A collection to be sorted.
	fieldName	java.lang.String	A fieldName to be sorted and the sorting direction.
Return	-	java.util.Collection	A sorted collection.

For example:

```
#foreach ($rel in $sorter.humanSort($package, "name:desc"))
$rel.name
#end
```

- $package is the collection to be sorted.
- "name:desc" is separated by ":" in two parts:
  - The first part is to identify fieldName to be sorted.
  - The second part is the option to identify the sorting scheme. Sometimes, the order is not identified. It is ascending by default.

$sorter.humanSort(collection)

This is a special function to sort text in a human-like order. It splits text into numeric and non-numeric chunks and sorts them in numerical order. For example, "foo10" is ordered after "foo2".

	Name	Type	Description
Parameter(s)	Collection	java.util.Collection	A collection to be sorted.
Return	-	java.util.Collection	A sorted collection.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

For example:

```
#foreach ($rel in $sorter.humanSort($package))
$rel.name
#end
```

- $package is the collection to be sorted.

$sorter.containmentTreeSort(collection)

This function is for sorting elements that match the Containment tree order.

	Name	Type	Description
Parameter(s)	Collection	java.util.Collection	A collection to be sorted.
Return	-	java.util.Collection	A sorted collection.

Template tool

$template is the tool used for getting a template's information.

$template.getName()

Return the name of a template.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	java.lang.String	Return a template name.

$template.getResourcesLocation()

Return the folder location of the resource file.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	java.lang.String	The location of a resource file folder.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

$template.getTemplateFile()

Return a template filename.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	java.lang.String	Return a template filename.

$template.getTemplateLocation()

In case, there is an included template, the latest included template location would be returned, otherwise, the main location of a template file will be returned.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	java.lang.String	The location of the latest included template file folder.

$template.getMainTemplateLocation()

Return the folder location of the main template file.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	java.lang.String	The location of a main template file folder.

$template.getOutputFile()

Return the output filename.

	Name	Type	Description
Parameter(s)	-	-	-

	Name	Type	Description
Return	-	java.lang.String	Return an output filename.

$template.getOutputFileNoExt()

Return the output name.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	java.lang.String	An output name without a filename extension.

$template.getOutputLocation()

Return the folder location of the output file.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	java.lang.String	Return the location of an output file folder.

File tool

On this page

- $file.silentCreate(templateFilename)(see page 1682)
- $file.silentCreate(attachedFile)(see page 1682)
- $file.silentCreate(templateFilename, importObject)(see page 1683)
- $file.silentCreate(attachedFile, importObject)(see page 1683)
- $file.silentCreate(templateFilename, outputFileName, importObject)(see page 1684)
- $file.silentCreate(attachedFile, outputFileName, importObject)(see page 1684)
- $file.silentCreate(templateType, templateFilename, outputname, importObject)(see page 1685)
- $file.silentCreate(templateType, attachedFile, outputname, importObject)(see page 1685)
- $file.create(templateFilename)(see page 1686)
- $file.create(attachedFile)(see page 1686)
- $file.create(templateFilename, importObject)(see page 1687)
- $file.create(attachedFile, importObject)(see page 1687)
- $file.create(templateFilename, outputFileName, importObject)(see page 1688)
- $file.create(attachedFile, outputFileName, importObject)(see page 1688)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- $file.create(templateType, templateFilename, outputname, importObject)(see page 1689)
- $file.create(templateType, attachedFile, outputname, importObject)(see page 1689)
- $file.createAndWait(templateFilename)(see page 1690)
- $file.createAndWait(attachedFile)(see page 1690)
- $file.createAndWait(templateFilename, contextValue)(see page 1691)
- $file.createAndWait(attachedFile, contextValue)(see page 1691)
- $file.createAndWait(templateFileName, outputFileName, contextValue)(see page 1692)
- $file.createAndWait(attachedFile, outputFileName, contextValue)(see page 1692)
- $file.createAndWait(templateFileName, outputFileName, ContextName, contextValue)(see page 1693)
- $file.createAndWait(attachedFile, outputFileName, ContextName, contextValue)(see page 1693)
- $file.createAndWait(templateFileName, outputFileName, context)(see page 1694)
- $file.createAndWait(attachedFile, outputFileName, context)(see page 1694)
- $file.copy(inputFilename)(see page 1695)
- $file.copy(inputFilename, outputFilename)(see page 1695)
- $file.exists(pathname)(see page 1696)
- $file.computeName(directory, name)(see page 1696)
- $file.computeName(directory, name, fileType)(see page 1696)

$file allows generating an output report file in a template file.

$file.silentCreate(templateFilename)

A shortcut to create a file, the output filename of which is the template name, not an import context object. For example:

```
$file.silentCreate('overview.html')
```

	Name	Type	Description
Parameter(s)	templateFilename	java.lang.String	• The **templateFilename** parameter can be either an absolute or relative path to a file in the file system. • If the file is not found in the file system, the filename would be searched as an Attached File Element in a MagicDraw project in the File property.
Return	-	void	

$file.silentCreate(attachedFile)

A shortcut to create a file, the output filename of which is the template name, not an import context object. For example:

```
$file.silentCreate($attachedFile)
```

	Name	Type	Description
Parameter(s)	attachedFile	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An Attached File Element.
Return	-	void	

$file.silentCreate(templateFilename, importObject)

A shortcut to create a file, the output filename of which is the template name. For example:

```
$file.silentCreate('overview.html','')
```

	Name	Type	Description
Parameter(s)	templateFilename	java.lang.String	• The **templateFilename** parameter can be either an absolute or relative path to a file in the file system. • If the file is not found in the file system, the filename would be searched as an Attached File Element in a MagicDraw project in the File property.
	importObject	java.lang.Object	An object reference, which will be $importer in the template file. You can use the $importer variable in the template file to get data.
Return	-	void	-

$file.silentCreate(attachedFile, importObject)

A shortcut to create a file, the output filename of which is the template name. For example:

```
$file.silentCreate($attachedFile,'')
```

	Name	Type	Description
Parameter(s)	attachedFile	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An Attached File Element.

	Name	Type	Description
	importObject	java.lang.Object	An object reference, which will be $importer in the template file. You can use the $importer variable in the template file to get data.
Return	-	void	-

$file.silentCreate(templateFilename, outputFileName, importObject)

Generate a report output from a given template name. For example:

```
$file.silentCreate('overview.html','overview.html','')
```

	Name	Type	Description
Parameter(s)	templateFilename	java.lang.String	• The **templateFilename** parameter can be either an absolute or relative path to a file in the file system. • If the file is not found in the file system, the filename would be searched as an Attached File Element in a MagicDraw project in the File property.
	outputFileName	java.lang.String	The output filename.
	importObject	java.lang.Object	An object reference, which will be $importer in the template file. You can use the $importer variable in the template file to get data.
Return	-	void	-

$file.silentCreate(attachedFile, outputFileName, importObject)

Generate a report output from a given template name. For example:

```
$file.silentCreate($attachedFile,'overview.html','')
```

	Name	Type	Description
Parameter(s)	attachedFile	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An Attached File Element.

	Name	Type	Description
	outputFileName	java.lang.String	The output filename.
	importObject	java.lang.Object	An object reference, which will be $importer in the template file. You can use the $importer variable in the template file to get data.
Return	-	void	-

$file.silentCreate(templateType, templateFilename, outputname, importObject)

Generate a report output from a given template name. For example:

```
$file.silentCreate('html','overview','overview','')
```

	Name	Type	Description
Parameter(s)	templateType	java.lang.String	A template type such as rtf, html, or htm.
	templateFilename	java.lang.String	• The **templateFilename** parameter can be either an absolute or relative path to a file in the file system. • If the file is not found in the file system, the filename would be searched as an Attached File Element in a MagicDraw project in the File property.
	outputFileName	java.lang.String	An output filename without a filename extension.
	importObject	java.lang.Object	An object reference, which will be $importer in the template file. You can use the $importer variable in the template file to get data.
Return	-	void	-

$file.silentCreate(templateType, attachedFile, outputname, importObject)

Generate a report output from a given template name. For example:

```
$file.silentCreate('html',$attachedFile,'overview','')
```

	Name	Type	Description
Parameter(s)	templateType	java.lang.String	A template type such as rtf, html, or htm.
	attachedFile	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An Attached File Element.
	outputFileName	java.lang.String	An output filename without a filename extension.
	importObject	java.lang.Object	An object reference, which will be $importer in the template file. You can use the $importer variable in the template file to get data.
Return	-	void	-

$file.create(templateFilename)

A shortcut to create a file, the output filename of which is the template name, not an import context object. For example:

```
$file.create('overview.html')
```

	Name	Type	Description
Parameter(s)	templateFilename	java.lang.String	• The **templateFilename** parameter can be either an absolute or relative path to a file in the file system. • If the file is not found in the file system, the filename would be searched as an Attached File Element in a MagicDraw project in the File property.
Return	-	java.lang.String	An output pathname. It will return an empty string if there is an error.

$file.create(attachedFile)

A shortcut to create a file, the output filename of which is the template name, not an import context object. For example:

```
$file.create($attachedFile)
```

	Name	Type	Description
Parameter(s)	attachedFile	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An Attached File Element.
Return	-	java.lang.String	An output pathname. It will return an empty string if there is an error.

$file.create(templateFilename, importObject)

A shortcut to create a file, the output filename of which is the template name. For example:

```
$file.create('overview.html','')
```

	Name	Type	Description
Parameter(s)	templateFilename	java.lang.String	• The **templateFilename** parameter can be either an absolute or relative path to a file in the file system. • If the file is not found in the file system, the filename would be searched as an Attached File Element in a MagicDraw project in the File property.
	importObject	java.lang.Object	An object reference, which will be importer in the template file. You can use the $importer variable in the template file to get data.
Return	-	java.lang.String	An output pathname. It will return an empty string if there is an error.

$file.create(attachedFile, importObject)

A shortcut to create a file, the output filename of which is the template name. For example:

```
$file.create($attachedFile,'')
```

	Name	Type	Description
Parameter(s)	attachedFile	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An Attached File Element.

	Name	Type	Description
	importObject	java.lang.Object	An object reference, which will be importer in the template file. You can use the $importer variable in the template file to get data.
Return	-	java.lang.String	An output pathname. It will return an empty string if there is an error.

$file.create(templateFilename, outputFileName, importObject)

Generate a report output from a given template name. For example:

```
$file.create('overview.html','overview.html','')
```

	Name	Type	Description
Parameter(s)	templateFilename	java.lang.String	• The **templateFilename** parameter can be either an absolute or relative path to a file in the file system. • If the file is not found in the file system, the filename would be searched as an Attached File Element in a MagicDraw project in the File property.
	outputFileName	java.lang.String	An output filename.
	importObject	java.lang.Object	An object reference, which will be $importer in the template file. You can use the $importer variable in the template file to get data.
Return	-	java.lang.String	An output pathname. It will return an empty string if there is an error.

$file.create(attachedFile, outputFileName, importObject)

Generate a report output from a given template name. For example:

```
$file.create($attachedFile,'overview.html','')
```

	Name	Type	Description
Parameter(s)	attachedFile	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An Attached File Element.
	outputFileName	java.lang.String	An output filename.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

	Name	Type	Description
	importObject	java.lang.Object	An object reference, which will be $importer in the template file. You can use the $importer variable in the template file to get data.
Return	-	java.lang.String	An output pathname. It will return an empty string if there is an error.

$file.create(templateType, templateFilename, outputname, importObject)

Generate a report output from a given template name. For example:

```
$file.create('html','overview','overview','')
```

	Name	Type	Description
Parameter(s)	templateType	java.lang.String	A template type such rtf, html, and htm.
	templateFilename	java.lang.String	• The **templateFilename** parameter can be either an absolute or relative path to a file in the file system. • If the file is not found in the file system, the filename would be searched as an Attached File Element in a MagicDraw project in the File property.
	outputname	java.lang.String	An output filename without a filename extension.
	importObject	java.lang.Object	An object reference, which will be $importer in the template file. You can use the $importer variable in the template file to get data.
Return	-	java.lang.String	An output pathname. It will return an empty string if there is an error.

$file.create(templateType, attachedFile, outputname, importObject)

Generate a report output from a given template name. For example:

```
$file.create('html',$attachedFile,'overview','')
```

	Name	Type	Description
Parameter(s)	templateType	java.lang.String	A template type such rtf, html, and htm.
	attachedFile	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An Attached File Element.
	outputname	java.lang.String	An output filename without a filename extension.
	importObject	java.lang.Object	An object reference, which will be $importer in the template file. You can use the $importer variable in the template file to get data.
Return	-	java.lang.String	An output pathname. It will return an empty string if there is an error.

$file.createAndWait(templateFilename)

Open a new template engine to generate a report and return a path to generate the file. The generated report will be given the template filename. For example:

```
$file.createAndWait('overview.html')
```

	Name	Type	Description
Parameter(s)	templateFilename	java.lang.String	• The **templateFilename** parameter can be either an absolute or relative path to a file in the file system. • If the file is not found in the file system, the filename would be searched as an Attached File Element in a MagicDraw project in the File property.
Return	-	java.lang.String	An absolute path to generate a file.

$file.createAndWait(attachedFile)

Open a new template engine to generate a report and return a path to generate the file. The generated report will be given the template filename. For example:

```
$file.createAndWait($attachedFile)
```

	Name	Type	Description
Parameter(s)	attachedFile	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An Attached File Element.
Return	-	java.lang.String	An absolute path to generate a file.

$file.createAndWait(templateFilename, contextValue)

Open a new template engine to generate a report and return a path to generate the file. The generated report will be named after the template filename. For example:

```
$file.createAndWait('overview.html',$var)
```

	Name	Type	Description
Parameter(s)	templateFilename	java.lang.String	• The **templateFilename** parameter can be either an absolute or relative path to a file in the file system. • If the file is not found in the file system, the filename would be searched as an Attached File Element in a MagicDraw project in the File property.
	contextValue	java.lang.Object	An additional object added to a new template context with a default $importer name.
Return		java.lang.String	

$file.createAndWait(attachedFile, contextValue)

Open a new template engine to generate a report and return a path to generate the file. The generated report will be named after the template filename. For example:

```
$file.createAndWait($attachedFile,$var)
```

	Name	Type	Description
Parameter(s)	attachedFile	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An Attached File Element.

	Name	Type	Description
	contextValue	java.lang.Object	An additional object added to a new template context with a default $importer name.
Return		java.lang.String	

$file.createAndWait(templateFileName, outputFileName, contextValue)

Open a new template engine to generate a report and return a path to generate the file. "importer" will be used as the name of a context value. For example:

```
$file.createAndWait('overview.html','overview.html',$var)
```

	Name	Type	Description
Parameter(s)	templateFilename	java.lang.String	• The **templateFilename** parameter can be either an absolute or relative path to a file in the file system. • If the file is not found in the file system, the filename would be searched as an Attached File Element in a MagicDraw project in the File property.
	outputFilename	java.lang.String	An output report filename.
	contextValue	java.lang.Object	An additional object added to a new template context with a default $importer name.
Return		java.lang.String	An absolute path to generate a file.

$file.createAndWait(attachedFile, outputFileName, contextValue)

Open a new template engine to generate a report and return a path to generate the file. "importer" will be used as the name of a context value. For example:

```
$file.createAndWait($attachedFile,'overview.html',$var)
```

	Name	Type	Description
Parameter(s)	attachedFile	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An Attached File Element.

	Name	Type	Description
	outputFilename	java.lang.String	An output report filename.
	contextValue	java.lang.Object	An additional object added to a new template context with a default $importer name.
Return		java.lang.String	An absolute path to generate a file.

$file.createAndWait(templateFileName, outputFileName, ContextName, contextValue)

Open a new template engine to generate a report and return a path to generate the file. For example:

```
$file.createAndWait('overview.html','overview.html','var',$var)
```

	Name	Type	Description
Parameter(s)	templateFilename	java.lang.String	• The **templateFilename** parameter can be either an absolute or relative path to a file in the file system. • If the file is not found in the file system, the filename would be searched as an Attached File Element in a MagicDraw project in the File property.
	outputFilename	java.lang.String	An output report filename.
	contextName	java.lang.String	Add an object context's name.
	contextValue	java.lang.Object	An additional object added to a new template context with a default $importer name.
Return	-	java.lang.String	An absolute path to generate a file.

$file.createAndWait(attachedFile, outputFileName, ContextName, contextValue)

Open a new template engine to generate a report and return a path to generate the file. For example:

```
$file.createAndWait($attachedFile,'overview.html','var',$var)
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

	Name	Type	Description
Parameter(s)	attachedFile	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An Attached File Element.
	outputFilename	java.lang.String	An output report filename.
	contextName	java.lang.String	Add an object context's name.
	contextValue	java.lang.Object	An additional object added to a new template context with a default $importer name.
Return	-	java.lang.String	An absolute path to generate a file.

$file.createAndWait(templateFileName, outputFileName, context)

Open a new template engine to generate a report and return a path to generate the file. For example:

```
$file.createAndWait('overview.html','overview.html',$map)
```

	Name	Type	Description
Parameter(s)	templateFilename	java.lang.String	• The **templateFilename** parameter can be either an absolute or relative path to a file in the file system. • If the file is not found in the file system, the filename would be searched as an Attached File Element in a MagicDraw project in the File property.
	outputFilename	java.lang.String	An output report filename.
	context	java.util.Map<String, Object>	An additional context added to a new template.
Return	-	java.lang.String	An absolute path to generate a file.

$file.createAndWait(attachedFile, outputFileName, context)

Open a new template engine to generate a report and return a path to generate the file. For example:

```
$file.createAndWait($attachedFile,'overview.html',$map)
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

	Name	Type	Description
Parameter(s)	attachedFile	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An Attached File Element.
	outputFilename	java.lang.String	An output report filename.
	context	java.util.Map<String, Object>	An additional context added to a new template.
Return	-	java.lang.String	An absolute path to generate a file.

$file.copy(inputFilename)

Copy an input file to an output file using the same name in the binary format. For example:

```
$file.copy('icon.gif')
```

	Name	Type	Description
Parameter(s)	inputFilename	java.lang.String	An input filename
Return	-	java.lang.String	An output pathname. It will return an empty string if there is an error.

$file.copy(inputFilename, outputFilename)

Copy an input file to an output file in the binary format. For example:

```
$file.copy('icon.gif','icon.gif')
```

	Name	Type	Description
Parameter(s)	inputFilename	java.lang.String	An input filename.
	outputFilename	java.lang.String	An output filename.
Return	-	java.lang.String	An output pathname. It will return an empty string if there is an error.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

$file.exists(pathname)

Test if a file denoted by a specific pathname exists. By default, the current directory refers to a template location. For example:

```
$file.exists("$template.resourcesLocation/myimage.png")
$file.exists("C:/myfolder/myimage.png")
$file.exists("mytemplate.txt")
```

	Name	Type	Description
Parameter(s)	pathname	java.lang.String	A pathname string.
Return	-	java.lang.String	True if and only if a file or directory denoted by a specific pathname exists; otherwise, false.

$file.computeName(directory, name)

Create a pathname string from a given directory and name.

	Name	Type	Description
Parameter(s)	directory	java.lang.String	A parent directory of a file.
	name	java.lang.String	A filename (excludes the filename extension).
Return	-	java.lang.String	A pathname from the given directory and name.

For example:

```
$file.computeName('actors', $ac.ID, 'html')
output: 'actor/_123456789.html'
```

$file.computeName(directory, name, fileType)

Create a pathname string from a given directory, name, and file type.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

	Name	Type	Description
Parameter(s)	directory	java.lang.String	A parent directory of a file.
	name	java.lang.String	A filename (excluding the extension).
	fileType	java.lang.String	A file type, such as rtf, txt, and html.
Return	-	java.lang.String	A pathname from the given directory, name, and file type.

Array tool

Use $array to create an Array or a HashSet instance.

$array.createArray()

Create an empty ArrayList.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	java.util.List	An instance of ArrayList.

$array.createArray(collection)

Construct an ArrayList containing elements of the specified collection, in the order they are returned by the collection's iterator. Return a zero size ArrayList if the given collection is null.

	Name	Type	Description
Parameter(s)	collection	java.util.Collection	A collection of instances whose elements are to be placed into a list.
Return	-	java.util.List	An ArrayList containing elements of the specified collection.

$array.subList(list, size)

Create an ArrayList of a portion of the list in the given size.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

	Name	Type	Description
Parameter(s)	list	java.util.List	An original list.
	size	int	A view size of a given list.
Return	-	java.util.List	An ArrayList of a specified view size within a given list.

$array.addCollection(parent, child)

Add a child list into a parent collection. It helps the template handle the case where the child is null.

	Name	Type	Description
Parameter(s)	parent	java.util.Collection	A parent collection.
	child	java.util.Collection	A child collection.
Return	-	void	

$array.createHashSet()

Create a HashSet instance.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	java.util.Set	An instance of HashSet.

Group tool

$group.create()

Create a new instance of a group tool.

	Name	Type	Description
Parameter(s)	-	-	-

	Name	Type	Description
Return	-	GroupTool	An instance of a group tool.

$group.init()

Initialize a group tool.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	void	-

$group.groupNames()

Return a set of group names.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	java.util.Set	A set of group names.

$group.contains(groupName)

Test whether a group name is contained in a set of group names.

	Name	Type	Description
Parameter(s)	groupName	java.lang.String	A group name.
Return	-	-	True if the group name is contained in a set of group names, otherwise false.

$group.put(groupName, object)

Add an object into a group.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

	Name	Type	Description
Parameter(s)	groupName	java.lang.String	A group name.
	object	java.lang.Object	An object being added.
Return	-	void	-

$group.get(groupName)

Return a list of group objects.

	Name	Type	Description
Parameter(s)	groupName	java.lang.String	A group name.
Return	-	java.util.List	A list of group objects.

$group.remove(groupName)

Remove a group from the specified group name.

	Name	Type	Description
Parameter(s)	groupName	java.lang.String	A group name.
Return	-	java.util.List	A list of group objects previously associated with a specified group name, or null if there is no group corresponding to the key.

$group.removeAll()

Remove all groups.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	void	

$group.clear()

Remove all mappings.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	void	-

Map tool

$map.createHashMap()

Create a HashMap instance.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	java.util.HashMap	An instance of HashMap.

Date tool

$date is a method to display and format the current date and time.

$date

Display the current date and time, for example, Oct 19, 2003 and 9:54:50 PM respectively.

$date.long

Display the current date and time in a long form, for example, October 19, 2003 and 9:54:50 PM. respectively.

$date.full_date

Display the current date in a full form, for example, Sunday, October 19, 2003.

$date.get('format')

Display the current date in a specific format, for example, $date.get('yyyy-M-d H:m:s') is displayed as 2003-10-19 21:54:50.

Letter	Date or Time Component	Example
G	Era designator	AD
y	Year 1996	96
M	Month in year	July; Jul; 07

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Letter	Date or Time Component	Example
w	Week in year	27
W	Week in month	2
D	Day in year	189
d	Day in month	10
F	Day of week in month	2
E	Day in week	Tuesday; Tue
a	Am/pm marker	PM
H	Hour in day (0-23)	0
k	Hour in day (1-24)	24
K	Hour in am/pm (0-11)	0
h	Hour in am/pm (1-12)	12
m	Minute in hour	30
s	Second in minute	55
S	Millisecond	978
z	Time zone	Pacific Standard Time; PST; GMT-08.00
Z	Time zone	Time zone -0800

Year

- If the number of letters in a pattern letters is four or more, a calendar specific long form is used.
- Otherwise, a calendar specific short or abbreviated form is used.

Date and Time Pattern	Result
yy	10
yyyy	2010

Month

- If the number of letters in a pattern is less than 3, the number form is used.
- If the number of letters in a pattern is 3, a short or abbreviated form is used.
- If the number of letters in a pattern is 4 or more, the full form is used.

Date and Time Pattern	Result
M	4

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Date and Time Pattern	Result
MM	04
MMM	Apr
MMMM	April

Days

- If the number of letters in a pattern is four or more, the full form is used.
- Otherwise, a calendar specific short or abbreviated form is used.

Date and Time Pattern	Result
E	Mon
EEEE	

Profiling tool

You can use $profiling to access MagicDraw meta-model information.

$profiling.getGeneralizationName(modelName)

Return a generalization model of modelName.

	Name	Type	Description
Parameter(s)	modelName	java.lang.String	A meta-model name.
Return	-	java.util.Collection<String>	A list of generalization model names.

$profiling.getDeclaringElementName (modelName, propertyName)

Retrieve the meta-model name which is the declared property name.

	Name	Type	Description
Parameter(s)	modelName	java.lang.String	A meta-model name.
	propertyName	java.lang.String	A property name.
Return	-	java.lang.String	A meta-model name.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

$profiling.getPropertyTypeName (modelName, propertyName)

Retrieve property types from meta-model names.

	Name	Type	Description
Parameter(s)	modelName	java.lang.String	A meta-model name.
	propertyName	java.lang.String	A property name.
Return	-	java.lang.String	A property type name.

$profiling.getPropertyTypeName (element, propertyName)

Retrieve property types from an element.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.
	propertyName	java.lang.String	A property name.
Return	-	java.lang.String	A property type name.

$profiling.getElementProperties(modelName)

Retrieve a collection of element property names from meta-model names.

	Name	Type	Description
Parameter(s)	modelName	java.lang.String	A meta-model name.
Return	-	java.util.Collection<String>	A collection of element property names.

$profiling.getElementProperties(element)

Retrieve a collection of element property names from elements.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.
Return	-	java.util.Collection<String>	A collection of element property names.

$profiling.getElementProperty(element, propertyName)

Retrieve property values of specified elements and property names.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.
	propertyName	java.lang.String	A property name.
Return	-	java.lang.Object	A property object.

$profiling. getHumanPropertyName(element, propertyName)

Return text representing a property name.

	Name	Type	Description
Parameter(s)	propertyName	java.lang.String	A property name.
	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.
Return	-	java.lang.String	Text representing a property name.

Image tool

$image provides a rich set of image manipulation methods that enable you to transform images during report generation. With **$image**, you can scale, rotate, split, and resize images, include external images, and export them for TXT and HTML templates.

Syntax

These manipulation methods can be broadly divided into seven categories:

1. Scaling images
2. Rotating images
3. Splitting images
4. Resizing fixed-pixel images
5. Resizing fixed-measurement images
6. Including external images
7. Exporting images for TXT and HTML templates

Multiple image transformations will have cumulative effects. Desired sizes can be specified either as pixel counts or as measurements in inches, centimeters, or millimeters. While the parameters for resizing an image with a specific measurement (in, cm, mm, pt, or px) must be specified in a pair of quote marks, all other size parameters do not need to be specified in a pair of quote marks.

The **keepRatio** parameter takes either a true or false value; a true value will resize the image and keep the original image height/width ratio; a false value will only resize one axis, as set by the method name, resulting in a stretched image.

Related pages

Scaling images

There are two scaling methods that you can use to scale an image using a given factor. For example, Method 1 provides height and width parameters (scaleWidth and scaleHeight), while Method 2 provides a single parameter that will scale both axes equally (scaleFactor). These three parameters must be positive real numbers.

$image.scale(image, scaleWidth, scaleHeight)

Return an image icon for an element.

	Name	Type	Description
Parameter(s)	image	com.nomagic.magicreport.Image	An image object for the element icon.
	scaleWidth	java.lang.Double	The width parameter of the image.
	scaleHeight	java.lang.Double	The height parameter of the image.
Return	-	com.nomagic.magicreport.Image	The resized image object for the element icon.

$image.scale(image, scaleFactor)

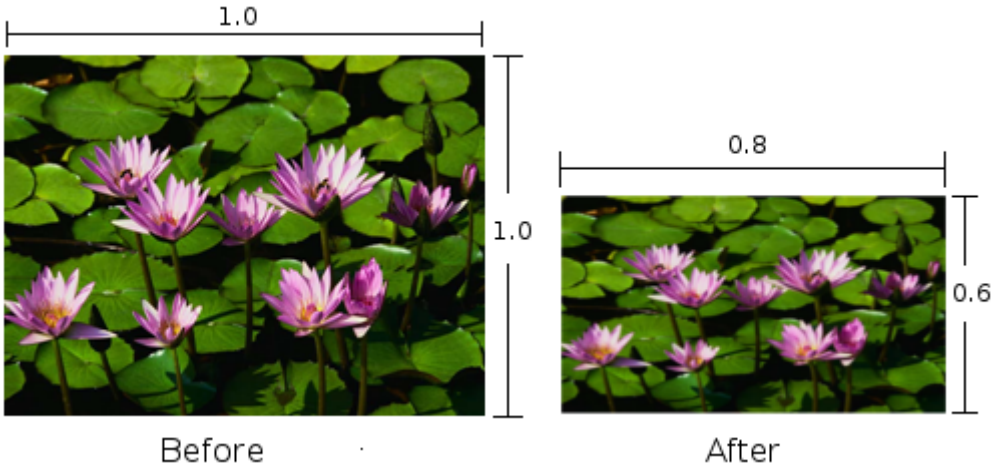
Return an image icon for an element.

	Name	Type	Description
Parameter(s)	image	com.nomagic.magicreport.Image	An image object for the element icon.
	scaleFactor	java.lang.Double	The scaling parameter of the image.
Return	-	com.nomagic.magicreport.Image	The resized image object for the element icon.

Use **$image.scale($diagram.image, 0.5)**, for example, to scale down the image to half the original size. The following photos show the result.

Results of using the image scale function.

Use **$image.scale($diagram.image, 0.8, 0.6)** to scale the image's width down to 80% and height to 60%. The following photos show the result.



Results of using the image scale function.

Alternative scaling methods

The images used in generated reports can also be scaled by specifying the **Exported Image Size** property in one of the following locations:

- The **Properties** dialog(see page 587), which is used to define image export properties for the **Save as Image** functionality.
- The **Environment Option** dialog. To open the dialog, in the main menu, select **Options** → **Environment**. Image export properties are located under the **General** group.

Note that the **Exported Image Size** setting is applied in generated reports only if you select **No resize** from the **Auto image size** drop down list located in the **Output Options** area of the **Report Wizard**.

Scaling Quality

Because the image tool provides several methods for scaling images, the quality of the scaled image depends on the algorithm that is used. To set the image scaling quality, $image.setScalingQuality() code must be inserted before scaling images. For example:

```
$image.setScalingQuality(5)
$image.scale(0.5)
```

The code above will set the image scaling quality to 'highest'. The possible values are from 1 to 5, where 5 is 'highest' and 1 is 'lowest'. The following table provides the description of each value.

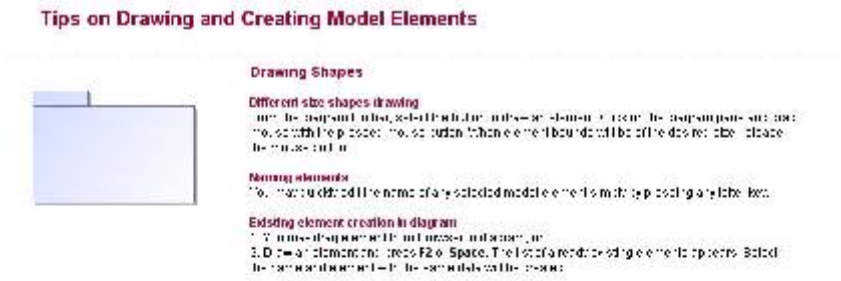
Value	Description
1	lowest' – Use one-step nearest neighbor interpolation.
2	'low' – Use one-step bi-cubic interpolation.
3	'medium – Use multi-step, bi-linear interpolation.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

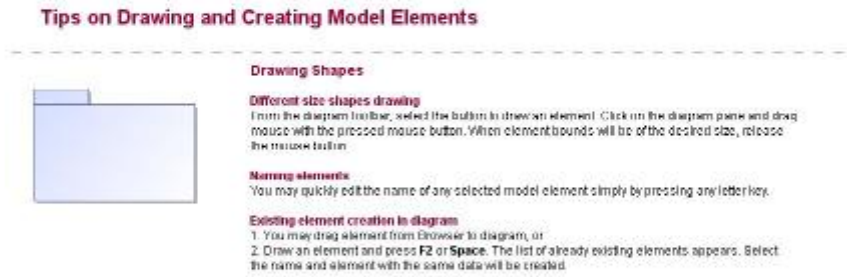
Value	Description
4	'high' – Use area average image scaling algorithm.
5	'highest' – Use a lossless transformation when the template is ODF, DOCX, or RTF; otherwise, area average image scaling algorithm will be used instead.

Sample pictures from different image scaling qualities with their sizes scaled down to 50% are indicated in the pictures below (ranging from the value 1 'lowest' to 5 'highest'):

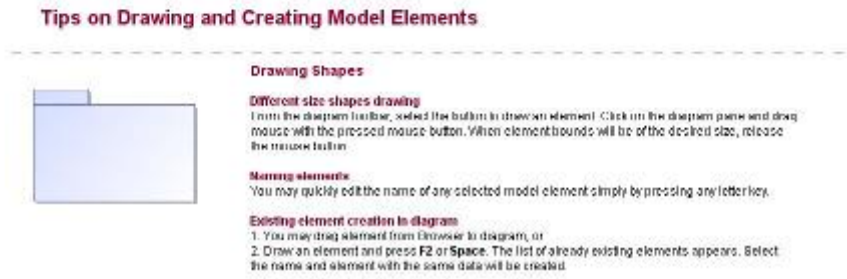
$image.setScalingQuality(1)



$image.setScalingQuality(2)

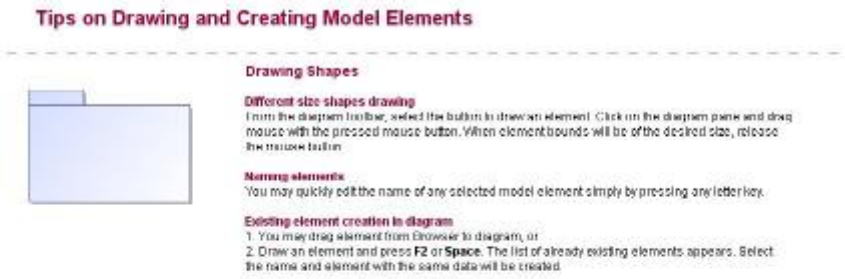


$image.setScalingQuality(3)

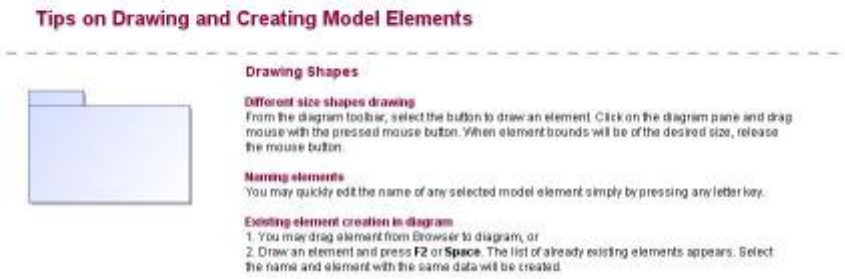


Different results of using the image set scaling quality function.

$image.setScalingQuality(4)



$image.setScalingQuality(5)



Different results of using the image set scaling quality function.

Rotating images

There are two rotating methods that you can use to rotate a given image by 90 degrees: (i) **rotateRight** for clockwise rotation and (ii) **rotateLeft** for counterclockwise rotation.

$image.rotateRight(image)

Return an image icon for an element.

	Name	Type	Description
Parameter(s)	image	com.nomagic.magicreport.Image	An image object for the element icon.
Return	-	com.nomagic.magicreport.Image	The rotated image object for the element icon.

$image.rotateLeft(image)

Return an image icon for an element.

	Name	Type	Description
Parameter(s)	image	com.nomagic.magicreport.Image	An image object for the element icon.
Return	-	com.nomagic.magicreport.Image	The rotated image object for the element icon.

Use **$image.rotateRight($diagram.image)**, for example, to rotate the image 90 degrees clockwise. The following photos show the result.

Before After

Results of using the image rotate right function.

Use **$image.rotateLeft($diagram.image)**, for example, to rotate the image 90 degrees counterclockwise. The following photos show the result.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Before

After

Results of using the image rotate left function.

> RTF Image rotation, with the scaling quality set to 'highest,' is not supported in any RTF document when opened with OpenOffice.org[339].

Splitting images

If you have an image that is larger than the paper size, you can split the image into smaller tiles.

> **Note**
> **$image.split()** supports only the JPG and PNG image formats.

$image.split($diagram.image, columns, rows)

Split an image into rows and columns. The returned array of the image contains chunks of images arranged from left-to-right and top-to-bottom.

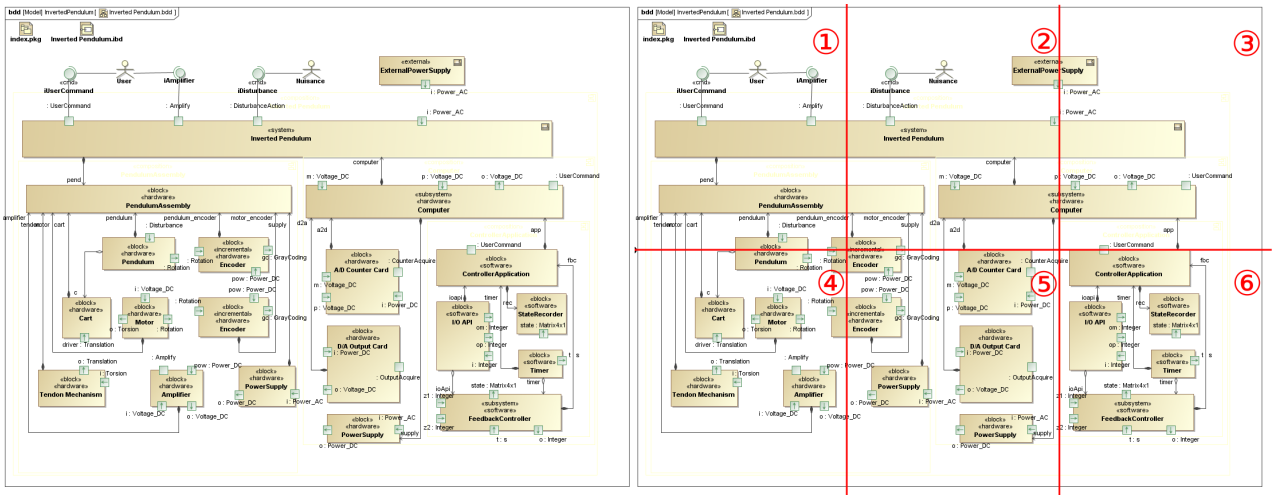
	Name	Type	Description
Parameter(s)	image	com.nomagic.magicreport.Image	An image object.
	column	int	The number of columns. Specifies -1 to automate the calculated number of columns from the paper width.

339 http://OpenOffice.org

	Name	Type	Description
	row	int	The number of rows. Specifies -1 to automate the calculated number of rows from the paper height.
Return		java.util.List<com.nomagic.magicreport.Image>	The array of the split image ordered from left-to-right and top-to-bottom.

For example, to split an image into 3x2 tiles, type the following lines:

```
#foreach ($d in $Diagram)
#foreach ($chunk in $image.split($d.image, 3, 2))
$chunk
#end
#end
```



The original image vs. 3x2 tiled image.

$image.split($diagram.image)

Split an image into rows and columns by automatically calculating the number of rows and columns based on the paper size. The returned array of the image contains divided images arranged from left-to-right and top-to-bottom.

	Name	Type	Description
Parameter(s)	image	com.nomagic.magicreport.Image	An image object.
Return		java.util.List<com.nomagic.magicreport.Image>	The array of the split image ordered from left-to-right and top-to-bottom.

Resizing fixed-pixel images

There are five methods to specify the resulting dimensions for an image to be resized in pixel size (the size parameters must be positive numbers or -1 number). If the value is -1, it will resize an image to the document paper dimensions. For example:

```
$image.setWidth($diagram.image, -1)
```

As shown by the example, the image will be resized to a paper width while maintaining the aspect ratio. The value -1 applies only to certain template types such as *RTF*, *ODT*, and *DOCX*. If the value is -2, it will resize an image to document paper bounds if and only if image bounds are larger than paper bounds. For example:

```
$image.setWidth($diagram.image, -2)
```

Using the value -2 also maintains the image aspect ratio. However, it can only be applied to certain template types such as *RTF*, *ODT*, and *DOCX*.

$image.setSize(image, sizeWidth, sizeHeight)

Return an image icon for an element. This method is used to resize the image to an exact size; the width and height in pixels.

	Name	Type	Description
Parameter(s)	image	com.nomagic.magicreport.Image	An image object for the element icon.
	sizeWidth	java.lang.Integer	The width of the image in pixels.
	sizeHeight	java.lang.Integer	The height of the image in pixels.
Return	-	com.nomagic.magicreport.Image	The resized image object for the element icon.

$image.setHeight(image, size)

Return an image icon for an element. This method is used to resize the image to a specific height (in pixels), while maintaining the image aspect ratio.

	Name	Type	Description
Parameter(s)	image	com.nomagic.magicreport.Image	An image object for the element icon.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

	Name	Type	Description
	size	java.lang.Integer	The height of the image in pixels.
Return	-	com.nomagic.magicreport.Image	The resized image object for the element icon.

$image.setHeight(image, size, keepRatio)

Return an image icon for an element. This method is used to resize the image to a specific height (in pixels) and to specify whether the image aspect ratio is to be maintained or not (depending on the keepRatio parameter).

	Name	Type	Description
Parameter(s)	image	com.nomagic.magicreport.Image	An image object for the element icon.
	size	java.lang.Integer	The height of the image in pixels.
	keepRatio	java.lang.Boolean	A flag to maintain the image aspect ratio
Return	-	com.nomagic.magicreport.Image	The resized image object for the element icon.

$image.setWidth(image, size)

Return an image icon for an element. This method is used to resize the image to a specific width (in pixels), while maintaining the image aspect ratio.

	Name	Type	Description
Parameter(s)	image	com.nomagic.magicreport.Image	An image object for the element icon.
	size	java.lang.Integer	The width of the image in pixels.
Return	-	com.nomagic.magicreport.Image	The resized image object for the element icon.

$image.setWidth(image, size, keepRatio)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Return an image icon for an element. This method is used to resize the image to a specific width (in pixels) and to specify whether the image aspect ratio is to be maintained or not (depending on the keepRatio parameter).

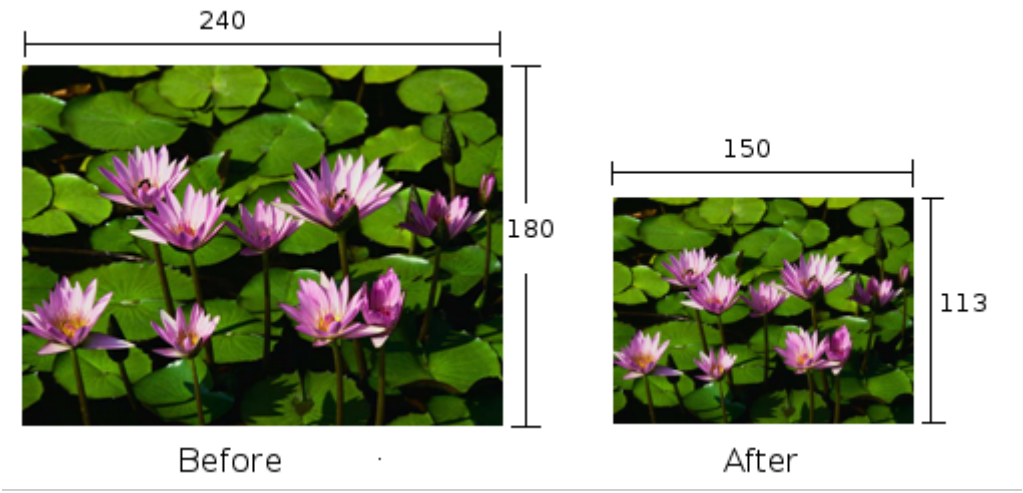
	Name	Type	Description
Parameter(s)	image	com.nomagic.magicreport.Image	An image object for the element icon.
	size	java.lang.Integer	The width of the image in pixels.
	keepRatio	java.lang.Boolean	A flag to maintain the image aspect ratio
Return	-	com.nomagic.magicreport.Image	The resized image object for the element icon.

Use **$image.setSize($diagram.image, 200, 200)**, for example, to resize the image's width and height to 200 pixels. The following photos show the result.

Results of using the image set size function.

Use either **$image.setWidth($diagram.image, 150)** or **$image.setWidth($diagram.image, 150, true)**, for example, to resize the image's width to 150 pixels and maintain the image aspect ratio. The following photos show the result.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Results of using the image set width function.

Use **$image.setWidth($diagram.image, 150, false)**, for example, to resize the image's width to 150 pixels and ignore the image aspect ratio. The following photos show the result.

Results of using the image set width function.

Resizing fixed-measurement images

There are six methods to specify the resulting dimensions in terms of measurements to resize an image, for example, in inch (in), centimeter (cm), millimeter (mm), point (pt), or pixel (px)], and specifying the dot-per-inch (*DPI*) value. Every measurement parameter must be specified in a pair of quote marks.

$image.setSize(image, measureWidth, measureHeight)

Return an image icon for an element. This method is used to resize the image to an exact size; the width and height in terms of measurements.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

	Name	Type	Description
Parameter(s)	image	com.nomagic.magicreport.Image	An image object for the element icon.
Parameter(s)	measureWidth	java.lang.String	The width of the image in a term of measurement.
	measureHeight	java.lang.String	The height of the image in a term of measurement.
Return	-	com.nomagic.magicreport.Image	The resized image object for the element icon.

$image.setHeight(image, measureSize)

Return an image icon for an element. This method is used to resize the image to a specific height in terms of measurement while maintaining the image aspect ratio.

	Name	Type	Description
Parameter(s)	image	com.nomagic.magicreport.Image	An image object for the element icon.
	measureSize	java.lang.String	The height of the image in a term of measurement.
Return	-	com.nomagic.magicreport.Image	The resized image object for the element icon.

$image.setHeight(image, measureSize, keepRatio)

Return an image icon for an element. This method is used to resize the image to a specific height (a term of measurement), and to specify whether the image aspect ratio is to be maintained or not (depending on the keepRatio parameter).

	Name	Type	Description
Parameter(s)	image	com.nomagic.magicreport.Image	An image object for the element icon.
	measureSize	java.lang.String	The height of the image in a term of measurement.
	keepRatio	java.lang.Boolean	A flag to maintain the image aspect ratio.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

	Name	Type	Description
Return	-	com.nomagic.magicreport.Image	The resized image object for the element icon.

$image.setWidth(image, measureSize)

Return an image icon for an element. This method is used to resize the image to a specific width in a term of measurement, while maintaining the image aspect ratio.

	Name	Type	Description
Parameter(s)	image	com.nomagic.magicreport.Image	An image object for the element icon.
	measureSize	java.lang.String	The width of the image in a term of measurement.
Return	-	com.nomagic.magicreport.Image	The resized image object for the element icon.

$image.setWidth(image, measureSize, keepRatio)

Return an image icon for an element. This method is used to resize the image to a specific width (a term of measurement), and to specify whether the image aspect ratio is to be maintained (depending on the keepRatio parameter).

	Name	Type	Description
Parameter(s)	image	com.nomagic.magicreport.Image	An image object for the element icon.
	measureSize	java.lang.String	The width of the image in a term of measurement.
	keepRatio	java.lang.Boolean	A flag to maintain the image aspect ratio.
Return	-	com.nomagic.magicreport.Image	The resized image object for the element icon.

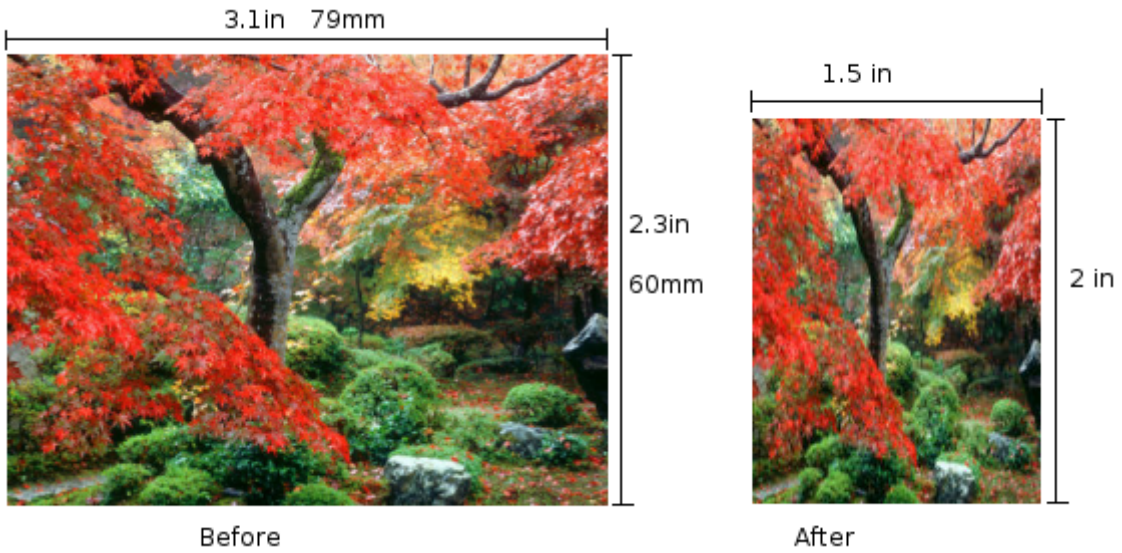
$image.setDPI(dotsPerInches)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

This method is used to set the resolution (dpi) of images. The default value is 96 dpi. The dpi value will only affect the methods that take inches, centimeters, and millimeters as their parameters.

	Name	Type	Description
Parameter(s)	dotsPerInches	java.lang.Integer	Image resolution (dpi).
Return	-	-	-

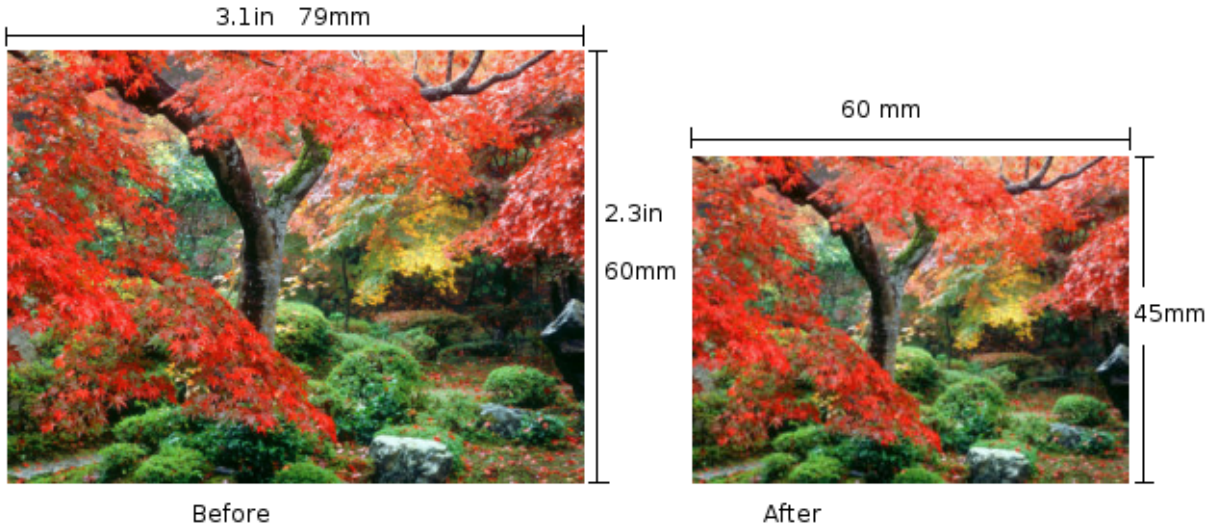
Use **$image.setSize($diagram.image, '1.5in', '2in')**, for example, to resize the image's width to 1.5 inches and height to 2 inches. The following photos show the result.



Results of using the image set size function.

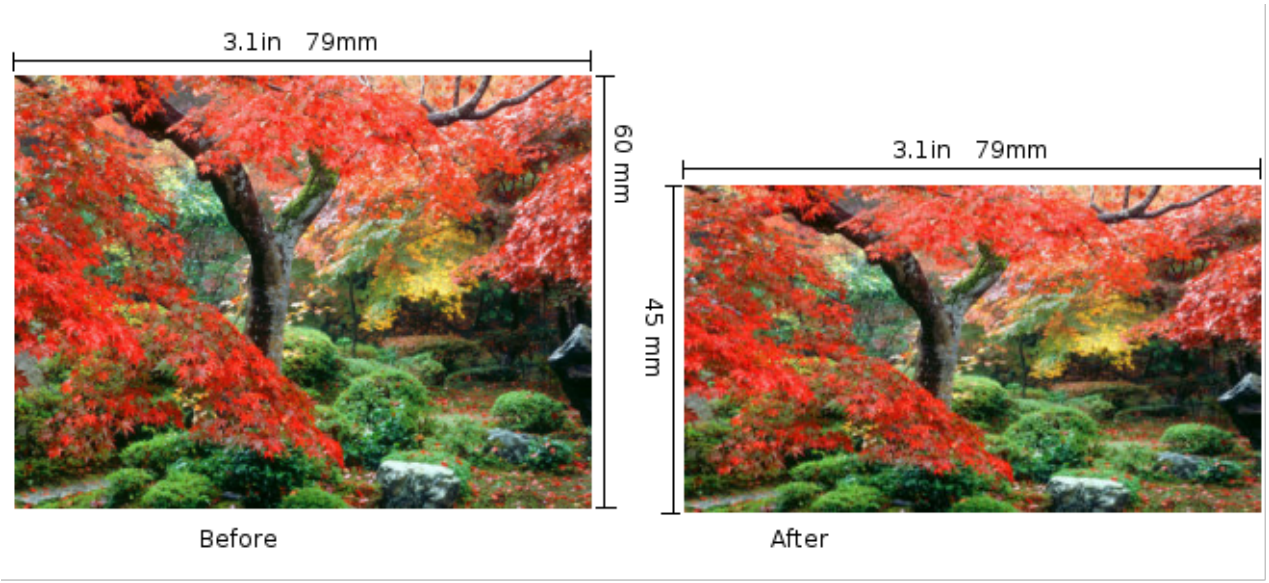
Use either **$image.setHeight($diagram.image, '45mm')** or **$image.setHeight($diagram.image, '45mm', true)**, for example, to resize the image's height to 45 millimeters and maintain the image aspect ratio. The following photos show the result.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Results of using the image set height function.

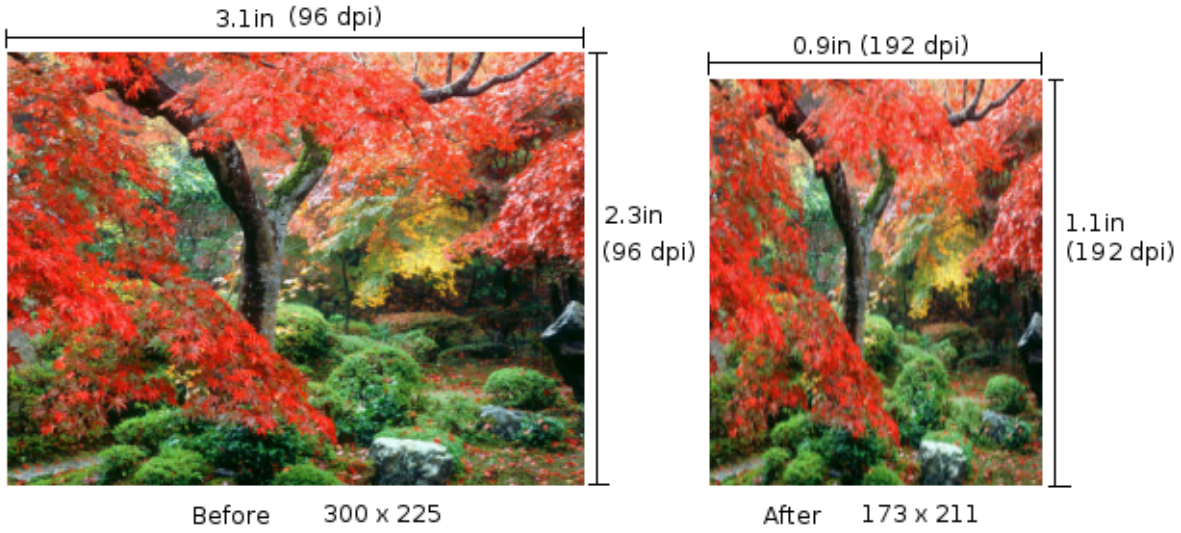
Use **$image.setHeight($diagram.image, '45mm', false)**, for example, to resize the image's height to 45 millimeters and ignore the image aspect ratio. The following photos show the result.



Results of using the image set height function.

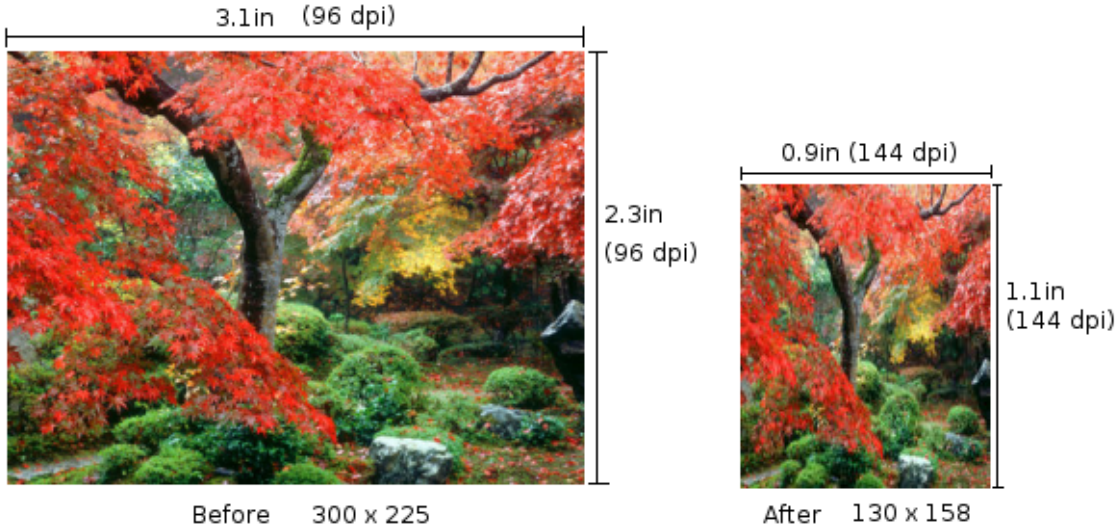
Use **$image.setDPI(192)** and then **$image.setSize($diagram.image, '0.9in', '1.1in')**, for example, to set the resolution (dpi) of the image to 192 dots per inch, and then resize the image width to 0.9 inches and height to 1.1 inches. The resulting image size will become 173x211 pixels. The following photos show the result.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Results of using the image set DPI function.

Instead of using **$image.setDPI(192)**, if you set the resolution (dpi) to 144 using **$image.setDPI(144)**, the resulting image size will become 130x158 pixels. The following photos show the result.



Results of using the image set DPI function.

Including external images

You can include images from an external source in a report. The supported image types are *BMP*, *JPG*, *WBMP*, *GIF*, and *PNG*.

$image.include(location)

Include an external image from a local file or *URL* in a report.

	Name	Type	Description
Parameter(s)	location	java.lang.String	An image's location. The location format can be either a URL or a local file.
Return	-	com.nomagic.magicreport.Image	An image included in a report

For example:

```
$image.include("c:/my document/logo.gif")
$image.include("http:/www.nomagic.com/images/product_boxes/MD.gif")
```

Exporting images for TXT and HTML templates

Exporting images for TXT and HTML templates returns absolute paths, e.g., *file:/C:/Users/username/Desktop/output/out_files/_18_5_1_133803de_1519700709957_140433_4668.jpg*. There are five functions for handling paths of exported images as follows.

$image.createSilentImage(image)
Create an image without returning a path.

	Name	Type	Description
Parameter(s)	image	com.nomagic.magicreport.Image	An image object for the element icon.
Return	-	com.nomagic.magicreport.Image	An exported image.

$image.createImageWithRelativePath(image)
Create an image and return a relative path.

	Name	Type	Description
Parameter(s)	image	com.nomagic.magicreport.Image	An image object for the element icon.
Return	-	com.nomagic.magicreport.Image	An exported image.
	-	java.lang.String	A relative path of the exported image.

$image.getRelativePath(image)
Return a relative path only.

	Name	Type	Description
Parameter(s)	image	com.nomagic.magicreport.Image	An image object for the element icon.
Return	-	java.lang.String	A relative path of the exported image.

$image.createImageWithAbsolutePath(image)
Create an image and return an absolute path.

	Name	Type	Description
Parameter(s)	image	com.nomagic.magicreport.Image	An image object for the element icon.
Return	-	com.nomagic.magicreport.Image	An exported image.
	-	java.lang.String	An absolute path of the exported image.

$image.getAbsolutePath(image)
Return an absolute path only.

	Name	Type	Description
Parameter(s)	image	com.nomagic.magicreport.Image	An image object for the element icon.
Return	-	java.lang.String	An absolute path of the exported image.

Tableprop tool

$tableprop is a table property tool to manage table style. In this version, you can set merged rows vertically and columns horizontally in *DOCX*, *PPTX*, and *XLSX* templates using **$tableprop**.

> **Note**
> This statement must be defined in a table.

Related pages

Creating merged column horizontally

There are two methods that you can use to set column to merged column.

$tableprop.mergeColumns(int number)

When you put the statement in to a column, it will be set as merged column in a specified number.

For example:

Template Code:

Project	**$tableprop.mergeColumns($Package.size())Packages**
	#forcol($p in $Package)$p.name#endcol

Output:

Project	Packages		
	Package A	Package C	Package B

$tableprop.mergeColumns(String stringNumber)

When you put the statement in to a column, it will be set as merged column in a specified string of number.

For example:

Template Code:

Project	**$tableprop.mergeColumns("3")Packages**
	#forcol($p in $Package)$p.name#endcol

Output:

Project	Packages		
	Package A	Package C	Package B

See Creating Merged Column Horizontally for DOCX(see page 1725).

Creating merged row vertically

There are two methods that you can use to merge rows.

$tableprop.mergeRows(Int number)

When you put the statement into a row, it will be set as a merged row in a specified number.

For example:

Template code:

#forrow($c in $sorter.sort($Class)) $c.name$tableprop.mergeRows($c.ownedElement.size())# set($count=0) #foreach($attr in $c.ownedElement)#set($count = $count + 1)	$attr.name#if($count < $c.ownedElement.size())
#end#if(false)	#end#end#endrow

Output:

ClassA	Att1
	Att2
	Att3
ClassB	Att1

$tableprop.mergeRows(String stringNumber)

When you put the statement into a row, it will be set as a merged row in a specified String of number. For example:

Template code:

#set($a = $array.createArray()) #set($void = $a.add("A")) #set($void = $a.add("B")) #set($void = $a.add("C"))	
#forrow($c in $a)**Name**$tableprop.mergeRows("3")	$c#endrow

Output:

Name	A
	B
	C

Report Wizard template editor

Template Editor is a Microsoft Word Add-In. It provides a list of fields that can be used in a Report Wizard template.

Installation

To install Report Wizard Template Editor in Microsoft Word

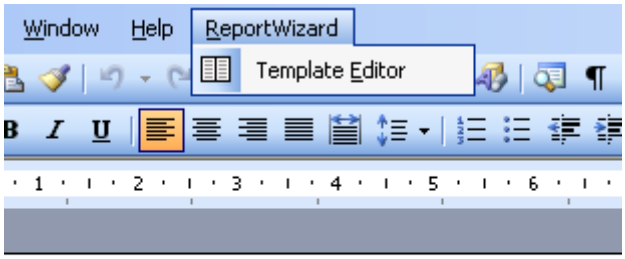
---

1. Download and install Microsoft XML Services (MSXML) from the following URL (You can skip this step if the latest version of MSXML is already installed in your system): http://www.microsoft.com/downloads/details.aspx?FamilyID=d21c292c-368b-4ce1-9dab-3e9827b70604&DisplayLang=en
2. Copy all files and subdirectories from the *<install.dir>\plugins\com.nomagic.magicdraw.reportwizard\vba* folder to *%APPDATA%\Microsoft\Word\STARTUP* (%APPDATA% is the location where user data is stored). For example,
   - On Windows 2000 and Windows XP: *C:\Documents and Settings\User Name\Application Data\Microsoft\Word\STARTUP*
   - On Windows Vista: *C:\Users\UserName\AppData\Roaming\Microsoft\Word\STARTUP*
   - On Windows 7 and 8: *C:\Users\UserName\AppData\Roaming\Microsoft\Word\STARTUP*
3. Restart Microsoft Word.

Opening template editor

After installation has been completed, the Template Editor menu will appear on the Microsoft Word menu bar, see figure below.

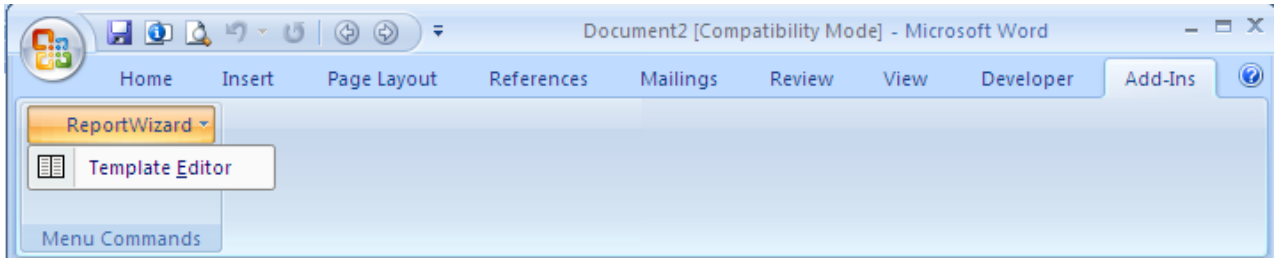
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Microsoft Word 2003 Template Editor Menu.

How to open Template Editor, either

- On the Microsoft Word 2000 – 2003 menu, click **ReportWizard** > **Template Editor**, see figure above, or
- On the Microsoft Word 2007 menu, click **Add-Ins** > **ReportWizard** > **Template Editor**, see figure below.

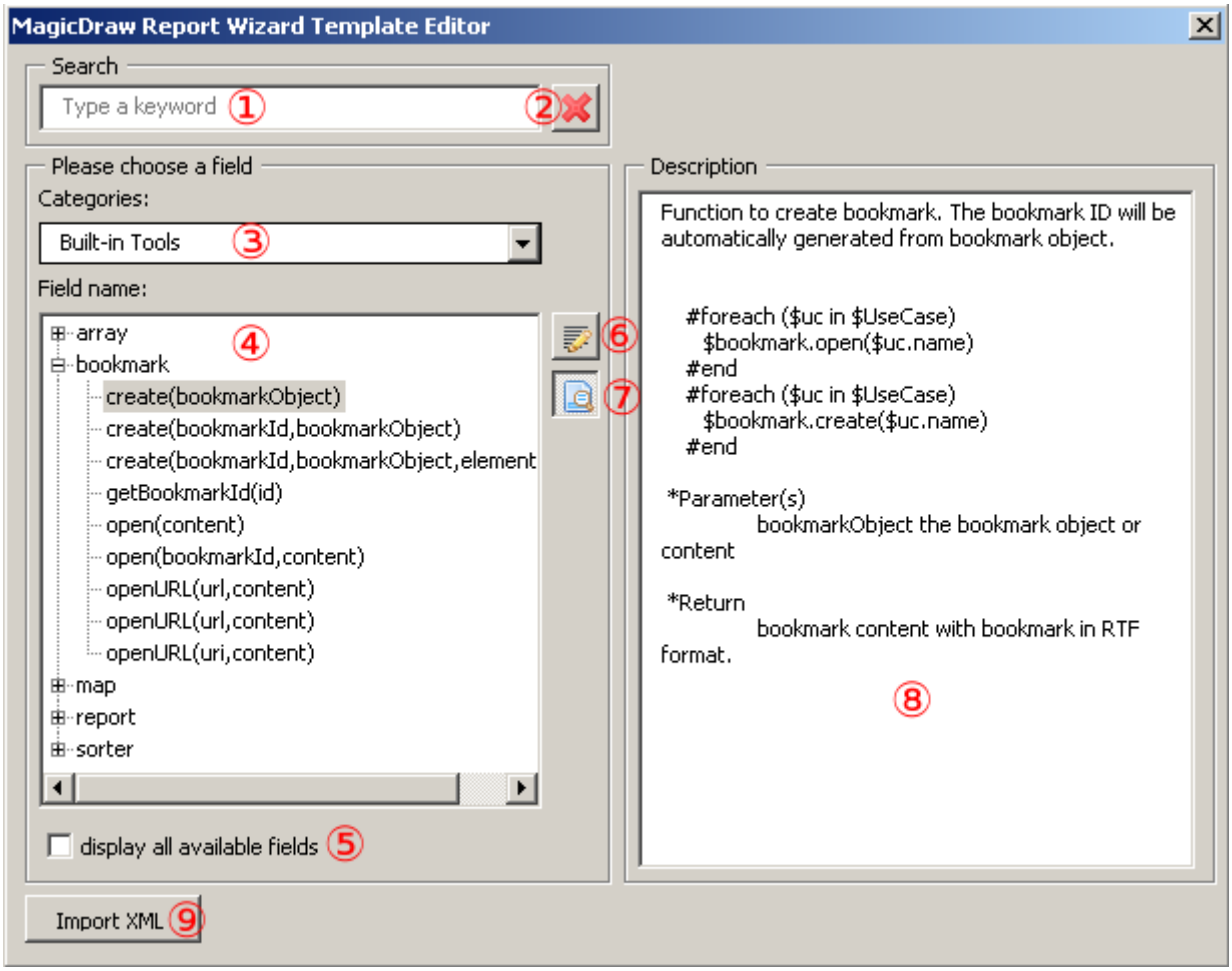


Microsoft Word 2007 Template Editor Menu.

> ⓘ The macro-enabled option in Microsoft Word is required to open Template Editor.

When Template Editor is open, the Report Wizard Template Editor dialog will open.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The Report Wizard Template Editor Dialog.

The table below shows the Report Wizard Template Editor Options

Name	Function
Search box	Filter a list of fields. Only fields that contain a keyword of search as part of their names can be shown in the (4) **List of fields**.
Clear search results	Clear the current search result.
Categories combo box	Select categories of fields. Fields are shown in (4) **List of fields** according to their categories.
List of fields	Show a list of fields. Double-click a field name to insert the code.

Name	Function
Display all available fields check box	Show all fields, otherwise it will show only commonly used fields.
Insert button	Insert a code for a selected field into the document.
Show/Hide description button	Show or hide (8) **description pane**.
Description pane	Show the description of a selected field.
Import button	Import a data file.

Data file

The Template Editor allows you to create a new data file and import it to the Report Wizard Template Editor dialog. Click the Import button to import the data file. The data file is written in the *XML* format. The following is the example of an *XML* data file schema, see the code snippet below.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/
XMLSchema">
  <xs:element name="data">
   <xs:complexType>
    <xs:sequence>
     <xs:element minOccurs="0" maxOccurs="1" name="elements">
      <xs:complexType>
       <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="unbounded" name="element">
         <xs:complexType>
          <xs:all>
           <xs:element minOccurs="1" maxOccurs="1" name="name" type="xs:string" />
           <xs:element minOccurs="0" maxOccurs="1" name="description"
type="xs:string" />
           <xs:element minOccurs="0" maxOccurs="1" name="members">
            <xs:complexType>
             <xs:sequence>
             <xs:element minOccurs="0" maxOccurs="unbounded" name="function">
              <xs:complexType>
               <xs:sequence>
                <xs:element minOccurs="1" maxOccurs="1" name="name"
type="xs:string" />
                <xs:element minOccurs="0" maxOccurs="1" name="description"
type="xs:string" />
                <xs:element minOccurs="0" maxOccurs="unbounded" name="param">
                 <xs:complexType>
                  <xs:all>
                   <xs:element minOccurs="1" maxOccurs="1" name="name"
type="xs:string" />
                   <xs:element minOccurs="0" maxOccurs="1" name="description"
type="xs:string" />
                   <xs:element minOccurs="0" maxOccurs="1" name="type"
nillable="true" type="xs:string" />
                   <xs:element minOccurs="0" maxOccurs="1" name="direction">
                    <xs:simpleType>
                     <xs:restriction base="xs:string">
                      <xs:enumeration value="in" />
                      <xs:enumeration value="out" />
                     </xs:restriction>
                    </xs:simpleType>
                   </xs:element>
                  </xs:all>
                 </xs:complexType>
                </xs:element>
               </xs:sequence>
              </xs:complexType>
             </xs:element>
              <xs:element minOccurs="0" maxOccurs="unbounded" name="attribute">
               <xs:complexType>
                <xs:all>
                 <xs:element minOccurs="1" maxOccurs="1" name="name"
type="xs:string" />
                 <xs:element minOccurs="0" maxOccurs="1" name="description"
type="xs:string" />
```

```
                  <xs:element minOccurs="0" maxOccurs="1" name="type" nillable="true"
type="xs:string" />
                 </xs:all>
                </xs:complexType>
               </xs:element>
              </xs:sequence>
             </xs:complexType>
            </xs:element>
           </xs:all>
           <xs:attribute name="often" type="xs:boolean" />
          </xs:complexType>
         </xs:element>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" />
   </xs:complexType>
  </xs:element>
</xs:schema>
```

The table below lists the XML Data File Schema Elements and Their Descriptions

Element	Attribute / Element	Description
data		A root element.
	name : string	A data name. This value will be used as a category name in (3) Category.
	elements : complexType	A group of <element> elements.
elements		A group of <element> elements.
element		An element display on (4) List of fields.
	name : string	An element's name.
	description : string	An element description.
	often : boolean	If an attribute is true, the element will be managed as a suggested element.
	members : complexType	A group of <function> or <attribute>.
function		A function member.

Element	Attribute / Element	Description
	name : string	A function's name.
	description : string	A function description.
	param : complexType	A group of function parameters.
param		A function parameter.
	name : string	A parameter's name.
	description : string	A parameter description.
	type : string	A parameter type.
	direction : simpleType	The direction of a parameter: • in indicates this parameter is an input. • out indicates this parameter is an output.
attribute		An attribute member.
	name : string	An attribute's name.
	description : string	An attribute description.
	type : string	An attribute type.

The following code snippet is an example of an XML data file.

```
<?xml version="1.0" encoding="UTF-8"?>
<data name="Built-in Tools" xmlns:xsi="http://www.w3.org/2001/XMLSchemainstance"
xsi:noNamespaceSchemaLocation="fields.xsd">
  <elements>
    <element often="true">
      <name>array</name>
      <description>Use for creating the Array or HashSet instance.</description>
      <members>
        <function>
          <name>subList</name>
          <description>Create ArrayList of the portion of list in given size.</
description>
          <param>
            <name>list</name>
            <direction>in</direction>
            <type>List</type>
            <description>a original list.</description>
          </param>
          <param>
            <name>size</name>
            <direction>in</direction>
            <type>int</type>
            <description>view size of given list</description>
          </param>
          <param>
            <name>list</name>
            <direction>out</direction>
            <type/>
            <description>a of view of the specified size within given list.</
description>
          </param>
        </function>
      </members>
    </element>
  </elements>
</data>
```

Generating Reports from Report Wizard

In addition to default document templates, Report Wizard allows you to create customized specification document templates. Templates may contain your own custom fields not related to model elements, as well as fields that correspond to model elements. Once customized, a Report Wizard template can be used with data from any project.

You can also format your template to create the style you want including tables of contents, headers and footers, and page numbers. You can apply most character and paragraph formatting available from your rtf editor for the rtf template or specify with html tags in the HTML template, including keywords. Once the template has been completed, you can run Report Wizard to create a report in a format that corresponds to the template.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Report Wizard templates can be in *txt, rtf, html, odt, odf, odp, docx, xlsx, pptx*, and *xml* for DocBook or FO files. If you prefer, you can work in Report Wizard Template Editor until you have finished the template file and use the Report Wizard template windows to save it in the template folder. The default templates are located in the *<MagicDraw home>/plugins/com.nomagic.magicdraw.reportwizard/data* folder.

> All built-in rtf report templates are combined into the RTF (depreciated) report template category.

Related pages

Concepts

Report Wizard includes the following concepts:

- **Template**
  specifies the document layout, format, corresponding model elements, and custom defined fields.
- **Report Data**
  specifies a set of custom fields and data in the selected template. A template can have a number of different custom
  defined fields for a project or different projects organized in a report.
- **Report**
  a generated user document with data from the project displayed in the selected template style.

Default templates

There are 12 default templates that come with the report engine.

1. **Class Specification Report**
   describes the specification of class, interface, and enumeration type of the project.
2. **Data Dictionary Report**
   enables you to print elements of MagicDraw data dictionary.
3. **IEEE 1233**
   the standard template based on the IEEE 1233 recommendation.
4. **Model Extension**
   describes common UML model extension mechanisms (stereotypes, tagged values, and constraints). It is useful for reporting custom UML profiles.
5. **Use Case (Simple)**
   describes system functionalities and actors in a simple format.
6. **Use Case (Modern)**
   describes system functionalities and actors that are useful in support of use-case driven analysis.
7. **Web Publisher (Simple HTML)**
   enables you to publish MagicDraw models (including diagrams) in simple HTML. The report is viewable using a standard browser such as Internet Explorer or Firefox.
8. **Web Publisher 2.0**
   enables you to publish MagicDraw models (including diagrams) in HTML with rich features. The report is viewable using a standard browser such as Internet Explorer or Firefox.
9. **Activity Diagram**
   provides a standard report for activity diagrams. The content of the report is a list of all the nodes (ordered by node hierarchy in each activity diagram).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

10. **Activity Diagram Specification**
    provides a standard report for activities. The content of the report is a list of all the nodes in each activity.
11. **Sequence Diagram**
    provides a report that shows sequence diagrams and a list of all the messages in the diagrams.
12. **Sequence Diagram Specification**
    provides a report that shows sequence diagrams, and a list of all the messages and lifelines in the diagrams.

Architecture templates

There are five architecture report templates that come with the report engine:

- Behavioral Report(see page 1736)
- Environment Report(see page 1736)
- Implementation Report(see page 1736)
- Structural Report(see page 1736)
- Use Case Report(see page 1737)

Behavioral Report

A Behavioral report is a standard report for the Behavioral view. This report will be generated by a selected package that consists of Sequence, Communication, State Machine, and Activity diagrams. The report lists all packages in a project and arranges them hierarchically. Each package consists of the specification of messages in the Sequence diagram, the specification of lifelines in the Communication diagram, the specification of states in the State Machine diagram, and the actions in the Activity diagram. All elements in the report are ordered by their names.

Environment Report

An Environment report is a standard report for the Environment view. The report will be generated by a selected package that consists of an Implementation diagram (Deployment diagram). The report lists all packages in a project and arranges them hierarchically. Each package describes the specification of nodes, interfaces, components, and artifacts which are ordered by the element's name.

Implementation Report

An Implementation report is a standard report for the Implementation view. The report will be generated by a selected package that consists of an Implementation diagram (Component diagram). The report lists all packages in a project and arranges them hierarchically. Each package consists of a list of interfaces, components, and artifacts. All elements in the report are ordered by their names.

Structural Report

A Structural report is a standard report for the Structural view. This report will be generated by a selected package that consists of a Class diagram. The report lists all packages in a project and arranges them hierarchically. Each package describes the specification of interfaces, classes, and enumerations which are ordered by the element's name.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Use Case Report

A Use Case report is a standard report for the Use Case view. This report will be generated by a selected package that consists of Use Case diagrams. The report lists all packages in a project and arranges them hierarchically. Each package describes the specification of actors and use case. All elements in the report are ordered by their names.

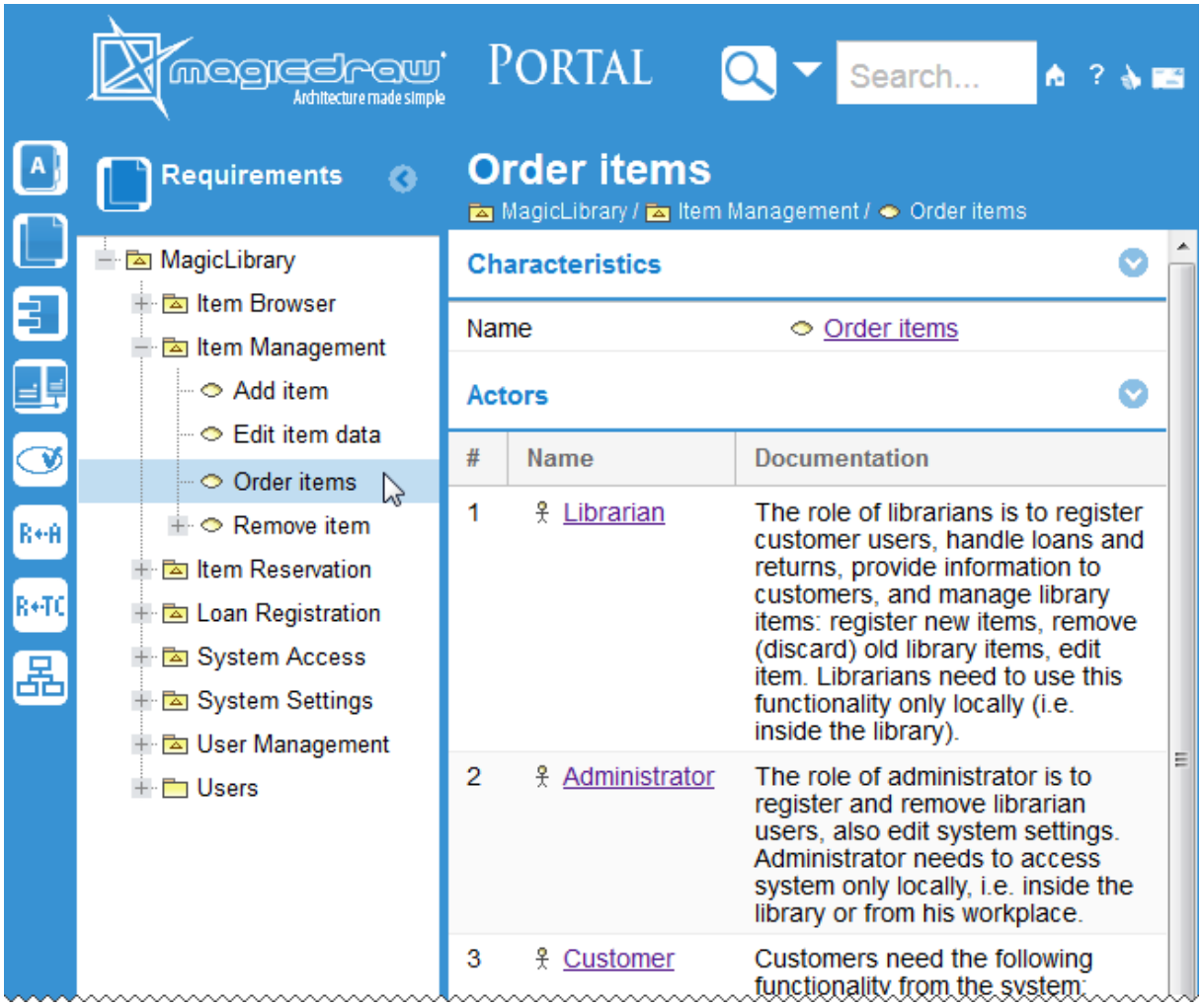
Web portal reports

Report Wizard provides a report template that you can use to generate a web portal report. A web portal report is a report published in HTML format so it can be viewed in a web browser. With the web portal report, you can share your project and virtually make your models readable to the other stakeholders, including those who do not understand MagicDraw models. The web portal report allows the readers to provide feedback on any element or diagram in the model by commenting on it. This web portal report template is only available in MagicDraw Enterprise edition. In other editions, MagicDraw provides a demo version of the web report template, which can contain, at most, 10 symbols per view.

A web portal report simplifies the complexity of a model by reducing less needed elements from the model tree and displays the most relevant views of your project. For example, a MagicDraw project report includes views such as Glossary, Requirements, Architecture, Implementation, and so forth, and a SysML Plugin project report includes views such as Requirements, Structure, Interfaces, Constraints, and so forth. It helps you generate your project in a user-friendly and easy-to-browse environment.

The web portal report features a quick search to help you find elements and diagrams for faster content viewing.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



An Example of Web Portal Report - Software Engineering Portal.

Generating reports

Generating a web portal report is as easy as generating any other reports with Report Wizard.

To generate a web portal report from a MagicDraw project

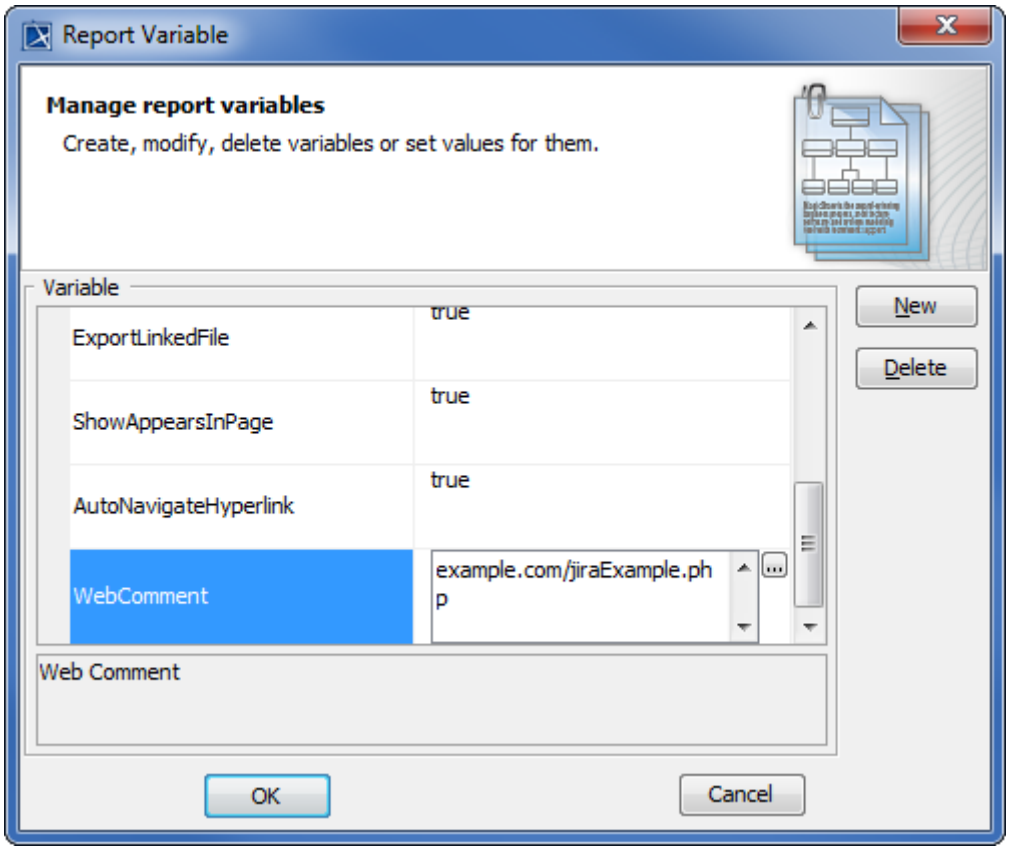
1. Press **Ctrl+Shift+G** to open the **Report Wizard** dialog.
2. In the **Select Template** pane, expand the **Web Reports** package and select **Software Engineering Portal**.

> If your MagicDraw has a plugin that provides another template for generating a web portal report, you can find this template in a separate package. For example, if you have installed SysML Plugin, you should expand the **SysML - Web Reports** package and select **SysML - Web Reports**.

3. Click **Next**.

4. If you need to allow comments on the report, do the following:

   - Click the **Variable** button. The **Report Variable** dialog will open.
   - In the value cell of the **WebComment** variable, type the global address for comments delivery, for example, example.com/jiraExample.php[340], see figure below.
   - Click OK.



Enabling Feedback on a Web Portal Report.

5. If you want ScreenTips to appear on the report when it opens for the first time, do the following:

   - Click the **Variable** button. The **Report Variable** dialog will open.
   - In the value cell of the **ShowTip** variable, type either True or true.
   - Click **OK**.

6. Click **Next**.

7. Select the elements you need to include in the report and click **Next**. For more information, refer to **Select Element Scope Pane**.

9. Save your report in a specific location, type the report name (for example, report.html), and specify other output options if desired.

10. Click **Generate**.

340 http://example.com/jiraExample.php

As a web portal report includes a number of files and folders, it is strongly recommended that you save the report in a separate folder. For more information, refer to **Generate Output Pane**.

11. You can open the report by double-clicking the HTML/HTM file in the location specified in step 6 (in this example, report.html).

12. You can place the report in a shared directory or on a web server to make it available for the other stakeholders.

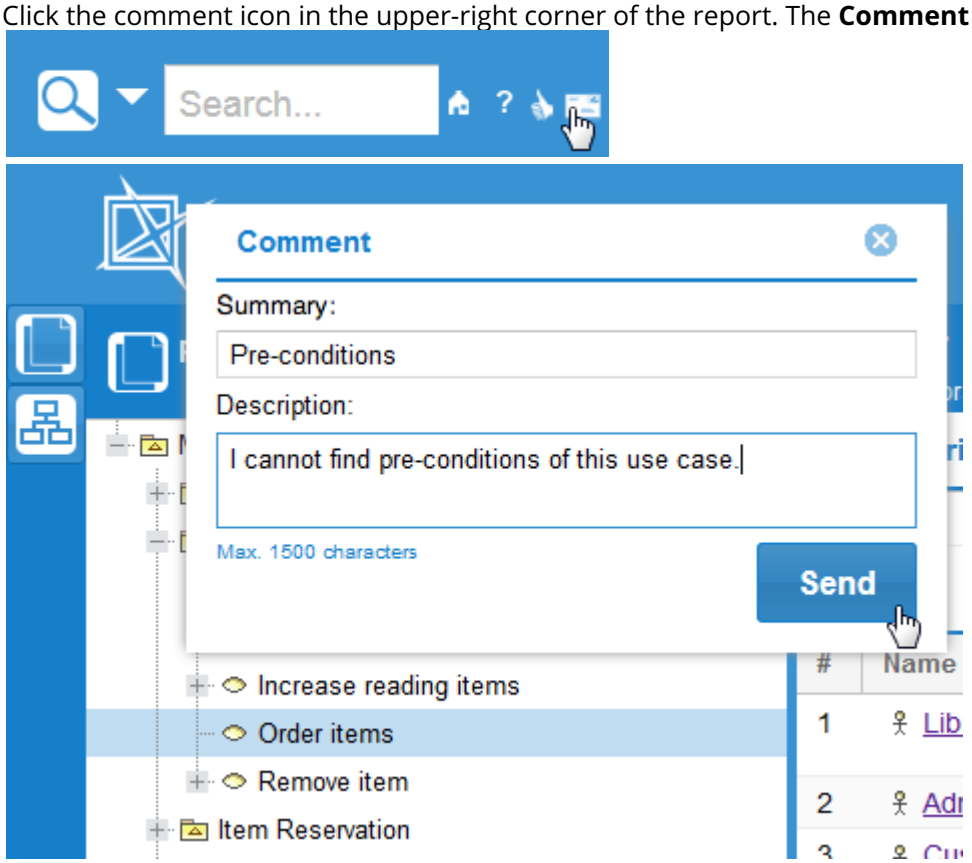
Commenting on a report

If a web portal report enables feedback or comments, you can write something about a model element or a diagram in the report. See step 4 in Generating Reports(see page 1734) for more information on how to enable feedback or comments in a web portal report.

To write a comment on a web portal report

---

1. Select an element or diagram you need to comment in the model tree view on the left-hand side of the web portal report.
2. Click the comment icon in the upper-right corner of the report. The **Comment** dialog opens.



Commenting on a Web Portal Report.

3. Type a short statement that describes your comment or feedback in the Summary box.
4. Type your comment in details in the Description box.

> IMPORTANT! Your comments in the **Description** box should not exceed 1500 characters. Comments exceeding the limit will not be included.

5. Click **Send**. The global address of the selected element or diagram, the name of that element or diagram, and the contents in both the **Summary** and **Description** boxes will be sent to the web server.

Reading comments

There are several ways to deliver comments on elements and diagrams in a web portal report and allow others to read them. You can choose to deliver them through an issue tracking system, such as JIRA, or an e-mail platform, such as Gmail or Outlook.

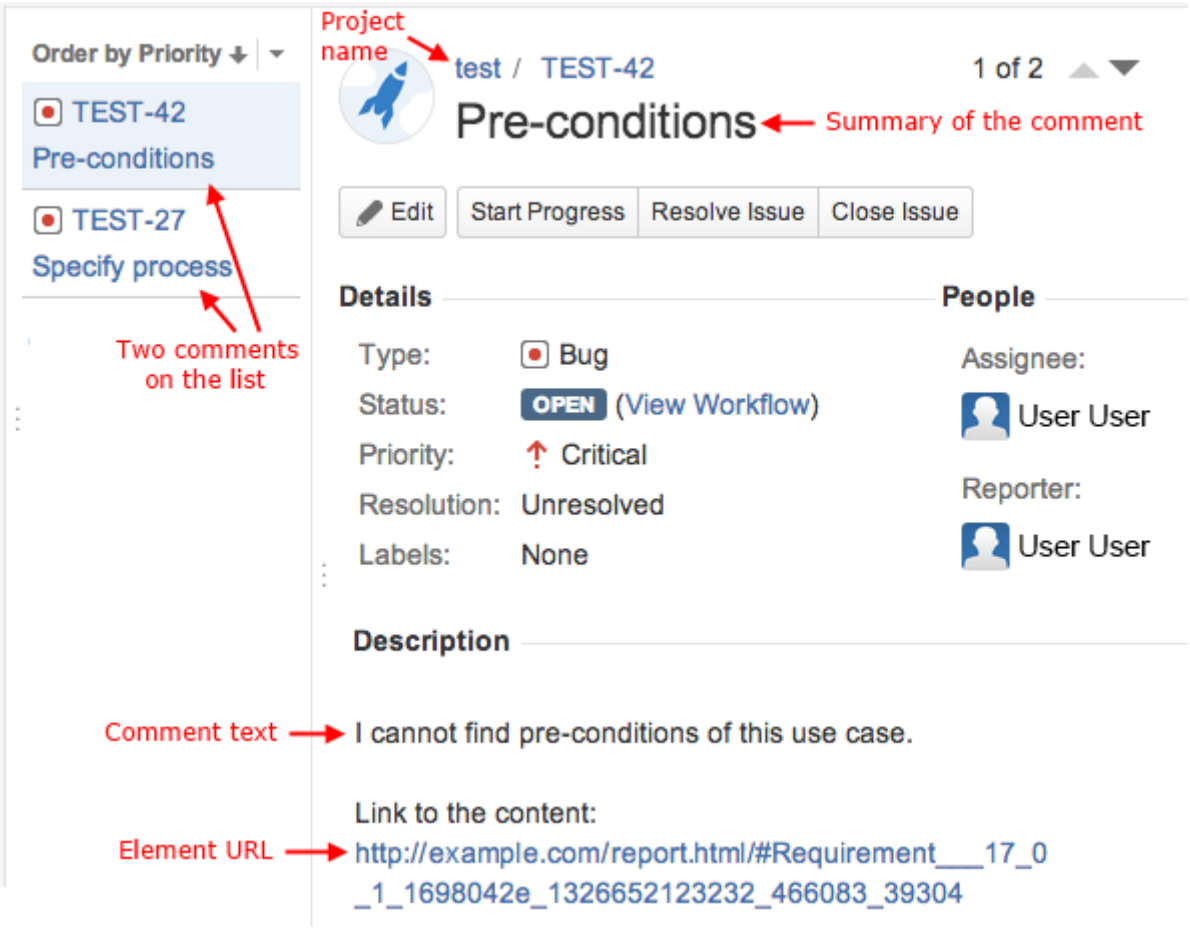
The following example shows how to deliver comments, if you need to get the comments to a certain account on JIRA as issues of the type 'Bug':

1. Copy the following script:

```
<?php
$contentUrl = $_GET['url'];                            // Global address of the
commented
                                                       // element or diagram.
$contentTitle = $_GET['contentTitle'];                 // Name of the commented
element
                                                       // or diagram.
$contentSummary = $_GET['commentSummary'];             // Summary text of the
comment.
$contentCommentArray = $_GET['comment'];               // Text of the comment.
$contentComment = implode(" ", $contentCommentArray);
$description = $contentTitle . " \\\\ " . $contentComment . " \\\\ " . "Link to
the content: \\\\" . $contentUrl;
$issue = array(
    'type'=>'1',                                        // Type of JIRA issue

'project'=>'TEST',                                                          //
that stores the comment.
                                                        // '1' means 'Bug'.
                                                       // Name of JIRA project
                                                        // that stores the
comments (3.1).
    'description'=> $description,                       // Description contents
    'summary'=> $contentSummary,                        // of JIRA issue that
    'priority'=>'2',                                    // stores the comment.
    'assignee'=>'username',                             // Summary text of JIRA
issue
    'reporter'=>'username',                             // that stores the
comment.
                                                        // '2' means
'Critical'.
                                     // Username of JIRA account,
                                                        // to which the comment
is sent (3.2).
                                  // Username of JIRA account,
                                                        // which sends the
comment (3.3).
);
$soapClient = new SoapClient("https://example.com/rpc/soap/jirasoapservice-v2?
wsdl");                                     // (3.4)
$token = $soapClient->login('username',
'password');
 // (3.5)
$soapClient->createIssue($token, $issue);
?>
```

2. Create a new file and paste the script onto the file.
3. Change the following values (the red ones in the script):
   a. The name of a JIRA project.
   b. The assignee's username.
   c. The reporter's username.
   d. The global address of a web service, which provides JIRA.
   e. The username and password of the JIRA account wherein a new issue is created from the comment data.
4. Save the file as jiraExample.php.

5. Upload the file to your web server, for example, Apache Tomcat.
6. When enabling comments in your web portal report (Step 4 in Generating Reports(see page 1734)), specify <web server address>/jiraExample.php (for example, example.com/jiraExample.php[341]) as the value of the WebComment variable.



The Representation of Web Portal Report Comments on JIRA (a Fragment of JIRA Window).

Adding variables into an output report filename

You can use the following variables and include them in an output report filename.

- Date variable
- Template variable
- Random number
- Project attribute

Date variable

A date variable is a variable for the current date and time. (See section **$date** in the previous chapter). The date variable consists of:

- A date variable with the default format.
- A date variable with a customized date format.

341 http://example.com/jiraExample.php

Date variables with default format

Using a date variable without an attribute causes it to format the current date and time into a default format.

The syntax: **${date}**

For example:

```
C:\OutputReport\index_${date}.html
```

The output report filename would be "*index_2011-12-21 11.55.44.html*".

Date variable with custom format

Using a date variable with an attribute causes it to format the current date and time into a specific format.

The syntax: **${date.get('format')}** or **${date.get("format")}**

For example:

```
C:\OutputReport\index_${date.get('yyyyMMdd-hhmmss')}.html
```

The output filename would be "*index_20111221-115544.html*".

Template variable

A template variable is a variable for a template that is created in Report Wizard. It allows you to add all template variables into an output report filename (see Including Variables in a Template(see page 1743)).

Including template variables in an output report filename

When you include template variables created in an output report filename, the output filename which consists of the variables will be converted into value. This section uses the following template variables as examples, see figure below.

⊟ Parent	parent value
Child	child value

Adding a value of a top-level variable

You can retrieve a top-level variable in an output report filename by using the following syntax.

The syntax: **${Parent}**

For example:

```
C:\OutputReport\index_${Parent}.html
```

The report output filename would be "*index_parent value.html*".

Adding a value of a child variable

You can retrieve any of the following child variable values in an output report filename.

1. Getting a child variable at an index position:
   The syntax: **${Parent.get(0)}**
2. Getting a child variable by child names:
   The syntax: **${Parent.get('Child')} or ${Parent.get("Child")}**
3. Getting a child value by referencing its name (in this case "Child"):
   The syntax: **${Parent.Child}**

For example:

```
C:\OutputReport\index_${Parent.get('Child')}.html
```

The output report filename would be "*index_child value.html*".

Random number

Report Wizard supports only pseudorandom numbers in an output report filename. The random number consists of:

- A random number with the default distributed int value.
- A random number with a distributed int value between 0 (inclusive) and specified value (exclusive).

Random number with the default distributed integer value

You can use Report Wizard to generate pseudorandom numbers with the default distributed integer value.

The syntax: ${random}

For example:

```
C:\OutputReport\index_${random}.html
```

The output report filename would be "*index_21547.html*".

Random number with a distributed integer value between 0 (inclusive) and specified value (exclusive)

You can use Report Wizard to generate pseudorandom numbers with the default distributed integer value between 0 and a specific value.

The syntax: ${random.nextInt(n)}

The Parameters:

"n" - the bound on the random number to be returned. The "n" value must be a positive integer.

For example:

```
C:\OutputReport\index_${random.nextInt(1000)}.html
```

The output report filename would be "*index_457.html*".

Project attribute

You can add a project attribute or information about a project into an output report filename.

The project attribute consists of:

- Project name
- Teamwork version

Project name

You can retrieve the name of a project by using the following syntax.

The syntax: ${project.name}

For example:

```
C:\OutputReport\index_${project.name}.html
```

The output report filename would be "*index_myProject.html*".

Teamwork version

You can retrieve a teamwork version of a project by using the following syntax.

The syntax: ${project.version}

For example:

```
C:\OutputReport\index_${project.version}.html
```

The output report filename would be "*index_1234.html*".

Exception flow

You may accidentally specify an invalid output report filename such as:

- An invalid syntax variable.

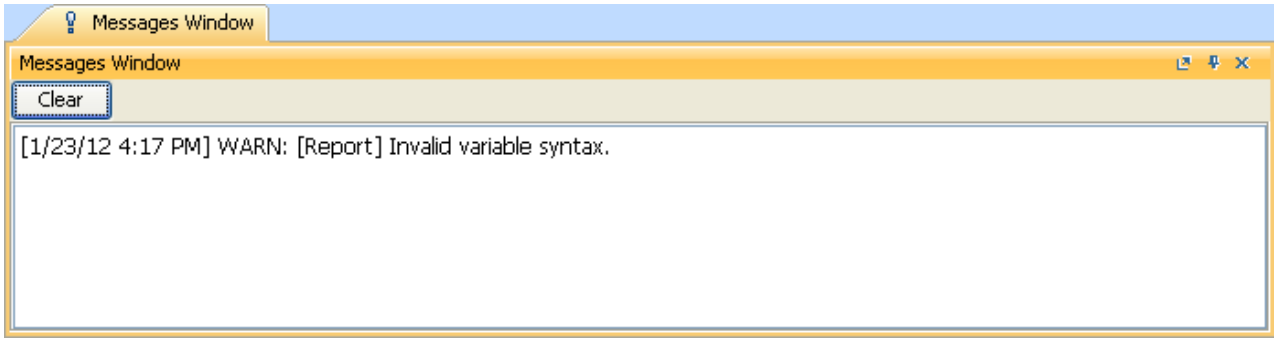
- A variable that does not exist in a template variable.
- An invalid parameter.

If this happens, Report Wizard will use this variable as a filename and a warning message will open in a MagicDraw log file (*\.magicdraw\version\md.log*).

For example:

```
C:\OutputReport\index_${date.get('yyyyMMdd-hhmmss)}.html
```

A single quote after “yyyyMMdd-hhmmss” was missing. Therefore, the output filename would be “index_${date.get('yyyyMMdd-hhmmss)}.html” and a warning message would open in MagicDraw, see figure below.

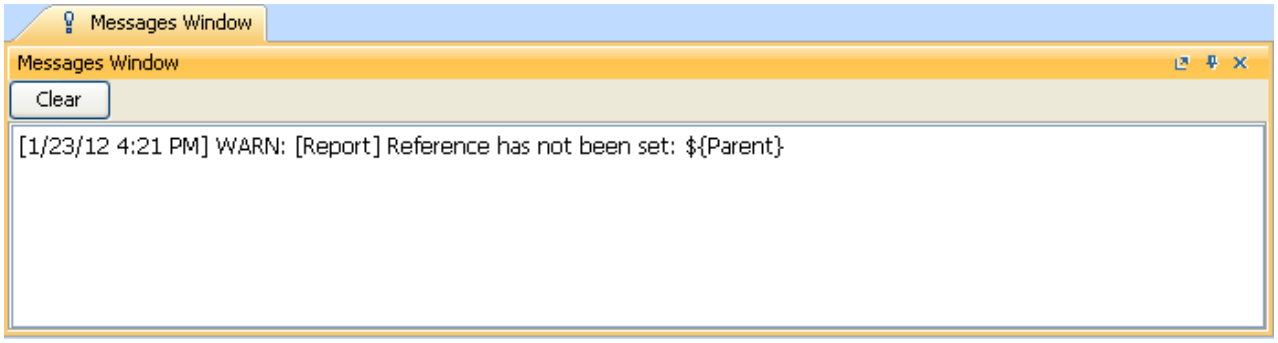


A Warning Message of an Invalid Variable Syntax.

For example:

```
C:\OutputReport\index_${Parent}.html
```

“${Parent}” was not in the report template variable. Therefore, the output filename would be "index_${Parent}.html" and a warning message would open in MagicDraw, see figure below.

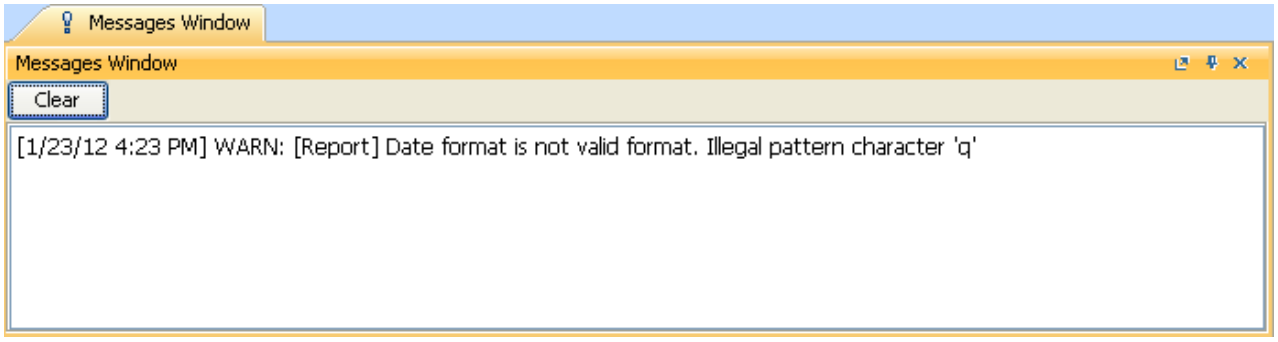


A Warning Message of a Variable that Does not Exist in a Template Variable.

For example:

```
C:\OutputReport\index_${date.get('yyyy-MM-dd qq')}.html
```

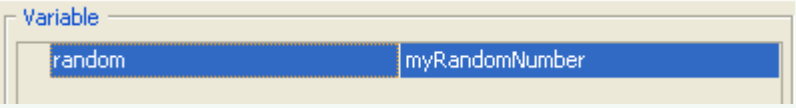
In this example, ‘yyyy-MM-dd qq’ is an invalid parameter. Therefore, output filename would be “index_${date.get('yyyy-MM-dd qq').html” and a warning message would open in MagicDraw, see figure below.



A Warning Message of an Invalid Parameter.

- All variables should be able to be combined together. For example:
  C:\OutputReport\index_${project.name}_${date}.html
  The output filename would be “*index_myProject_2011-12-21 11.55.44.html*”.
- The template variable is the highest priority of all other variables. For example:
  C:\OutputReport\index_${random}.html

If you specify a template variable whose name is at random in the Report Data, the variable value will be used as the filename.



The output filename would be “*index_myRandomNumber.html*”.
If you do not specify a template variable whose name is at random in the Report Data, the random number with the default distributed integer value will be used as the filename.
The output filename would be “*index_12345.html*”.

FAQs

To generate an output report document from Report Wizard

1. On the **Tools** menu, click **Report Wizard**. The **Report Wizard** dialog will open.
2. Select the template or create a new one. Click **Next**.
3. Specify Report Data and add information for the variables. Click **Next**.
4. Select the document scope from the corresponding packages. Click **Next**.
5. Select the output location, images format, and text for blank fields.
6. Click **Generate**.

To add a new report template

1. On the **Tools** menu, click **Report Wizard**. The **Report Wizard** dialog will open.
2. Click **New**. The **New** dialog will open.
3. Enter the template name and description. Click the ... button to specify the template file location.
4. Click **Create**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Once a new template has been created, Report Wizard will add a folder with the entered template name and save the selected template in this folder.

To remove a template

1. On the **Tools** menu, click **Report Wizard**. The **Report Wizard** dialog will open.
2. Select a template from the list and click **Delete**.

Once removed, the selected template with its folder and reports cannot be recovered.

How to modify a template file

1. On the **Tools** menu, click **Report Wizard**. The **Report Wizard** dialog will open.
2. Select a template from the list and click **Open**. The default editor and a template file for editing will open.
3. Modify the template and perform the save command in the editor.

A different editor will be used for each template format. For example, MS Word could be used for **.rtf* template modification, or Macromedia Dreamweaver could be used for **.html* template editing.

To add a **Report Data** into the template

1. On the **Tools** menu, click **Report Wizard**. The **Report Wizard** dialog will open.
2. Select a template or create a new one. Click **Next**. The **Select Report Data** pane will appear.
3. Click **New**. The **New** dialog will appear.
4. Enter the Report Data **name** and **description**. Click **Create**. A new **Report Data** will be created. In the next step, you may add new fields or delete the existing ones.

To remove **Report Data** from the template

1. On the **Tools** menu, click **Report Wizard**. The **Report Wizard** dialog will open.
2. Select a template or create a new one. Click **Next**. The **Select Report Data** pane will appear.
3. Select the **Report Data** from the list and click **Delete**.

To set the default viewer option for the report file

1. On the **Tools** menu, click **Report Wizard**. The **Report Wizard** dialog will open.
2. Select a template and **Report Data** and specify the variables and package scope. Click the **Next** button to proceed.
3. At the last wizard step, select the **Display in viewer** after generating report check box. The report output will be displayed in the default editor or browser.

To change an output image format

1. On the **Tools** menu, click **Report Wizard**. The **Report Wizard** dialog will open.
2. Select a template and **Report Data** and specify the variables and package scope. Click the **Next** button to proceed.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

3. At the last wizard step, select the output image format from the **Report Image Format** box.

> The supported image formats include:
> ***.PNG*** and ***.JPG***
> for all supported report templates.
> ***.SVG***
> for *HTML, XML* and Text report templates.
> ***.EMF*** and ***.WMF***
> for *RTF, OOXML* and *Text* report templates

To change an empty value configuration

1. On the **Tools** menu, click **Report Wizard**. The **Report Wizard** dialog will open.
2. Select a template and **Report Data** and specify the variables and package scope each time.Click the **Next** button to proceed.
3. At the last wizard step, select an option for output on blank fields:
   - Select **Display empty value** or **Display value as**, and select **NA** or
   - Enter the text to represent other than null value when the template query fields return empty.

To add a new variable

1. On the **Tools** menu, click **Report Wizard**. The **Report Wizard** dialog will open.
2. Select a template and **Report Data**. Click the **Next** button to proceed.
3. Click the **New** button when the **Variable** pane opens. The **Variables** dialog will open.
4. Enter the name of a new variable and its value (the value can be modified later after the variable has been created).
5. Click **Create**.

To delete a variable

1. On the **Tools** menu, click **Report Wizard**. The **Report Wizard** dialog will open.
2. Select a template and **Report Data**. Click the **Next** button to proceed.
3. Select a field and click **Delete** when the **Variable** step opens.

> You cannot recover any deleted **Report Data**.

To modify a variable

1. On the **Tools** menu, click **Report Wizard**. The **Report Wizard** dialog will open.
2. Select a template and **Report Data**. Click the **Next** button to proceed with the steps.
3. Select a field and modify its value when the **Variable** step opens. The value can be modified in the properties list or in the **Field Value** box below the properties list.

To select a package for report generation

1. On the **Tools** menu, click **Report Wizard**. The **Report Wizard** dialog will open.
2. Select a template and **Report Data** and specify the variables. Click the **Next** button to proceed.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

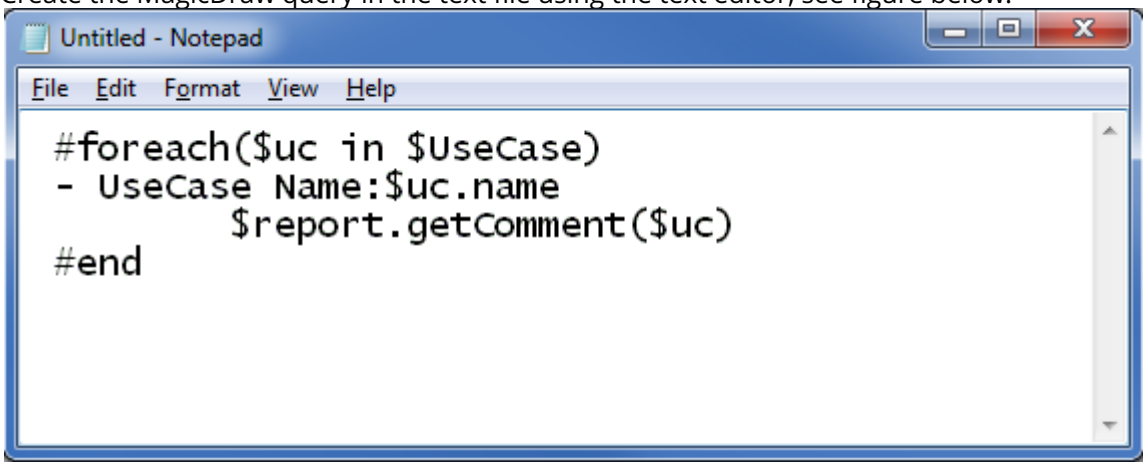
3. The package tree will open. Select the package of the project and click the **Add** button to add the elements from the package to the report.

> Clicking the **Add** button will add only the selected element, not its children, to the report scope. In order to include all children, click the **Add Recursively** button instead.

To generate a normal text output format

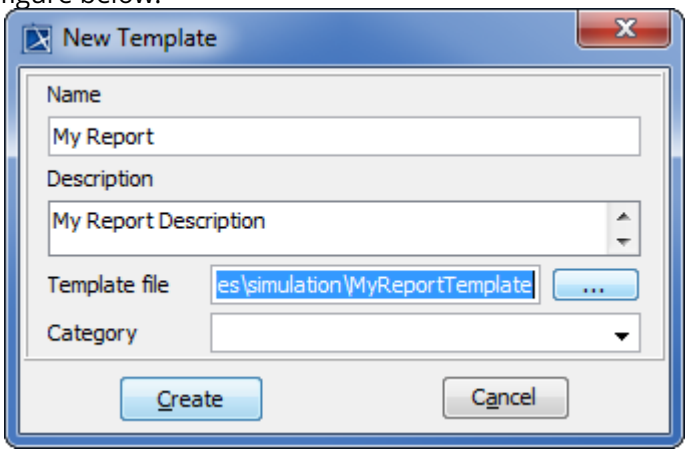
---

1. Create the MagicDraw query in the text file using the text editor, see figure below.



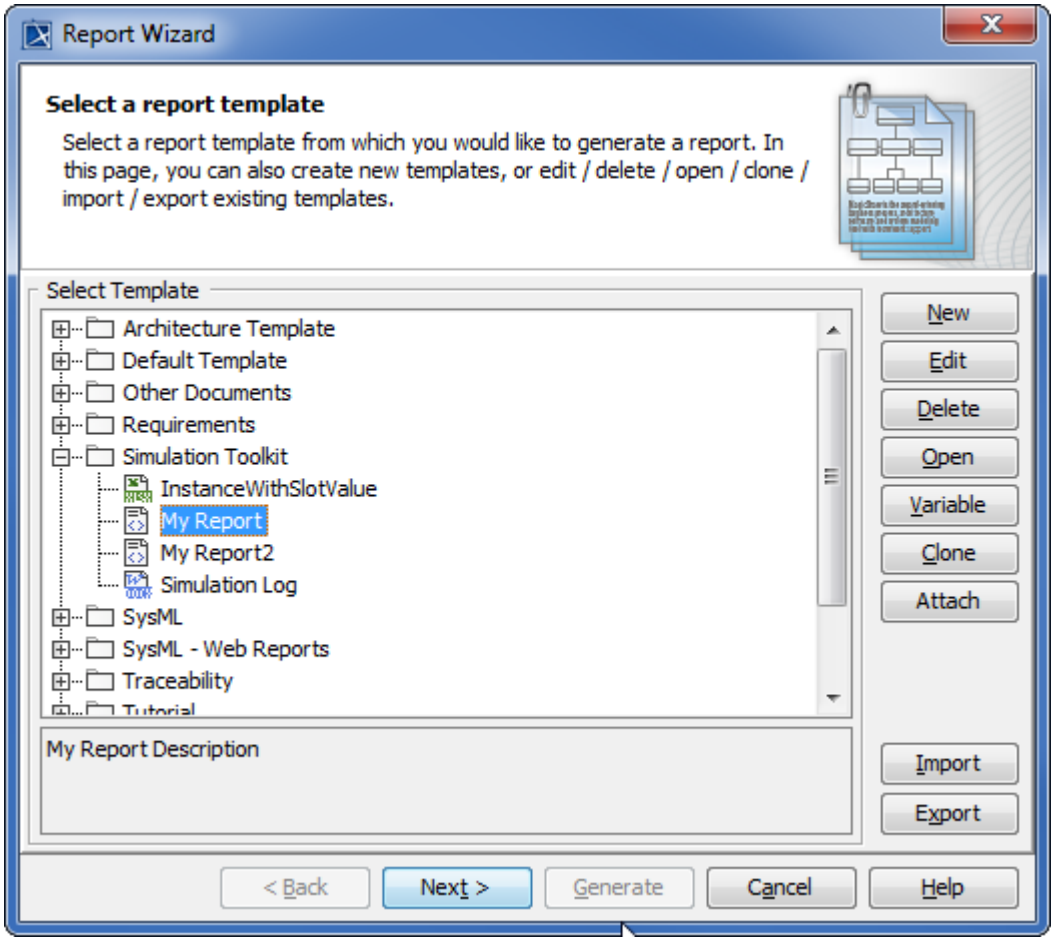
Entering Query in the Normal Text Template.

2. Create a new template in the **Report Wizard** dialog through the **New Template** dialog, see figure below.

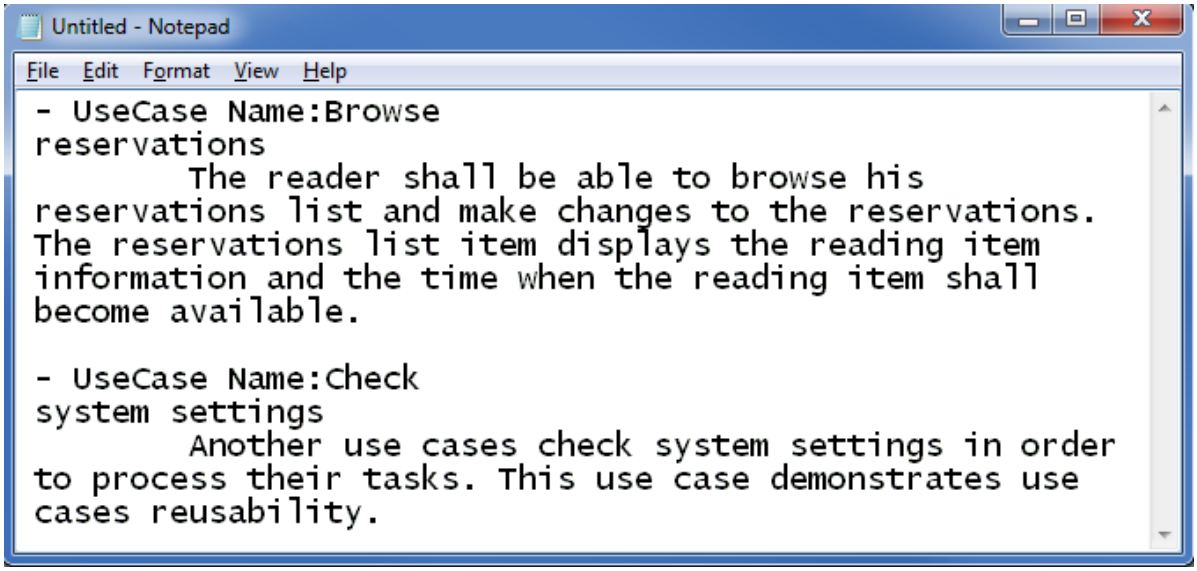


Creating New Text Template.

3. Select Text Template, see first figure below, and generate the output report, see second figure below.



Selecting Text Template to Generate Report Output.



Text Output Report.

Generating Use Case Description Reports

On this page

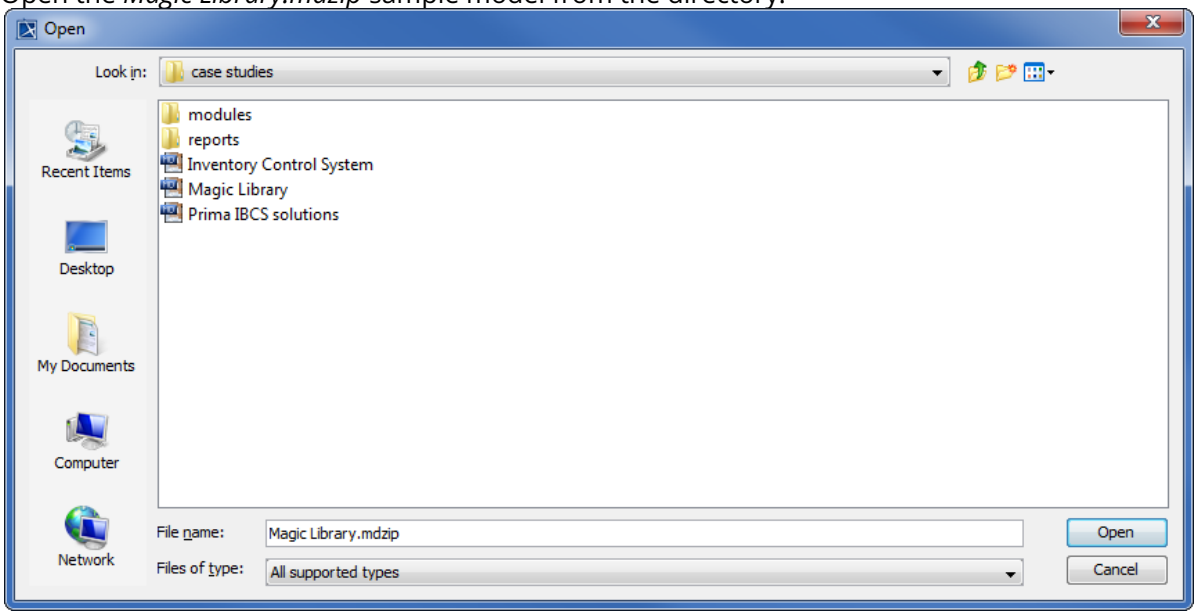
- Use Case Report Template(see page 1753)
- Glossary(see page 1759)
- Sample model(see page 1759)
- More Information on Generating Reports(see page 1759)

Use Case Report Template

Report Wizard provides a Use Case description document template that is designed to capture all Use Case-related artifacts, e.g. Use Cases, actors, Use Case descriptions, in a report. It is used to describe systems functionality and actors in simple format. This template includes all of the Use Case diagram-related keywords. When you select the highest level of the packages (in MagicDraw, Model is the highest level of the packages) as the scope of the report, only Use Case diagrams and their related elements will be added to the document.

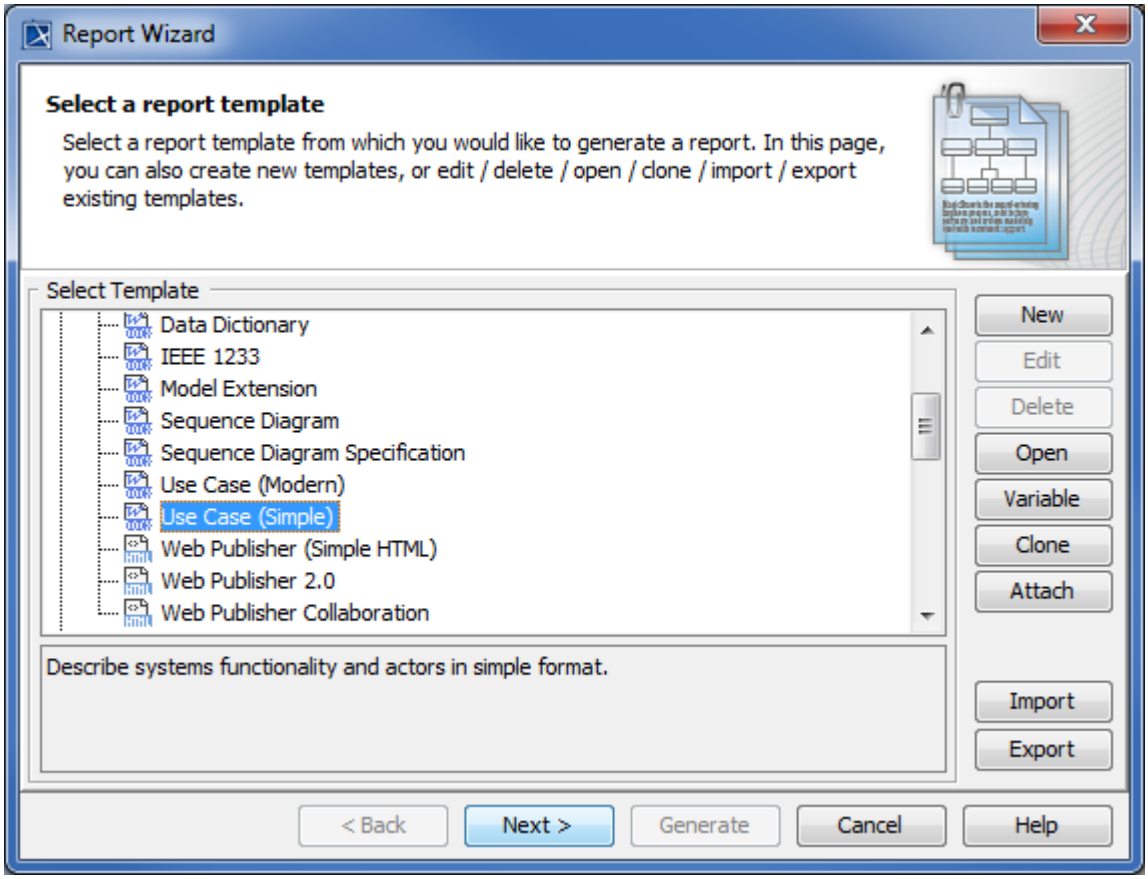
To generate a Use Case description document

1. Open the *Magic Library.mdzip* sample model from the directory.

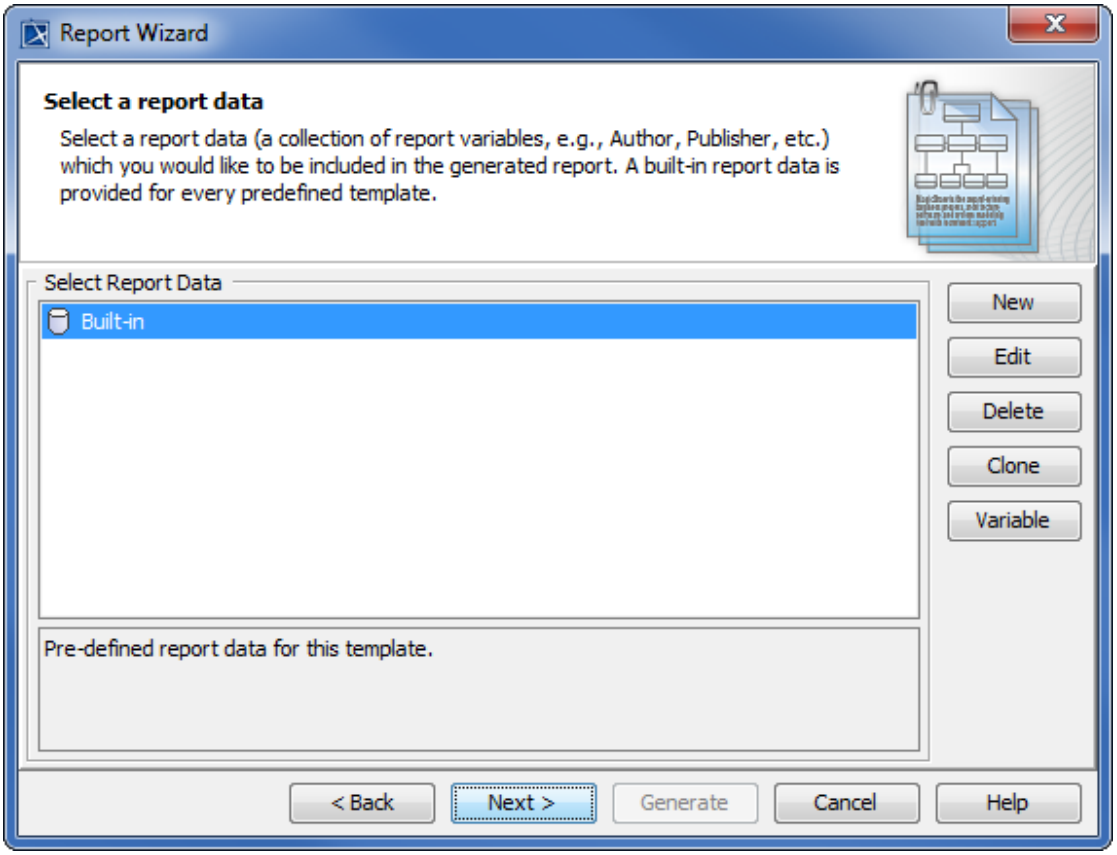


> This step is necessary for illustration purposes, as well as other steps within this scenario.

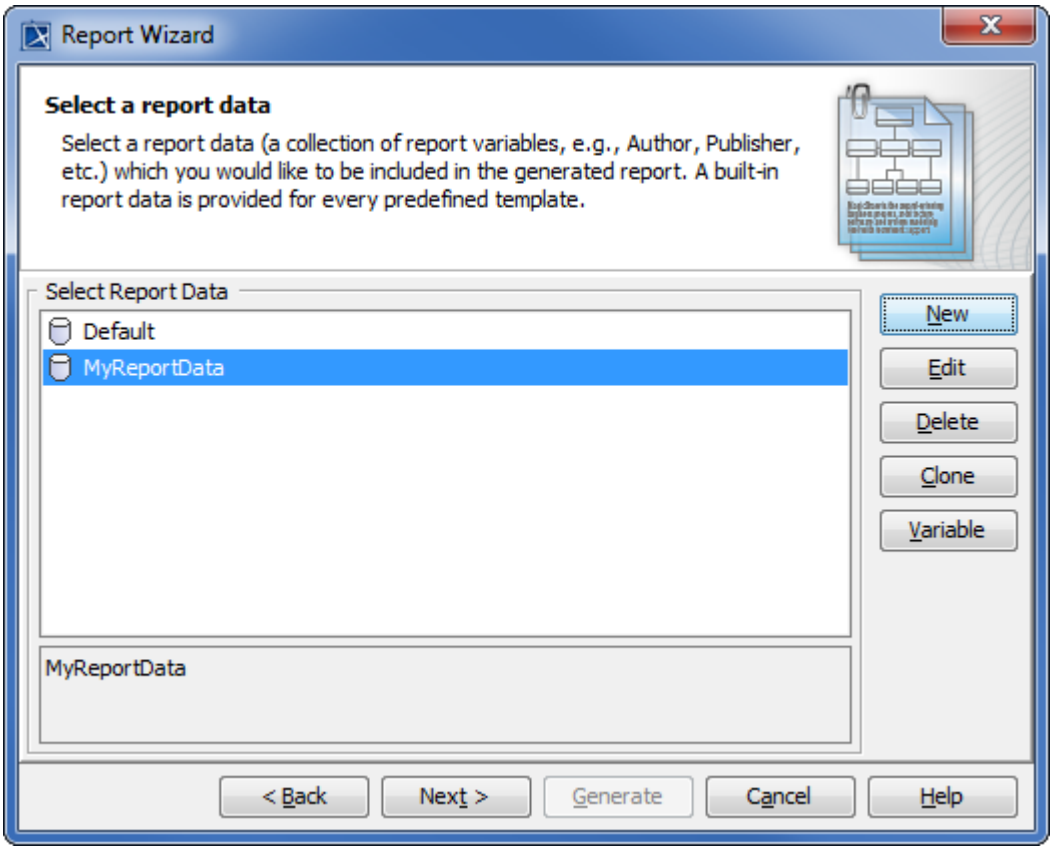
2. Click **Tools** > **Report Wizard** to open the **Report Wizard** dialog.
3. Select a template, for example, **Default Template** > **Use Case (Simple),** and click Next > .



4. You can choose to:
   a. Select the **built-in** Report Data. You can modify or delete some variables of the **Built-in** Report Data in the **Report Variable** dialog by clicking [Variable]
   or
   b. Click [New] to create a new one. Once you have created a new Report Data, for example, MyReportData, you will see it in the Select Report Data pane. You can select it and click [Variable] to create its report variables.

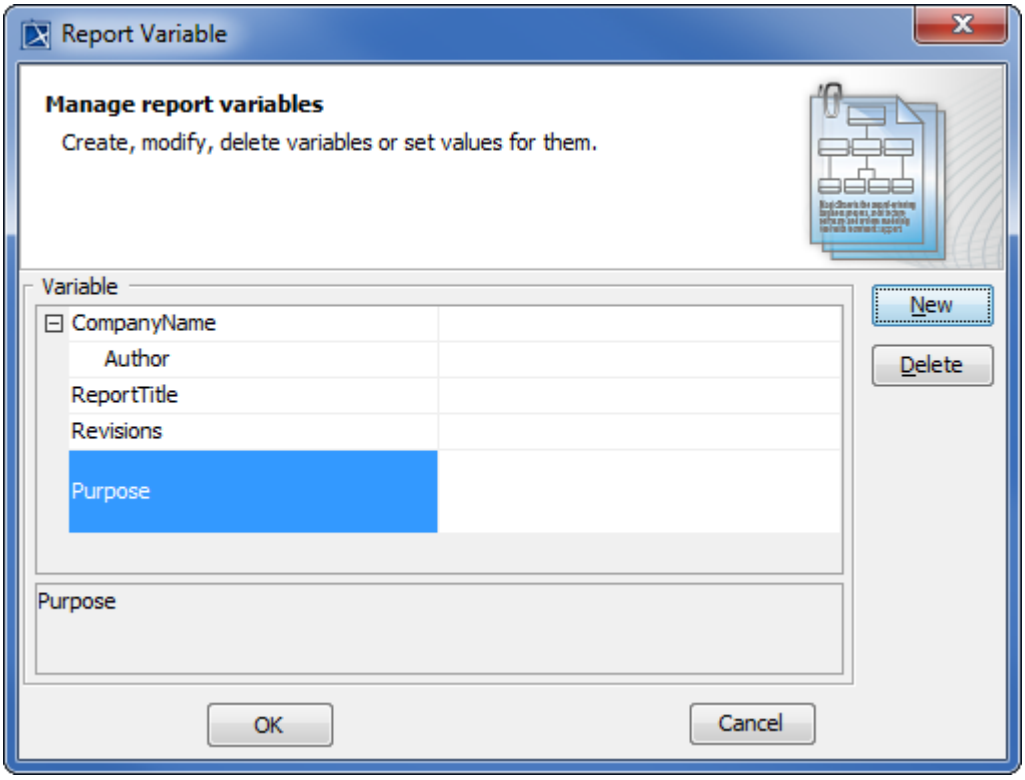


Selecting the Built-in report data.



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Selecting a custom report data .

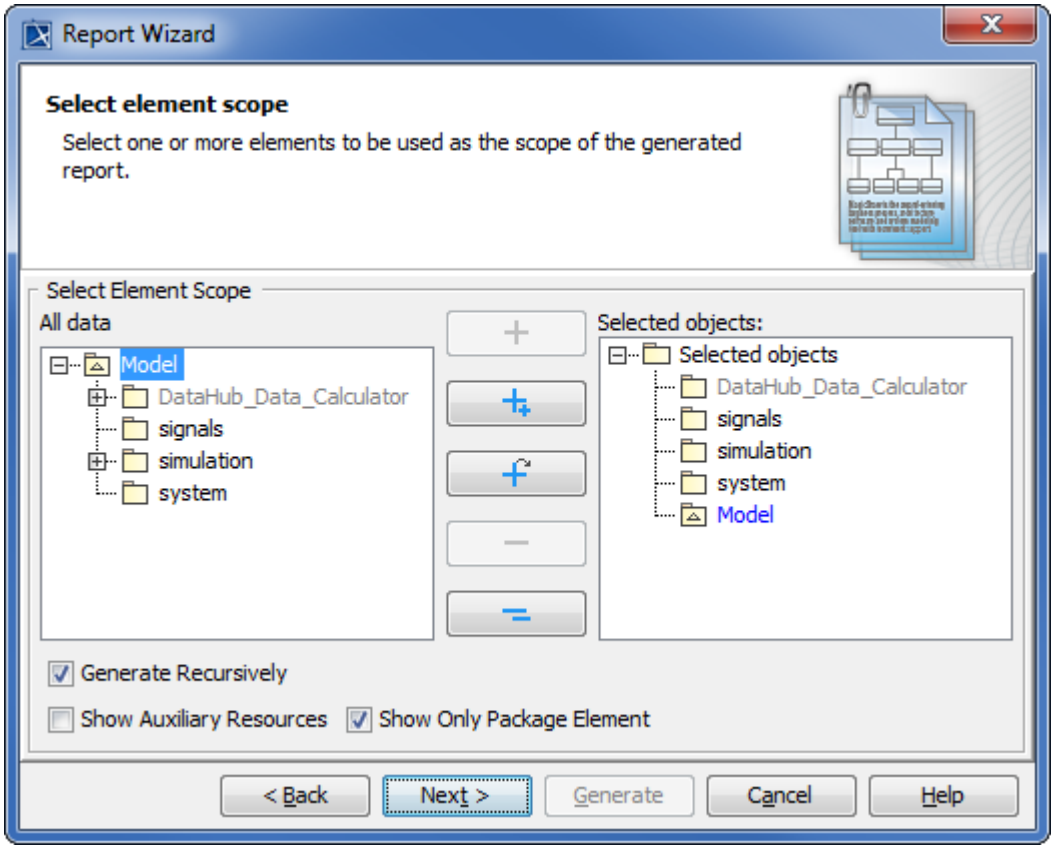


The Report Variable dialog.

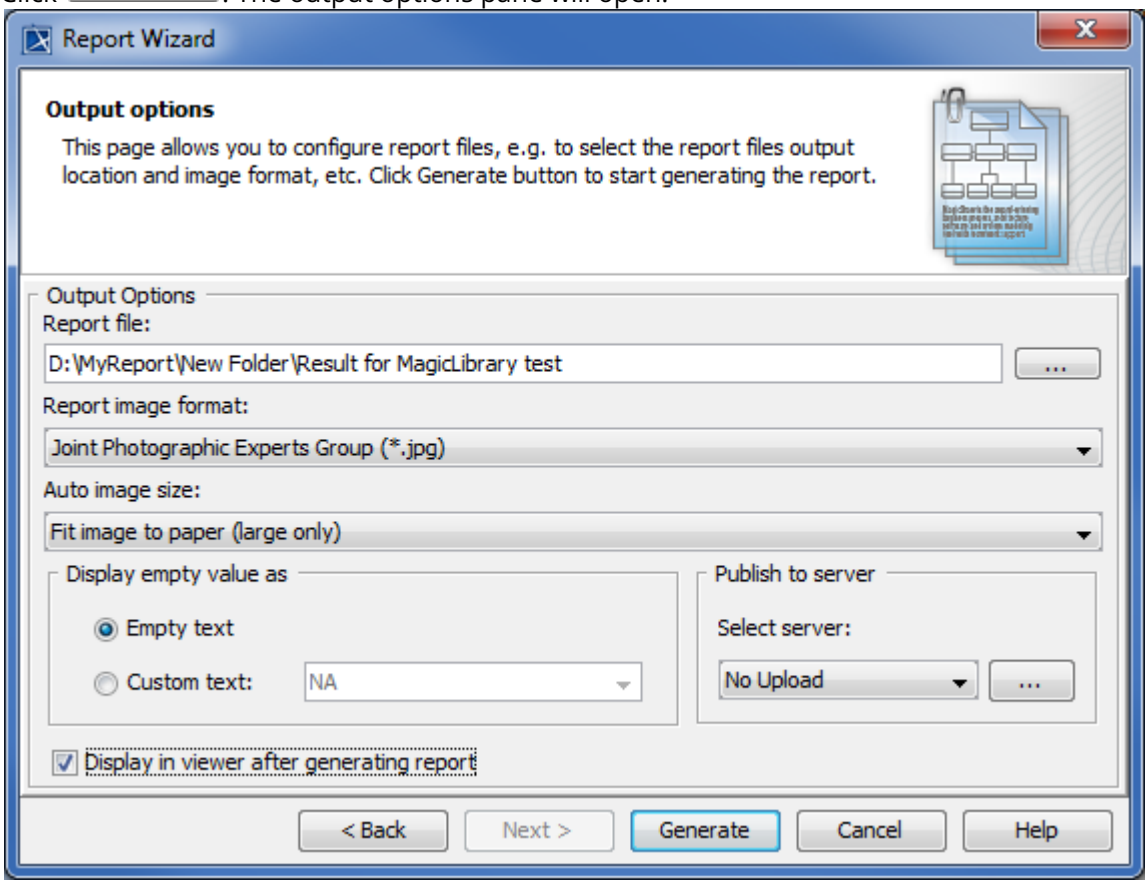
> You can create a new report variable, and modify or delete an existing one through the **Report Variable** dialog.

5. Click Next > once you have completed modifying or creating the report variables. The **Select Element Scope** option will open in the **Report Wizard** dialog.
6. Select the scope of the report from the package tree on the left-hand side. In this case, it will be the *MagicLibrary Requirements[MagicLibraryRequirements.mdzip]* model package, as all Use Case requirements are stored in this package.
7. Click +. The model package will be moved to the **Selected objects** tree on the right-hand side.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



8. Click Next >. The output options pane will open.

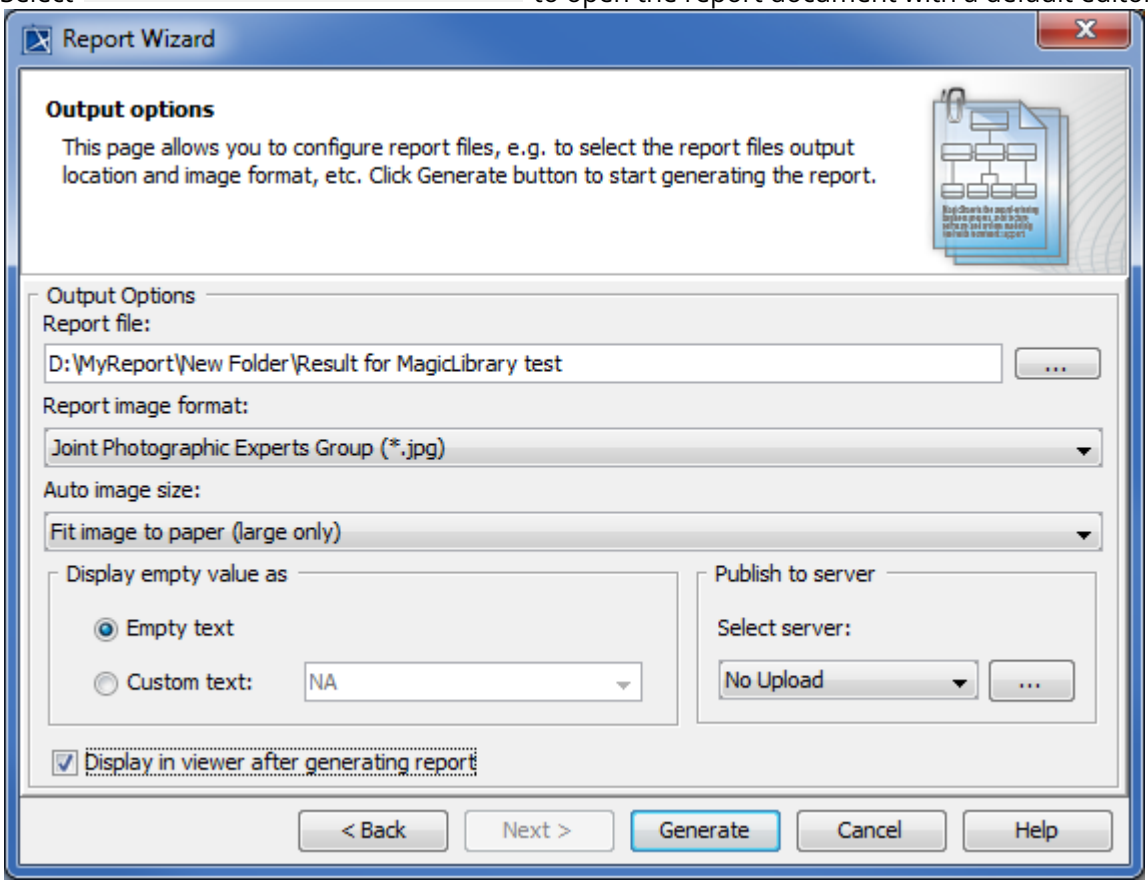


Selecting the location to generate the report, the image format and size, and other output options before generating the report.

9. Click ... to specify the location in which you want to save the report.
10. Select the file location, enter the filename, and click Save. The report file format varies depending on the template file format. For example, if the template file format is **.rtf*, the report file format will be **.rtf* as well.
11. Select the report image format, **.png*, **.jpg*, **.emf*, or **.wmf,* and select an image size option.

> **Tip**
> In some cases, a query may return an empty value that causes blank fields in the report. The **Display empty value as** option is useful when you have a standard representation for blank fields.

12. Select either the **Empty text** or **Custom text** option to display empty value information. If you select the latter, type the information you want to display for every empty value.
13. Select Display in viewer after generating report to open the report document with a default editor.



14. Click Generate.

The Use Case template is one of the default templates from which you can use and select in the Report Wizard dialog. You can see all of the default templates on this page Default templates(see page 1735).

Glossary

- **Use case**
  A Use Case is a list of actions or event steps of interactions between a system and the users to achieve certain goals.
- **Model**
  Model is the highest-level package in the Containment tree in MagicDraw.
- **Report data**
  Report Data is a collection of report variables that can be included in a report.

Sample model

Magic Library.mdzip[342]

More Information on Generating Reports

You can generate different types of reports with Report Wizard. Learn more about generating the other types of reports on the following pages.

- Web Publisher 2.0 Report(see page 1759)
- Web Portal Reports(see page 1737)

Web Publisher 2.0 report

Web Publisher 2.0 is a Java-Doc-like report template with a clickable navigation and an image map for all diagrams and elements. It allows you to publish MagicDraw models (including diagrams) in HTML with rich features, and to open them using your browser, e.g. Internet Explorer, Opera, or Firefox.

This section describes the features of Web Publisher 2.0, illustrates how to generate a Web Publisher 2.0 report template by using the Library.mdzip sample project, and gives you examples on how to work with each feature.

342 https://docs.nomagic.com/download/attachments/36336975/Magic%20Library.mdzip?api=v2&modificationDate=1475219260001&version=1

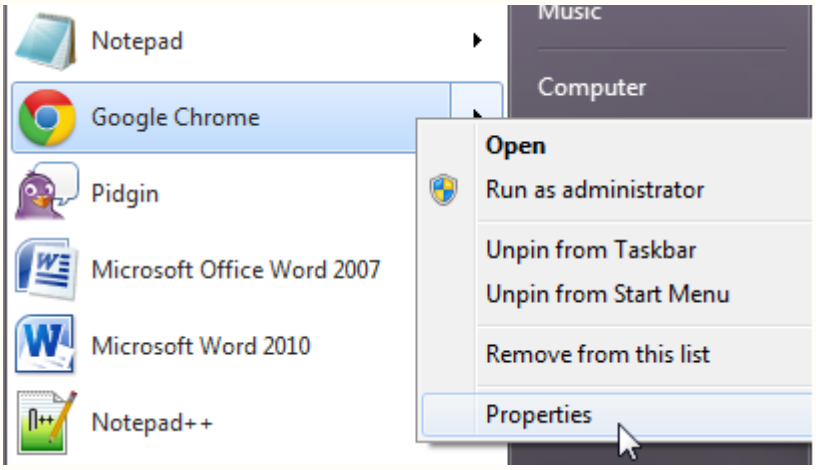
A Web Publisher 2.0 report.

Note

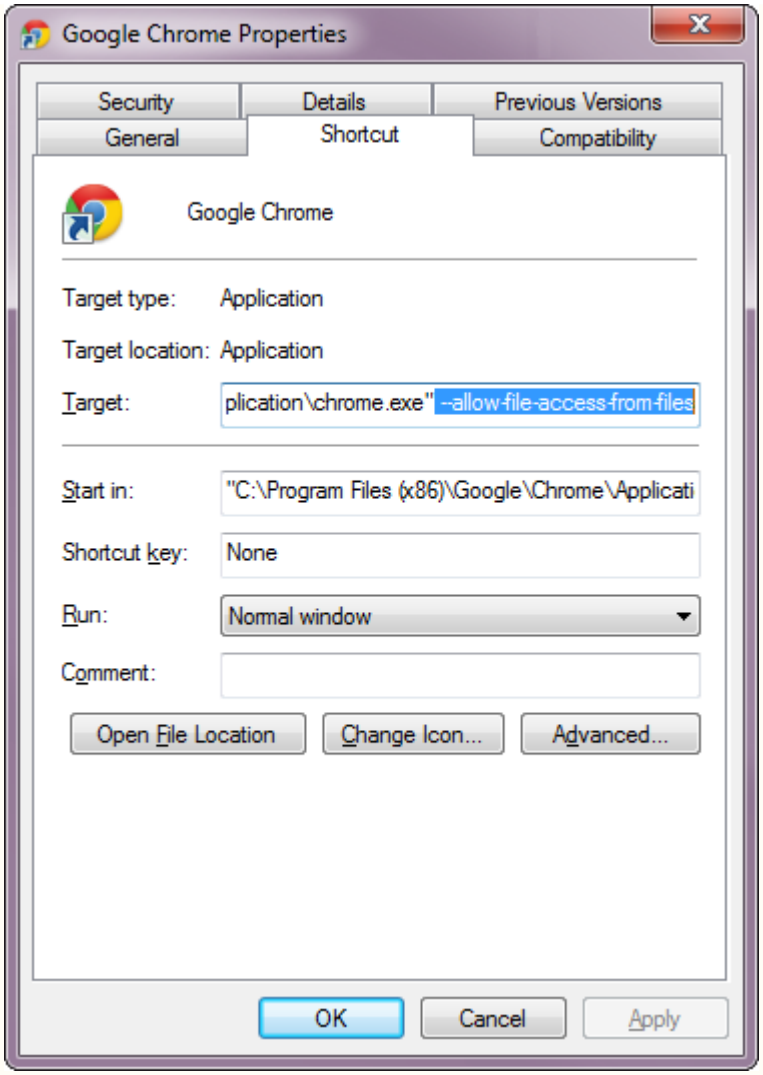
Web Publisher 2.0 report is not fully supported by Google Chrome and Opera; therefore, the data tree structure in the Containment tree may be missing because they do not allow JavaScript to load local files via **XMLHttpRequest**. You can solve this problem by doing one of the following:

- Deploy Web Publisher 2.0 report on web server, or
- Start Google Chrome with "**--allow-file-access-from-files**" argument by following the steps below:
    a. Close the web browser (Chrome) instances.
    b. Open the web browser with "**--allow-file-access-from-files**" argument.

    2.1 Right click on web browser launcher and select **Properties**.



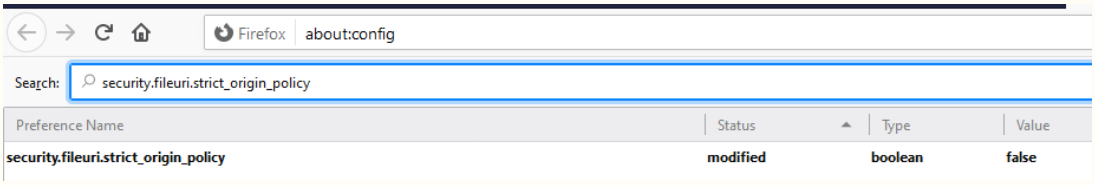
2.2 Add this argument "**--allow-file-access-from-files**" next to the web browser installation path.



2.3 Open the web browser instance from the launcher.
2.4 Open the generated Web 2.0 report.
2.5 Start the web browser from the launcher or command line that contains "**--allow-file-access-from-files**" argument before.

Note
Web Publisher 2.0 report is also supported by Firefox. You can configure Firefox before opening Web Publisher 2.0 report by doing the following:

1. In the Firefox address box, type **about:config**.



2. Find the **security.fileuri.strict_origin_policy** parameter and set its parameter value to **false**.

Cameo Collaborator is developed for replacing Web Publisher Collaboration report. This is a web-based solution for publishing and reviewing models online. See more detail at www.nomagic.com/products/cameo-collaborator-for-alfresco[343].

Related pages

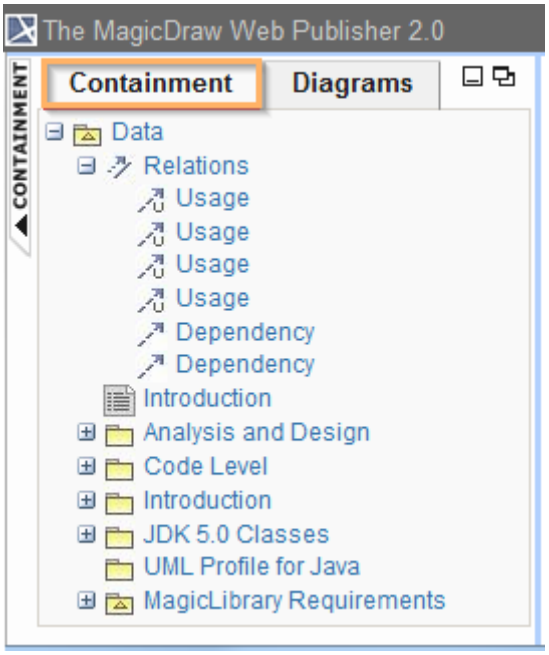
Features and layout

The Web Publisher 2.0 report consists of three panels:

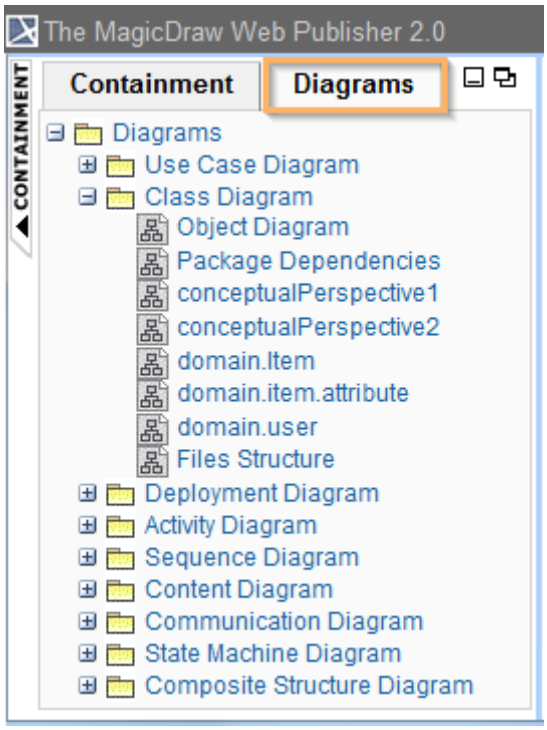
1. Containment panel-This panel contains two tabs: **Containment** and **Diagrams**.
   The **Containment** tab shows data of a project in a tree structure, and the **Diagrams** tab shows all diagrams in a project.
2. Content panel-This panel shows an element's content.
3. Search panel-This panel contains a **Quick Search** box. You can search for an element in a project by either typing in a specific keyword or by using a regular expression as a keyword.

343 http://www.nomagic.com/products/cameo-collaborator-for-alfresco

Web Publisher 2.0.



The Containment tab.

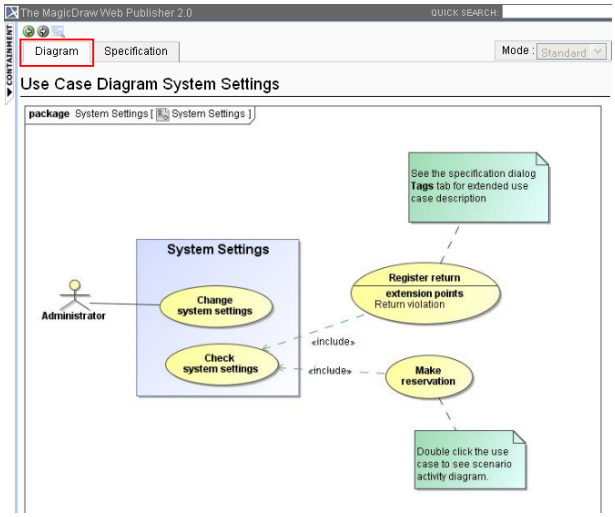
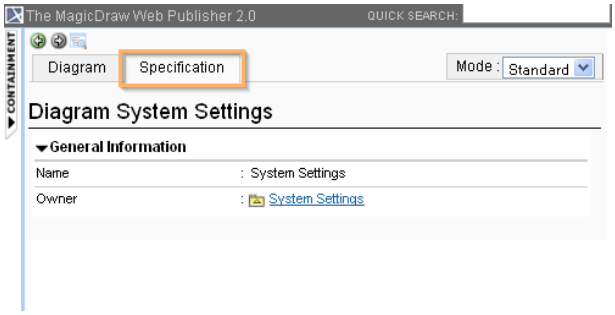
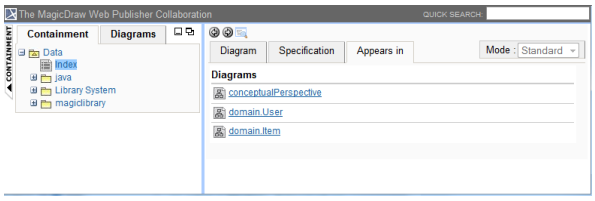


The Diagrams tab.

The Content panel of Web Publisher contains three tabs: Diagram, Specification, and Appears in. You can click any element in the Containment tree of Web Publisher to open three tabs.

- The **Diagram** tab shows diagram images.

- The **Specification** tab shows elements specification.
- The **Appears in tab** shows you all diagrams in which a particular element is present. You need to select the element in the Containment tree to open the **Appears** in tab.

Tab name	UI component
Diagram	
Specification	
Appears in	

Related page

- Working with Web Publisher 2.0(see page 1764)

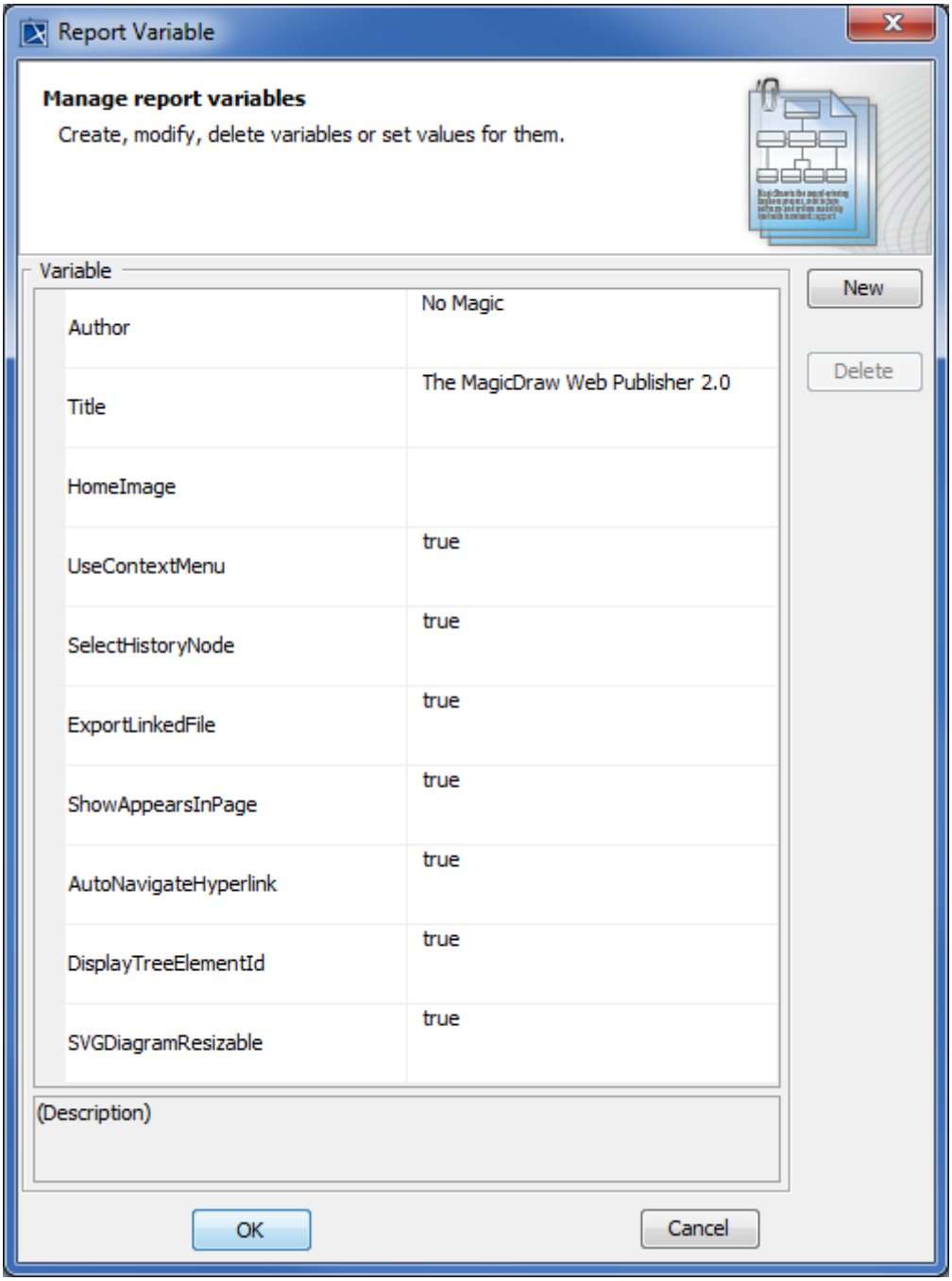
Working with Web Publisher 2.0

Note
The Magic Library[344] sample model, which comes with MagicDraw(see page 19), is used in the figures of this page. To open the sample properly, you need to install MagicDraw(see page 19) and download Magic Library.mdzip[345] or find in the modeling tool *<modeling tool installation directory>\samples\case studies\Magic Library.mdzip.*

The Web Publisher 2.0 report template comes with predefined variables and their values. Before generating a report based on the Web Publisher 2.0 template, you can add more variables to it, edit the existing ones, or delete those you do not want to include in your report.

[344] https://docs.nomagic.com/download/attachments/36336926/Magic%20Library.mdzip?api=v2&modificationDate=1492485589181&version=1
[345] https://docs.nomagic.com/download/attachments/36336926/Magic%20Library.mdzip?api=v2&modificationDate=1492485589181&version=1

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

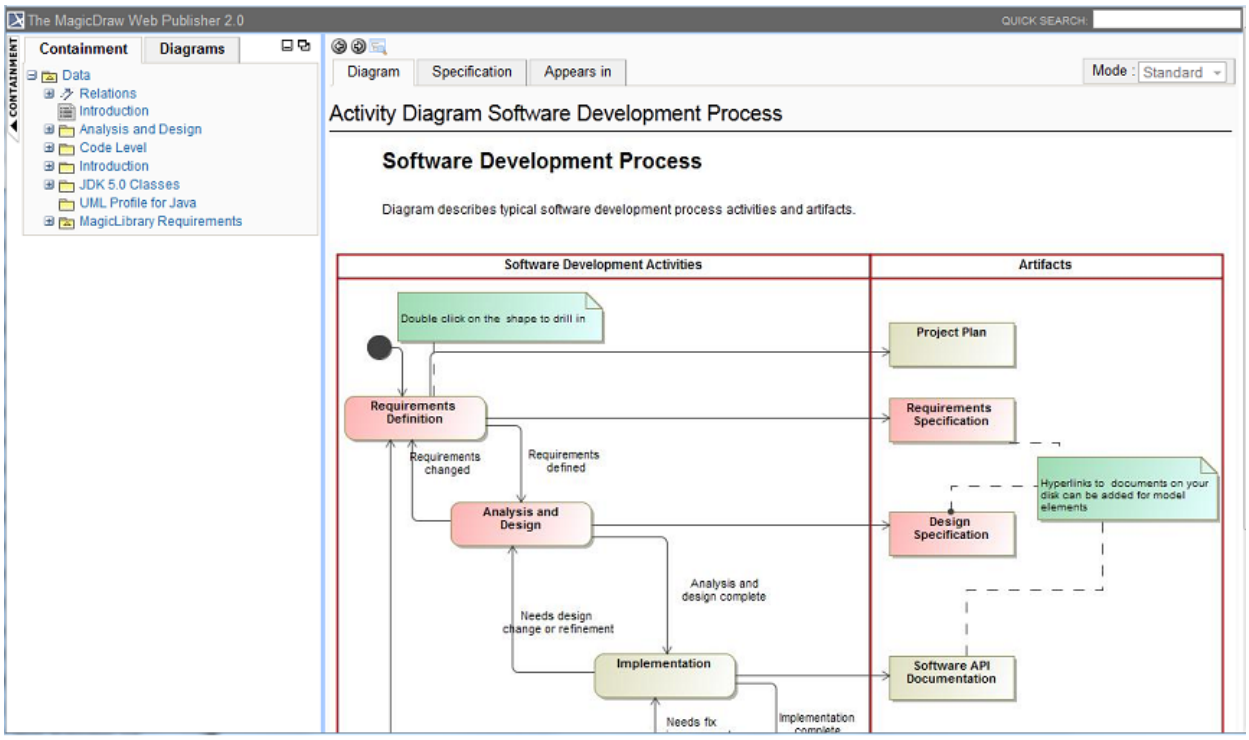


Report variables of Web Publisher 2.0.

The following table shows the variable names and values of the Web Publisher 2.0 template.

Variable	Value
Author	The report's creator.
Title	The report's title.

Variable	Value
HomeImage	Possible values: • Leave it empty or unspecified to use the MagicDraw icon as the homepage image. • Enter a diagram name (plain text) to display the selected diagram as the homepage image, for example, Software Development Process. • Enter "mdel://" followed by a model element ID to display the element as the homepage image, for example, "mdel://_10_0EAPbeta2_8740266_1126593738250_35764_172(see page 1764)". • Enter the location of an image file on your computer to display the image as the homepage image, for example, "*file://d://picture//image.jpg*". • Enter the location of an image on the Web to display the image as the homepage image, for example, http://www.photobucket.com/image.jpg.

Variable	Value
 Diagram as the homepage image.	
UseContextMenu	True: Shows a context menu when right-clicking a diagram element. False: Shows an element specification or opens an element, a diagram, or a page specified in any existing active hyperlink when right-clicking a diagram element.
SelectHistoryNode	True: Selects a node when you move backwards or forwards. False: Type "false" if you want it another way.
ExportLinkedFile	True: Copies a linked file from a model hyperlink into an output report folder. False: Keeps the link to an absolute path only.

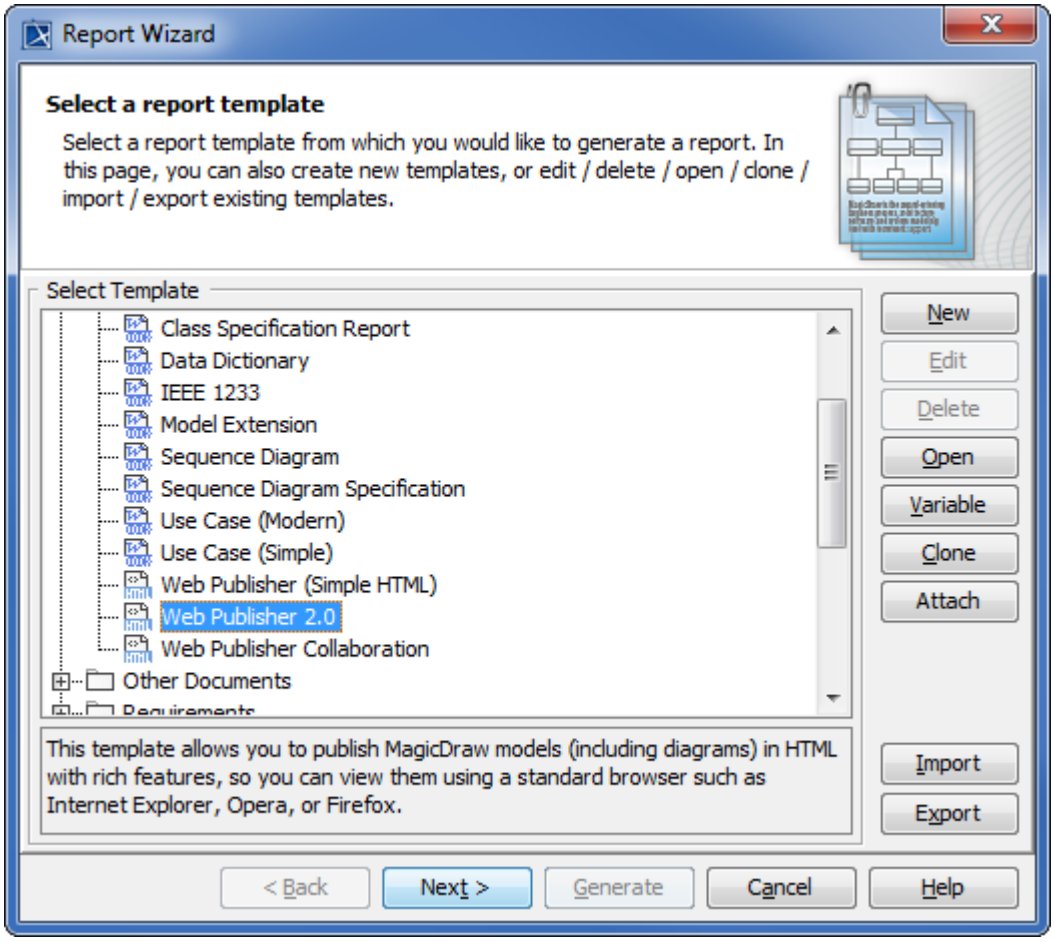
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Variable	Value
ShowAppearsInPage	True: Shows data in a report, and opens the **Appears in** tab when clicking an element in the Containment tree or a diagram pane. False: Hides the Appears in tab in a generated report.
AutoNavigateHyperlink	True: Automatically navigate the active hyperlink of each element. For example, if A has an active hyperlink to B and B has an active hyperlink to C, then C will be shown when you click A. False: The behavior of navigation to active hyperlink is the same as that of MagicDraw. For example, if A has an active hyperlink to B and B has an active hyperlink to C, then B will be shown when you click A.
DisplayTreeElementId	True: Shows the property ID of an element in the Containment tree. False: Type "false" if you want it another way.
SVGDiagramResizable	True: Resize SVG images when the browser is minimized. False: Enable SVG image linkable capability, but the resizable capability would be disabled. If the "SVGDiagramResizable" is set to True, the SVG image would be resizable on web browser, but it would not be linkable.

The following section contains the instructions on how to work with your Web Publisher report.

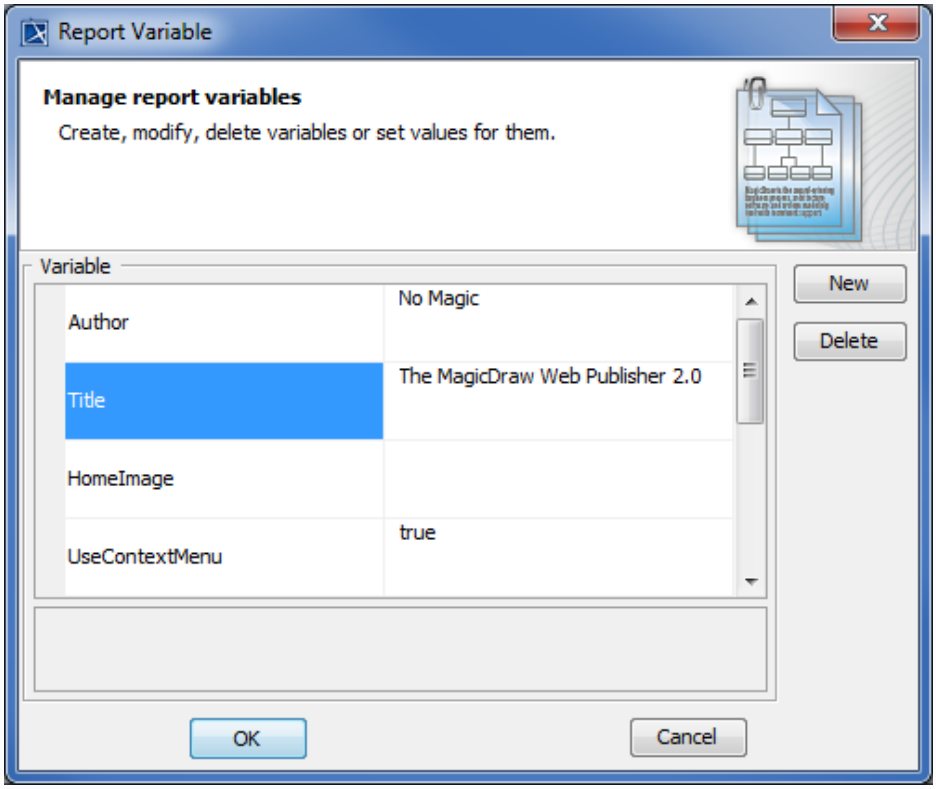
To generate a Web Publisher report

1. Open the *Magic Library.mdzip* sample project from the *<MagicDraw_home>/samples/case studies* directory.
2. Click **Tools** > **Report Wizard**. The **Report Wizard** dialog will open.

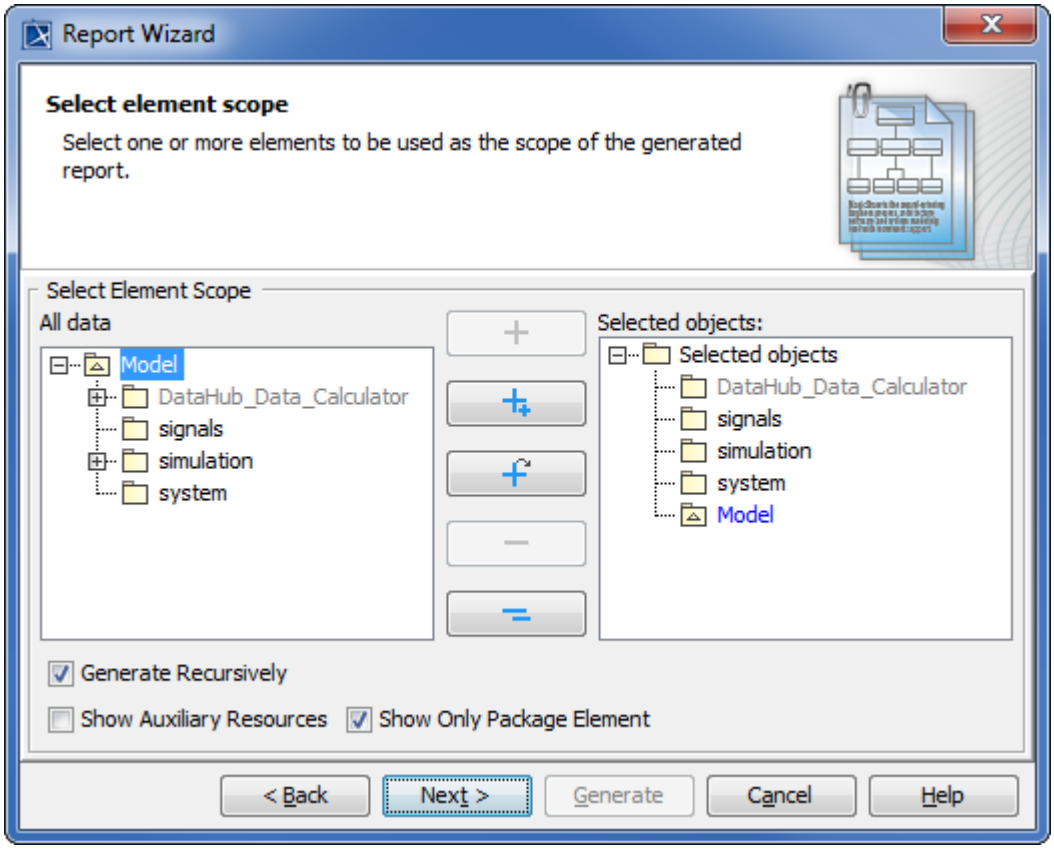


3. In the Select Template pane, select **Default Template** > **Web Publisher 2.0**, and click Next > .
4. You can either:
   - select the built-in report data or click New to create a new one. Type the new report name and description, and then click Create .

- In the Select Report Data pane, you can create a new set of Report Data for the Web Publisher template. The Report Data is a container for a set of custom-defined fields in the template. It can be used to group different report versions.
- To edit or create a variable in the report data by clicking [Variable], edit or delete information for predefined custom fields or create a new one, and click [OK].



5. Click [Next >].
6. Select the element(s) from the model that you want to include in the report data, and click [+]. Select the scope of the report in the open package tree. Select the **Model** package if you want to have a web-based report of your entire project.



7. Click Next >.
8. Click the ... button to locate the report file location.
9. Select the file location, type the report name, and click Save. A generated web report will include a number of folders and files.
10. Select the report image format: *.*png*, *.*jpg* or *.*svg*.
11. Select an option to display empty value information, either **Empty text** or **Custom text**.

> ⓘ In some cases, the query may return an empty value that creates blank fields in the report. The **Display empty value as** option is useful when you have a standard representation for blank fields.

12. Select the ☑ Display in viewer after generating report check box to open the report document with the default editor.
13. After all options have been selected, click Generate.

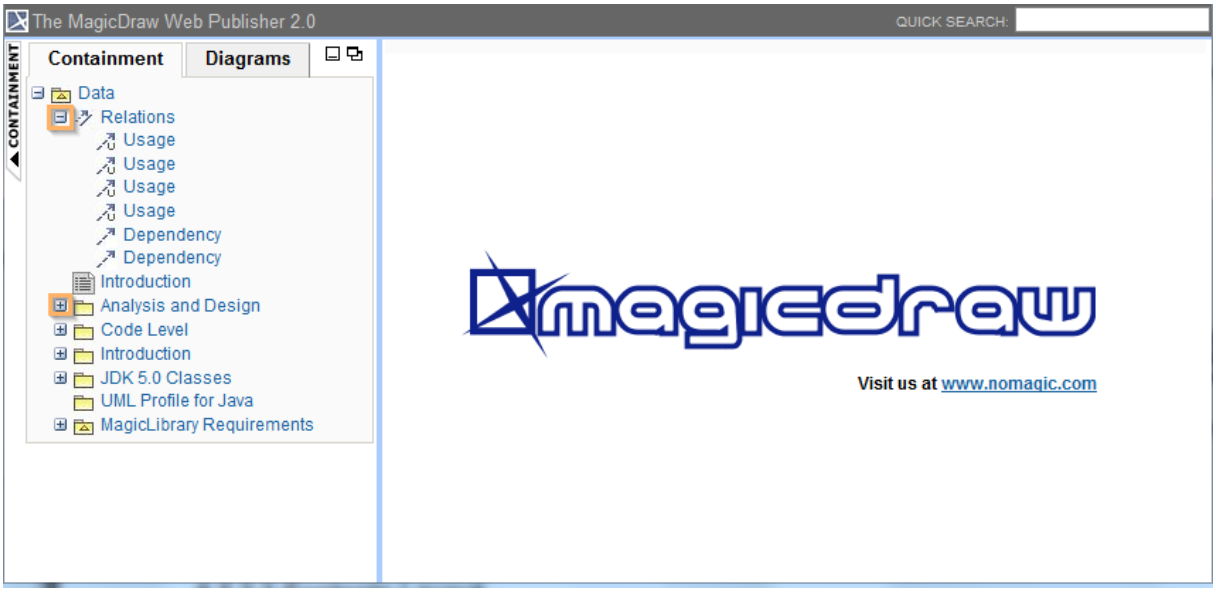
To hide or show the Containment menu

- Click the **CONTAINMENT** menu to hide it, and click it again to show it.

Hiding the Containment Menu.

To expand or collapse the Containment tree

- Click the "**+**" button in the Containment tree to expand it, or click the "**-**" button to collapse it.



The button to expand or collapse the Containment tree.

To reduce or increase the Containment tree width

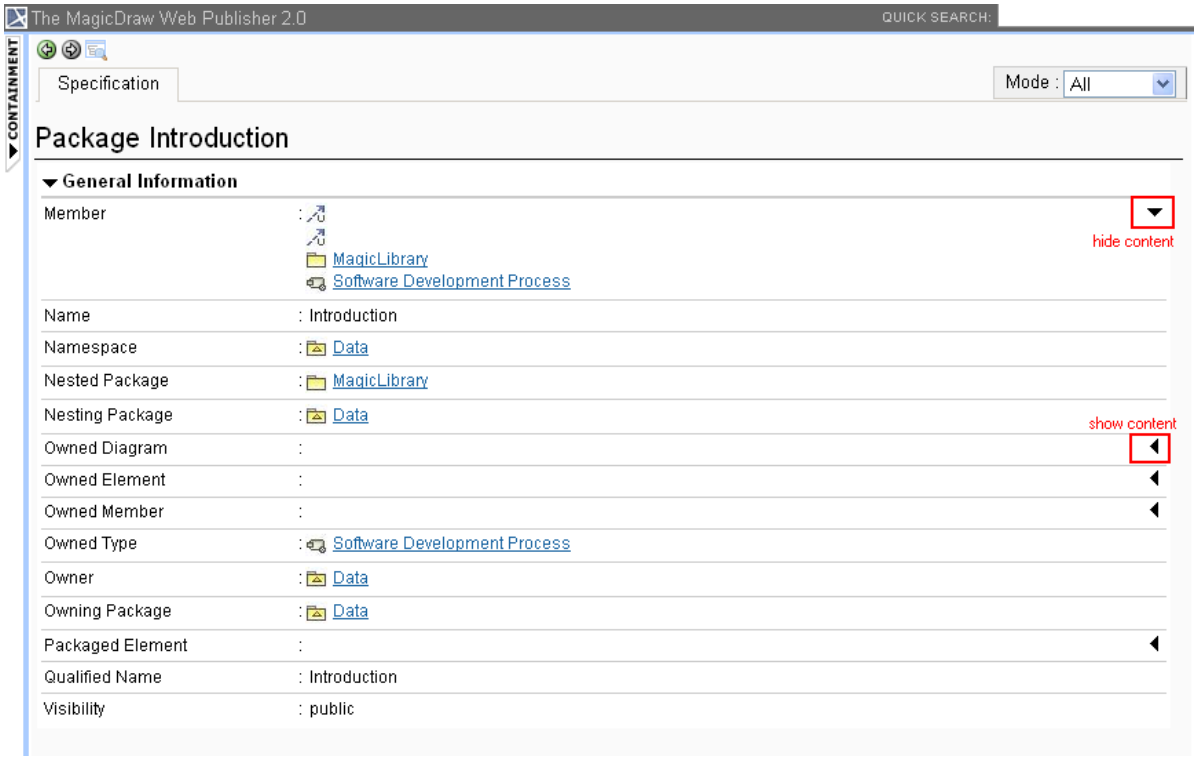
- Drag the separator line between the containment and the content panel to reduce or increase the width of the Containment tree.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

> The **Web Publisher 2.0** report interface uses a horizontal scrollbar, which appears when the length of an element's name exceeds the current width of the Containment tree. The default width of the containment tree is **20%** of the **web browser** width.

To show or hide element contents

- Click and re-click the arrow button to show and hide the contents.

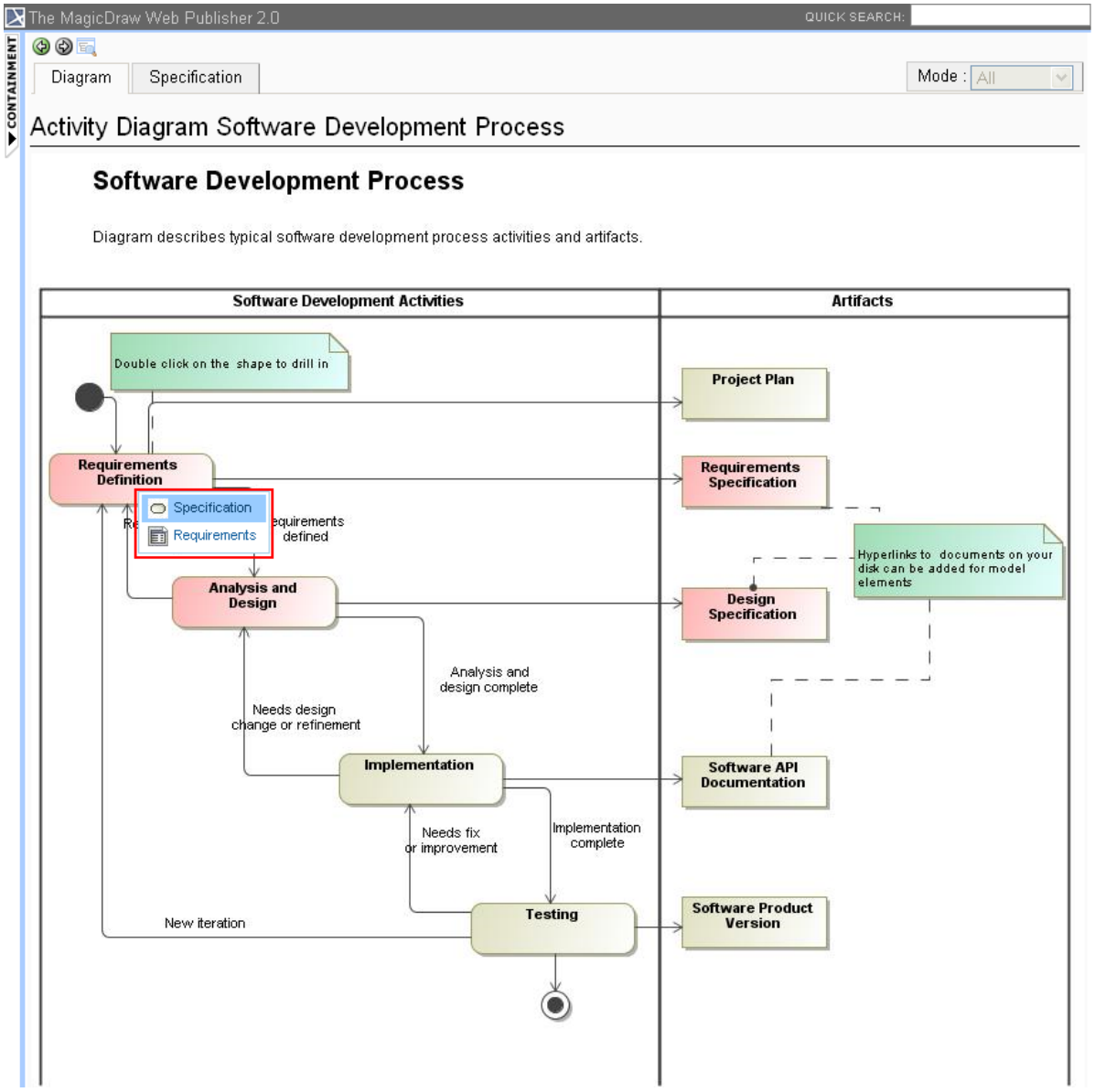


Showing and hiding element contents.

To show an element specification, active hyperlink, Submachine of state, or behavior of the Call Behavior action

- Click a diagram's element to show the shorcut menu for opening its specification, Active Hyperlink, Hyperlink, submachine of state, or behavior of the call behavior action.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

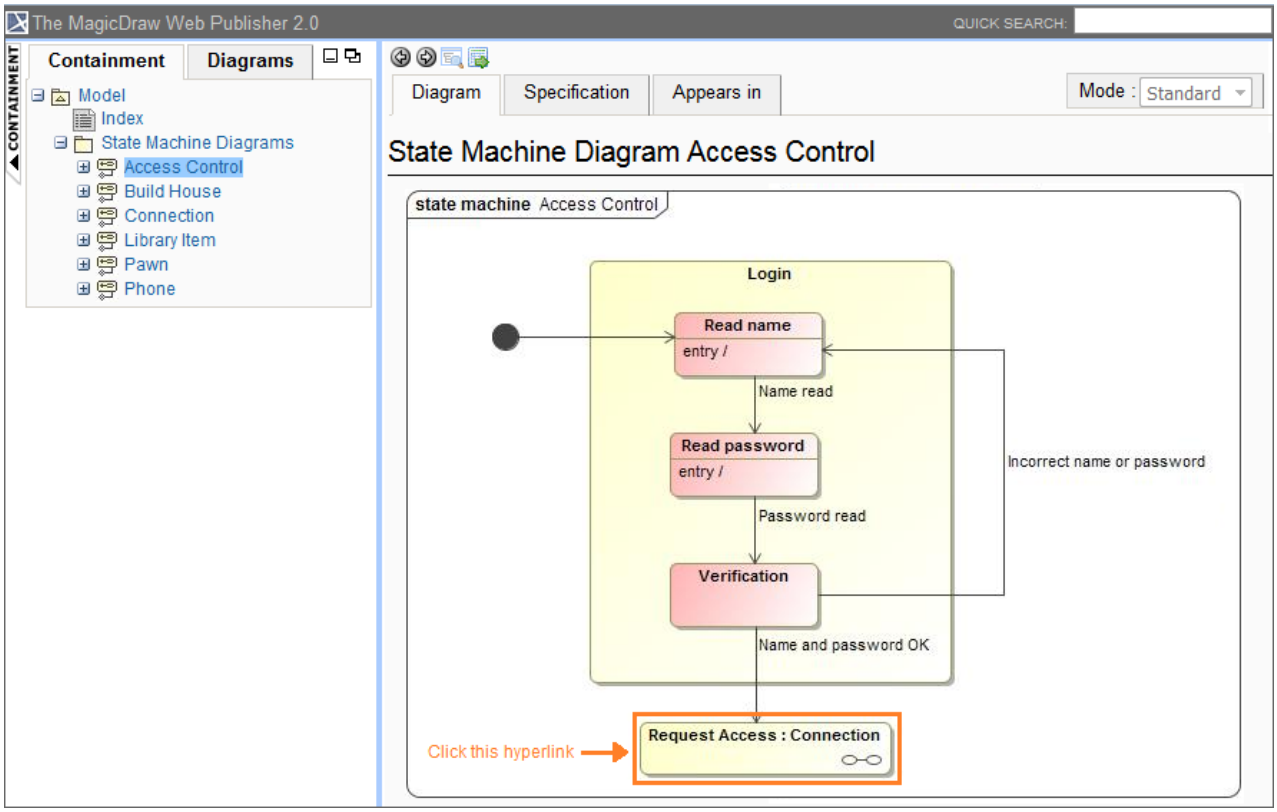


Web Publisher 2.0 context menu.

To add an Active Hyperlink to a model

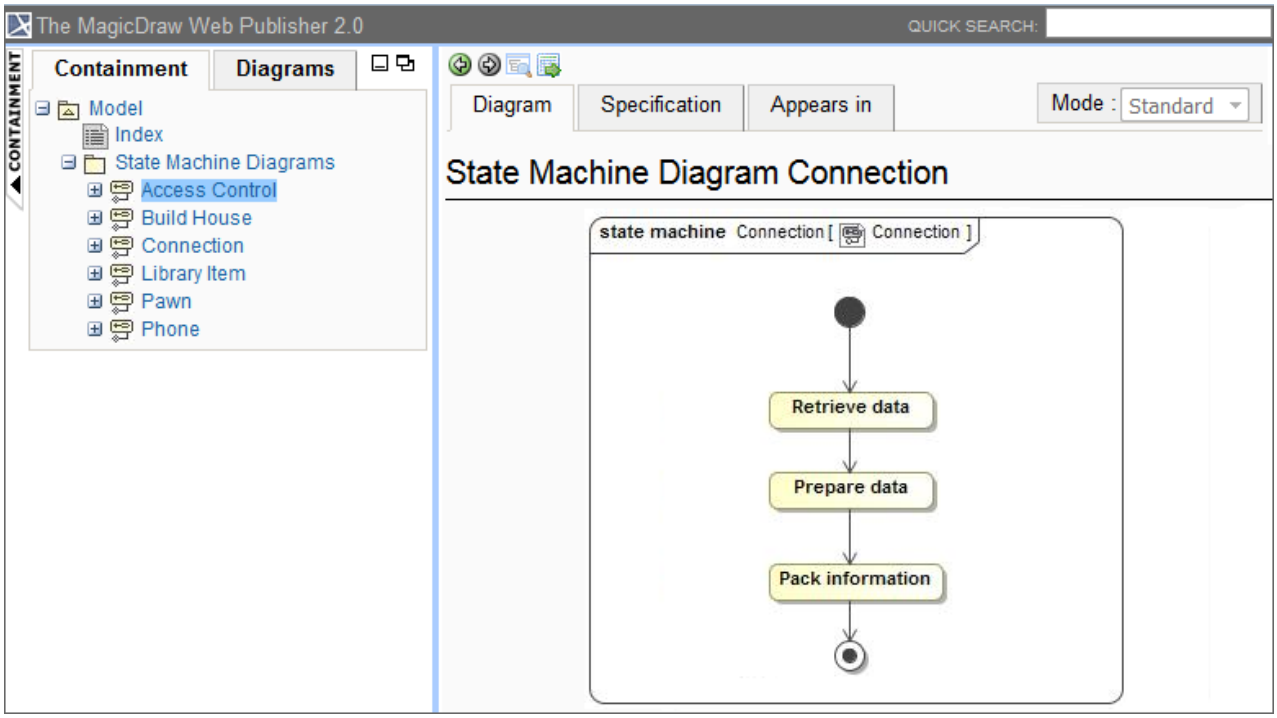
You can add an active hyperlink to any symbol on a diagram(see page 1008). There are three kinds of hyperlinks: Element, File, and URL. However, the Element type hyperlink is available for the Active Hyperlink column on the Generic Table diagram(see page 1777) image only.

You can navigate to the target if you double-click the hyperlink on the diagram in the Web Publisher report. The following figure below shows the Submachine State hyperlink in the Web Publisher report.



The Submachine State hyperlink in the Web Publisher report.

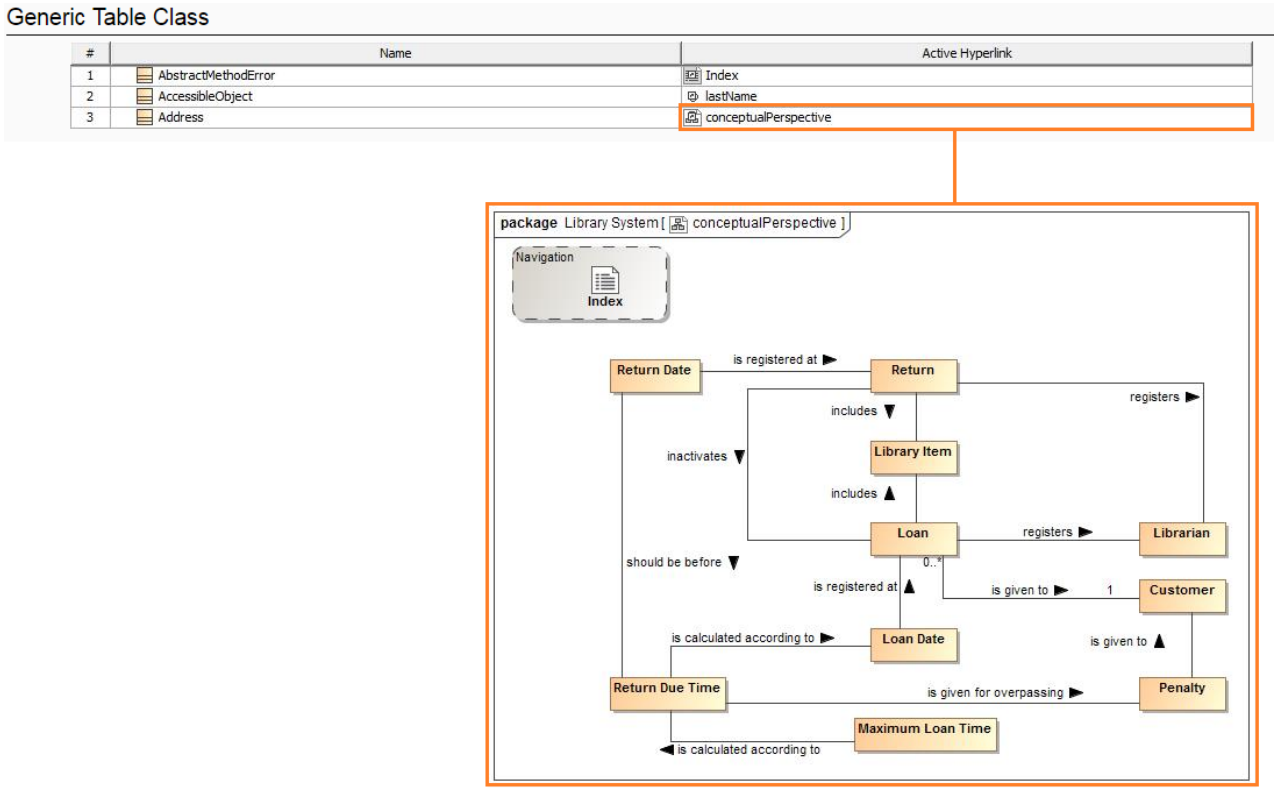
If you double-click the Submachine State in the above example, it navigates to the target and opens it for you as shown in the following figure.



The hyperlink navigates to the target in the Web Publisher report.

MagicDraw elements in the **Active Hyperlink** column in the **Generic Table** diagram link to the associate elements. If you click an element in this column, you will be navigated to the Specification page of the associated element.

> **Note**
> Only MagicDraw element type is available. File and URL address cannot navigate to the associated artifacts.

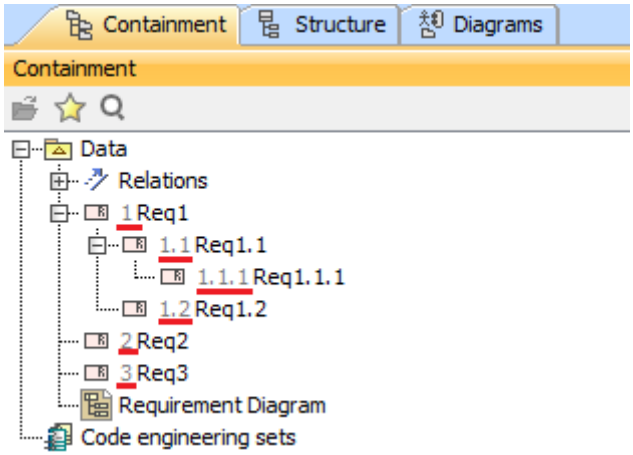


MagicDraw elements in the Active Hyperlink column in the Generic Table diagram links to the associated MagicDraw elements in the Specification page.

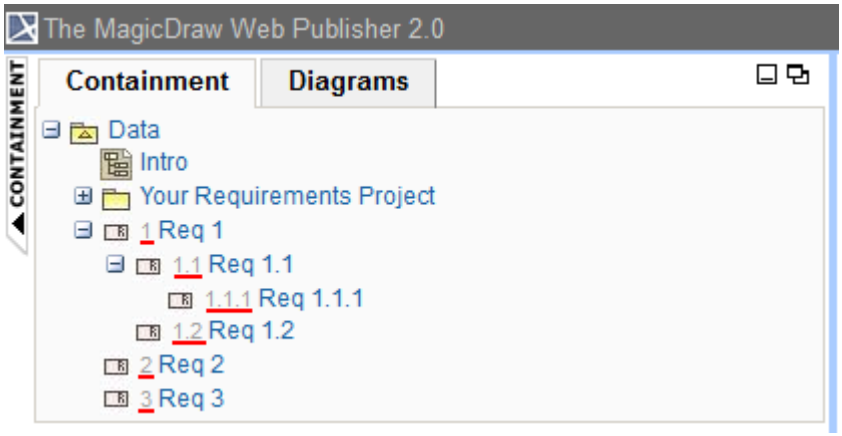
To display a property ID in the Containment tree

1. Click **Tools** > **Report Wizard** > **Variable** to open the **Report Variable** dialog.
2. Select the **DisplayTreeElementId** variable and enter the value **true**.
3. Click **OK** to show the requirement property IDs in the **Containment** tree.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

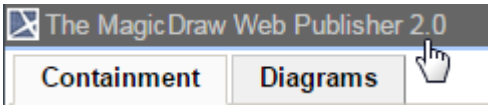


The requirement property IDs also appear in the Containment tree of Web Publisher.



To go to the index page

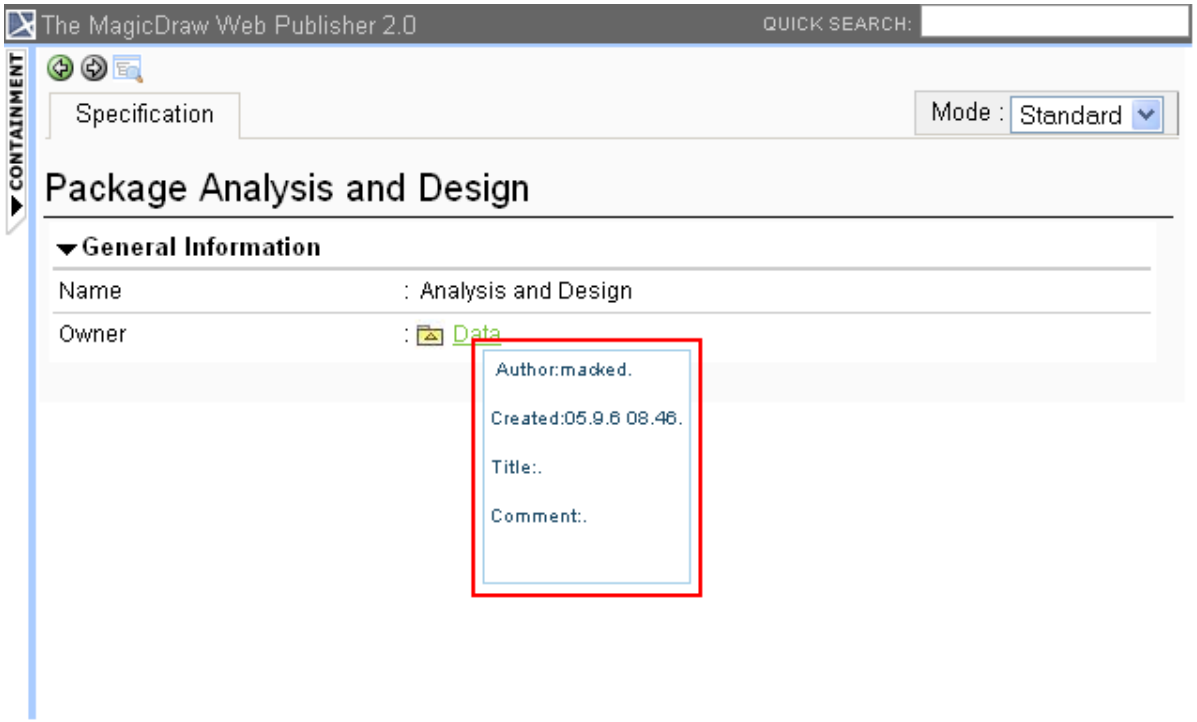
- Click the MagicDraw Web Publisher 2.0 at the top-left corner of the screen to go to the index page.



Shortcut to the index page.

To see the description of an element

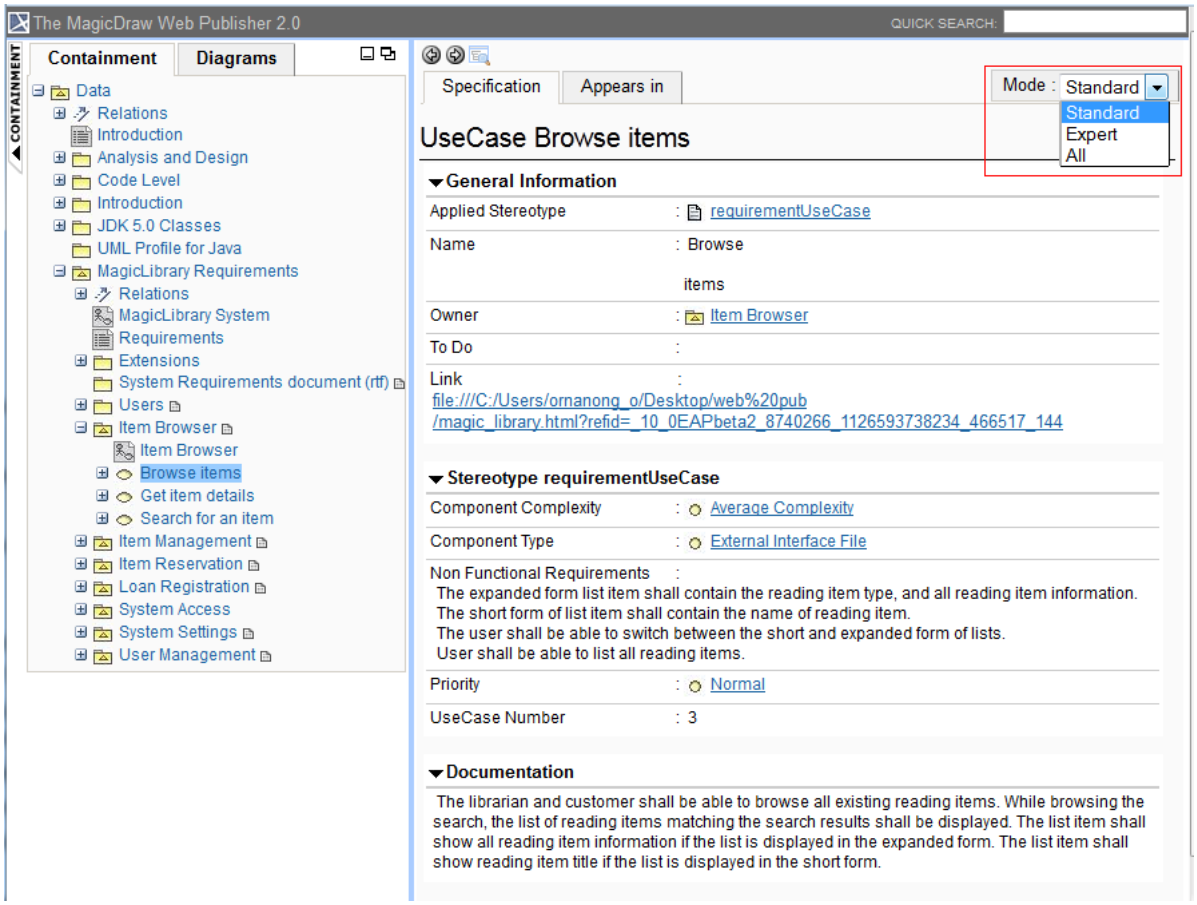
- On the **Specification** tab, move your mouse over an element to see the description in a tooltip.



Showing an element description.

To show property visibility of your report

- In the **Specification** tab, select **Standard**, **Expert**, or **All** from the **Mode** drop-down menu. The mode appears in the property visibility depends upon the mode that you have selected in MagicDraw.

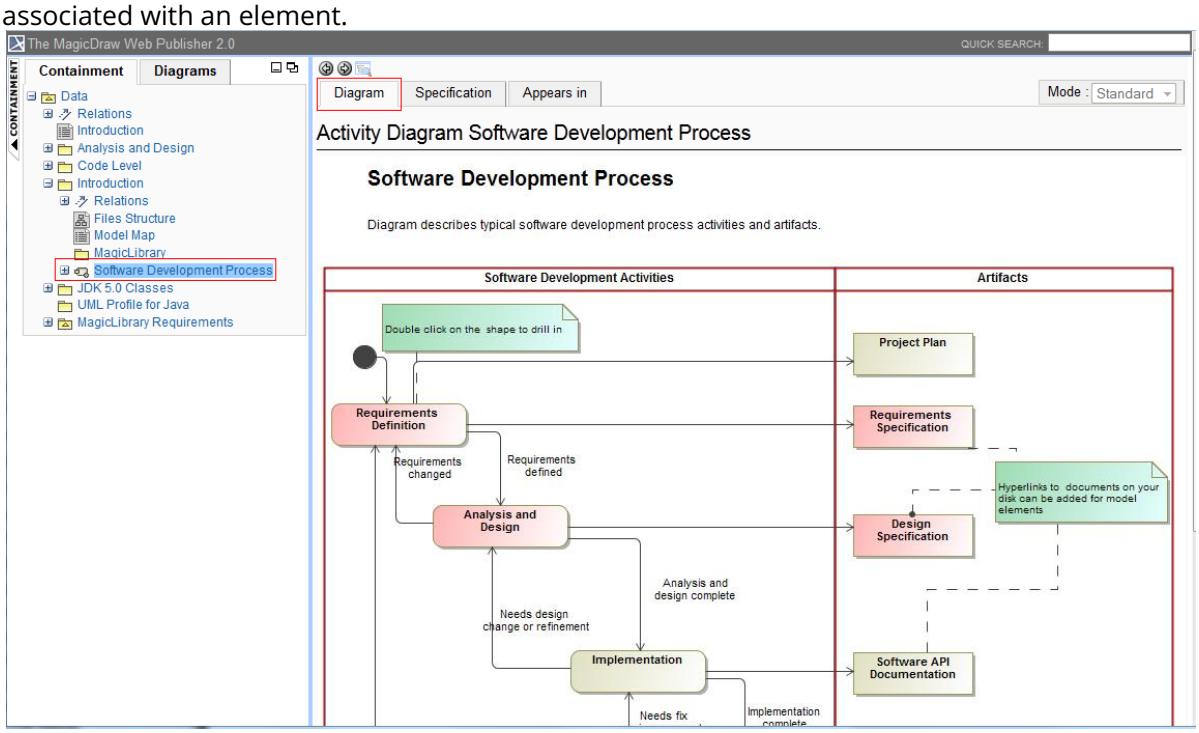


Property visibility mode types.

To open an Activity, a State Machine, a Collaboration, or an Interaction sub-diagram

---

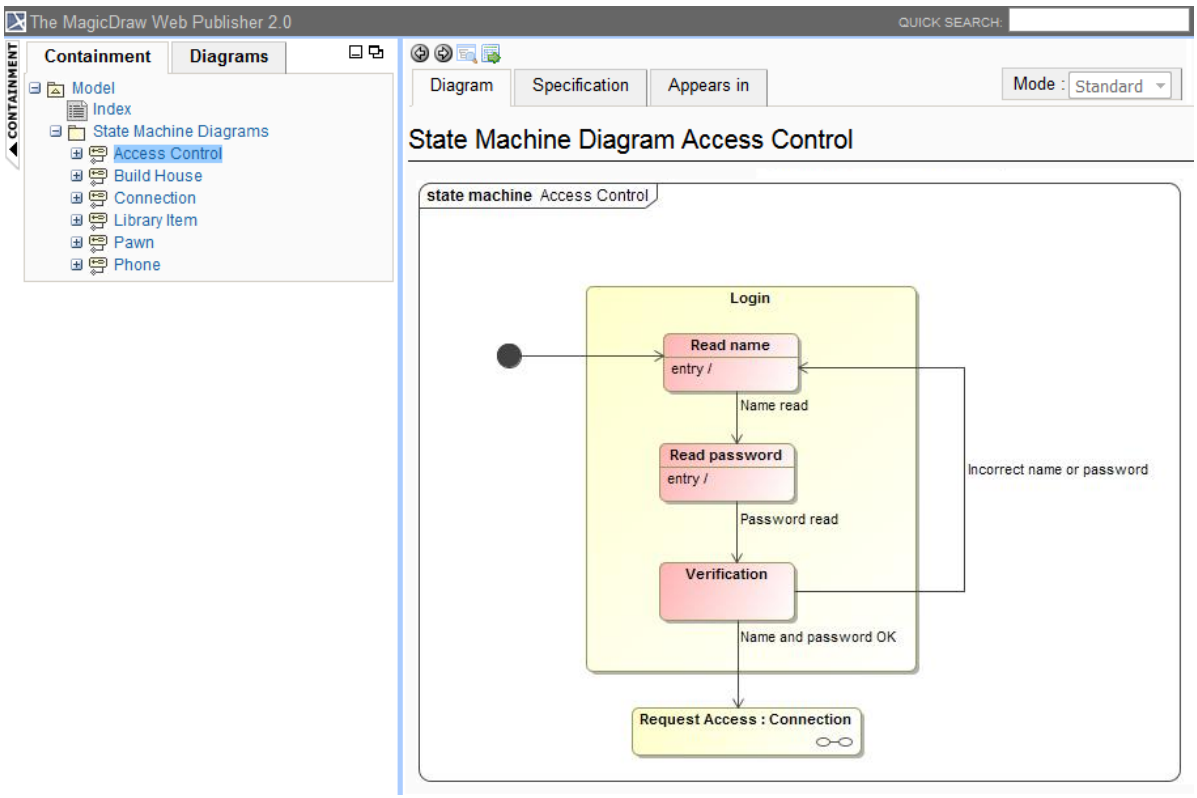
- Click an Activity, State Machine, Collaboration, or Interaction diagram to open a sub-diagram associated with an element.



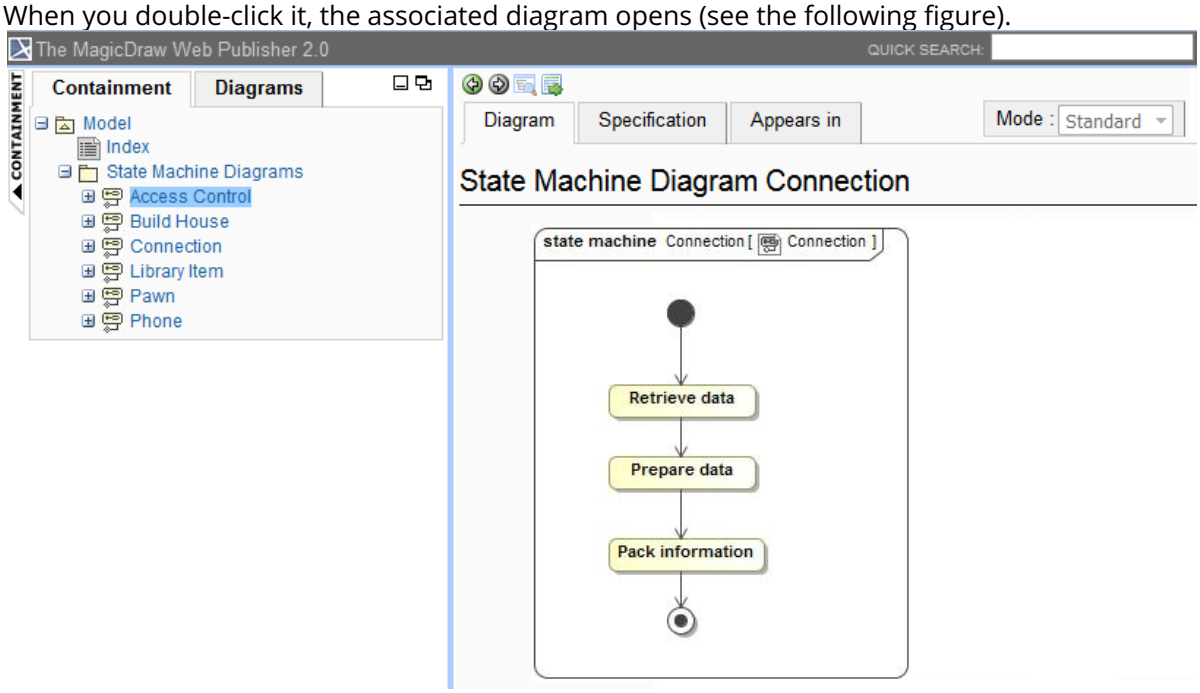
Clicking a diagram name to open its subdiagrams.

To open the sub-diagrams of a state with Submachine

1. Double-click either a state with a Submachine or a call behavior action of which behavior is specified. For example, double-clicking the last Submachine State *Request Access : Connection* in the following figure.

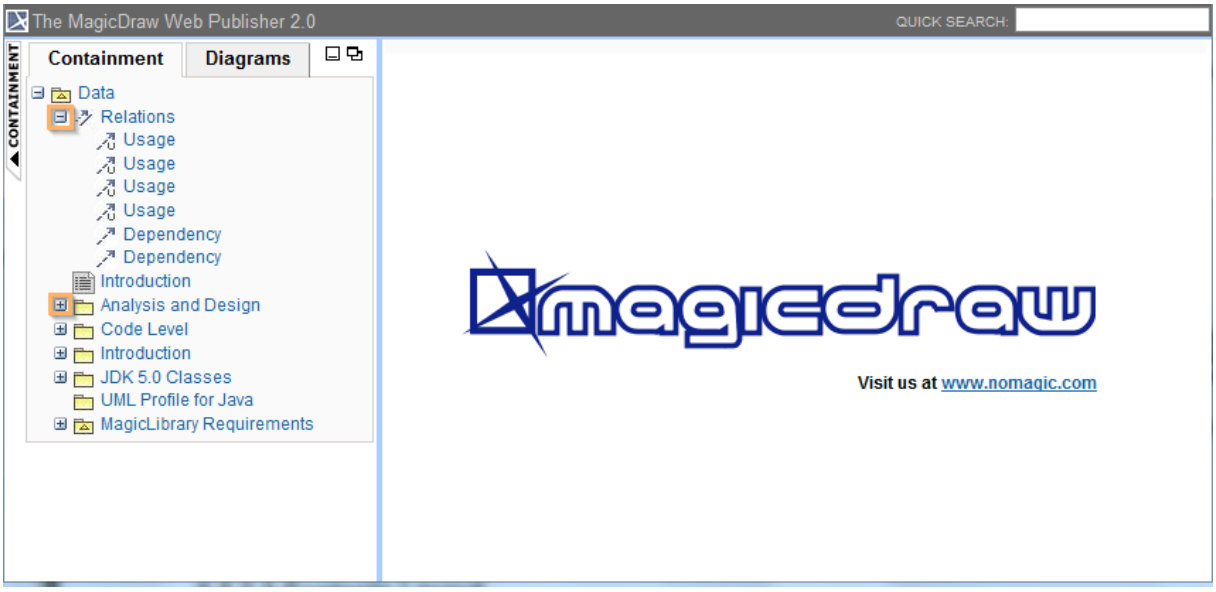


When you double-click it, the associated diagram opens (see the following figure).



State machine diagram viewed in web publisher interface.

2. Click the "**+**" button in the Containment tree to expand it, or click the "**-**" button to collapse it.



The button to expand or collapse the Containment tree.

Related page

- Working with your Web Publisher report(see page 1769)

Uploading reports to remote locations

On this page

- Uploading reports to a new server(see page 1783)
- Uploading reports to an existing server(see page 1788)

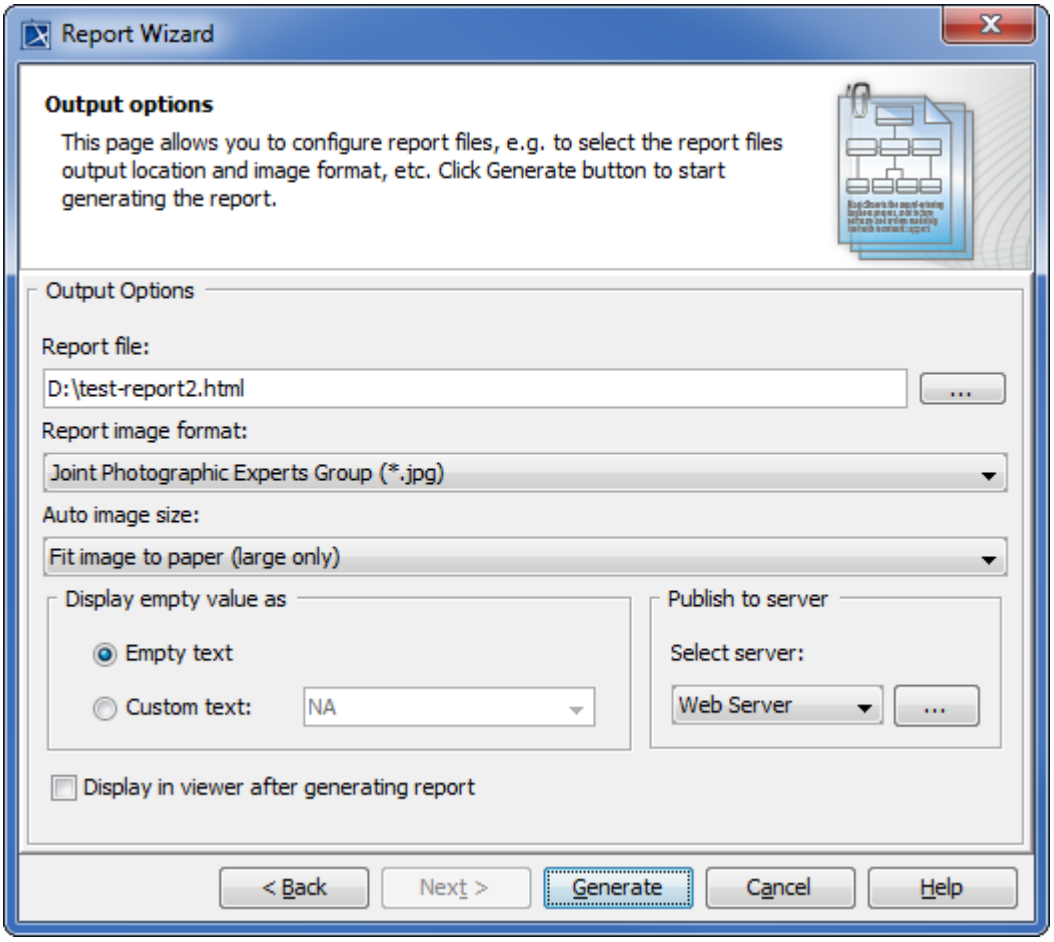
You can use the Report Wizard to upload a generated report to a predefined remote location server. Each remote location server has a profile that holds all necessary information, e.g., protocol, URL, username, and password. The Profile Management dialog(see page 1789) is used to create, edit, or delete profiles containing the configuration for remote location servers.

In this section, you will learn how to create a new remote location server step by step to upload a generated report to a newly created server, and how to upload a generated report to an existing server.

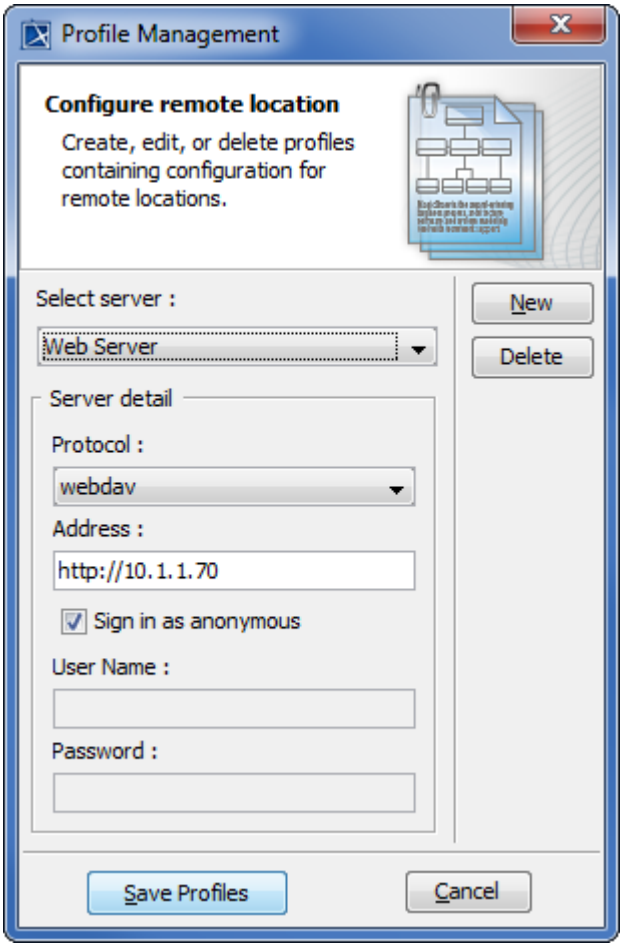
Uploading reports to a new server

To upload a generated report to a newly created server

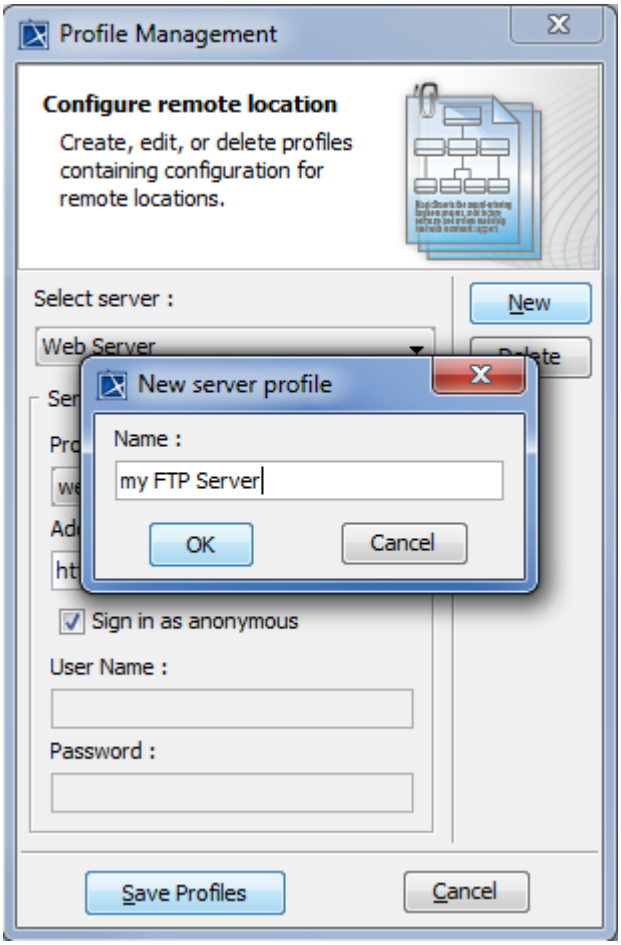
1. Open the **Report Wizard** dialog. On the main menu, click **Tools** > **Report Wizard**. Proceed through each step by selecting a report template, report data, and Element scope. You will be at the **Output options** step. In the **Report Wizard** dialog, click ... in the **Publish to server** area.



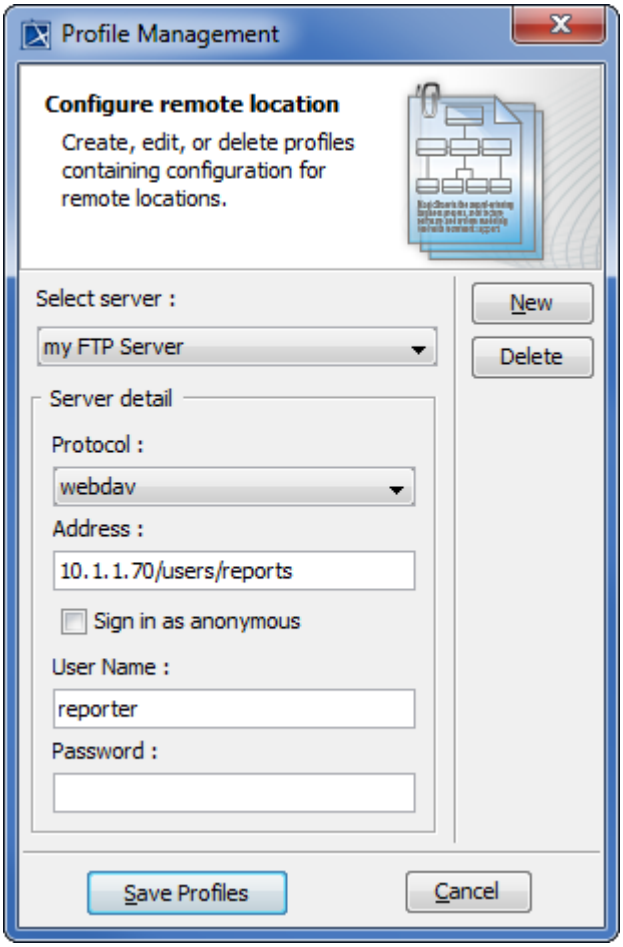
2. The **Profile Management** dialog opens. Click **New** to create a new server.



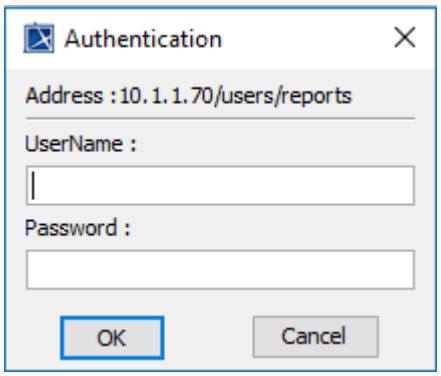
3. The **New server profile** dialog opens. Enter the name of the server to create. You cannot leave the **Name** box empty or assign a name that has already been used in an existing set of servers. Click **OK**.



4. Choose a protocol from the **Protocol** drop-down list and fill out a URL address in the **Address** box. You can leave out the **User Name** and/or **Password** boxes if desired. Click **Save Profiles** to save the newly added server, and close the **Profile Management** dialog.



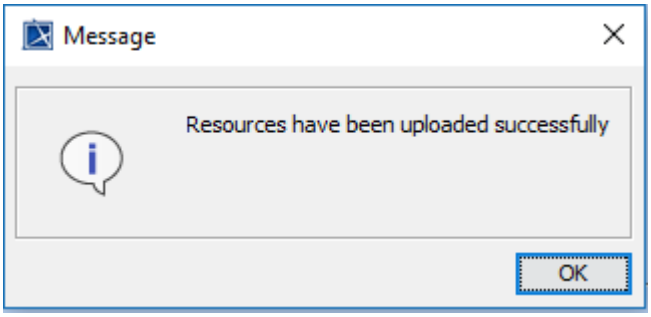
Note
If you leave out the **User Name** and/or **Password** boxes in the **Profile Management** dialog when creating a new server, you must enter the authentication information in the **Authentication** dialog after the report generation, then click **OK** to proceed.



5. You are now back to the **Report Wizard** dialog. Select the newly created server from the **Select server** drop-down list. Click **Generate** to generate the report to upload it to the newly created server. Once the report has been generated, it will be uploaded to the server previously created.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

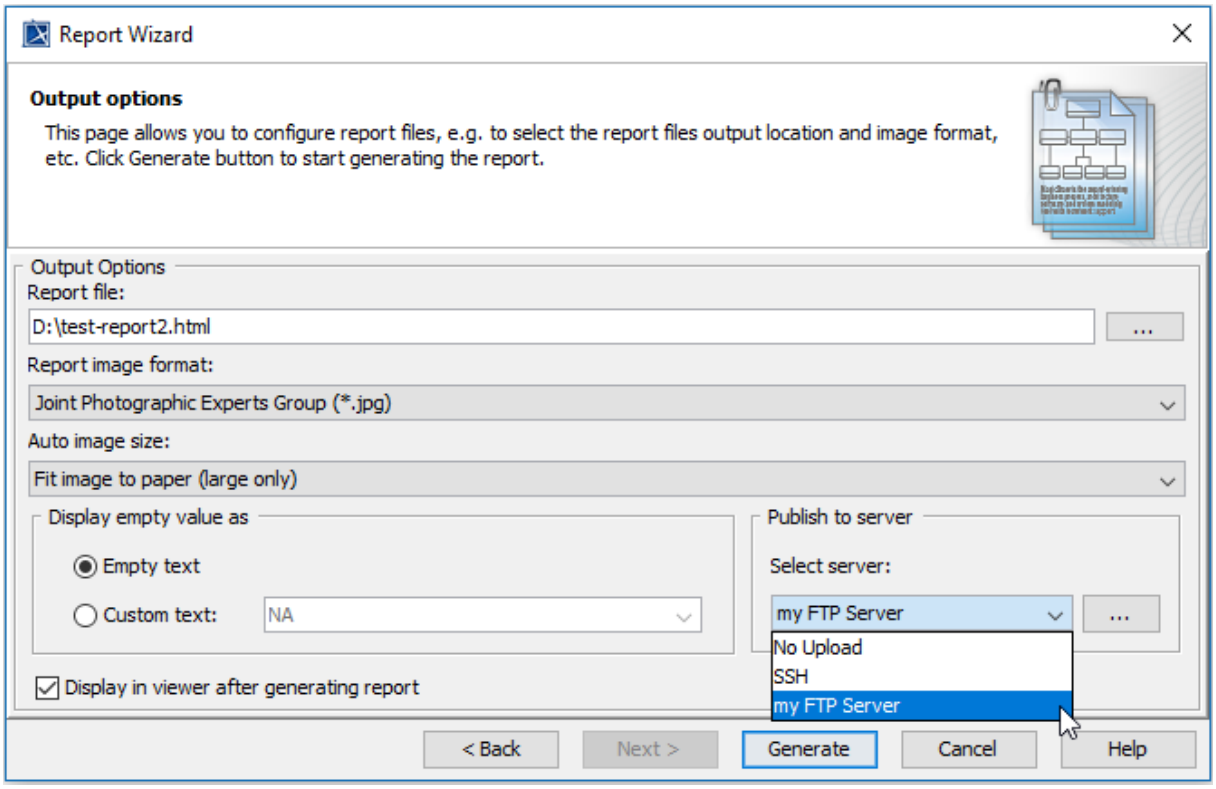
6. If the report has been successfully uploaded, a message box informing that the resources have been uploaded successfully opens. Click **OK**.



Uploading reports to an existing server

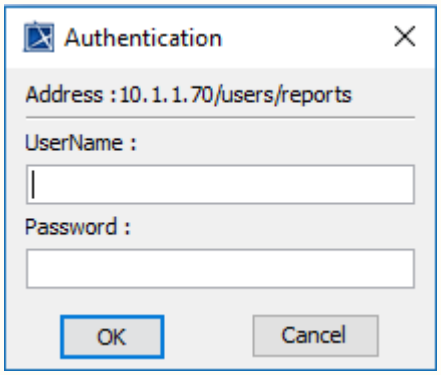
To upload a generated report to an existing server

1. Open the **Report Wizard** dialog. On the main menu, click **Tools** > **Report Wizard**. Proceed through each step by selecting a report template, report data, and Element scope. You will be at the **Output options** step. Select an existing server from the **Select server** drop-down list.

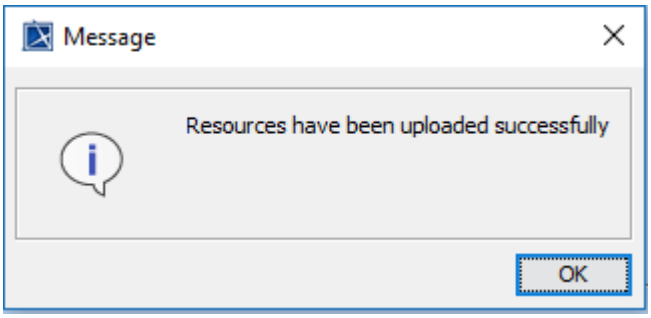


2. Click **Generate** to generate the report to upload it to the chosen existing server. Once the report has been generated, it will be uploaded to the server previously selected.

Note
If the selected existing server does not have the username and/or password specified in the **Profile Management** dialog when the server was created, you must enter the authentication information in the **Authentication** dialog after the report generation, then click **OK** to proceed.



3. If the report has been successfully uploaded, a message box informing that the resources have been uploaded successfully opens. Click **OK**.



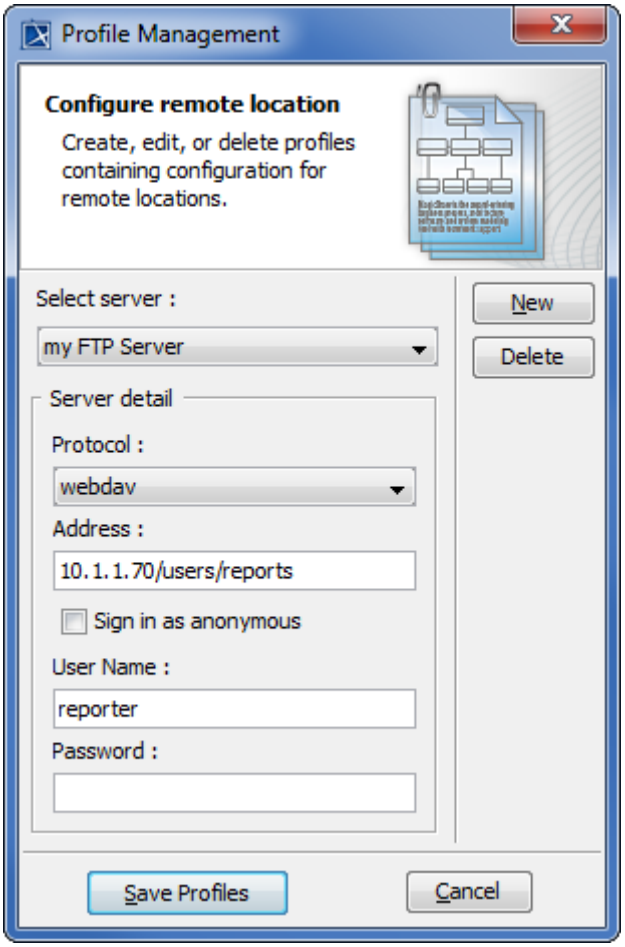
Related pages

- Profile Management dialog(see page 1789)

Profile Management dialog

This section explains the following items

- Profile Management dialog buttons(see page 1790)
- Profile Management dialog fields(see page 1791)



Profile Management dialog.

Profile Management dialog buttons

The Profile Management dialog contains five buttons:

New	Adds a new profile to the list of existing Profiles. You can neither create a profile with a blank name, nor with a name that already been used. You can add and edit multiple Profiles. However, these profiles will not be saved until you click the **Save Profiles** button.
Delete	Removes the currently selected profile. Note that you cannot remove the "No Upload" profile.
Save Profiles	Saves all profiles. If new profiles have been added or existing Profiles have been modified, then these changes will be saved. All profiles will be checked for consistency or information correctness. You will be asked to fix issues, if any. It is not possible to save profiles as long as invalid information is detected. See Profile Management dialog fields(see page 0) below for the description of "valid profile".

Cancel	Abandons all newly-added profiles and all changes made to existing profiles. If you do not make any changes to any profiles, clicking this button will bring you back to the **Output Options** pane of the Report Wizard dialog. Otherwise, the **Confirm Cancel** dialog will open. In this dialog, you will see the names of the profiles whose content you have changed. You can choose to either abandon all changes, or return to the **Profile Management** dialog and save the changes.
✕	Works similar to the **Cancel** button.

Profile Management dialog fields

Profiles describe information necessary for sending a report to a remote location. The report will be saved remotely with the same name as the locally-generated report, and will overwrite any file with the same name on the remote location.

The Address field is URL sensitive, and can hold protocol as well as username information. If the protocol or username information is found in the Address field, the corresponding fields, namely the Protocol and Username fields, will be updated with the values from the Address field. At the very least, a profile must have a hostname in the Address field.

You cannot save a profile unless this condition is met. The Username and Password fields contain sensitive data and, therefore, are optional. If these fields are empty in a profile, you will be prompted for your username and password when the server asks for your authentication.

- **Profile name**: This drop-down list holds all the existing Profiles. They are alphabetically sorted. The "No upload" profile is always available, and you can neither modify, nor remove it. Select the "No Upload" profile if you do not want to upload a report.
- **Protocol**: Five different protocols are supported: *FTP, FTP over SSL (Secure Socket Layer), Webdav, Webdav over SSL, and SSH (Secure Shell)*. You can either choose to select a protocol from the protocol drop-down list, or fill in the scheme in the address part of the profile. Certificates that are sent by servers using secure traffic (FTP over SSL, Webdav over SSL, and SSH) are silently accepted. The scheme identifiers and default port numbers of the five supported protocols are listed in the table below.

The table below lists Scheme Identifier and Default Port Numbers of Five Supported Protocols

Protocol Name	Scheme	Default Port Number	Traffic Mode
ftp	ftp://	21	Plain text
webdav	http://	80	Plain text
ftp + ssl	ftps://	990	Encrypted & Secure
webdav + ssl	https://	443	Encrypted & Secure
ssh	ssh:// or sftp://	22	Encrypted & Secure

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- **Address**

The Address field is the central part of a profile. It can set other fields, except the Password field. An Address contains five different parts: scheme, username, hostname, port (number), and (remote) path. Except hostname, every part is optional. Depending on the circumstances, you might need to add a special port number and/or the remote path where you have write access. Even though an address is valid without the path part, most servers will require a path, and will not let you write to the root directory. In every case, scheme and username parts will take precedence over the selection in the Protocol Field or the content in the Username field. The following is the basic pattern for the address field:

[scheme://][username@]hostname[:port][path]

1. **scheme**:
   The supported schemes are ftp, ftps, http, https, ssh, or sftp. A scheme must be followed by “://”.
2. **username**:
   A username is needed for user authentication. It must be followed by “@”.
3. **hostname**:
   Must be either an IP address (for example, 10.1.1.195) or a human readable URL (for example, www.ftpserver.com[346]), and a valid address.
4. **port**:
   A port number where the server is listed. It is only necessary if the port number is different from the default port for well-known services. It must be preceded by ":".
5. **path**:
   The predefined path will start from the home directory of the specified username. Therefore, the report will be generated at the home directory of the username, e.g., if the address is ssh://10.1.1.97:22/mymodel/publish, and the username is *developer*, the report will be generated at the home directory of the *developer* username. The designated path will be */home/developer/mymodel/publish*.

Examples of valid addresses:

1. 10.1.1.195 (hostname only)
2. 10.1.1.195/companyDav (hostname and path)
3. 10.1.1.195:80/companyDav (hostname, port number, and path)
4. ftp://www.ftpserver.com[347] (scheme and hostname)
5. john@10.1.1.195[348] (username and hostname)
6. john@www.ftpserver.com[349]/users/john/reports (username, hostname, and path)
7. ssh://john@10.1.2.35:22/reports (all parts)

- **User Name**

This field specifies a username that will be used for authentication. If the address contains a username part, the content of the Username field will be ignored. It will be either filled in or overwritten with the username from the Address. A Profile with an empty username field is valid, but you will be prompted for a username and password every time you upload to the server using this profile.

- **Password**

A profile with a password must always be accompanied by a username. A Profile with a password, but not a username, is invalid and, therefore, cannot be saved. The password field uses an '*' to represent a character in a password. Saving a profile with a password will save the password in an encrypted form. For example, if the password is 'test', it will be encrypted to '0013BCA5590DE'. A

346 http://www.ftpserver.com/
347 ftp://www.ftpserver.com/
348 mailto:john@10.1.1.195
349 mailto:john@www.ftpserver.com

Profile with an empty password field is valid, but you will be prompted for a password every time you upload a report to the server in this profile.

Generating reports from the command line

To generate reports and schedule report printing, type the command in the terminal. This feature allows you to generate reports without opening the MagicDraw application and using Report Wizard.

Related pages

Using the Generate command to generate reports

On this page

- Using the Generate command(see page 1793)
- Description(see page 1794)
- Options(see page 1795)
- Using command line(see page 1800)
- Generating multiple reports (see page 1801)

Using the Generate command

You can enter the Generate command in the command line to generate a report.

To generate a report from the package scope

- `generate - project projectFileName -output outputFileName -template templateName -package packageNameList`

To generate a report from the element scope

- `generate -project projectFileName -output outputFileName -template templateName -element elementNameList`

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To generate a report from the option specified in the properties file

- `generate -properties propertiesFileName`

To generate a Teamwork password for a properties file

- `generate -generatepassword password`

Description

The Generate command creates a report document with a received set of information from arguments and parameters. The information will then be generated as a report document to the specified output file. By default, an argument is the specified data of the invoked parameters. If the -properties option is specified, the argument is the name of a properties file. A properties file contains other parameters, along with the specified data of each parameter.

Synopsis

Command	Function
`-project projectFileName`	Specifies the MagicDraw project path as a file name.
`-output outputFileName`	Specifies the output file path as a file name.
`-template templateName`	Specifies a template name used to generate a report document. **Note** • The template must exist in MagicDraw prior to using this command, otherwise an error will occur. • If the template name is not specified, the template specified in the Report Data (specified in **-report reportName**) will be used instead.
`-templatePath templatePath`	Specifies a template path used to generate a report document. Both absolute and relative paths are allowed for this command. The specified template file must have proper structure as it has been added through the **Report Wizard** dialog. **Note** If both **-template** and **-templatePath** are specified in the same command, the error will occur.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Command	Function
`-package packageNameList`	Specifies the name of one or more packages in a MagicDraw project to be included in the report. Multiple package entries must be separated by a semicolon (;). **Warning: Duplicate qualified name** Do not enter a duplicate name of a root Model package or its immediate child package to the -package option.
`-element elementNameList`	Specifies the name of one or more elements from a MagicDraw project to be included in a report. Element entries must be separated by a semicolon (;).
`-elementID`	Specifies the element ID of one or more elements in a MagicDraw project to be included in the report. Multiple element entries must be separated by a semicolon (;). **Note** • You can get an element ID by using **$element.elementID**. • The element ID will change when the local project is added to Teamwork Cloud. Please make sure the element ID is up-to-date.
`[options]`	Command-line options.
`-properties propertiesFileName`	Specifies the name of a properties file to utilize. Use only with -properties.
`-generatepassword password`	Generates an encrypted password to be used with a properties file.

Options

The command line feature supports a set of options useful in adding additional configuration possibilities.

Command	Function
`-report reportName`	Specifies the name of a Report Data(see page 0) file or Report Data element. The Report Data file must exist in MagicDraw prior to using this command, otherwise an error will occur. If the reportName is not specified, the previous Report Data will be set and used instead.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Command	Function
`-autoImage 0|1|2|3`	Specifies how an image will be shown in a report document. The default value for this argument is **1**. • 0 – No resize. • 1 – Fit image to paper (large only). • 2 – Fit and rotate image (clockwise) to paper (large only). • 3 – Fit and rotate image (counter-clockwise) to paper (large only)
`-imageFormat jpg|png|svg|emf|wmf`	Specifies an image format in a report document. The default value for this argument is **jpg**. • *jpg* – Joint Photographic Expert Group • *png* – Portable Network Graphics • *svg* – Scaling Vector Image • *emf* – Microsoft Windows Enhanced Metafile • *wmf* – Windows Metafile
`-recursive true|false`	Specifies how to select a package in a MagicDraw project. The default value for this argument is **true**. • **true** – Select the specified package and its recursive package. • **false** – Select only the specified package.
`-includeAuxiliary true|false`	Selects packages, including auxiliary packages. The default value is **false**.
`-outputOnBlankField stringValue`	Shows a string value in a blank field in a report document. The default value for this argument is “”.
`-category categoryName`	Specifies a template category.
`-fields[name=value]`	Specifies a variable.
`-server serverName|IPAddress`	Specifies the name or IP Address of a Teamwork Server project.
`-login loginName`	Specifies a login name to log on to Teamwork Server.
`-password password`	Specifies a password to log on to Teamwork Server.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Command	Function
`-spassword encryptedPassword`	Specifies an encrypted password to log on to the server. This option is available only in a properties file. You can get encrypted password by entering this command below in command line prompt. `generate -generatepassword "enter_your_password_here"`
`-upload serverAddress`	Specifies a serverAddress consisting of a scheme, userInfo, host, port, and path to connect and upload a generated report to a server (see Upload generated report to server(see page 1807)).

Command	Function
`-overwrite true|false`	Overwrites any previously generated files. The default value for this argument is **true**. • **true** – Overwrites all files automatically (removes any existing file before generating a report.) • **false** – If the file exists, the following prompt message will open: `Outputfilealreadyexists.`

Command	Function
	`Would you like to replace the existing file? [y][n]:` If you type ‘y’ or ‘yes’, the Report Wizard will generate a new report that replaces the previous one. If you type ‘n’ or ‘no’, the Report Wizard will not generate a report and terminate the process.
`-pversion projectVersion`	Specifies a Teamwork project version.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Command	Function
`-servertype tw | twcloud`	Specifies a type of server to connect. The default value is **tw**. • **tw** - Connect to Teamwork Server • **twcloud** - Connect to Teamwork Cloud
`-ssl true | false`	Specifies if a Secured Connection (SSL) is used. The default value is **false**. • **true** - Use Secured Connection (SSL) • **false** - Do not use Secured Connection (SSL)
`-tag tagValue`	An additional argument to generate reports from a server project by specifying a server project tag name. The tagValue is non-case sensitive. More than one project version may be returned from specifying one tag value. Hence, one report generation process may generate multiple reports. You cannot use this command with –pversion.
`-incremental true|false`	Specifies the incremental method. The default value is true. • **true** - The output report shall be generated by appending a new generated output file name with the incremental number. • **false** - The latest project version in the returned project collection. Use this command with -tag only.
`-leaveprojectopen`	Leaves the project open per property file after generating the report, in a series of report generation. The default value is **false**.

Using command line

You can use the command line on Windows, Linux, or Unix.

To use the command line on Windows

1. Run a command line console. In the Console window, go to the local installation of the MagicDraw application.
2. Go to the plugins\com.nomagic.magicdraw.reportwizard folder, and type the command line there.

To use the command line on Linux or Unix

1. Run a terminal (command line console). In the Console window, go to the local installation of the MagicDraw application.

2. Go to the plugins\com.nomagic.magicdraw.reportwizard folder, and type the command line there.

See the following examples of command line on Windows and Linux or Unix respectively.

On Windows

```
generate -properties "C:\\output\\prop.properties"
```

On Windows

```
generate -project "C:\\MagicDraw\\samples\\diagrams\\class diagram.mdzip" -output "C:
\\output\\output.rtf" -template "Class Specification Report" -package "Model" -report
"Built-in" -autoImage 1 -imageFormat png -recursive false -outputOnBlankField "#NA"
-fields "[Author=NMA][Company=No Magic]"
```

On Linux or Unix

```
./generate.sh -project "/home/project/project.mdzip" -output "/home/output/
output.rtf" -template "Class Specification Report" -package "Model" -report "Built-
in" -autoImage 1 -imageFormat png -recursive false -outputOnBlankField "#NA" -fields
"[Author=NMA][Company=No Magic]"
```

Generating multiple reports

You can open multiple projects and generate multiple reports before MagicDraw shuts down. The **generate** command with the -properties option accepts multiple properties files as shown below.

```
generate -properties <file1.properties> <file2.properties> <file3.properties>...
```

For example

```
generate -properties “property1.properties” “property2.properties”
```

Note
Template, **output**, **package**, and **project** are the required data for each property file.
The following parameters are reset when finishing generating a report for each property file. Other parameter values affect subsequent property files, if there are no overridden values.

- -project
- -output
- -template
- -templatePath
- -tag
- -pVersion
- -recursive
- -category
- -includeAuxiliary

Tip
When large-size projects need to be loaded and closed for many property files, it takes longer time to generate reports. In this case, you can use **-leaveprojectopen** as an option to reduce memory for loading each project when starting to generate a new report.

```
generate -leaveprojectopen true -properties “property1.properties”
“property2.properties”
```

Note
-leaveprojectopen needs to be specified before *-properties.*

Related pages

- The Generate command to generate reports[350]
- Generating a report from Teamwork Cloud Server(see page 1804)
- Properties filename(see page 1805)
- Uploading generated reports to servers(see page 1807)
- Syntax rules(see page 1810)

Generating a Report from Teamwork Server

You can use the following command arguments in the command line to generate a report from Teamwork Server.

- **-server**, **-login**, and **-password** are key arguments to log on to Teamwork Server. You will be prompted for a password when trying to enter **-server** and **-login** without **-password**.
- If **-spassword** is used with a properties file, an encrypted password must be used for this property.
- **-pversion** is a key argument to specify a Teamwork project version. You can specify a project version after **-pversion**. If either you do not specify any version, or the version you specify does not match any version in Teamwork Server, Report Wizard will generate the latest project version.

350 https://docs.nomagic.com/display/MD190SP4/The+Generate+command+to+generate+reports

- **-tag** is an option to specify a Teamwork project. You can specify tagName to search for a project version. The returned project version may be more than one. You may use this command argument with "**-incremental true**".
- You can specify a project name and its branch after **-project**. For example:
  - "*MyProject*" means generating MyProject without specifying its branch.
  - "*MyProject##release*" means generating MyProject from the branch ["release"].
  - "*MyProject##release##sp1*" means generating MyProject from the branch and sub-branch ["release", "sp1"].

See the following for more examples of how to generate a project from a specific branch and/or a specific version.

Example	Command argument
Generating a project without specifying its branch.	generate -project "MyProject" -template "Web Publisher 2.0" -output "D:\output\output.html" -package "Model" -server "localhost" -login "Administrator" -password "Administrator"
Generating a project from the branch release.	generate -project "MyProject##release" -template "Web Publisher 2.0" -output "D:\output\output.html" -package "Model" -server "localhost" -login "Administrator" -password "Administrator"
Generating a specific version of a project without specifying its branch.	generate -project "MyProject" -pversion "2" -template "Web Publisher 2.0" -output "D:\output\output.html" -package "Model" -server "localhost" -login "Administrator" -password "Administrator"
Generating a specific version of a project from the branch release.	generate -project "MyProject##release" -pversion "2" -template "Web Publisher 2.0" -output "D:\output\output.html" -package "Model" -server "localhost" -login "Administrator" -password "Administrator"
Generating a set of reports from a specific tag.	generate -project "MyProject" -tag "Finish" -incremental true -ouput "C:/output/out.docx" -template "Class Specification Report" -package "Model" -server "localhost" -login "Administrator" -password "Administrator"

Related pages

- Generating a report from Teamwork Cloud Server(see page 1804)
- Properties filename(see page 1805)
- Uploading generated reports to servers(see page 1807)
- Syntax rules(see page 1810)
- Using the Generate command to generate reports(see page 1793)

Generating a report from Teamwork Cloud Server

Apart from the mandatory commands described in using the Generate command to generate reports(see page 1793), you can use the following command arguments in the command line to generate a report from the Teamwork Cloud (TWCloud) server.

- **-server**, **-login**, and **-password** are key arguments to log on to TWCloud server. You will be prompted for a password when trying to enter **-server** and **-login** without a password.
- **-servertype** is for selecting a connecting server. "**twcloud**" is a required value for this argument. It is for getting a project from TWCloud server. If either you do not specify any server type, or the value is "**tw**", the Teamwork server will be connected.
- **-pversion** is a key argument to specify a TWCloud project version. If either you do not specify any version, or the version you specify does not match any version in the TWCloud repository, Report Wizard will generate the latest project version of the specified branch.
- **-branch** option is used to retrieve a particular project branch. If you do not specify any branch, the trunk would be specified.
- **-updatemodule** A Boolean parameter. True for updating the used module of the project to the latest version. False by default, when the user does not specify this option in the command.
- **-tag** is an option to specify a Teamwork project. You can specify the tagName to search for a project version. The returned project version may be more than one. You may use this command argument with "**-incremental true**". If you use this command with **-branch,** it will return a matching project version in the specified branch.
- **-projectpassword** is required to specify a password to open a password-protected Teamwork Cloud project. The password is required for the project to generate a report.
- **-usagepackage** specifies the name of one or more packages in Project Usages of a TWCloud project to be included in the report. Package entries must be separated by a semi-colon (;).
- **-usageelement** specifies the name of one or more elements in the Project Usages of TWCloud pr oject to be included in the report. Element entries must be separated by a semi-colon (;).

> **Note**
> For TWCloud projects, **-package** / **-element** is used to select one or more packages/ elements only in the Primary Model of TWCloud projects. To select packages/elements in Project Usages, specify **-usagepackage** / **-usageelement**. You can also set **-includeAuxiliary** as true if you want to make auxiliary resources available for selection. This **-usagepackage** / **-usageelement** argument can be used with **-package** / **-element** to select packages/elements in both Primary Model and Project Usages.

If, for example, you want to generate a Web Publisher 2.0 report from a TWCloud project by getting the latest project version of the trunk branch, type the following:

```
generate -server "localhost" -login "Administrator" -password "Administrator"
-servertype "twcloud" -project "MyProject" -package "SamplePackage" -template "Web
Publisher 2.0" -output "D:\output\output.html"
```

The following code generates a Class Specification Report report from the TWCloud project by getting the project version number 10 from MyBranch branch of the MyProject project.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

```
generate -server "localhost" -login "Administrator" -password "Administrator"
-servertype "twcloud" -project "MyProject" -branch "MyBranch" -pversion "10" -package
"SamplePackage" -template "Web Publisher 2.0" -output "D:\output\output.html"
```

The following code generates a specific project version by specifying tagName.

```
generate -project "MyProject" -branch "TestingLine" -tag "Finish" -incremental true
-ouput "C:/output/out.docx" -template "Class Specification Report" -package
"SamplePackage" -server "localhost" -servertype "twcloud" -login "Administrator"
-password "Administrator"
```

Tip
The specified project version would be used to generate the report as a first priority, in case the specified project version does not belong in the specified branch.

Note
If the project version specified is not the latest, the **-updatemodule** option would not affect the **generate** command.

Related pages

- Generating a report from Teamwork Cloud Server(see page 1804)
- Properties filename(see page 1805)
- Uploading generated reports to servers(see page 1807)
- Syntax rules(see page 1810)
- Using the Generate command to generate reports(see page 1793)

Properties filename

On this page

- XML Properties File(see page 1806)

When you use -properties, other command actions are no longer necessary; the properties file contains the configuration information. The information in the properties file is the same as the information used in the command line.

Four parameters are needed to specify data for a report: **project**, **output**, **template**, and **package**. There are five more parameters called options, which are additional configuration information such as **report**, **autoImage**, **imageFormat**, **recursive**, and **outputOnBlankField**. All nine parameters are described in Generate - the Command to Generate Reports(see page 1793).

A properties file is a simple text file. You can create and maintain a properties file with any text editors (see the following example).

Properties file sample

```
#----main argument ----#
project = C:\\MagicDraw\\samples\\diagrams\\class diagram.mdzip
output = C:\\output\\output.rtf
template = Class Specification Report
package = java;Library System;magiclibrary;objects


#---- optional ----#
report = Built-in
autoImage = 1
imageFormat = png
recursive = false
outputOnBlankField = #NA
```

In the figure above, the command lines begin with a pound sign (#). The other lines contain key-value pairs. The key is on the left side of the equal sign, and the value is on the right. For instance, template is the key that corresponds to the value of the **Class Specification Report**.

The string representing the key and value in the properties file has a special character. The special character is an escape character. It must be used to prevent interpretation errors. Each escape character starts with a backslash.

Escape Sequence	Character
\n	new line
\t	tab
\b	backspace
\f	form feed
\r	return
\”	“ (double quote)
\’	‘ (single quote)
\\	\ (backslash)
\uDDDD	character from the Unicode character set (DDDD is four hex digits)

XML Properties File

J2SE version 1.5 and Report Wizard version 15.5 allow you to use XML files to load and save key-value pairs.

Properties DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- DTD for properties -->
<!ELEMENT properties ( comment?, entry* ) >
<!ATTLIST properties version CDATA #FIXED "1.0">
<!ELEMENT comment (#PCDATA) >
<!ELEMENT entry (#PCDATA) >
<!ATTLIST entry key CDATA #REQUIRED>
```

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPEproperties SYSTEM
"http://java.sun.com/dtd/properties.dtd">
<properties>
    <entry key = "project">C:\MagicDraw\samples\diagrams\class diagram.mdzip</entry>
    <entry key = "output">C:\\output\\output.rtf<\entry>
    <entry key = "template">Class Specification Report</entry>
    <entry key = "package">java;Library system;magiclibrary;objects</entry>
</properties>
```

Related pages

- Properties filename(see page 1805)
- Generating a report from Teamwork Cloud Server(see page 1804)
- Uploading generated reports to servers(see page 1807)
- Syntax rules(see page 1810)
- The Generate command to generate reports[351]

Uploading generated reports to servers

The available command arguments include:

- **-upload** is the key argument to upload a generated report to a server.

The following code is the uploaded command syntax of the upload value.

```
-upload
"{scheme}://[{userInfo}@]{host}[:{port}][/{path}][?{query}][#{fragment}]"
```

where square brackets [...] are delineated as optional.

- **{scheme}**
  The {scheme} defines the namespace, purpose, and the syntax of the remaining part of a URL. This field is mandatory. A scheme must be followed by "://". The supported schemes are:
  - ftp
  - ftps

351 https://docs.nomagic.com/display/MD190SP4/The+Generate+command+to+generate+reports

- sftp
- http
- https

- **{userInfo}**
  The {userInfo} defines the username and password for authentication. The syntax is displayed as follows.

```
{username}[:{password}]
```

  This optional password must be predicated by ":", and it can be an empty string. If the password is omitted, the command line will prompt for the password.
  UserInfo must be followed by "@", and this field is optional.

  > **Warning**
  > {username} and {password} should not contain special characters such as ":", "/", and "@" for they may cause an invalid result or error when the uploaded command is parsed.

- **{host}**
  The {host} defines a host name or an IP address that gives the destination location of a URL. This field is mandatory.
- **{port}**
  The {port} defines a port number where the server is listed. The port value has to be a value from 1 to 65535. It is necessary when the port number is different from the default port for well-known services. If the port is omitted, the default port will be used to connect to the server. The table below shows a port number for each scheme.
  The table below lists the Default Port Numbers for Schemes

Scheme	Port Number
ftp	21
ftps	990
sftp	22
http	80
https	443

The port must be preceded by ":". This field is optional.

- **{path}**
  The {path} defines a remote location where the report will be saved. Paths are Unix-style paths; therefore use "/" (forward-slash) as delimiters. This field is optional.
- **{query}**
  The {query} defines a query string that contains data to be passed to software running on the server. It may contain name or value pairs separated by ampersands.
- **{fragment}**
  The {fragment} defines a fragment identifier that, if present, specifies a part or a position within the overall resource or document.

> **Info**
> For this feature, the parser ignores query and fragment. Both fields are included in {path}.

The following are some examples of valid server addresses.

- ftp://magicreport:1234@10.1.2.4:25/report

Scheme	ftp
User name	magicreport
Password	1234
Host	10.1.2.4
Port	25
Path	report

- ftp://10.1.2.4:25/report

Scheme	ftp
Host	10.1.2.4
Port	25
Path	report

- ftp://10.1.2.4

Scheme	ftp
Host	10.1.2.4
Port	21 (Default port for ftp)

- ftp://magicreport@10.1.2.4:25/report

Scheme	ftp
User name	magicreport
Host	10.1.2.4
Port	25
Path	report

> **Tip**
> The command line prompts for password.

- ftp://magicreport:@10.1.2.4:25/report

Scheme	ftp
User name	magicreport
Password	
Host	10.1.2.4
Port	25
Path	report

> **Tip**
> The command line uses an empty password.

The following are some examples of invalid server addresses.

Invalid Server Address	Cause
10.1.2.4:25	The scheme is required.
Xx://10.1.2.4:25	The scheme is invalid.
ftp://magicreport@	The host is required.
ftp://@10.1.2.4:25	The username is required.

Related pages

- Uploading generated reports to servers(see page 1807)
- Generating a report from Teamwork Cloud Server(see page 1804)
- Properties filename(see page 1805)
- Syntax rules(see page 1810)
- Using the Generate command to generate reports(see page 1793)

Syntax rules

- Command parameters containing spaces must be enclosed in quotes, for example, "UML Standard Profile".
- Command parameters are case sensitive, except project, imageFormat and recursive which are not case-sensitive.
- Blank spaces (" ") separate commands and parameters.
- To specify a package name that contains a semicolon (;), you can use a backslash (\). For example, the package name "com;nomagic" can be typed by using "com\;nomagic". The backslash followed by a semicolon (\;) is not considered to be a character separator.
- For a parameter that contains unicode characters such as Thai, you can use the xml properties file to generate a report. The properties file does not support the unicode encoding.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- A field entry consists of a name-value pair. The name and value formats are specified by [name=value] pattern strings. A valid name must start with a letter (a-z or A-Z) and can be followed by any letter, digit, or underscore.
- You can specify a field name that consists of brackets ([ ]) by using backslashes (\ ). For example, "[author=NoMagic][Revision=[1.0]]" will be typed as "[author=NoMagic][Revision=\[1.0\]]".
- Either -package or -element can be used with the Generate command. For example, you can use -package without -element.
- If -upload is specified to the command line. Its value is validated and executed after a report has been generated.

Related pages

- Syntax rules(see page 1810)
- Generating a report from Teamwork Cloud Server(see page 1804)
- Properties filename(see page 1805)
- Uploading generated reports to servers(see page 1807)
- Using the Generate command to generate reports(see page 1793)

Configuring Report Wizard

Report Wizard Environment options

The Report Wizard Environment options in the MagicDraw **Environment Options** dialog allow you to configure the report engine. There are three things that you can configure: (i) Report engine properties, (ii) Template mappings, and (iii) Template folder.

(i) Report engine properties

Report engine properties can be configured as follows:

- **Template process size (template.process.size)**
  This integer value defines the size of allocated memory for processing a template. If the template is larger than the allocated size, the engine will create temporary files on a local disk and process the template from these files.

- **Template pool size (template.pool.size)**
  This integer value defines the number of processing threads for evaluating a template. An increase in the number of threads may improve the performance of the engine, which will also depend on your hardware and JVM.

- **Velocity properties**
  These properties are a key-value pair. You can add any number for these properties. You will need a basic knowledge of Velocity to enter the value. (See http://velocity.apache.org/engine/devel/apidocs/org/apache/velocity/runtime/RuntimeConstants.html)
  - velocimacro.max.depth

    - velocimacro.context.localscope

- **Debug report template**.
  You can configure this value to enable or disable any invalid properties, references, or exception messages on the message window of MagicDraw. See Debug report template(see page 1820) for more details.
    - template.warn.invalid.property
    - template.warn.invalid.reference
    - template.warn.invalid.method
    - template.warn.exception

- **rememberOutputOptions**.
  This value determines whether Report Wizard will remember output options of the report file when generating report. **Yes** if this value is true, and **No** if this value is false. See the last figure on this page.

  Other properties that are not in the **Configure report properties** table are as follows:

    - template.rtf.newline
      Specify the new line character for the RTF documentation. The values are **line** (by default) and **paragraph**.

    - template.docx.ignorechildstyle
      A key for ignoring styles from included templates in the generated report. When **False** (by default), the styles from included templates are included in the generated report. When **True**, the styles from included templates are not included in the generated report, and matching styles in the main template are applied instead.

    - template.xlsx.html.link
      A key for selecting the first or last hyperlink in the HTML string of any cell and applying the selected hyperlink to the cell. The values are **first**, **last**, and **skip**.

    - element.additional.include
      Use an additional element and owned element to get children of the element.

    - template.odp.use.virtual.name
      A key for enabling the engine to replace an ODP hyperlink with a page name.

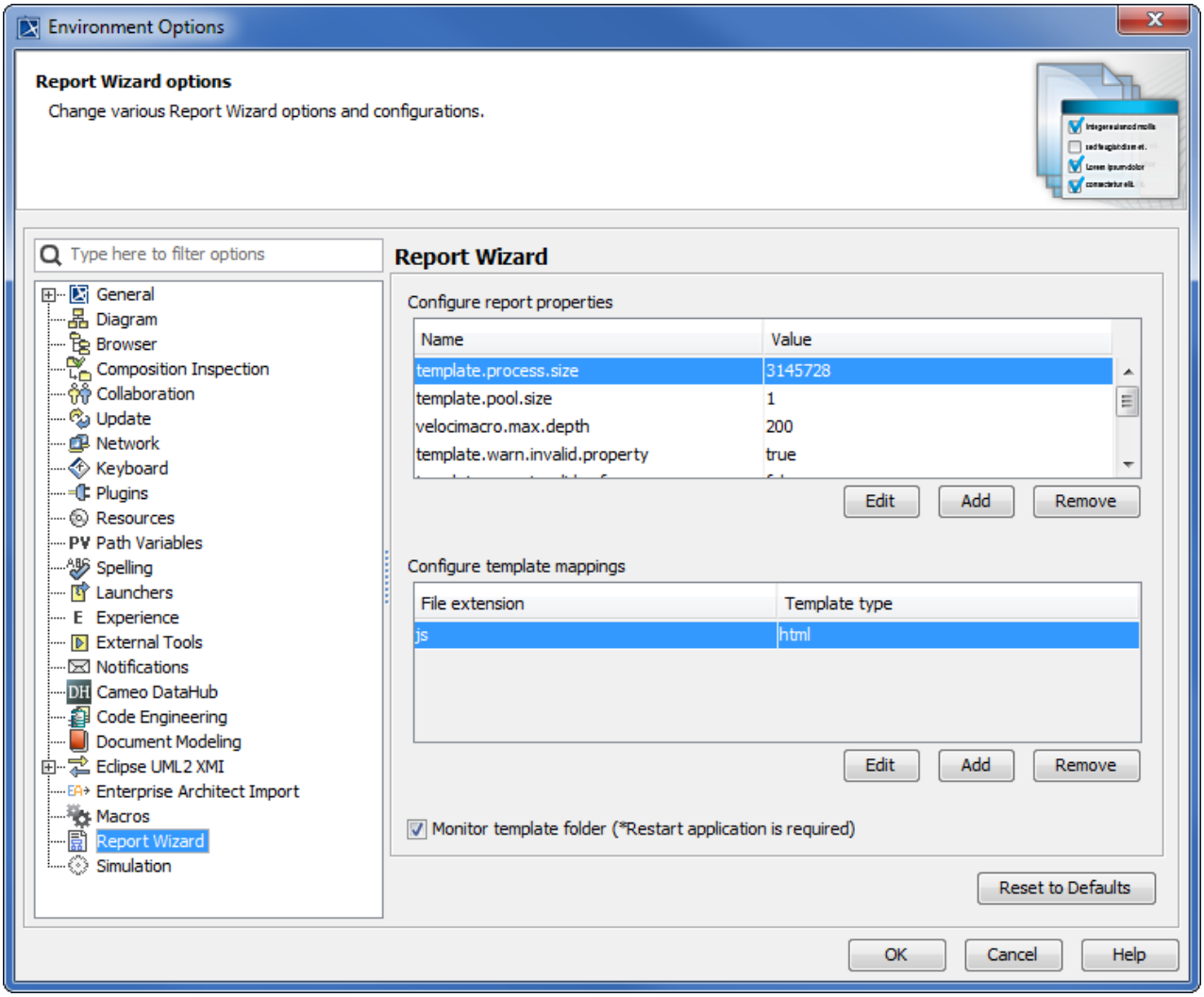
(ii) Template mappings

Template mappings associate a template type with the template engine. The Report engine allows you to determine how to handle different types of files, such as JS or DOCBOOK files. All you have to do is specify a filename extension and select an engine name. The Report engine also allows developers to add a user custom engine to this option.

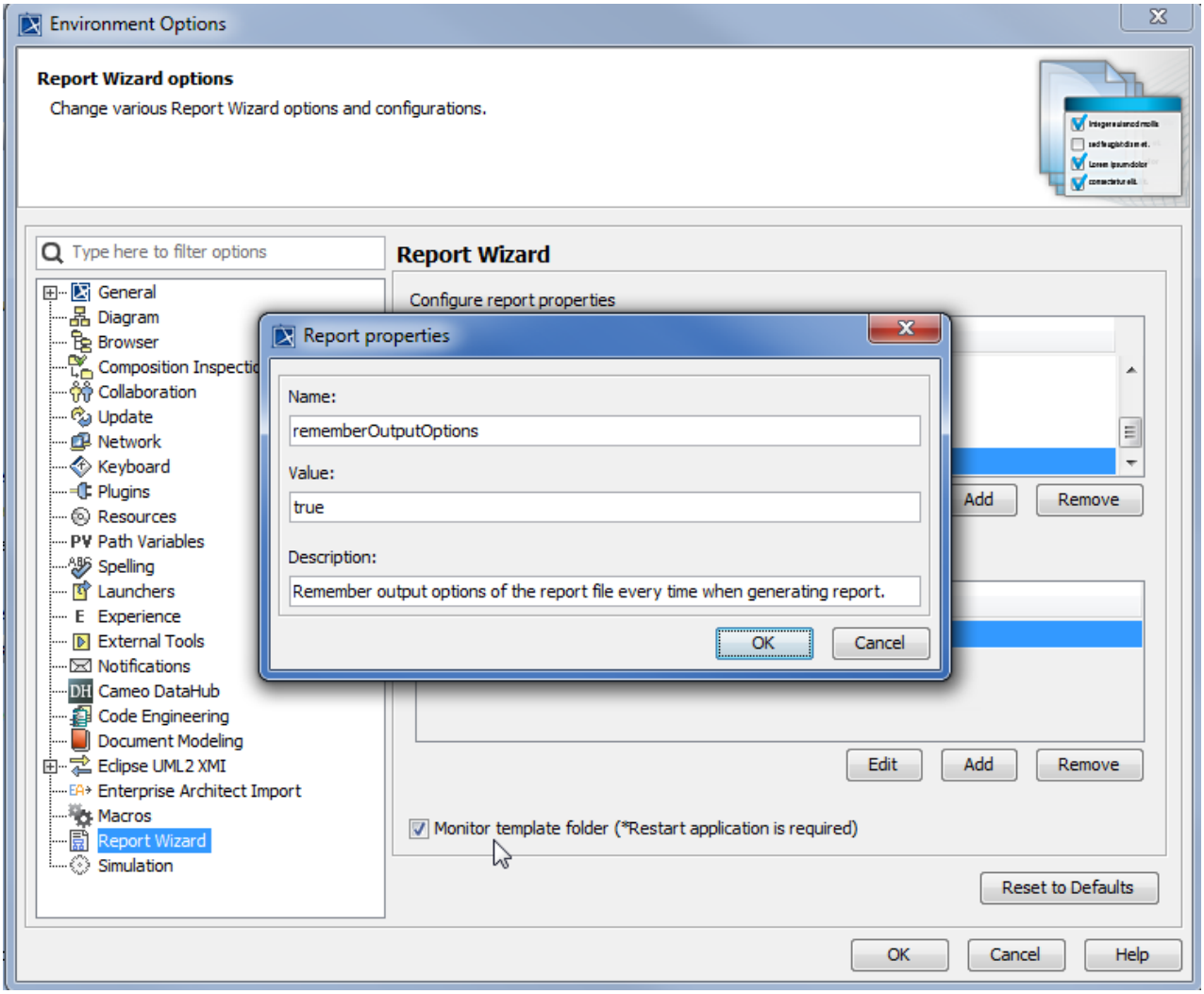
(iii) Template folder

You can either select or clear the **Monitor template folder** check box in the Report Wizard Environment options in the MagicDraw **Environments Options** dialog to enable or disable the option to deploy an MRZIP template file. If you select the check box, Report Wizard will automatically deploy the MRZIP template file to the Report Wizard dialog whenever the file is added to your folder.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Report Wizard GUI configuration in the MagicDraw Environment Options dialog.



The Report properties dialog in the Report Wizard Environment options.

> **Note**
> After updating the Report Wizard Environment options, restart MagicDraw to take effect of the changes.

Related pages

Reset to defaults option

Click the **Reset to Defaults** button to reset data to the default settings, see figure below.

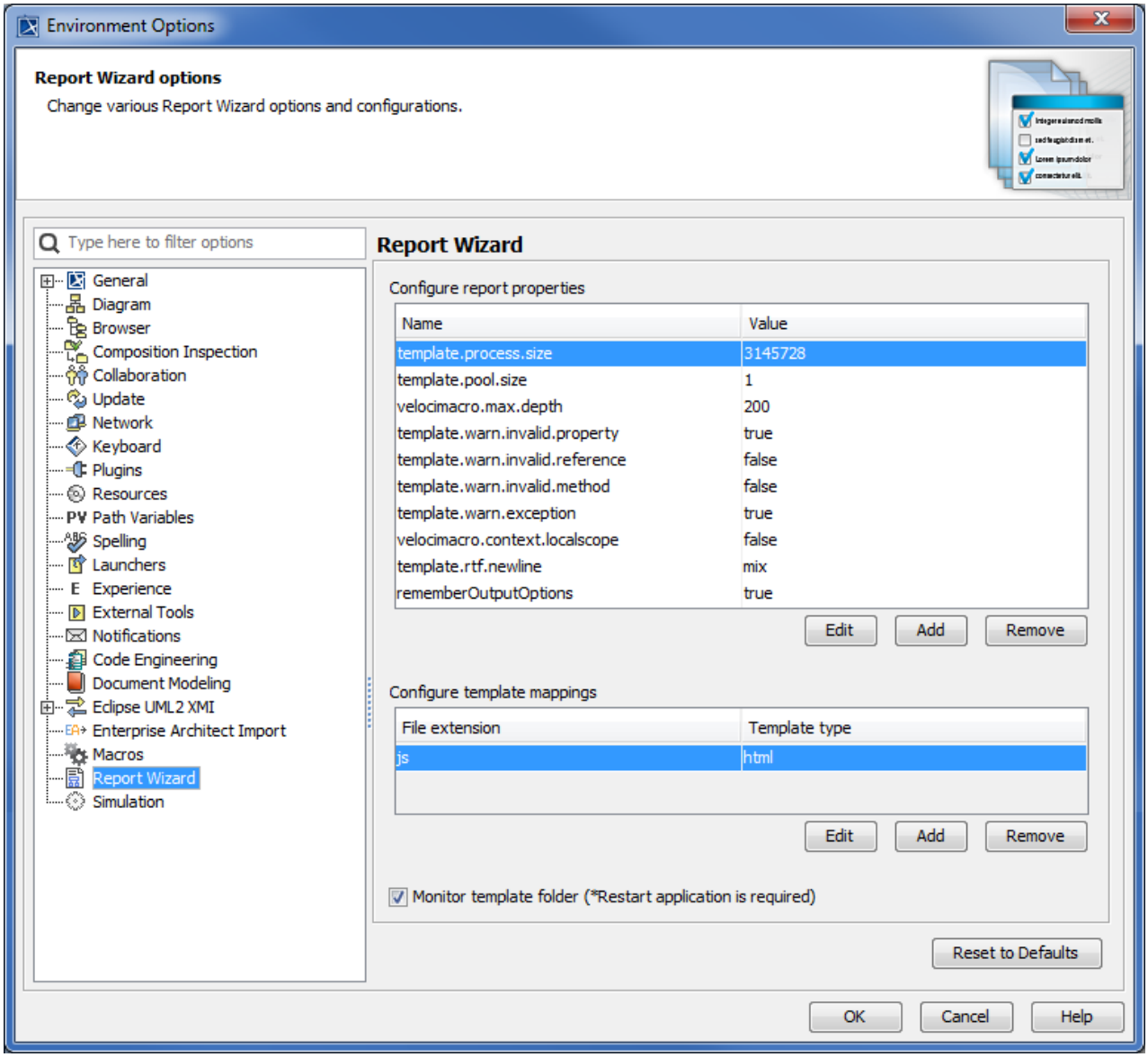
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

The Reset to Defaults button.

- MagicDraw allows you to configure the report engine settings in three levels: (i) Environment option, (ii) global *config.xml*, and (iii) template *config.xml*.
- The template *config.xml* configuration settings will override the environment option configuration settings, which will override the global *config.xml* configuration settings.

Configuring report engine properties

You can configure the report properties of Report Wizard in the **Environment Options** dialog.



The Configure report properties pane.

> **Tip**
> You can select a report property in the **Configure report properties** pane, and click Edit to see its description.

To edit, add, or delete a report property

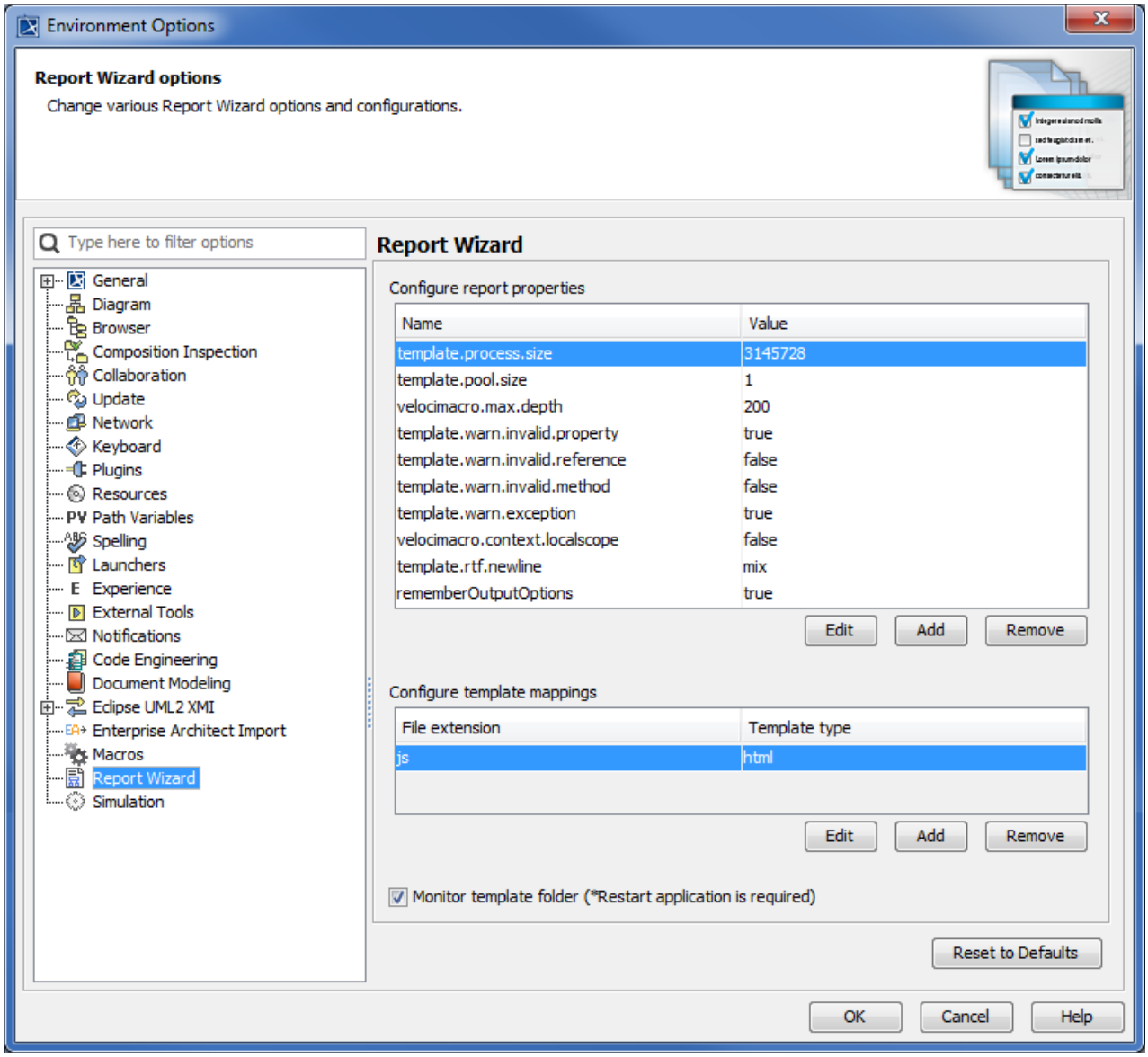
1. Click **Options** > **Environment** to open the **Environment Options** dialog.
2. Select **Report Wizard** from the tree view to open the **Configure report properties** pane.
3. Click:
   Edit to edit a selected property name and value in the **Report Properties** dialog, and click OK.
   Add to create a property name and value in the **Report Properties** dialog, and click OK.
   Remove to delete a selected property name and value from the **Configure report properties** pane.

The following table lists the fields of the **Report Properties** dialog:

Field Name	Function	Default Value	Possible Value	Required
Name	The property name	Blank	Text	Yes
Value	The property value	Blank	Text	No
Description	The property description		Text	No

Configuring template mappings

You can use the **Environment Options** dialog to configure the template mappings of Report Wizard.



The Configure template mappings pane.

To edit, create, or delete a template mapping

1. Click **Options** > **Environment**. The **Environment Options** dialog will open.
2. Select **Report Wizard** from the tree view. The **Configure template mappings** pane will open.
3. Click:
   Edit to edit a selected filename extension and template type in the **Template mappings** dialog, and click OK.
   Add to create a filename extension and template type in the **Template mappings** dialog, and click OK.
   Remove to delete a selected filename extension and template type from the **Configure template mappings** pane.

The following table lists the fields of the **Template mappings** dialog:

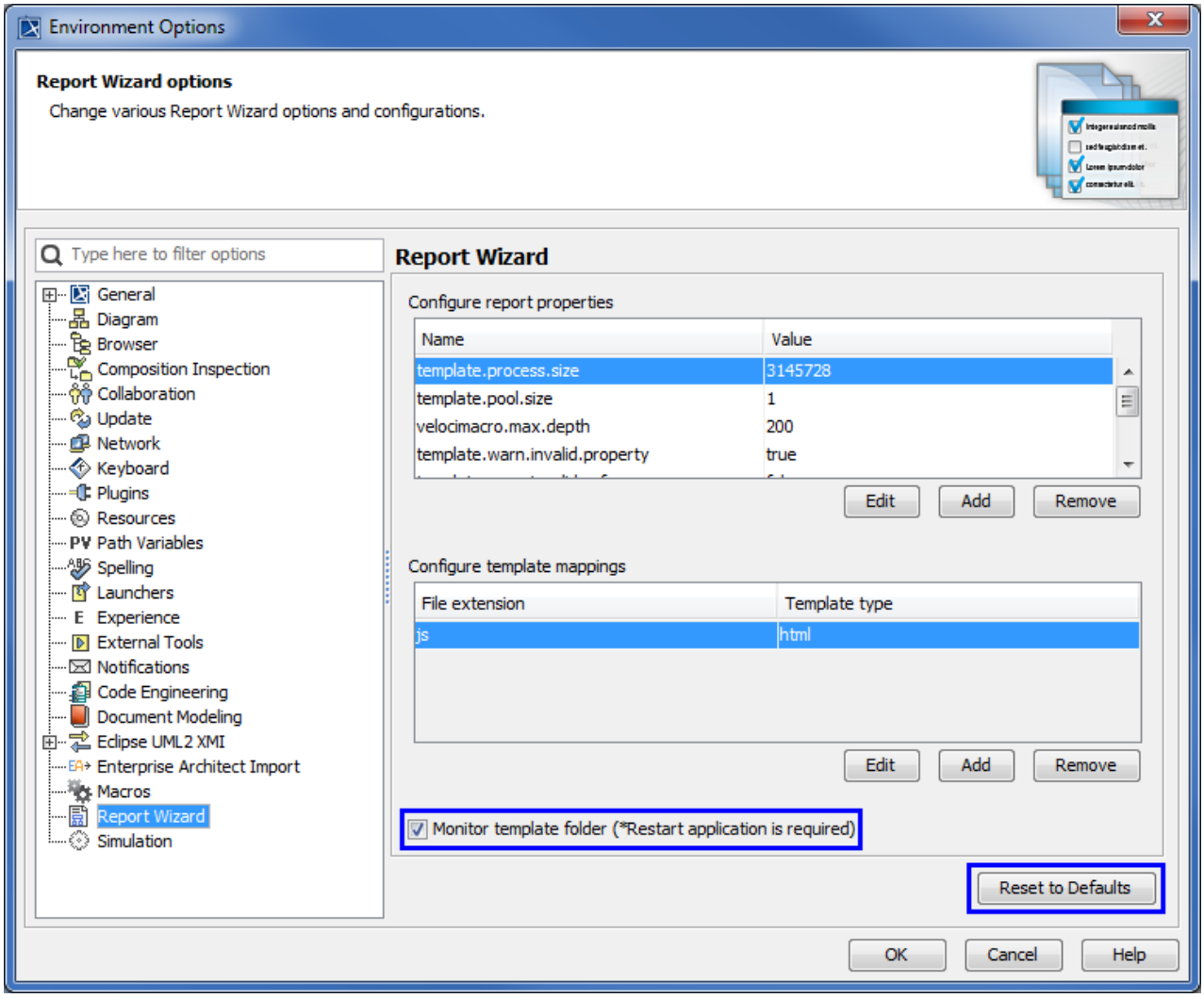
Field Name	Function	Default Value	Possible Value	Required
File extension	The filename extension (not case-sensitive).	Blank	Text	Yes
Template type	The template type. You can select any of the following: html, odp, docbook, dbk, odt, ods, docx.	Blank	Text	Yes

Warning
Deleting a template mapping from the **Environment Options** dialog will cause the file extension mapping to be removed from MagicDraw. To restore the default setting, restart MagicDraw.

Monitor template folder and reset to defaults options

You can select the Report Wizard options for monitoring template folder, and resetting to defaults options in the **Environment Options** dialog.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The Monitor template folder and Reset to Defaults options of Report Wizard in the MagicDraw's Environment Options dialog.

Monitor template folder

You can either enable or disable the MRZIP template file automatic deployment option in the Report Wizard environment options by selecting or clearing the **Monitor template folder** check box.

Reset to defaults

MagicDraw allows you to configure the report engine settings in three levels: Environment Options, global *config.xml*, and template *config.xml*. The template *config.xml* configuration settings will override the **Environment Options** configuration settings, which will override the global *config.xml* configuration settings.

If you have made changes to the settings in the Report Wizard's **Environment Options** dialog, but would then like to cancel them, you can click Reset to Defaults to reset data to the default settings.

Adding more report data directories

The default report data directories for operating systems are as follows:

- For Windows:

```
C:\Users\<username>\AppData\Local\.magicdraw\<version>\data\reports
```

- For Linux:

```
<user.home>/.<modeling tool name>/< modeling tool version number>
```

Report Wizard provides a configuration to add more report data directories by creating a **report.properties** file in the Report Wizard plugin directory, *<MagicDraw>\plugins\com.nomagic.magicdraw.reportwizard*.

You can declare the **template_loc** parameter in the **report.properties** file to specify another report data directory as follows:

```
template_loc=C:/Users/MD_user/AppData/Local/.magicdraw/new Location
```

You can also specify the **template_loc** parameter for multiple report data directories by using a semicolon (;) to separate each of the directories.

```
template_loc=//template_server/MagicReport/;C:/Users/MD_user/AppData/
Local/.magicdraw/new Location
```

All of templates in the default report data directory and other locations specified in the **report.properties** file are available for generating in the **Report Wizard** dialog(see page 1547) and can be called through the **-template** parameter in command line report generation.

> **Note**
> The template names defined in the **template.xml** file in each report data directory must be unique.

Related pages

- Configuring Report(see page 1811) Wizard(see page 1811)
- Report Wizard Environment options(see page 1811)

Debug report template

The Debug Report template features warning messages that describe template errors. A warning message will appear in a Messages Window while the report template is being processed. Warning messages have no effect on the output report, thus can be ignored. They are intended for informative purposes only.

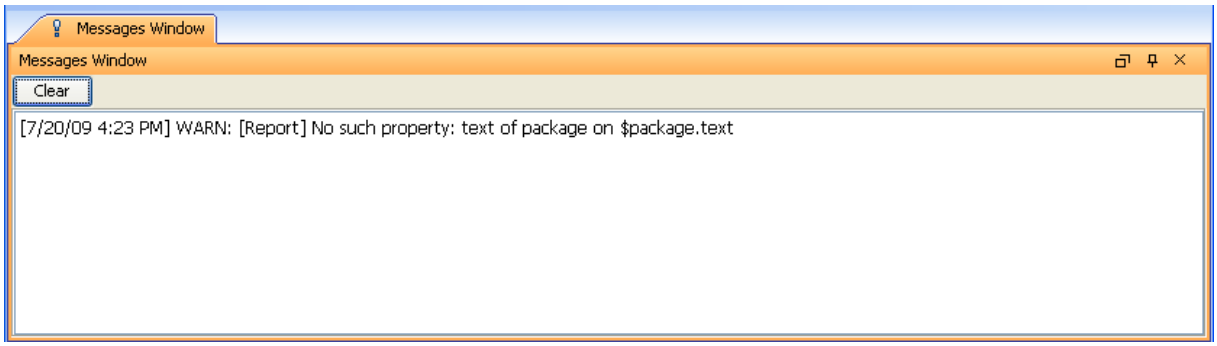
There are 5 warning message types:

1. Invalid Property
2. Invalid Reference
3. Invalid Method Reference
4. Exception
5. Invalid Syntax

Related pages

Invalid property

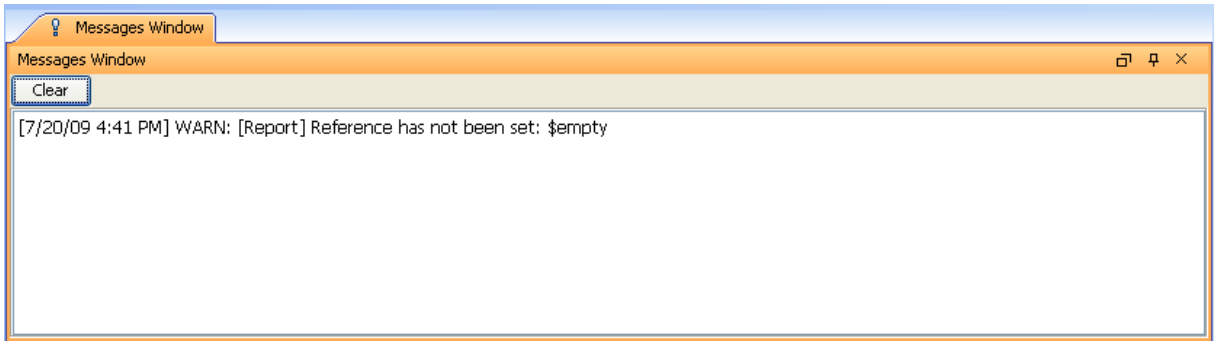
The warning, see figure below will open when an invalid property is encountered during the report generation. For example, getting a text property from the **$package** variable, when the text property is not a valid property for **$package**, is a variable with an invalid property that will trigger a warning.



Invalid Property Warning Message.

Invalid reference

The warning will open, see the figure below, whenever using a variable, for example, **$empty** that has not been specified.

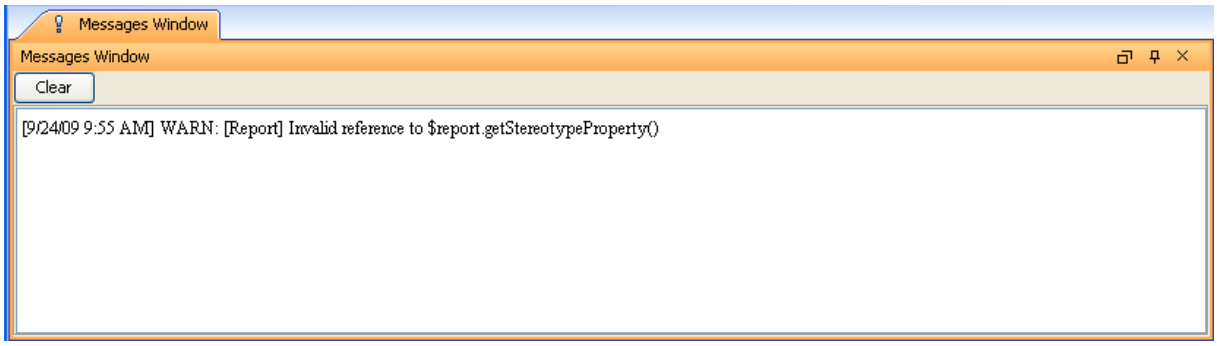


Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Invalid Reference Warning Message.

Invalid method reference

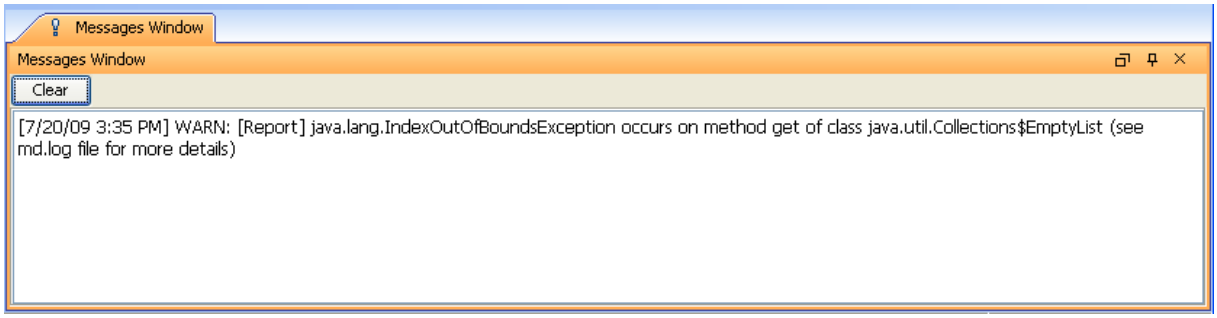
The warning message will open, see the figure below, when the use of an invalid method from a variable is encountered during the report generation. In this example, the "**getStereotypeProperty**" method is not a valid method of the **$report** variable.



Invalid Method Reference Warning Message.

Exception

The warning message, see the figure below, will open whenever an exception, such as IndexOutOfBoundException, occurs during the report generation.

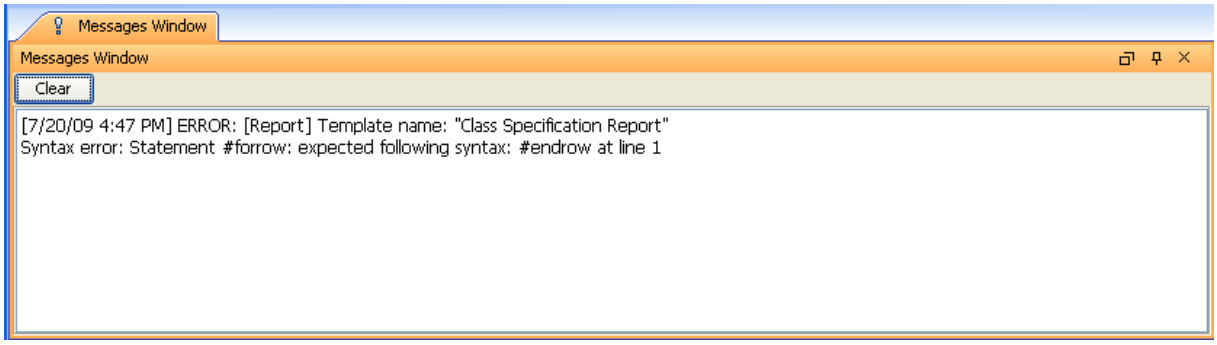


Exception Warning Message.

Invalid syntax

The warning message, see the figure below, will open whenever a template contains any invalid syntax, such as using **#forrow** without **#endrow**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Invalid Syntax Warning Message.

Enabling or Disabling Warning Messages

You can enable or disable any invalid properties, references, and exception messages except invalid syntax messages by editing the config.xml file and tags and values. Invalid syntax messages are always enabled.

Modifying config.xml

To modify the config.xml file

- Either (i) edit the *config.xml* file in the *...\plugins\com.nomagic.magicdraw.reportwizard\data* folder to enable or disable all templates' warning messages or (ii) edit the *config.xml* file or create a file in the *...\plugins\com.nomagic.magicdraw.reportwizard\data\templatefolder* folder for each template.

For example, to enable or disable a warning message of Class Specification Report you need to edit the *config.xml* file in the *...\plugins\com.nomagic.magicdraw.reportwizard\data\Class* Specification Report folder.

Adding Tags and Values

```
<tag>value</tag>
```

The table below lists the Warning Message Types.

Warning Message Type	Tag
Invalid Property	<template.warn.invalid.property>
Invalid Reference	<template.warn.invalid.reference>
Invalid Method Reference	<template.warn.invalid.method>
Exception	<template.warn.exception>

Set this boolean value to:

- "true" to enable warning messages, or

- “false” to disable warning messages.

An example of how an Invalid Reference warning message of Class Specification Report can be enabled is shown in the figure below.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<configuration>
<template.warn.invalid.reference>true</template.warn.invalid.reference>
</configuration>
```

Use Case driven

Report Wizard provides a set of templates to support the Use Case Driven approach methodology. A MagicDraw project that creates the Use Case Driven approach development steps can generate a report to capture elements.

Report Wizard provides 3 templates for the Use Case Driven approach:

1. Use Case Specification Report
2. Method Specification Report
3. Use Case Project Estimation Report

You can create a new MagicDraw project using the Use Case Driven approach either as:

(i) A blank project with the *UseCase_Profile.xml* module or
(ii) A Use Case project

To create a new project as a blank project using *UseCase_Profile.xml*

---

1. On the main menu, click **File**.
2. Click **Use Module** . The **Use Module** dialog will open.
3. Select the *UseCase_Profile.xml* module.
4. Click **Finish**. The MagicDraw project will apply the **Use Case Description** profile.

To create a new project as a Use Case Project

- A MagicDraw project will apply the Use Case Description profile automatically.

Related pages

Use Case specification report

The Use Case Specification report shows the details of actors and use cases in the project. The first part, which is the actor summary, shows the list of all actors and their associated use cases. The second part presents the use case specification in general information of relations and scenarios. The report supports use case and SysML use case diagrams.

Method specification report

The purpose of the Method Specification report is to show the method design and its related use cases in detail to developers.

The report shows a class diagram and a list of all classes in the diagram. The list contains attributes and methods for each class. The report also shows the method specification, parameters, and algorithms.

Use Case project estimation report

The use case project estimation or 'Use Case Points' is a method for sizing and estimating projects developed with the object-oriented method, developed by Gustav Karner of Objectory (now Rational Software). The method is an extension of Function Point Analysis and Mk II Function Point Analysis (an adaption of FPA mainly used in the UK).
The Use Case Project Estimation report is intended to estimate the project size and duration of hourly manpower required based on the use case model. The report supports use case and SysML use case diagrams.

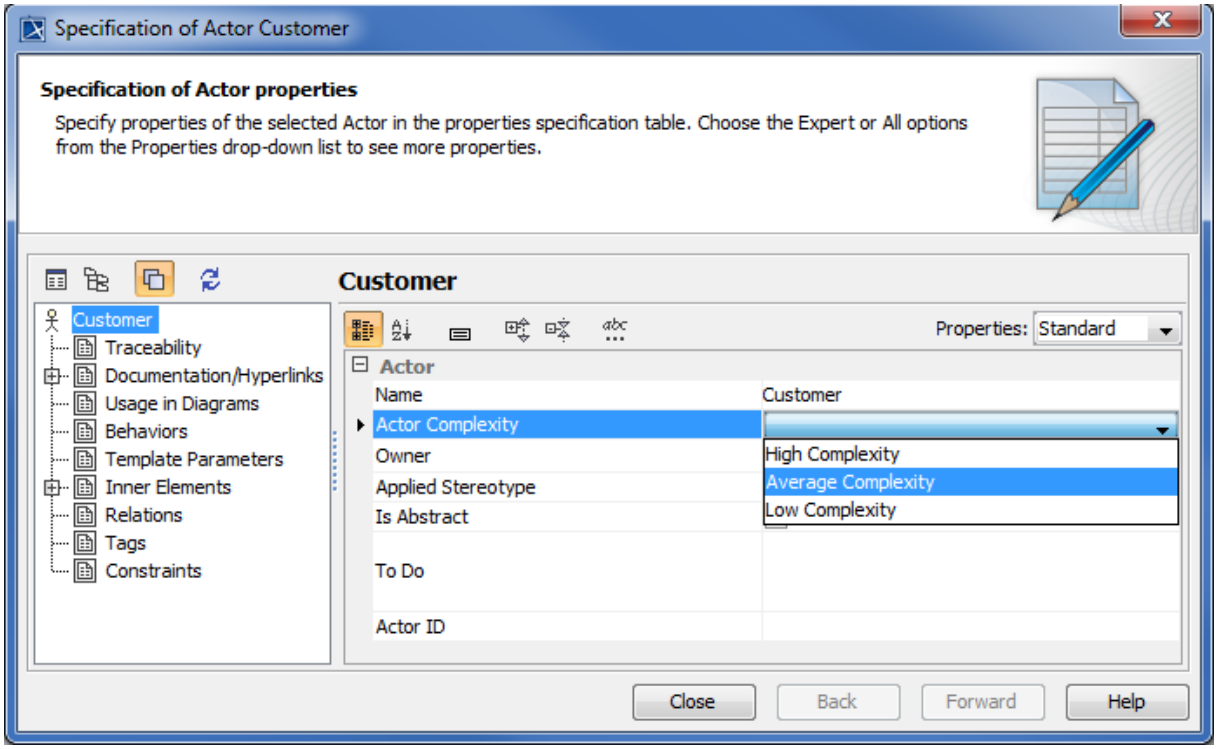
The use case point technique is applied to reestimate the project size. All the use cases used in the example below are based on the use case diagram. The following weight criteria are used:

Classifying Actors

The table below lists Actors Classifications

Actor Complexity	Litmus Test to recognize classifications	Weight
Low Complexity	An external system with a well-defined API.	1
Average complexity	Either a human with a command line interface or an external system via some protocols or a database.	2
High complexity	A human with a GUI or a web page.	3

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Specification of Actor.

Unadjusted Actor Weights

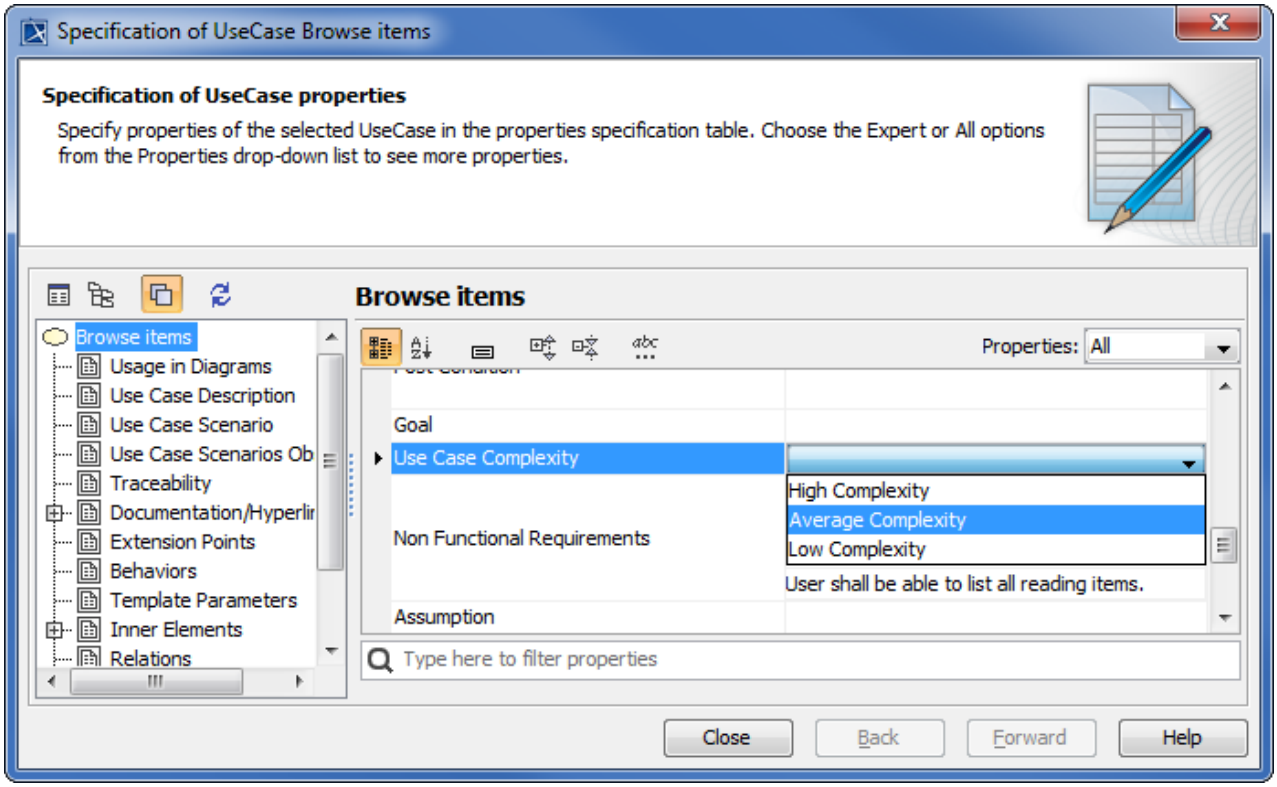
Unadjusted Actor Weights (UAW) is obtained by counting how many actors there are in each category, multiplying each total by its weight, and adding up the products.

Determining Scenarios and Transactions of Use Cases

The table below lists Use Case Complexity

Use Case Complexity	Litmus Test To Decide Classifications	Weight
Low complexity	1 - 3 transactions	5
Average complexity	4 - 7 transactions	10
High complexity	> 7 transactions	15

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Use Case Driven - Use Case Complexity.

Unadjusted Use Case Weights

A transaction is a set of activities, which is either performed entirely or not at all. To determine the number of transactions, you need to count the use case steps. Next, multiply each use case type by the weighting factor and add up the products to get **Unadjusted Use Case Weights (UUCW)**.

Unadjusted Use Case Point

Add UAW and UUCW to get Unadjusted Use Case Point (UUCP): UUCP = UAW + UUCW.

Reference Documents

- The Estimation of Effort Based on Use Cases, published by Rational software
  ftp://ftp.software.ibm.com/software/rational/web/whitepapers/2003/finalTP171.pdf
- COCOMO II
  http://sunset.usc.edu/csse/research/COCOMOII/cocomo_main.html

Project characteristics

The Technical and Environmental factors are associated with the weight of the project. You have to assign a value to each factor. The value assigned to a particular factor depends on the degree of influence a factor has. The relevance values range from 0 to 5, where 0 means no influence, 3 is the average level of influence, and 5 means a strong influence.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

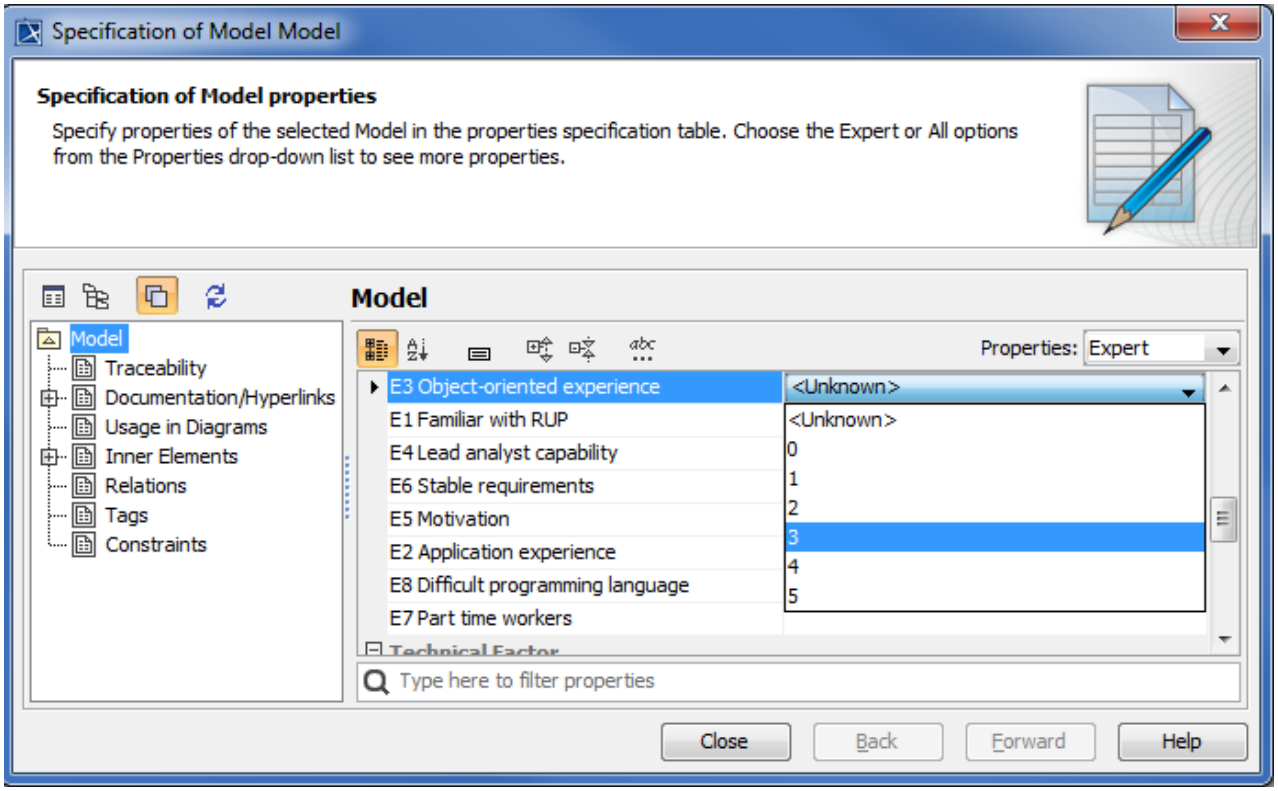
Technical Factors

The table below lists Technical Factors

Factor	Technical Factors	Weight	Description
T1	Distributed system	2.0	The system distributed architecture or centralized architecture.
T2	Response adjectives	1.0	The response time is one of the important criteria.
T3	End-user efficiency	1.0	The end-user efficiency.
T4	Complex processing	1.0	The business process is very complex.
T5	Reusable code	1.0	The project maintains high reusability. The design is complex.
T6	Easy to install	0.5	The project requires simple installation.
T7	Easy to use	0.5	User-friendly is one of the important criteria.
T8	Portable	2.0	The project requires cross-platform implementation.
T9	Easy to change	1.0	The project is highly customizable in the future. The architectural design is complex.
T10	Concurrent	1.0	The project has a large numbers of users working with locking support. The architecture increases the project complexity.
T11	Security features	1.0	The project has heavy security.
T12	Access for third parties	1.0	The project is dependent on the third party's control. Studying and understanding the third party is required.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Factor	Technical Factors	Weight	Description
T13	Special training required	1.0	The application is so complex for the user that training must be provided.



Use Case Driven Model Data E3.

Technical Factor Value

Technical Factor Value (TFactor) is obtained with the multiplication of the value of each Technical factor by its weight.

```
TFactor = value x weight
```

Technical Complexity Factor

Technical Complexity Factor (TCF) is obtained with the addition of 0.6 to the sum of TFactor multiplied by 0.01.

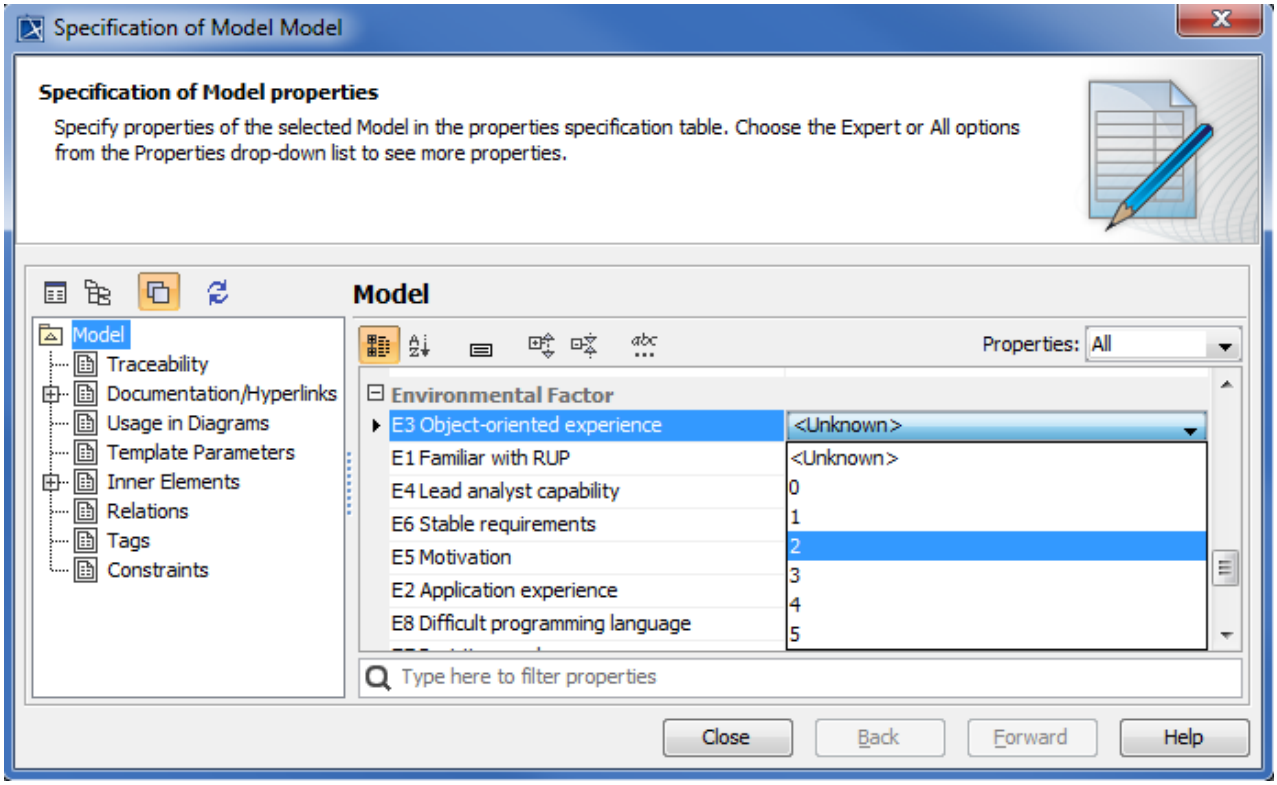
```
TCF = 0.6 + (0.01 * TFactor)
```

Environmental Factors

The Table below lists Environmental Factors

Factor	Environmental Factors	Weight	Description
E1	Familiar with RUP	1.5	Staff in the project are familiar with domain and technical details of the project.
E2	Application experience	0.5	The application experience level.
E3	Object-oriented experience	1.0	Staff in the project have basic knowledge of the OOP concept. The project is implemented on the object oriented design.
E4	Lead analyst capability	0.5	The analyst who is leading the project has enough domain knowledge.
E5	Motivation	1.0	The project motivates staff to work including how the software industry is going on.
E6	Stable requirements	2.0	The requirements are clear, stable, and unlikely to change in the future.
E7	Part-time workers	-1.0	Part-time staff are working on the project.
E8	Difficult programming language	-1.0	Complexity of a programming language.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Use Case Driven Model Data E3.

Environmental Factor Value

Environmental Factor Value (EFactor) is obtained with the multiplication of the value of each Environmental factor by its weight.

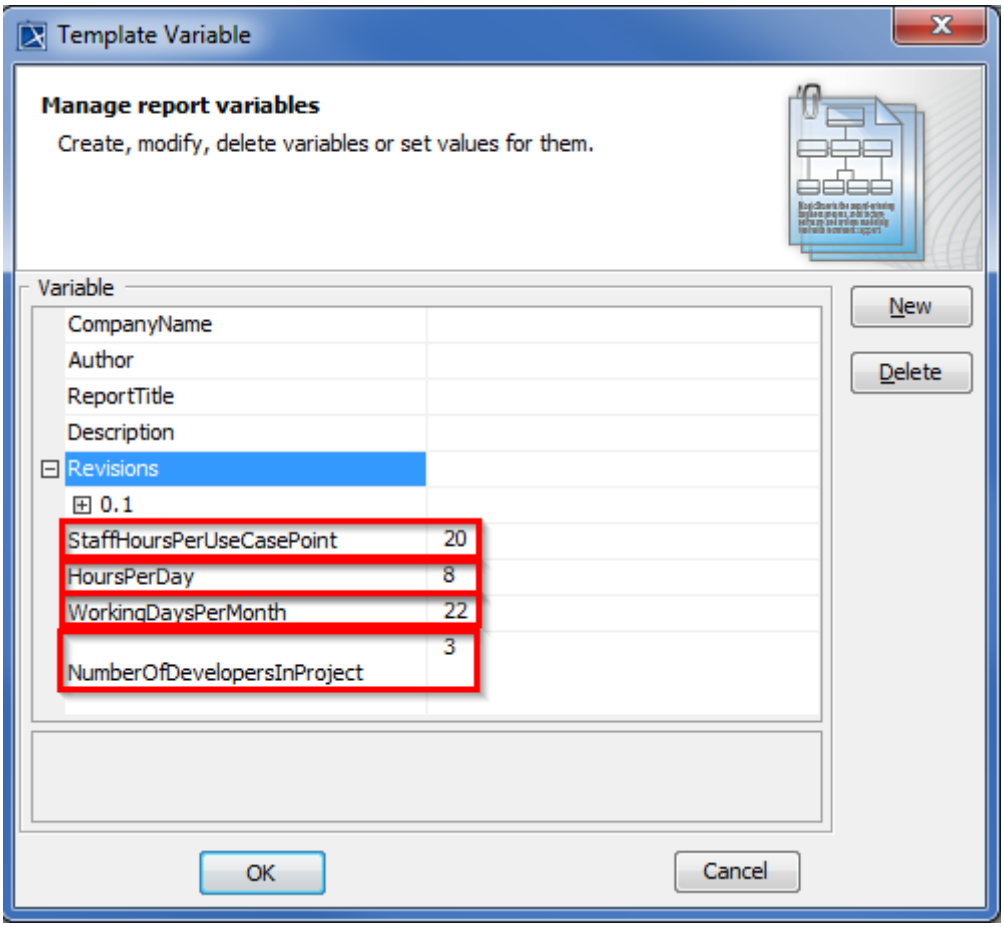
Environmental Factor

Environmental Factor (EF) is obtained with the addition of 1.4 to the sum of EFactor multiplied by -0.03.

```
EF = 1.4 + (-0.03 * EFactor)
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Project Estimation



Variables of Use Case Project Estimation.

Adjusted Use Case Points

The Adjusted Use Case Point (UCP) is determined with the multiplication of Unadjusted Use Case Point (UUCP) by Technical Complexity Factor (TCF) and Environmental Factor (EF).

```
UCP = UUCP * TCF * EF
```

Estimated Effort in Person Hours

The person hours multiplier or X hours is a ratio of the number of man hours (PHM) per Use Case Point based on past projects. If no historical data has been collected, industry experts suggest using a figure between 15 and 30. A typical value is 20.

```
X hours = UCP * PHM
```

Estimated Effort in Scheduled Time

Divide X hours by the number of developers working on the project and working hours per day to determine Estimated Effort in Scheduled Time or Y days. This means that with nDev developers, it would take Y days to complete the project.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

```
Y days = X / nDev / hay
```

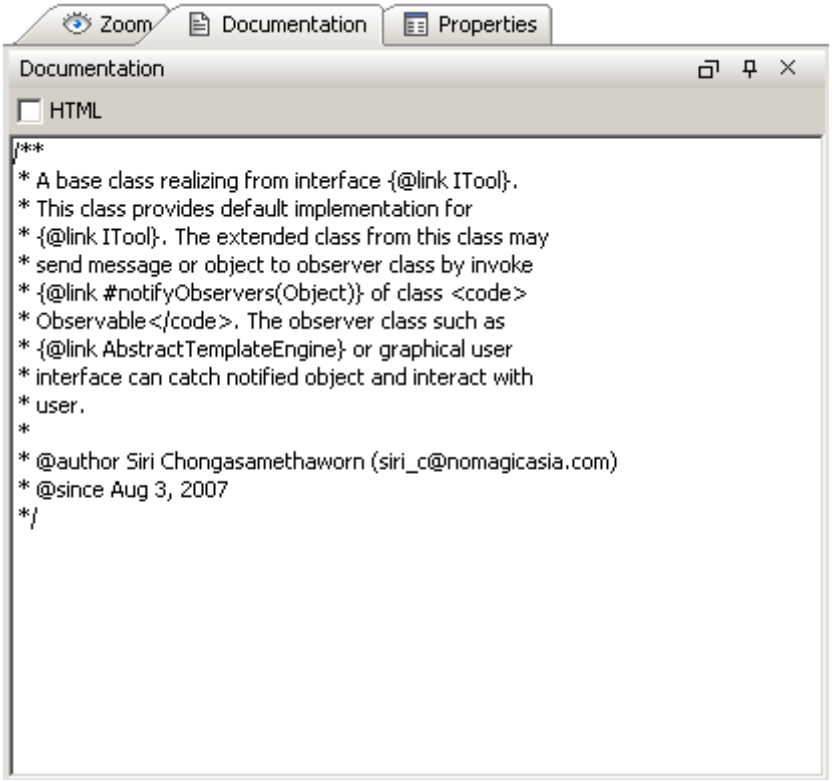
Estimated Effort in Working Days

Divide Y days by working days per month to get the estimated effort in working days or Z months. This means that with nDev developers, it would take Z month to complete the project.

```
Z months = Y / d Month
```

Javadoc syntax tool

Javadoc is the industry standard for documenting Java classes. Javadoc can be documented inside the documentation field, see figure below.



Documentation Field.

Report Wizard provides a template tool for translating Javadoc text into Javadoc properties. Javadoc tool will be loaded and used inside the template upon the user's request.

An example of Javadoc text is shown below:

```
/**
 * A base class realizing from interface {@link Itool}.
 * This class provides default implementation for
 * {@link Itool}. The extended class from this class may
 * send message or object to observer class by invoking
 * {@link #notifyObservers(Object)} of class <code>
 * Observable</code>. The observer class such as
 * {@link AbstractTemplateEngine} or graphical user
 * interface can catch notified object and interact with
 * user.
 *
 * @author Siri Chongasamethaworn (siri_c@nomagicasia.com)
 * @since Aug 3, 2007
 */
```

To print the Javadoc comment and author inside the template:

```
#import ("javadoc", "com.nomagic.reportwizard.tools.doc.JavaDoctool")
#set ($doc = $javadoc.create($class.documentation))
Comment:
$doc.comment
Author:
$doc.author
```

The output from this example will be:

```
Comment:
A base class realizing from interface Itool. This class provides default
implementation for Itool. The extended class from this class may send message or
object to observer class by invoke #notifyObservers(Object) of class Observable. The
observer class such as AbstractTemplateEngine or graphical user interface can catch
notified object and interact with user.
Author:
Siri Chongasamethaworn (siri_c@nomagicasia.com)
```

The Javadoc Syntax tool is implemented based on Custom tool (See Appendix A: report extensions(see page 1833)).

Related pages

Javadoc syntax

A doc comment consists of characters between the characters /** that begin the comment and the characters */ that end it. Leading asterisks are allowed on each line and are described below. Text in a comment can continue to multiple lines.

- **Leading asterisks**
  When Javadoc Tool parses a doc comment, leading asterisk (*) characters on each line are discarded.

- **First sentence**
  The first sentence of each doc comment is a summary sentence.
  This sentence can be retrieved from $doc.firstSentenceTags.
  The first sentence tags are a collection of inline tags until full stops (.).
- **Comment and tag sections**
  A comment is the main description which begins after the starting delimiters /** and continues until the tag section. A tag section starts with the first block tag. The comment can be retrieved by $doc.comment
- **Comments written in HTML**
  Text is written in HTML and will be rendered as HTML support before being printed to a report. (See Appendix D: HTML Tag Support(see page 1985))
- **Block and in-line tags**
  A tag is a special keyword within a doc comment that Javadoc Tool can process. There are two kinds of tags:
  (i) Block tags
  They can be placed only in the tag section. The block tag form is @tag.
  The block tag can be accessed directly from the root document such as $doc.param and $doc.author.
  The $doc.tags will provide a collection of all the tags appearing in this Javadoc.
  (ii) Inline tags
  They can be placed anywhere in the main description or in the block tags. The inline tag form is {@tag}.

The table below lists Current Supported Tags

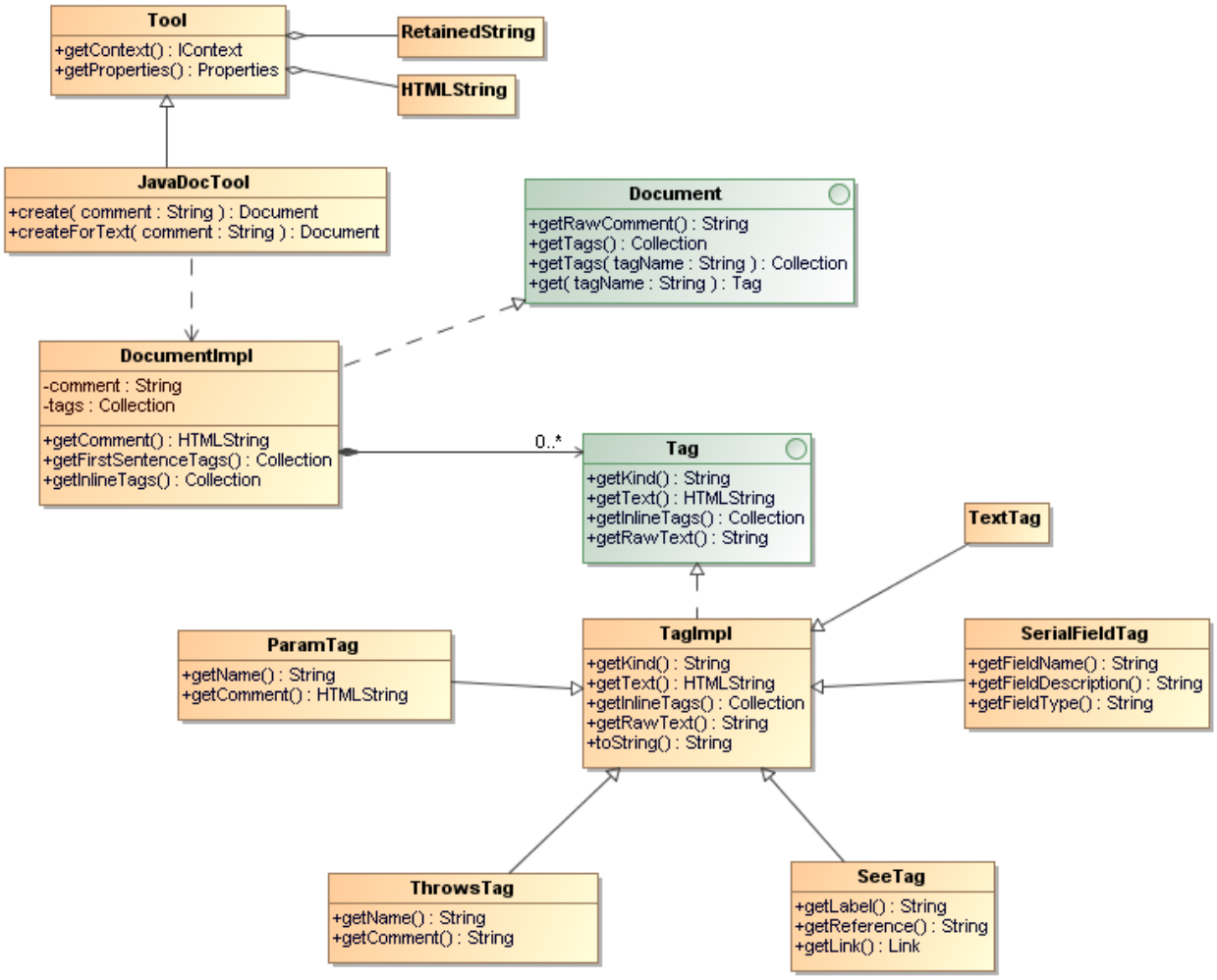
Tag	JDK	Report Wizard Support
@author	1.0	Yes
{@code}	1.5	Render as <code>text</code>
{@docRoot}	1.3	Not supported
@deprecated	1.0	Yes
@exception	1.0	Yes
{@inheritDoc}	1.4	Not supported
{@link}	1.2	Yes, with conditions (External link, model support, and class/method link will be ignored)
{@linkplain}	1.4	Yes, see {@link link}
{@literal}	1.5	Yes
@param	1.0	Yes
@return	1.0	Yes

Tag	JDK	Report Wizard Support
@see	1.0	Yes, with conditions (External link, model support, and class/method link will be return as plain text)
@serial	1.2	Yes
@serialData	1.2	Yes
@serialField	1.2	Yes
@since	1.1	Yes
@throws	1.2	Yes
{@value}	1.4	Not Supported
@version	1.0	Yes

Javadoc tool API

On this page

- Document(see page 1837)
- DocumentImpl(see page 1838)
- Tag(see page 1838)
- TagImpl(see page 1838)
- ParamTag(see page 1838)
- ThrowsTag(see page 1838)
- SeeTag(see page 1838)
- SerialFieldTag(see page 1838)



Javadoc Tool API.

JavaDocTool

JavaDoc Tool is a custom tool class used for creating a Document node.

- create() - Return a new instance of a Document node. Each JavaDoc comment must start with /** and ended with */.
- createForText() - Return a new instance of a Document node. This method allows in-complete JavaDoc (comments without /** and */) to be loaded with JavaDocTool.

Document

Document is an interface representing a Comment Document.

- getRawComment() - Return the full unprocessed text of the comment.
- getTags() - Return all tags in this document item.
- getTags(tagName : String) - Return a tag of the specified kind of this document item.
- get(tagName : String) - Return a tag of the specified kind of this document item. If any tags with the same kind are found, the first tag of that kind will be returned.

DocumentImpl

DocumentImpl is a default JavaDoc document implementation.

- getComment() - Return formatted text for this document.
- getInlineTags() - Return comment as a collection of tags.
- getFirstSentenceTags() - Return the first sentence of the comment as a collection of tags.

Tag

Tag is an interface representing a simple documentation tag, such as @since, @author, and @version

- getKind() - Return the kind of this tag.
- getText() - Return the text of this tag.
- getInlineTags() - Return a collection of tags for a documentation comment with embedded {@link} tags.

TagImpl

TagImpl is a default tag implementation.

- getRawText() - Return the full unprocessed text of this tag.
- toString() - Return the text of this tag. Usually calls getText().

ParamTag

ParamTag represents a @param documentation tag.

- getName() - Return the name of the parameter associated with this tag.
- getComment() - Return the parameter's comment.

ThrowsTag

ThrowsTag represents a @throws or @exception documentation tag.

- getName() - Return the name of the exception associated with this tag.
- getComment() - Return the exception's comment.

SeeTag

SeeTag represents a user-defined cross-reference to related documentation. The reference can be either inline with the comment using {@link} or a separate block comment using @see.

- getLabel() - Return the label of the @see tag.
- getReference() - Return the MODEL or URL of the @see reference.
- getLink() - Return the Link object representing this tag.

SerialFieldTag

SerialFieldTag represents a @serialField tag.

- getFieldName() - return the serial field name.
- getFieldDescription() - return the field comment.
- getFieldType() - return the field type string.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

For examples:

- Import the Javadoc tool and create an instance of Javadoc document:

```
#import ("javadoc",
"com.nomagic.reportwizard.tools.doc.JavaDocTool")
#set ($doc = $javadoc.create($comment))
```

- Print a Javadoc comment:

```
$doc.comment
```

- Print the first encountered author:

```
$doc.author
```

- Print all authors:

```
#foreach ($author in $doc.getTags("author"))
$author.text
#end
```

- Print the first sentence:

```
#foreach ($tag in $doc.firstSentenceTags)
$tag.text
#end
```

- Print the inline tags:

```
#foreach ($tag in $doc.inlineTags)
$tag.kind
$tag.text
#end
```

- Print a raw comment (plain text without a document parser or HTML renderer):

```
$doc.rawComment
```

- Print all block tags:

```
#foreach ($tag in $doc.tags)
$tag.kind : $tag.text
#end
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Print all param tags:

```
#foreach ($tag in $doc.getTags("param"))
$tag.name - $tag.comment
#end
```

- Print all throws tags:

```
#foreach ($tag in $doc.getTags("throws"))
$tag.name - $tag.comment
#end
```

- Print all see tags:

```
#foreach ($tag in $doc.getTags("see"))
$tag.label - $tag.reference
#end
```

Reference Documents

- Javadoc specification
  http://java.sun.com/j2se/1.5.0/docs/tooldocs/windows/javadoc.html#javadoctags
- Writing Javadoc
  http://java.sun.com/j2se/javadoc/writingdoccomments/index.html
- Javadoc Tool API
  <install.dir>/plugins/com.nomagic.magicdraw.reportwizard/api/javadoc.zip

Import tool

Import tool is a module of functions that allows importing documents to be in generated reports. Import tool enables you to dynamically import whole or parts of documents into reports, giving you greater flexibility when generating reports that require dynamic resources. You can now include documents whose location is only known at the actual translation time.

The Import tool allows you to:

- Parse and import *RTF*, *DOCX, HTML*, and text templates from any location in the file system
- Reuse the #sectionBegin(see page 0) and #sectionEnd(see page 0) syntax to import any part of a document. The Import tool uses the predefined syntaxes **#sectionBegin** and **#sectionEnd**, introduced with the #includeSection Directive(see page 0).

Related pages

- Import syntax(see page 1841)
- Import usage(see page 1842)

Import syntax

To use the Import tool syntax

1. To declare Import tool in the template, type **#import ('import', 'com.nomagic.reportwizard.tools.ImportTool')** in the document that you want to import its content to the report. The **#import** directive will load Import tool so that it can be accessed inside the template.
    - The first parameter **'import'** is for identifying the alias of Import tool (in this case, **'import'**, but it can be named anything).
    - The second parameter **'com.nomagic.reportwizard.tools.ImportTool'** has to be copied as it is (do not rename it). Otherwise, Import tool will not be loaded.
2. To import any content from a child template, type the following in the document that you want to import its content to the report:
    - **$import.include('child.docx')** to import a complete document at the current position of the template, and/or
    - **$import.includeSection('child.docx', 'Section A')** to include only the text contained in the named section of the document.

$import

This part names the alias assigned to Import tool when declaring it to the template.

$import.include(FileName)

FileName in $import.include(FileName)(see page 1842) specifies the location of a file in the file system. **FileName** can be either an absolute or relative path to a file in the file system. The **FileName** parameter can be set either statically or dynamically. If the file is not found in the file system, **FileName** would be searched as an Attached File element in a MagicDraw project in the **File** property.

$import.includeSection(FileName, SectionName)

FileName in $import.includeSection(FileName, SectionName)(see page 1843) functions the same as in the sub-section above. **FileName** can be either an absolute or relative path to a file in the file system. **SectionName** denotes a paragraph in the document named **FileName**, which is delimited by the **#sectionBegin(SectionName)** and **#sectionEnd** identifiers. The **SectionName** variable can also be set either statically or dynamically. If the file is not found in the file system, **FileName** would be searched as an Attached File element in a MagicDraw project in the **File** property.

> **Information**
> You can also use **$attachedFile** in the parameter of **$import.include(FileName)** and **$import.includeSection(FileName, SectionName)**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related page

- Ruby Script Tool(see page 1853)

Import usage

This section explains the examples of usage of the master and child templates. These examples allow you to create documents as the master template and reuse the content of child templates into it.

Preparatory Step

Define a child template that you will use as in Examples 1, 2, and 3. This child template contains the header line and named sections (**Section A** and **Section B**). Type the following lines in the document that you want to define the child template:

```
This is the child template.
#sectionBegin(Section A)
This is Section A.
#sectionEnd
#sectionBegin(Section B)
This is Section B.
#sectionEnd
```

Usage in Example 1

This example shows how to statically include the complete child template as shown in the Preparatory Step. To include it, call the **$import.include('child.docx')** method. Note that there is no specified path. This path to the child template shows you that the master and child templates reside in the same directory. Type the following lines in the document that you want to include all of its content to the report:

```
#import ('import', 'com.nomagic.reportwizard.tools.ImportTool')
This is the 1st master template
Include child template
$import.include('child.docx')
```

The output right below will include the entire text from the master template (right above) and from the child template minus the template directives. The included content will be then parsed and stripped of all velocity directives. The output is as follows:

```
This is the 1st master template
Include child template
This is the child template.
This is Section A.
This is Section B.
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Usage in Example 2

This example shows how to statically include a section of the child template as shown in the Preparatory Step. To include the section named **Section A** from the child template, call the **$import.includeSection('templates/child.rtf', 'Section A')** method. In this example, the master and child templates reside in different directories, which means that the child template will reside in a subdirectory called *templates*. This *templates* directory will thus reside in the same directory as the master template. Type the following lines in the document that you want to include its content in **Section A** to the report:

```
#import ('import', 'com.nomagic.reportwizard.tools.ImportTool')
This is the 2nd master template
Include Section A
$import.includeSection('templates/child.rtf', 'Section A')
```

The output will be:

```
This is the 2nd  master template
Include Section A
This is Section A.
```

Usage in Example 3

This example shows how to dynamically include the section of a child template as shown in the Preparatory Step. Before dynamically including the section of the child template named **Section B**, set a variable for the file location of the child template **[#set ($child = "***C:/ImportTool/child.rtf***")]** and another one for the section name to be included **[#set ($section = "Section B")]**. To include the **Section B** section from the child template, call the **$import.includeSection($child, $section)** method. In this example, the child's template location is provided by an absolute path. Note that dynamic values must not be specified in a pair of quote marks, otherwise the section will not be included. Type the following lines in the document that you want to include its content in **Section B** to the report:

```
#import ('import', 'com.nomagic.reportwizard.tools.ImportTool')
This is the 3rd master template
Include Section B
#set ($child = "C:/ImportTool/child.rtf")
#set ($section = "Section B")
$import.includeSection($child, $section)
```

The output will be:

```
This is the 3rd master template
Include Section B
This is Section B.
```

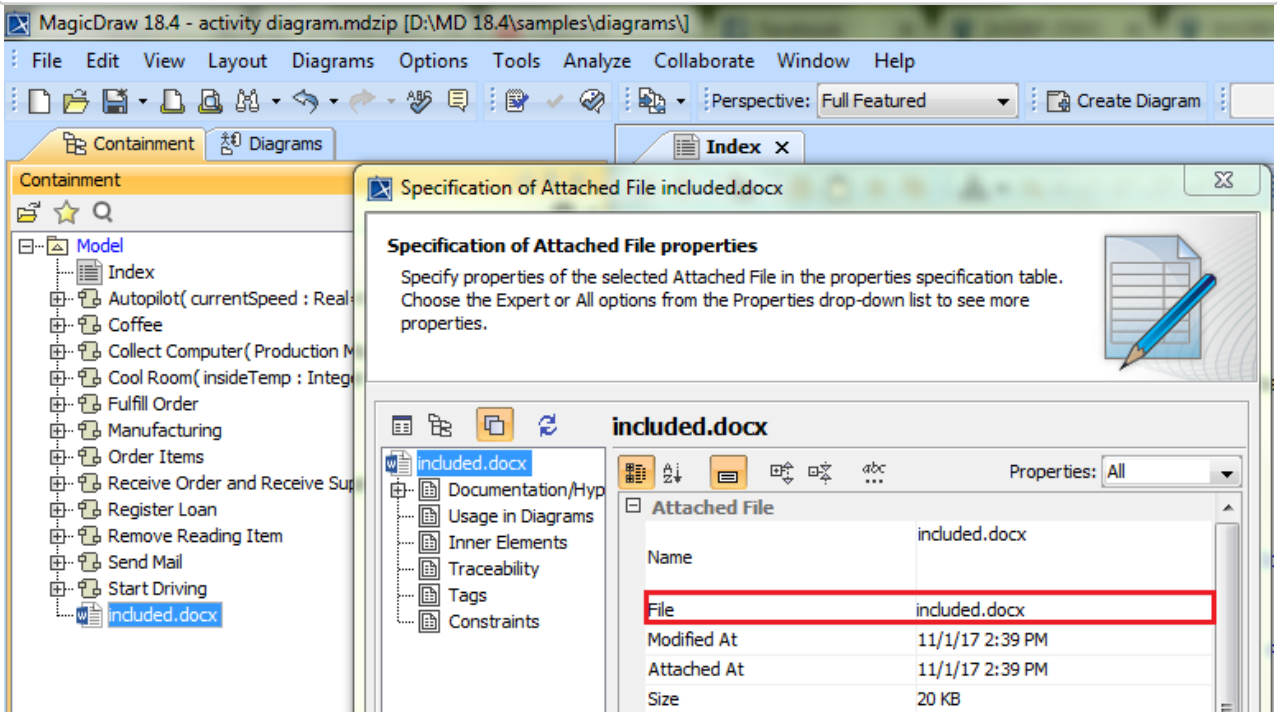
The Import tool supports *DOCX, RTF, HTML, XML*, and *text* files only.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Usage in Example 4

This example shows how to include a document as an Attached File element in a project. First, you need to create an Attached File(see page 209) element in a project. The **File** property will then be used as a parameter to make a reference to the Attached File element. Type the following lines in the document that you want to include the content of the Attached File element to the report:

```
#import ('import', 'com.nomagic.reportwizard.tools.ImportTool')
#foreach($included in $AttachedFile)
    #if($included.file == "included.docx")
        $import.include($included)
    #end
#end
```



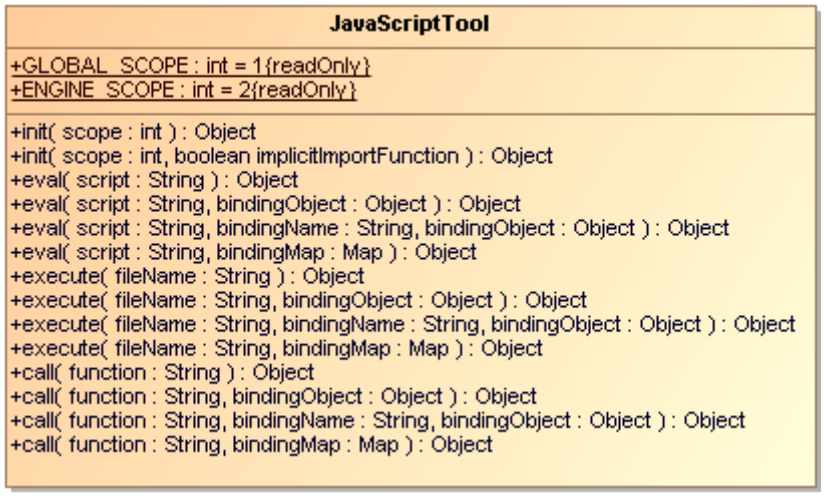
Including a document as an Attached File element in a project.

JavaScript tool

JavaScript tool enables report templates to evaluate or run JavaScript codes from templates and external JavaScript files, see figure below.

The general concept behind this JavaScript feature is to separate complex business logic from presentation logic. Complex logic should be executed by JavaScript, and presentation logic should be executed by Velocity code.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Class Diagram: JavaScript tool.

JavaScript tool includes 3 methods:

(i) **'eval' Method**
this method will evaluate JavaScript text, and then return the result.
(ii) **'execute' Method**
this method will execute a JavaScript file, and then return the result.
(iii) **'call' Method**
this method will call a JavaScript function, and then return the result.

Like other Custom tools, the JavaScript tool '*scripttool.jar*' must be installed in the 'extensions' folder of the Report Wizard plugin, and '*js.jar*' must be included in any class path or the 'extensions' folder. For further information about Custom tools and the installation, see Custom tool(see page 1952).

To import JavaScript tool to a template type, for example:

```
#import('js', 'com.nomagic.reportwizard.tools.script.JavaScripttool')
```

Related pages

JavaScript tool API

'eval' Method

eval(String script)

This method will evaluate a JavaScript code from a string and return the result.

```
$js.eval('script')
```

eval(String script, Object bindingObject)

This method will evaluate a JavaScript code with a single binding object. The code will be evaluated from a string. The "importer" name will be used as a binding name for this object.

```
#foreach ($class in $Class)
 $js.eval('importer.getName()', $class)
#end
```

eval(String script, String bindingName, Object bindingObject)

This method will evaluate a JavaScript code with a single binding object and specified binding name. The code will be evaluated from a string. The binding name will be used as the name for this object.

```
#foreach ($class in $Class)
 $js.eval('cls.getName()', 'cls', $class)
#end
```

eval(String script, Map bindingMap)

This method will evaluate a JavaScript code with a set of binding arguments (a name and an object). The code will be evaluated from a string. The binding map consists of key-value pairs for this binding name and binding object.

```
#set ($dict = $map.createHashMap())
#set ($void = $dict.put("first", "foo"))
#set ($void = $dict.put("last", "bar"))
$js.eval("first + ' ' + last", $dict)
```

'execute' Method

execute(String filename)

This method will execute a JavaScript file. The filename parameter is a file path to the JavaScript file.

```
$js.execute('filename.js')
```

execute(String filename, Object bindingObject)

This method will execute a JavaScript file with a single binding object. The filename parameter is a file path to the JavaScript file. The "importer" name will be used as the binding name for this object.

```
#foreach ($class in $Class)
 $js.execute('filename.js', $class)
#end
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

```
// filename.js
importer.getName();
```

execute(String filename, String bindingName, Object bindingObject)

This method will execute a JavaScript file with a single binding object and specified binding name. The filename parameter is a file path to the JavaScript file. The binding name will be used as the name for this object.

```
#foreach ($class in $Class)
 $js.execute('filename.js', 'cls', $class)
#end
```

```
// filename.js
cls.getName();
```

execute(String filename, Map bindingMap)

This method will execute a JavaScript file with a set of binding arguments (a name and an object). The filename parameter is a file path to the JavaScript file. The binding map consists of key-value pairs for this binding name and binding object.

```
#set ($map = $map.createHashMap())
#set ($void = $map.put("first", "foo"))
#set ($void = $map.put("last", "bar"))
$js.execute('filename.js', $map)
```

```
// filename.js
first + ' ' + last;
```

- **Absolute Path**: If a 'filename' is provided with an absolute path, JavaScript tool will read the JavaScript file from an absolute location such as $js.execute('*c:/mycode/readclass.js*').
- **Relative Path**: If a 'filename' is provided with a relative path, JavaScript tool will read the template from a relative location. This relative location starts from the current directory in which the template is located such as $js.execute('readclass.js').

'call' Method

call(String function)

This method will provide a short and reusable method to call a JavaScript function. This function must be defined before calling this method. This function can be defined by 'eval' or 'execute'.

```
$js.execute('javascript.js')
$js.call('calc(1, 3)')
```

```
// javascript.js
function calc(var1, var2)
{
 return var1 + var2;
}
```

call(String function, Object bindingObject)

This method will provide a short and reusable method to call a JavaScript function with a single binding object. The "importer" name will be used as the binding name for this object. This function must be defined before calling this method. This function can be defined by 'eval' or 'execute'.

```
$js.execute('javascript.js')
$js.call('calc(1, 3)', 10)
```

```
// javascript.js
function calc(var1, var2)
{
 var f = Number(importer ? importer : 0);
 return f + var1 + var2;
}
```

call(String function, String bindingName, Object bindingObject)

This method will provide a short and reusable method to call a JavaScript function with a single binding object and specified binding name. This binding name will be used as the name for this object. This function must be defined before calling this method. This function can be defined by 'eval' or 'execute'.

```
$js.execute('javascript.js')
$js.call('calc(1, 3)', 'factor', 10)
```

```
// javascript.js
function calc(var1, var2)
{
 var f = Number(factor ? factor : 0);
 return f + var1 + var2;
}
```

call(String function, Map bindingMap)

This method will provide a short and reusable method to call a JavaScript function with a set of binding arguments (a name and an object). The binding map consists of key-value pairs for this binding name and binding object.

```
#set ($map = $map.createHashMap())
#set ($void = $map.put("first", "foo"))
#set ($void = $map.put("last", "bar"))
$js.call('hello()', $map)
```

```
// filename.js
function hello()
{
 return 'hello ' + first + ' ' + last;
}
```

References to elements

References to elements are implicitly inserted into the JavaScript context when calling **eval()**, **execute()**, or **call()**. Examples of implicit variables include, for instance, **$Class**, **$UseCase**, **$sorter**, etc.

For example:

A "functions.js"

```
importPackage(java.util)
// variable $Dependency and $sorter can be accessed directly inside
function getSupplier()
{
   var supplierList = new ArrayList();
    var sortedDependencyList = $sorter.sort($Dependency);
   for (var i=0; i<sortedDependencyList.size(); i++)
   {
      var dependency = sortedDependencyList.get(i);
      supplierList.addAll(dependency.getSupplier());
   }
   return supplierList;
}
```

A template code

```
#import ('js', 'com.nomagic.reportwizard.tools.script.JavaScriptTool')
$js.execute("functions.js")
#set ($supplierList = $js.eval("getSupplier()"))
#foreach ($supplier in $supplierList)
$supplier.name
#end
```

Example:

A "functions.js"

```
var qualifiedName = "";
function packageQualifiedName(element)
{
   qualifiedName = "";
    packageName(element);
    return qualifiedName==""?"Default":qualifiedName;
}
function packageName(element)
{
    var parent = element.owner;
   if (parent.classType.simpleName == 'Package') {
       if (qualifiedName == "")
           qualifiedName = parent.name;
        else
          qualifiedName = parent.name + '.' + qualifiedName;
        packageName(parent);
    }
}
```

A template code

```
#import ('js', 'com.nomagic.reportwizard.tools.script.JavaScriptTool')
$js.execute("functions.js")
#foreach ($c in $Class)
   Package: $js.eval('packageQualifiedName($c)')
    Name: $c.name
#end
```

Reference Documents

- Mozilla JavaScript Tool
  http://developer.mozilla.org/en/Rhino_documentation
- Sun JavaScript programming guide
  http://java.sun.com/javase/6/docs/technotes/guides/scripting/programmer_guide/index.html

Groovy script tool

Groovy Script tool enables report templates to evaluate or run Groovy codes from templates and external Groovy files.

The Groovy tool includes two methods:

(i) 'eval' method
This method will evaluate Groovy text, and then return the result.
(ii) 'execute' method
This method will execute a Groovy file, and then return the result.

Like other Script tools, the Groovy Script tool '*scripttool.jar*' must be installed in the 'extensions' folder of the Report Wizard plugin, and '*groovy-all-1.7.9.jar*' must be included in any class path or the 'extensions' folder. For further information about Custom tools and the installation, see Custom tool(see page 1952).

To import Groovy Script tool to a template, type the following code:

```
#import ('groovy', 'com.nomagic.reportwizard.tools.script.Groovytool')
```

Related pages

Groovy script tool API

'eval' Method

eval(String script)

This method will evaluate a Groovy code from a string and return the result.

```
$groovy.eval("println 'Hello World!'")
```

eval(String script, String bindingName, Object bindingObject)

This method will evaluate a Groovy code with a single binding object and specified binding name. The code will be evaluated from a string. The binding name will be used as the name for this object.

```
#foreach ($c in $Class)
$groovy.eval("println classname", "classname", $c.name)
#end
```

eval(String script, Map bindingMap)

This method will evaluate a Groovy code with a set of binding arguments (a name and an object). The code will be evaluated from a string. The binding map consists of key-value pairs for the binding name and the binding object.

```
#set ($dict = $map.createHashMap())
#set ($void = $dict.put("first", "foo"))
#set ($void = $dict.put("last", "bar"))
$groovy.eval("println first + last", $dict)
```

Or

```
$groovy.eval("println first + ' ' + last", {"first":"foo", "last":"bar"})
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

The second code contains curly brackets; '{' and '}' characters, which are not allowed to be used in any *RTF* template. For the *RTF* template, use the first code instead.

'execute' method

execute(String filename)

This method will execute a Groovy file. The 'filename' parameter refers to a name of the Groovy file or an absolute path to the Groovy file.

```
$groovy.execute("filename.groovy")
```

execute(String filename, String bindingName, Object bindingObject)

This method will execute a Groovy file with a single binding object and specified binding name. The 'filename' parameter is a file path to the Groovy file. The binding name will be used as the name for this object.

File filename.groovy

```
"Class name is $c.name"
```

The template code

```
#foreach ($c in $Class)
$groovy.execute("filename.groovy", 'c', $c)
#end
```

execute(String filename, Map bindingMap)

This method will execute a Groovy file with a set of binding arguments (a name and an object). The 'filename' parameter is a file path to the Groovy file. The binding map consists of key-value pairs for the binding name and the binding object.

File filename.groovy

```
first + " " + last
```

The template code

```
#set ($dict = $map.createHashMap())
#set ($void = $dict.put("first", "foo"))
#set ($void = $dict.put("last", "bar"))
$groovy.execute("filename.groovy", $dict)
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- *Absolute Path*: If the 'filename' is provided with an absolute path, Groovy Tool will read the Groovy file from an absolute location such as **$groovy.execute('***c:/mycode/readclass.groovy***')**.
- *Relative Path*: If the 'filename' is provided with a relative path, Groovy Tool will read the template from a relative location. This relative location starts from the current directory in which the template is located such as **$groovy.execute('***readclass.groovy***')**.

References to elements - Groovy

References to elements are similar to ones in JavaScript Tool. The elements are implicitly inserted into the Groovy context when calling "eval()", or "execute()". Examples of implicit variables include $Class, $UseCase, $sorter, etc.

File 'AllAbstractClass.groovy'

```
// variable $Class can be accessed directly inside Groovy script
def list = []
for (c in $Class) {
    if (c.isAbstract()) {
        list.add(c)
    }
}
return list
```

The report template code is:

```
#import ('groovy', 'com.nomagic.reportwizard.tools.script.GroovyTool')
#set ($abstractClassList = $groovy.execute('AllAbstractClass.groovy'))
#foreach ($cls in $abstractClassList)
$cls.name
#end
```

Ruby Script tool

Ruby Script tool enables report templates to evaluate or run Ruby codes from templates and external Ruby files.
The Ruby tool includes 2 methods:

(i) 'eval' method: This method will evaluate Ruby text, and then return the result.
(ii) 'execute' method: This method will execute a Ruby file, and then return the result.

Like other Script tools, the Ruby Script tool 'scripttool.jar' must be installed in the 'extensions' folder of the Report Wizard plugin, and '*jruby-complete-1.6.3.jar*' must be included in any class path or the 'extensions' folder. For further information about Custom tools and the installation, see Custom tool(see page 1952).

To import Ruby Script tool to a template, type the following code:

```
#import ('ruby', 'com.nomagic.reportwizard.tools.script.RubyScripttool')
```

Related pages

References to elements - Ruby

References to elements are similar to JavaScript Tool and Groovy Tool. The elements are implicitly inserted into the Ruby context when “eval()” or “execute()” is called. Examples of implicit variables include $Class, $UseCase, $sorter, etc.

For example, File ‘Functions.rb’:

```
def supplierList
result = [];
$Dependency.each do |item|
item.supplier.each do |supplier|
result << supplier;
end
end
return result;
end
```

Indicated below is the template code.

```
#import ('ruby', 'com.nomagic.reportwizard.tools.script.RubyScriptTool')
$ruby.execute('Functions.rb')
#set ($supplierList = $ruby.eval('supplierList'))
#foreach ($e in $supplierList)
$e.name
#end
```

Ruby Script tool API

‘eval’ Method

eval(String script)

This method will evaluate a Ruby code from a string and return the result.

```
$ruby.eval("puts 'Hello World!'")
```

eval(String script, String bindingName, Object bindingObject)

This method will evaluate a Ruby code with a single binding object and specified binding name. The code will be evaluated from a string. The binding name will be used as the name for this object.

```
#foreach ($class in $Class)
$ruby.eval('"Class name is " + c.name', 'c', $class)
#end
```

eval(String script, Map bindingMap)

This method will evaluate a Ruby code with a set of binding arguments (a name and an object). The code will be evaluated from a string. The binding map consists of key-value pairs for the binding name and binding object.

```
#set ($dict = $map.createHashMap())
#set ($void = $dict.put("first", "foo"))
#set ($void = $dict.put("last", "bar"))
$ruby.eval("puts first + last", $dict)
```

Another alternative is as follows:

```
$ruby.eval("puts first + last", {"first":"foo", "last":"bar"})
```

The second code contains curly brackets; “{“ and “}” characters, which are not allowed to be used in any *RTF* template. For the *RTF* template, use the first code instead.

‘execute’ Method

execute(String filename)

This method will execute a Ruby file. The ‘filename’ parameter is a name of the Ruby file or an absolute path to the Ruby file.

After executing the Ruby file, the result of the execution will be stored in the single context for each report generation. If the Ruby file contains Ruby functions, you can recall the functions with the use of ‘eval()’ methods.

For example, File ‘String.rb’:

```
def deCamelCase(str)
return str.gsub!(/(.)([A-Z])/,'\1 \2')
end
```

Indicated below is the template code.

```
$ruby.execute("String.rb")
#foreach ($c in $Class)
$ruby.eval('deCamelCase($c.name)')
#end
```

execute(String filename, String bindingName, Object bindingObject)

This method will execute a Ruby file with a single binding object and specified binding name. The 'filename' parameter is a file path to the Ruby file. The binding name will be used as the name for this object.

```
File 'filename.rb'
puts c.name
```

The template code

```
#foreach ($c in $Class)
$ruby.execute("filename.rb", 'c', $c)
#end
```

execute(String filename, Map bindingMap)

This method will execute a Ruby file with a set of binding arguments (a name and an object). The 'filename' parameter is a file path to the Ruby file. The binding map consists of key-value pairs for the binding name and binding object.

File '*filename.rb*'

```
puts first+' '+last
```

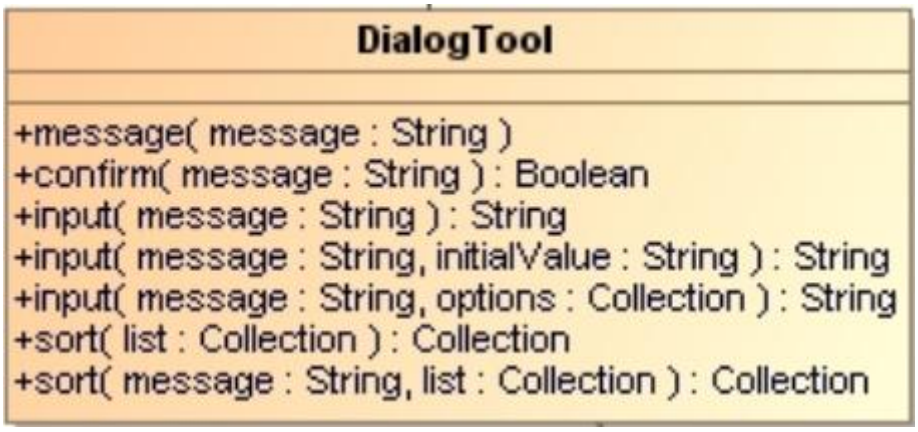
The template code

```
#set ($dict = $map.createHashMap())
#set ($void = $dict.put("first", "foo"))
#set ($void = $dict.put("last", "bar"))
$ruby.execute("filename.rb", $dict)
```

- **Absolute Path**
  If the 'filename' is provided with an absolute path, Ruby tool will read the Ruby file from an absolute location such as $ruby.execute('*c:/mycode/readclass.rb*').
- **Relative Path**
  If the 'filename' is provided with a relative path, Ruby tool will read the template from a relative location. This relative location starts from the current directory location of the template, such as $ruby.execute('*readclass.rb*').

Dialog tool

The purpose of a Dialog tool is to enable report templates to call functions for creating dialogs to interact with users during the report generation process. To modify the report templates, you can use the Dialog tool.



Class Diagram: DialogTool.

The Dialog tool uses 4 methods:

- **message**
  This method will create message dialogs.
- **confirm**
  This method will create confirmation dialogs and return the Boolean value 'true' when you click the OK button or return 'false' when you click the Cancel button.
- **input**
  This method will create input dialogs and return the input value as a string.
- **sort**
  This method will create Sort and Enable dialogs and return a collection of newly-sorted results.

Like other Custom tools, the Dialog tool (*dialogtool.jar*) must be installed in the 'extensions' folder of the Report Wizard plugin.

To import a Dialog tool to a template, type the following:

```
#import('dialog','com.nomagic.reportwizard.tools.Dialogtool')
```

Related page

Dialog tool API

On this page

- ‘message’ Method(see page 1858)
- ‘confirm’ Method(see page 1858)
- ‘input’ Method(see page 1859)
  - Input dialog with text(see page 1859)
  - Input dialog with text and initial value(see page 1860)
  - Input dialog with text and initial value Array(see page 1860)
- ‘sort’ Method(see page 1861)
  - Sort and Enable dialog(see page 1861)
  - Sort and Enable dialog with text(see page 1861)
- 'select' Method(see page 1862)
  - Default Selection dialog(see page 1862)
  - Selection dialog with a title(see page 1863)
  - Selection dialog without selection(see page 1864)
  - Selection dialog with description text(see page 1865)

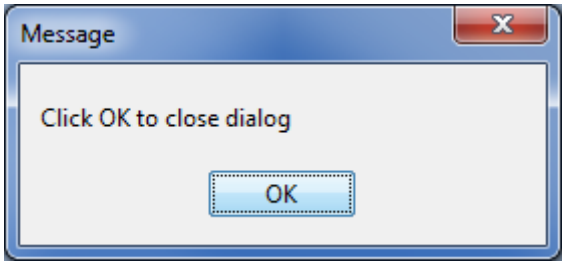
‘message’ Method

message(String message) : void. This method will create a **Message** dialog.

The code example is shown below:

```
$dialog.message("Click OK to close dialog")
```

The output will be as follows:



The Message dialog.

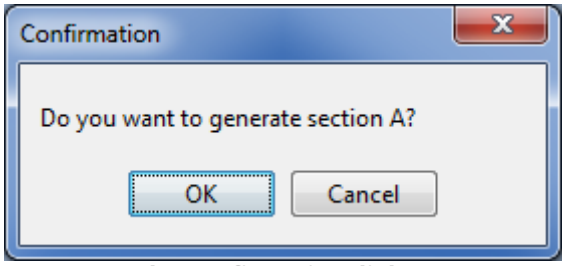
‘confirm’ Method

confirm(String message) : boolean. This method will create a **Confirmation** dialog. It will return the Boolean value ‘true’ when you click OK or ‘false’ when you click Cancel in the dialog.

The code example is shown below:

```
#if($dialog.confirm("Do you want to generate section A?"))
#end
```

The output will be as follows:



The Confirmation dialog.

'input' Method

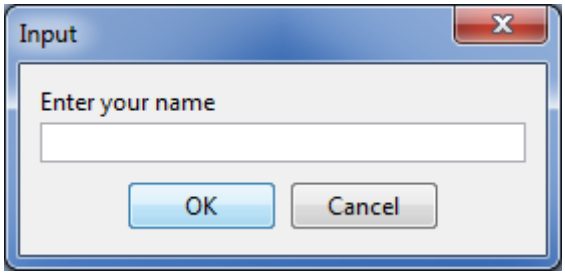
Input dialog with text

input(String message) : String. This method will create an **Input** dialog from a message. It will return the input values as Strings when you click OK or 'null' when you click Cancel in the dialog.

The code example is shown below:

```
#set ($userText = $dialog.input("Enter your name"))
#if ($userText)
Your name is $userText
#else
Please specify your name
#end
```

The output will be as follows:



The Input dialog with text.

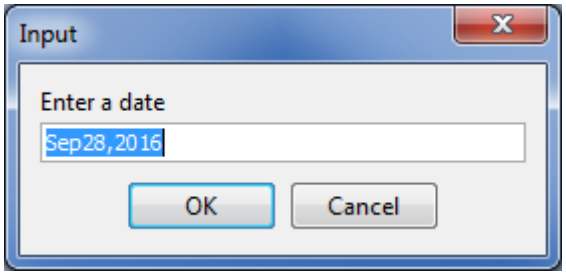
Input dialog with text and initial value

input(String message, String initialValue) : String. This method will create a message and add an initial value in an **Input** dialog. It will return the input values as Strings when you click OK or 'null' when you click Cancel.

The code example is shown below:

```
#set ($userText = $dialog.input("Enter a date", "Sep28,2016"))
Date is $userText
```

The output will be as follows:



The Input dialog with text and an initial value.

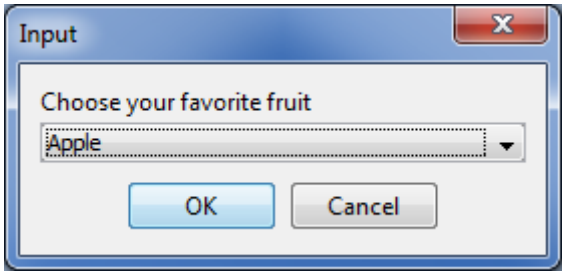
Input dialog with text and initial value Array

input(String message, Collection options) : String. This method will create a message and add initial value arrays in an **Input** dialog. It will return the input values as Strings when you click OK or 'null' when you click Cancel.

The code example is shown below:

```
#set ($selectedOption = $dialog.input("Choose your favorite fruit", ["Apple",
"Orange", "Banana"]))
Your favorite fruit is $selectedOption
```

The output will be as follows:



The Input dialog with text and initial value arrays.

‘sort’ Method

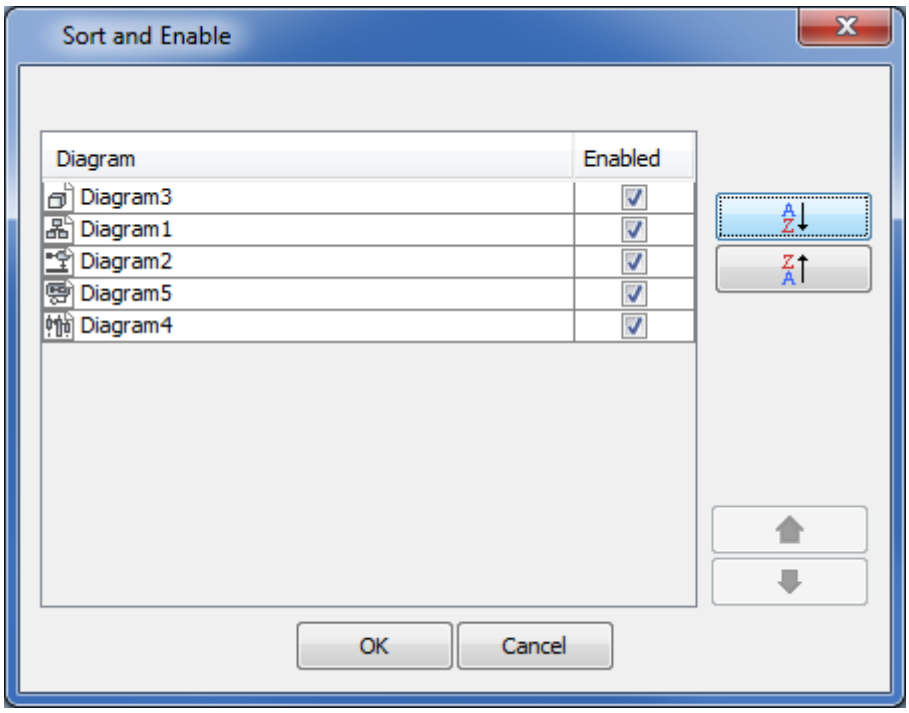
Sort and Enable dialog

sort(Collection list) : Collection. This method will create a **Sort and Enable** dialog from collections. It will return a collection of the newly sorted results when you click OK or return the original collection when you click Cancel.

The code example is shown below:

```
#foreach ($diagram in $dialog.sort($Diagram))
$diagram.name
#end
```

The output will be as follows:



The Sort and Enable dialog.

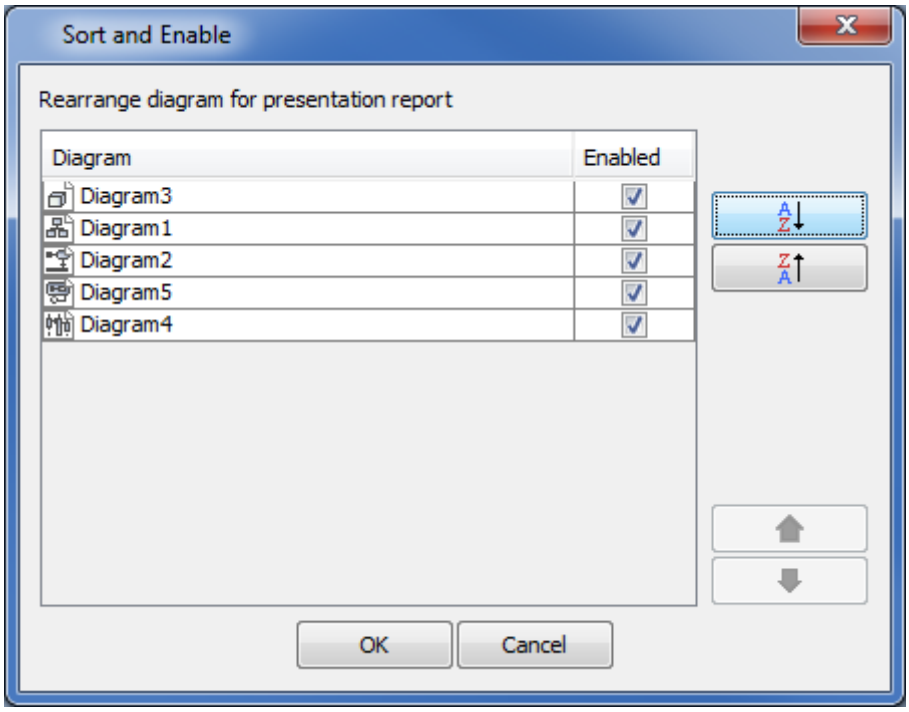
Sort and Enable dialog with text

sort(String message, Collection list). This method will create a **Sort and Enable** dialog from text and collections. It will return a collection of the newly sorted results when you click OK or return the original collection when you click Cancel.

The code example is shown below:

```
#foreach ($diagram in $dialog.sort("Rearrange diagram for presentation report",
$Diagram))
$diagram.name
#end
```

The output will be as follows:



The Sort and Enable dialog with text.

'select' Method

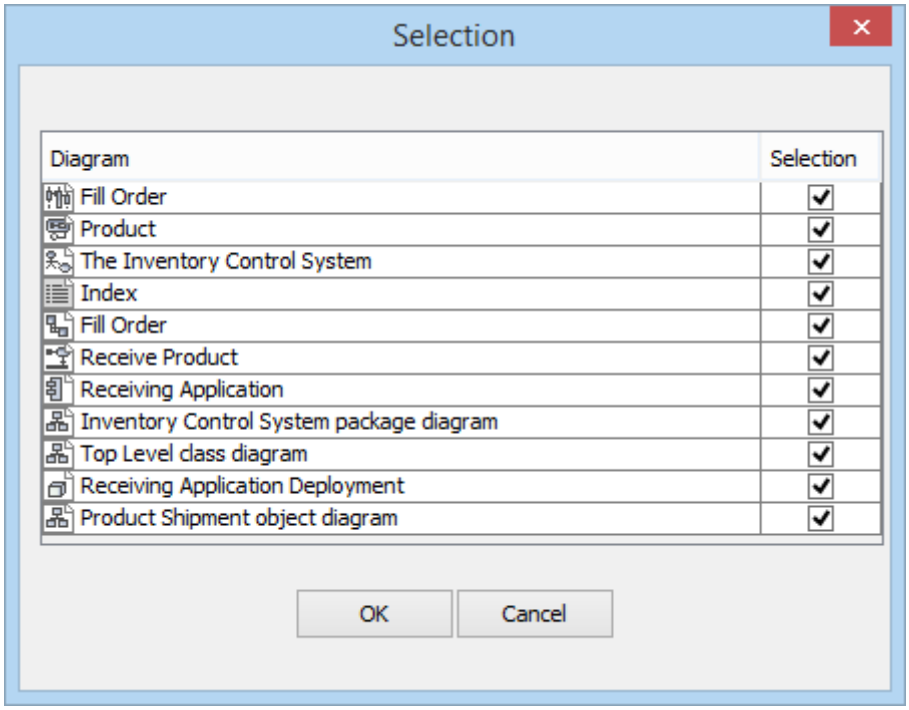
Default Selection dialog

select(Collection list) : Collection. This method will create a **Selection** dialog from a collection. Items in the collection will be listed in the **Selection** dialog. You can select the items and click OK to return a collection of the selected items or null when you click Cancel.

The code example is shown below:

```
#foreach ($class in $dialog.select($Diagram))
$diagram.name
#end
```

The output will be as follows:



The Selection dialog.

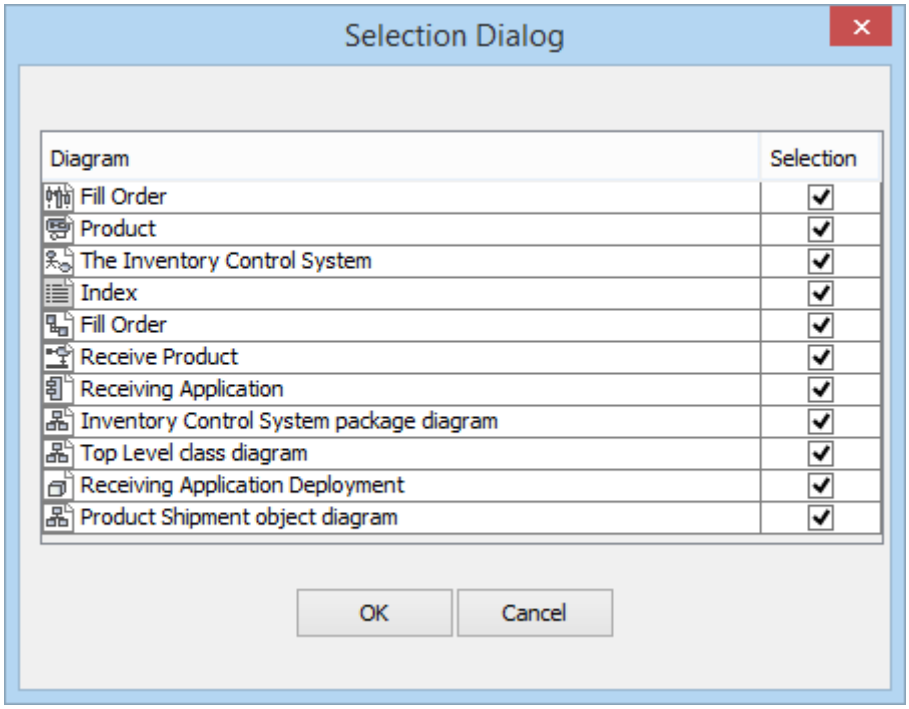
Selection dialog with a title

select(Collection list, String title) : Collection. This method will create a **Selection** dialog with a specified dialog title. The default **Selection** dialog title is **Selection**. You can select the items and click OK to return a collection of the selected items or null if you click Cancel.

The code example is shown below:

```
#foreach ($diagram in $dialog.select($Diagram, "Selection Dialog"))
$diagram.name
#end
```

The output will be as follows:



The Selection dialog with title.

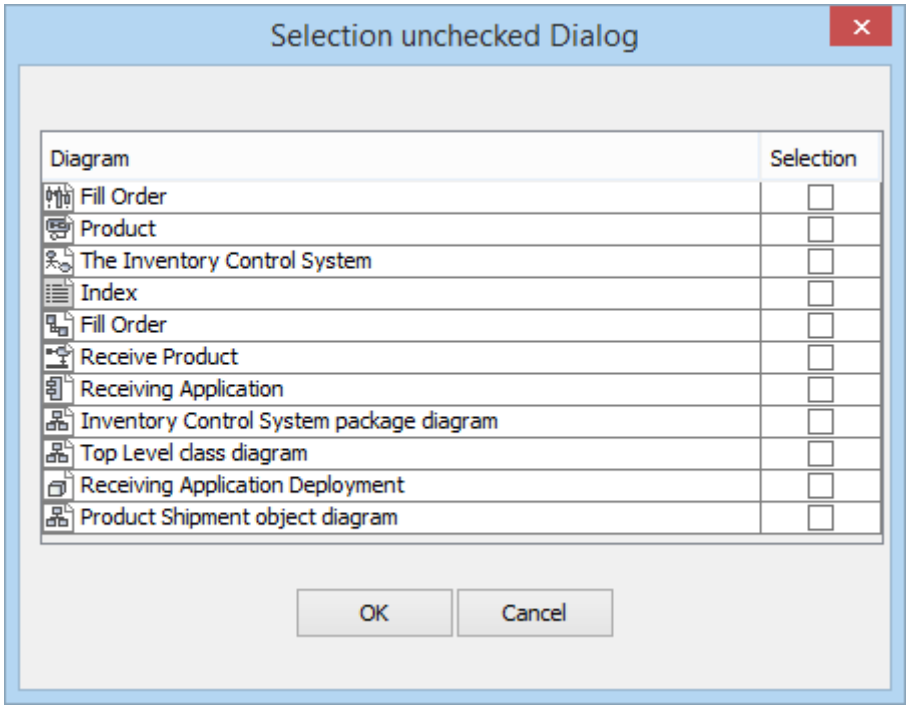
Selection dialog without selection

select(Collection list, String title, Boolean defaultSelected) : Collection. This method will create a **Selection** dialog from the collection without selecting the items in the dialog. You can specify whether all listed items will be selected or not. The default value for defaultSelected is true. Otherwise, all listed items will not be selected. You can select the items and click OK to return a collection of the selected items or null if you click Cancel.

The code example is shown below:

```
#foreach ($diagram in $dialog.select($Diagram, "Selection unchecked Dialog", false))
$diagram.name
#end
```

The output will be as follows:



The Selection dialog without selection.

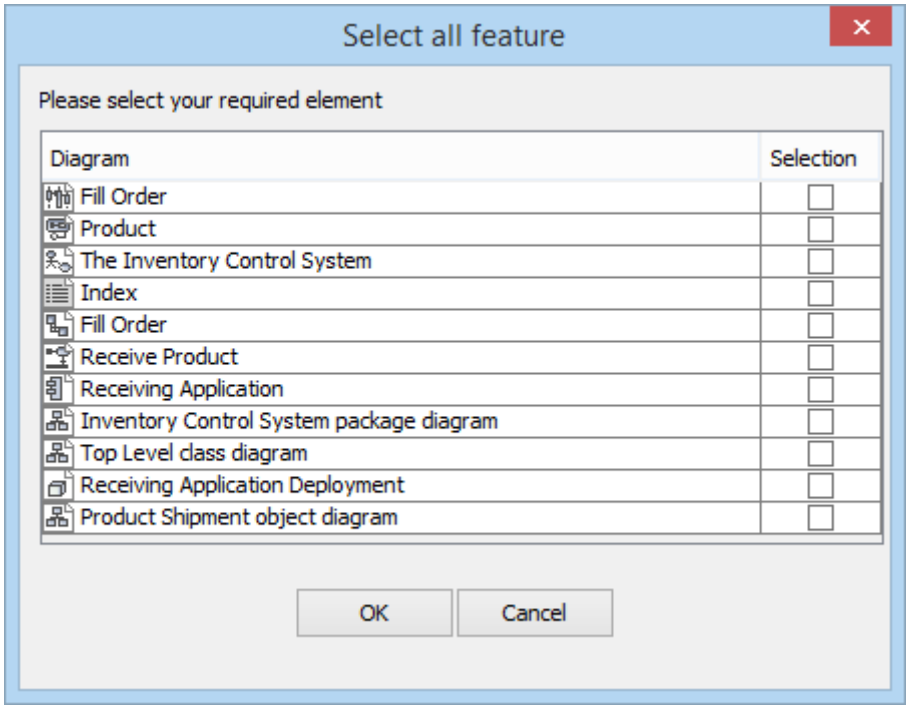
Selection dialog with description text

select(Collection list, String title, Boolean defaultSelected, String dialogText) : Collection. This method will create a **Selection** dialog from the collection. You can insert description text in the dialog to show the dialog behavior or more details of the listed items. The default value for the dialogText String is null. You can select the items and click OK to return a collection of the selected items or null if you click Cancel.

The code example is shown below:

```
#foreach ($diagram in $dialog.select($Diagram,“Select all feature”, false, "Please
select your required element"))
$diagram.name
#end
```

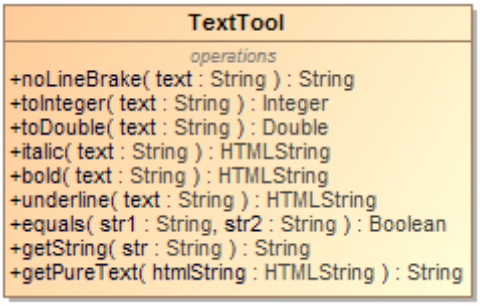
The output will be as follows:



The Selection dialog with text.

Text tool

A Text tool provides extra functions to convert or format text such as change text to bold, italics, add an underline, equals sign etc. The following figure shows a class diagram to represent the Texttool API.



A Class Diagram: TextTool.

Like other Custom tools, the Text tool (*text.jar*) must be installed in the 'extensions' folder of the Report Wizard plugin. For further information on Custom tools and installation, see Custom tool(see page 1952).

To use the APIs of Texttool or other custom tools

---

1. Place the jar file in the **extensions** folder of the Report Wizard plugin folder.
2. Start MagicDraw.
3. Import it to your report by typing the following:

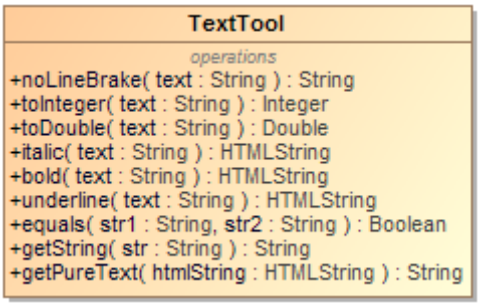
```
#import('text', 'com.nomagic.reportwizard.tools.TextTool')
```

Related pages

Text tool API

On this page

- noLineBreak(String text)(see page 1867)
- toInteger(String text)(see page 1868)
- toDouble(String text)(see page 1868)
- italic(String text)(see page 1868)
- bold(String text)(see page 1869)
- underline(String text)(see page 1869)
- html(String text)(see page 1869)
- getString(String text)(see page 1870)
- equals(String str1, String str2)(see page 1870)
- getPureText(htmlString HTMLString)(see page 1870)



A Class diagram: TextTool

noLineBreak(String text)

This method removes all control characters such as U+0009 (Tab), U+000A (Line Feed), and U+000D (Carriage Return) from a given text. This method also filters text under the category "Cc" in the Unicode specification.

An example of a use case that contains multiple lines
Code line:

```
$usecase.name
```

The output:

Line 1
Line 2
Line 3

An example of the $text.noLineBreak() method
Code line:

```
$text.noLineBreak($usecase.name)
```

The output:

```
Line 1 Line 2 Line 3
```

toInteger(String text)

This method converts a string argument to a signed decimal integer.

An example of the toInteger(String text) method
Code line:

```
#set ($int = $text.toInteger('2'))
#set ($result = $int + 1)
Result is $result
```

toDouble(String text)

This method converts a string argument to a signed decimal double.

An example of the toDouble(String text) method
Code line:

```
#set ($d = $text.toDouble('10.1'))
#set ($result = $d + 2)
Result is $result
```

italic(String text)

This method forces a report to render a given text in the italic style.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

An example of the italic(String text) method
Code line:

```
The $text.italic($class.name) must provide interface for web services.
```

The output:
The *BusinessServices* must provide interface for web services.

bold(String text)

This method forces a report to render a given text in the bold style.

An example of the bold(String text) method
Code line:

```
Method $text.bold($operator.name) must be implemented by implementation
class.
```

The output:
Method **doService** must be implemented by implementation class.

underline(String text)

This method forces a report to render a given text in the underlined style.

An example of the underline(String text) method
Code line:

```
Attribute $text.underline ($attribute.name) contains a service name.
```

The output:
Attribute serverName contains a service name.

html(String text)

This method forces a report to render a given text as HTML.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

An example of the html(String text) method
Code line:

```
$text.html('<ul><li>A</li><li>B</li></ul>')
```

The output:

- A
- B

For more information on supported *HTML* tags, see Appendix D: HTML Tag Support(see page 1867).

getString(String text)

This methods converts text from *RTF* to a Java String.

An example of the getString(String text) method
Code line:

```
#set ($str = $text.getString("Ubersetzer"))
```

The output:

Ubersetzer

equals(String str1, String str2)

This method compares two strings. The result will be true if and only if str1 represents the same sequence of characters as str2 does. This method supports *RTF* text comparison.

An example of the equals(String str1, String str2) method
Code line:

```
#set ($str = $text.getString("Ubersetzer"))
#if ($text.equals($str, "Ubersetzer"))
    true
#end
```

getPureText(htmlString HTMLString)

This method gets plain text from an HTML documentation. All HTML tags that are applied to the text will be removed from the text. Only plain text would be returned.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

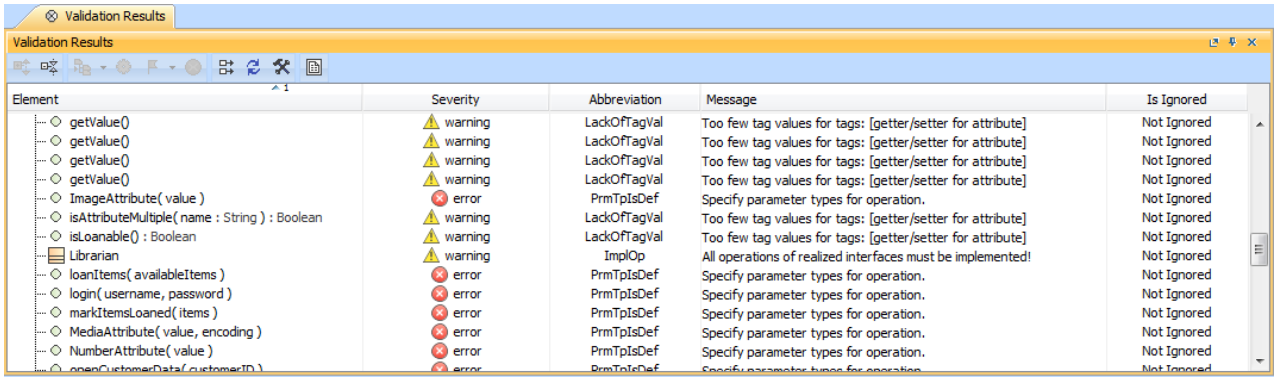
An example of the getPureText(htmlString HTMLString) method
Code line:

```
#set (\$html = '<html><body><b>Text</b></body></html>')
$text.getPureText($html)
```

Model validation tool

The Model Validation tool allows templates to:

- Invoke the Validation menu and validate the models from suite names
- Set the severity.
- Retrieve validation results.



The Model Validation dialog.

Like other Custom tools, the Model Validation tool '*validationtool.jar*' must be presented in the 'extensions' folder of the Report Wizard plugin. For further information on Custom tools and the installation, see Custom tool(see page 1952).

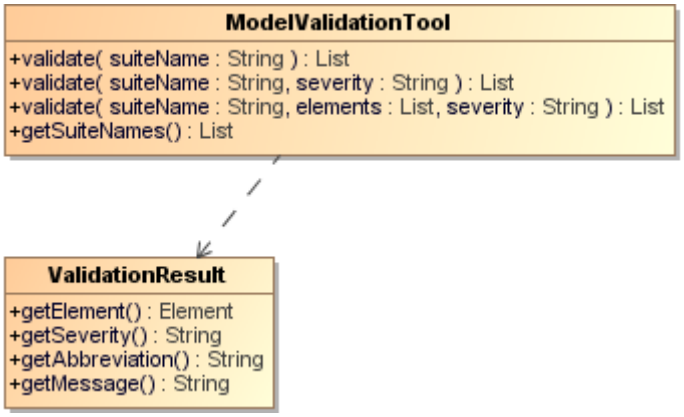
To import a Model Validation tool to a template, type this code in the template.

```
#import('validator', 'com.nomagic.reportwizard.tools.ModelValidationtool')
```

Model validation tool API

On this page

- Validate Methods(see page 1872)
- Getting the Suite Name List(see page 1873)
- Getting the Validation Data from Validation Results(see page 1873)



A Class diagram of a ModelValidationTool.

Class Metricstool

Methods:

- +validate(suiteName : String) : List<ValidationResult>
- +validate(suiteName : String, severity: String) : List<ValidationResult>
- +validate(suiteName : String, element : List, severity : String) : List<ValidationResult>
- +getSuiteNames() : List<String>

Class ValidationResult

Methods:

- +getElement( ) : Element
- +getSeverity( ) : String
- +getAbbreviation() : String
- +getMessage( ) : String

Validate Methods

validate(suiteName : String) : List<ValidationResult>

Uses only a suite name to validate a model from the whole project with default “debug” as severity.

```
#set ($results = $validator.validate("suitename"))
```

validate(suiteName : String, severity : String) : List<ValidationResult>

Uses a suite name and a severity type to validate a model from the whole project. The severity types are “debug”, “*info*”, “*warning*”, “*error*” and “*fatal*”.

```
#set ($results = $validator.validate("suitename", "severity"))
```

validate(suiteName : String, elementList : List<Element>, severity : String) : List<ValidationResult>

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Uses a suite name and a severity type to validate a model from an element list. The severity types are "*d ebug*", "*info*", "*warning*", "*error*", and "*fatal*".

```
#set ($results = $validator.validate("suitename", $elementList, "severity"))
```

To validate all classes by using 'UML completeness constraints' suite and the severity type is warning, for example, type the following code:

```
#set ($results = $validator.validate("UML completeness constraints", $Class,
"warning"))
```

Getting the Suite Name List

The template author can list the entire available Model Validation suite names from 'suiteNames' methods. The 'suiteNames' is available in the ModelValidationtool class.

getSuiteNames() : List<String>

```
#set ($nameList = $validator.suiteNames)
```

$validator represents the ModelValidationtool class.

Prints all Metrics names and their values of all Classes, for example, type the following code:

```
#foreach ($name in $validator.suiteNames)
$name
#end
```

Getting the Validation Data from Validation Results

The object returned from validate methods in the previous section is the list of validation results. The validation results contain columns of validation result with each row of validated models.

getElement() : Element

Gets a validated element.

```
#set ($element = $result.getElement())
```

To print all validated elements' names, for example, type the following code:

```
#set ($results = $validator.validate("UML completeness constraints"))
#foreach ($result in $results)
$result.element.name
#end
```

getSeverity() : String

Gets a severity type.

```
#set ($severity = $result.getSeverity())
or
$result.severity
```

getAbbreviation() : String

Gets an abbreviation.

```
#set ($abbr = $result.getAbbreviation())
or
$result.abbreviation
```

getMessage() : String

Gets a message.

```
#set ($message = $result.getMessage())
or
$result.message
```

isIgnored() : Boolean

Gets an isIgnored value to check whether the validated element is ignored or not.

```
#set ($isIgnored = $result.isIgnored())
or
$result.isIgnored
```

Code examples for model validation tool

The code in this section demonstrates some scenarios with Model Validation tool.

To print all validation results from a project by using the “UML completeness constraints” suite, for example, type the following code:

```
#import('validator', 'com.nomagic.reportwizard.tools.ModelValidationTool')
#set ($results = $validator.validate("UML completeness constraints"))
#foreach ($result in $results)
Element: $result.element.name
Severity: $result.severity
Abbreviation: $result.abbreviation
Message: $result.message
IsIgnored: $result.isIgnored
#end
```

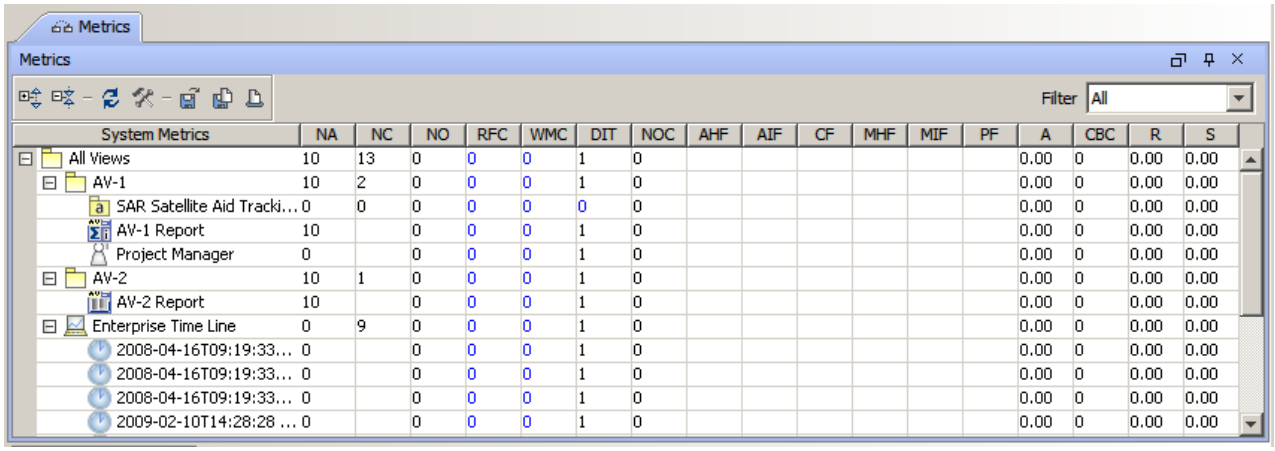
To print only the validation results from the Class by using the “UML Correctness” suite and the "warning" severity, for example, type the following code:

```
#import('validator', 'com.nomagic.reportwizard.tools.ModelValidationTool')
#set ($results = $validator.validate("UML Correctness", $Class, "warning"))
#foreach ($result in $results)
Element: $result.element.name
Abbreviation: $result.abbreviation
Message: $result.message
IsIgnored: $result.isIgnored
#end
```

Metrics tool

The Metrics tool allows templates to:

- Invoke the Metrics menu and calculate the Metric from its suite names.
- Set a filter.
- Retrieve row elements and the Metric value.



System Metrics	NA	NC	NO	RFC	WMC	DIT	NOC	AHF	AIF	CF	MHF	MIF	PF	A	CBC	R	S
All Views	10	13	0	0	0	1	0							0.00	0	0.00	0.00
AV-1	10	2	0	0	0	1	0							0.00	0	0.00	0.00
SAR Satellite Aid Tracki...	0	0	0	0	0	0	0							0.00	0	0.00	0.00
AV-1 Report	10		0	0	0	1	0							0.00	0	0.00	0.00
Project Manager	0		0	0	0	1	0							0.00	0	0.00	0.00
AV-2	10	1	0	0	0	1	0							0.00	0	0.00	0.00
AV-2 Report	10		0	0	0	1	0							0.00	0	0.00	0.00
Enterprise Time Line	0	9	0	0	0	1	0							0.00	0	0.00	0.00
2008-04-16T09:19:33...	0		0	0	0	1	0							0.00	0	0.00	0.00
2008-04-16T09:19:33...	0		0	0	0	1	0							0.00	0	0.00	0.00
2008-04-16T09:19:33...	0		0	0	0	1	0							0.00	0	0.00	0.00
2009-02-10T14:28:28 ...	0		0	0	0	1	0							0.00	0	0.00	0.00

The Metrics Dialog.

Like other Custom tools, the Metrics tool '*metrictool.jar*' must be presented in the 'extensions' folder of the Report Wizard plugin. For further information on Custom tools and the installation, see Custom tool(see page 1952).

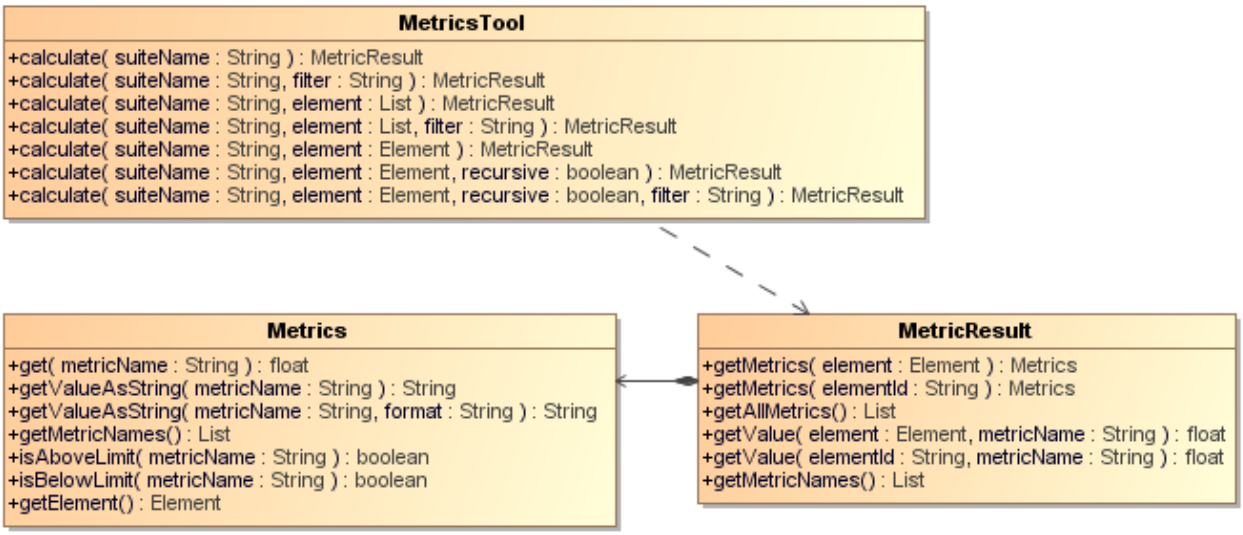
To import the Metrics tool to a template, type this following code in the template:

```
#import('metrics', 'com.nomagic.reportwizard.tools.Metricstool')
```

Metrics tool API

On this page

- Calculation Methods(see page 1877)
- Getting Metric Data from Metric Results(see page 1879)
- Getting Metric Values(see page 1880)
- Getting the Metric Name List(see page 1882)
- Getting the Result Attribute "is above limit" from a Metrics Name(see page 1882)
- Getting the Result Attribute "is below limit" from a Metric Name(see page 1883)



A Class diagram of the MetricsTool.

Class of MetricsTool

Methods:

- +calculate(suiteName : String) : MetricsResult
- +calculate(suiteName : String, filter : String) : MetricsResult
- +calculate(suiteName : String, element : java::util::List) : MetricsResult
- +calculate(suiteName : String, element : java::util::List, filter : String) : MetricsResult
- +calculate(suiteName : String, element : Element) : MetricsResult
- +calculate(suiteName : String, element : Element, recursive : boolean) : MetricsResult
- +calculate(suiteName : String, element : Element, filter : String) : MetricsResult

Class of MetricsResult

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Methods:

- +getMetrics(element : Element) : Metrics
- +getMetrics(elementId : String) : Metrics
- +getAllMetrics() : List<Metrics>
- +getValue(element : Element, metricName : String) : float
- +getValue(elementId : String, metricName : String) : float
- +getMetricNames() : List<String>

Class of Metric

Methods:

- +get(metricName : String) : float
- +getValueAsString(metricName : String) : String
- +getValueAsString(metricName : String, format : String) : String
- +getMetricNames() : List
- +isAboveLimit(metricName : String) : boolean
- +isBelowLimit(metricName : String) : boolean

Calculation Methods

calculate(suiteName : String) : MetricResult

Uses only the suite name to calculate the Metric from an entire project.

```
#set ($metricsResults = $metrics.calculate('suitename'))
```

Example code:

Calculates the Metric from the Actor name “Template Author”.

```
#set ($metricsResults = $metrics.calculate('System Metrics'))
```

calculate(suiteName : String, filter : String) : MetricResult

Uses a suite name and filter to calculate the Metric from an entire project. "filter" is a filter name displayed in the Metric dialog.

```
#set ($metricsResults = $metrics.calculate('suitename', "filter"))
```

> If the language used in the MagicDraw interface changes, the filter name needs to be replaced with a target language, for example: French will be "Violations de Paquetage” for "Package Violations".

calculate(suiteName : String, elementList : List<Element>) : MetricResult

Specifies a suite name and Element list to calculate the Metrics.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

```
#set ($metricsResults = $metrics.calculate('suitename', $elementList))
```

To calculate the Metrics from all classes by using the 'System Metrics' suite, for example, type the following code:

```
#set ($metricsResults = $metrics.calculate('System Metrics', $Class))
```

calculate(suiteName : String, elementList : List<Element>) : MetricResult

Specifies a suite name and Element list to calculate the Metric with a specific filter name.

```
#set ($metricsResults = $metrics.calculate('suitename', $elementList, "filter"))
```

To calculate the Metric from all classes by using the 'System Metrics' suite with the package violations filter, for example, type the following code:

```
#set ($metricsResults = $metrics.calculate('System Metrics', $Class, "Package
Violations"))
```

calculate(suiteName : String, element : Element) : MetricResult

Calculates the Metric from a single element.

```
#set ($metricsResults = $metrics.calculate('suitename', $element))
```

To calculate the Metric from the Actor name "Template Author", for example, type the following code:

```
#set ($actor = $report.findElementInCollection($Actor, "Template Author"))
#set ($metricsResults = $metrics.calculate('System Metrics', $actor))
```

calculate(suiteName : String, element : Element, recursive : boolean) : MetricResult

Calculates the Metric from a single element with the recursive option. If the recursive option is true, it will calculate the result from a given root element and recursive to all children; otherwise (false), calculates the result from the single element.

```
#set ($metricsResults = $metrics.calculate('suitename', $element, $recursive))
```

To calculate the Metric from a root element and recursive to all children. The result is similar to the calculated result from all elements in a project, for example, type the following code:

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

```
#set ($root = $project.model)
#set ($metricsResults = $metrics.calculate('System Metrics', $root, true))
```

calculate(suiteName : String, element : Element, recursive : Boolean, filter : String) : MetricResult

Calculates the Metric from a single element with the recursive option and a filter name. If the recursive option is true, it will calculate the result from a given root element and recursive to all children; otherwise (false), calculate the result from the single element.

```
#set ($metricsResults = $metrics.calculate('suitename', $element, $recursive,
"filter"))
```

Getting Metric Data from Metric Results

The object returned from calculating methods in the previous section is the Metrics results. The Metrics results represent the Metrics Table as shown in MagicDraw. It consists of Element and the Metrics. To get the Metrics and value, we need to pass the element (row) as parameter and get the Metrics (column).

getMetrics(element : Element) : Metrics

Gets the Metric of a specified element.

```
#set ($elementMetric = $metricsResults.getMetrics($element))
```

getMetrics(elementId : String) : Metrics

Gets the Metric of a specific element ID.

```
#set ($elementMetric = $metricsResults.getMetrics("element id"))
```

getAllMetrics() : List<Metrics>

Gets all Metrics at once.

To print all numbers of attribute (NA) value of all elements, for example, type the following code:

```
#set ($metricsResults = $metrics.calculate('System Metrics')
#foreach ($elementMetric in $metricsResults.allMetrics)
$elementMetric.element.name has $elementMetric.NA
#end
```

getValue(element : Element, metricName : String) : float

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

This is the shortcut method to get a specific Metric value of a target element. The Metric name is the abbreviation of Metric, for example, NA, NC, NO, RFC, etc. This value is case-sensitive. See section **Getting Metric Values**.

```
#set ($value = $metricsResults.getValue($element, "Metric name"))
```

getValue(elementId : String, metricName : String) : float

This is the shortcut method to get the specified Metrics value of a target element. The Metric name is the abbreviation of Metric, for example, NA, NC, NO, RFC, etc. This value is case-sensitive. See 21.1.3 Getting Metric Values.

```
#set ($value = $metricsResults.getValue("element id", "Metric name"))
```

Getting Metric Values

The method in Section **Getting Metric Data from Metric Results** will return a Metric object. The Metric object represents all Metric values of a single element. If the Metric contains no value, the returned value will be zero.

get(metricName : String) : float

The Metric name is the abbreviation of Metric, for example, NA, NC, NO, and RFC. This value is case-sensitive.
To return the number of attributes (NA) from Class SwingUtilities, for example, type the following code:

```
#set ($class = $report.findElementInCollection($Class, "SwingUtilities"))
#set ($metricsResults = $metrics.calculate('System Metrics', $class))
#set ($elementMetric = $metricsResults.getMetrics($class))
#set ($value = $elementMetric.get("NA"))
```

The user can use the Velocity technique for a shortcut when retrieving value from a Metrics name. The Metrics name is case-sensitive.

To return the number of an attribute (NA), for example, type the following code:

```
#set ($value = $elementMetric.NA)
```

getValueAsString(metricName : String) : String

You can format an output value as string. The Metric name is the abbreviation of Metric, for example, NA, NC, NO, and RFC. This value is case-sensitive. If the Metric contains no value, the returned string will be an empty string.

```
#set ($value = $elementMetric.getValueAsString('metricName'))
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

The characters in the table below are used in the patterns.

Symbol	Meaning
0	Digit
#	Digit, zero shows as absent
.	Decimal separator or monetary decimal separator
,	Grouping separator
E	Separates mantissa and exponent in scientific notation.
%	Multiply by 100 and show as percentage

The following are some examples of patterns.

Value	Pattern	Result
23.456	###.00	23.46
23.456	000.00	023.46
23.456	###.###	23.456
12345678	###,###.##	12,345,678.00
12345678	#.##	12345678

- Since the symbol contains # (hash character), which is a special character for Velocity Template Language, you needs to use a pair of single quotes instead of double quotes to pass the hash character into the parameter.
- The following code is not in the correct format:

  $elementMetric.getValueAsString("NA", "#.##")
  The code written above will return an unexpected result. Use the following code to get the formatted text.
  $elementMetric.getValueAsString("NA", '#.##')
  However, this method cannot use a single quote character in formatted text.

The following is the sample code:

Return the number of an attribute (NA) from Class SwingUtilities.

```
#set ($class = $report.findElementInCollection($Class, "SwingUtilities"))
#set ($metricsResults = $metrics.calculate('System Metrics', $class))
#set ($elementMetric = $metricsResults.getMetrics($class))
#set ($value = $elementMetric.getValueAsString("NA", '#.##'))
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Getting the Metric Name List

A template author can list an entire available Metric names from 'metricNames' methods. The 'metricNames' is available in both Metric results and Metric class.

getMetricNames() : List<String>

```
#set ($nameList = $elementMetric.metricNames)
```

Or

```
#set ($nameList = $metricsResults.metricNames)
```

$elementMetric represents a Metric class and $metricsResults represents a MetricsResults class.

To print all Metric names and all of their classes' values, for example, type the following code:

```
#set ($metricsResults = $metrics.calculate('System Metrics', $Class))
#foreach ($class in $Class)
$class.humanName
#foreach ($metricName in $metricsResults.metricNames)
$metricName value $metricsResults.getValue($class, $metricName)
#end
#end
```

To print all Metric names and their values of all Classes, for example, type the following code:

```
#set ($metricsResults = $metrics.calculate('System Metrics', $Class))
#foreach ($class in $Class)
#foreach ($elementMetric in $metricsResults.getMetrics($class))
#foreach ($metricName in $elementMetric.metricNames)
$metricName value $elementMetric.get($metricName)
#end
#end
#end
```

Getting the Result Attribute "is above limit" from a Metrics Name

The "is above limit" is a Boolean value that indicates that a value is larger than the value configured in the Metrics menu. The $elementMetric represents a Metric class and $metricsResults represents a MetricsResults class.

```
#set ($isAbove = $metricsResults.isAboveLimit('metricName'))
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To change the HTML text color to red if the number of classes in a project is larger than the limit, for example, type the following code:

```
#set ($metricsResults = $metrics.calculate('System Metrics', $project.model, true))
#set ($elementMetrics = $metricsResults.getMetrics($project.model))
#if ($elementMetrics.isAboveLimit('NC'))
 Total classes is <font color="red">$elementMetrics.get('NC')</font>
#end
#end
```

Getting the Result Attribute “is below limit” from a Metric Name

The “is below limit” is a Boolean value that indicates that a value is less than the value configured in the Metric menu.

The $elementMetric represents a Metric class.

```
#set ($isAbove = $elementMetric.isBelowLimit('metricName'))
```

Example code:
Change the text color to red if the number of responses for the class “Customer” is less than the limit.

```
#set ($class = $report.findElementInCollection($Class, "Customer"))
#set ($metricsResults = $metrics.calculate('System Metrics', $class))
#set ($elementMetrics = $metricsResults.getMetrics($class))
#if ($elementMetrics.isBelowLimit('RFC'))
$class.name is too complex. The total number of methods access to this class is <font
color="blue">$elementMetrics.get('RFC')</font>
#end
```

Code examples for metric tool

The code in this section demonstrates some scenarios with Metric Tool.

(i) Printing all Metric names and their values of all elements under the package “Design” by using the “System Metrics” suite.

```
#import('metrics', 'com.nomagic.reportwizard.tools.MetricsTool')
#set ($package = $report.findElementInCollection($Package, "Design"))
#set ($metricsResults = $metrics.calculate('System Metrics', $package))
Package $package.name contains the following metrics
#foreach ($metricName in $metricsResults.metricNames)
$metricName value $metricsResults.getValue($package, $metricName)
#end
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

(ii) Creating a hierarchy of elements of a project and printing all Metric names and their values by using the “System Metrics” suite.

```
#import('metrics', 'com.nomagic.reportwizard.tools.MetricsTool')
#macro (next $e)
#set ($metricsResults = $metrics.calculate('System Metrics', $e))
Element $e.name
#foreach ($metricName in $metricsResults.metricNames)
$metricName value $metricsResults.getValue($e, $metricName)
#end
#foreach ($child in $e.ownedElement)
#next($child)
#end
#end
#next ($project.model)
```

(iii) Creating a Metric table for Classes by using the “System Metrics” suite and formatting the result by using two-decimal digits.

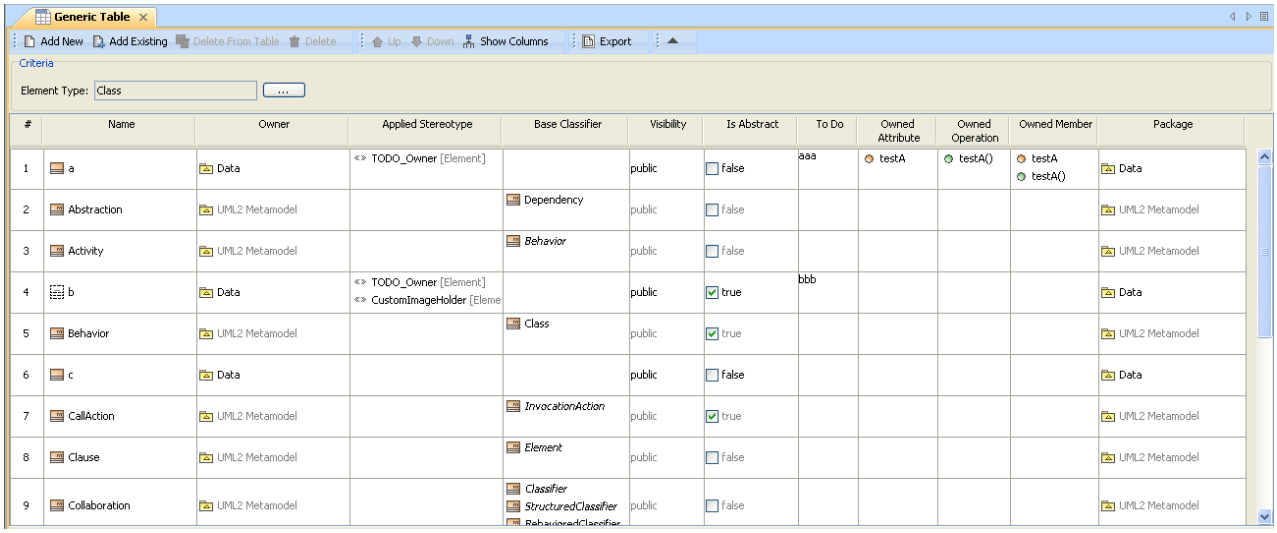
```
#import('metrics', 'com.nomagic.reportwizard.tools.MetricsTool')
#set ($metricsResults = $metrics.calculate('System Metrics', $Class))
#foreach ($cls in $Class)
Class $cls.name
#set ($elementMetric = $metricsResults.getMetrics($cls))
#foreach ($name in $elementMetric.metricNames)
$name : $elementMetric.getValueAsString($name, '##.00')
#end
#end
```

Generic table tool

The Generic Table tool allows templates to:

- Access data of the Generic Table.
- Use diagrams to get data from the Generic Table.
- Get the row elements.
- Get the column names.
- Get the cell values.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The Generic Table Dialog.

Like other Custom Tools, the Generic Table tool '*generictabletool.jar*' must be presented in the 'extensions' folder of the Report Wizard plugin. For further information on Custom Tools and the installation, see section Custom tool(see page 1952).

To import the Generic Table tool to a template, for example, type the following code in the template.

```
#import('generic', 'com.nomagic.reportwizard.tools.GenericTableTool')
```

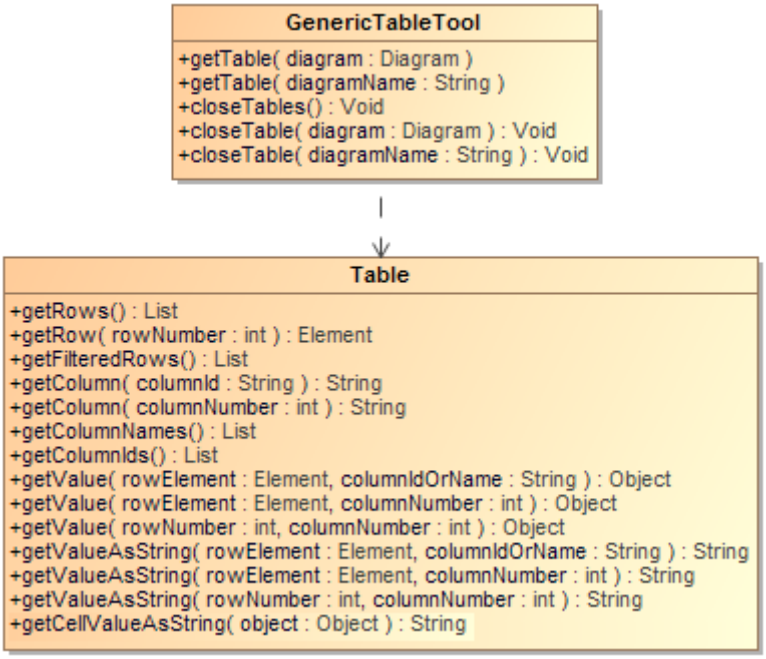
Related pages

Generic table tool API

On this page

- Getting generic table data(see page 1887)
  - Getting generic table instances from diagram elements(see page 1887)
  - Getting generic table instances from diagram names(see page 1888)
- Closing tables(see page 1888)
  - Closing all diagram tables(see page 1888)
  - Closing a specific table diagram(see page 1888)
  - Closing a table diagram with a specific diagram name(see page 1888)
- Getting row elements(see page 1889)
  - Getting all row elements(see page 1889)
  - Getting all filtered row elements(see page 1889)
  - Getting row elements in specific row numbers(see page 1890)
- Getting column names(see page 1890)
  - Getting column names from column ID(see page 1890)
  - Getting column names from column numbers(see page 1890)
  - Getting all column names(see page 1891)

    - Getting all column IDs(see page 1891)
- Getting cell values(see page 1891)
    - Getting values from row element and column ID(see page 1891)
    - Getting values from row element and column name(see page 1892)
    - Getting values from row element and column number(see page 1892)
    - Getting values from row and column numbers(see page 1892)
    - Getting values from row element and column IDs as string(see page 1893)
    - Getting values from row element and column name as string(see page 1893)
    - Getting values from row element and column number as string(see page 1893)
    - Getting values from row and column numbers as string(see page 1894)
    - Getting cell value from returned object as string(see page 1894)
- Getting visible column and cell values(see page 1894)
    - Getting all visible column IDs(see page 1895)
    - Getting visible column names from column numbers(see page 1895)
    - Getting visible values from row elements and column number(see page 1895)
    - Getting visible values from row number and column number(see page 1895)
    - Getting visible values from row element and column number as string(see page 1896)
    - Getting visible values from row and column numbers as string(see page 1896)



GenericTableTool diagram.

Class GenericTableTool:

Methods:

- +getTable(diagram : Diagram) : Table
- +getTable(diagramName : String) : Table
- +closeTables() : Void
- +closeTable(diagram : Diagram) : Void
- +closeTable(diagramName : String) : Void

Class Table:

Methods:

- +getRows() : List<Element>
- +getRow(rowNumber : int) : Element
- +getFilteredRows() : List<Element>
- +getColumn(columnId : String) : String
- +getColumn(columnNumber : int) : String
- +getColumnNames() : List<String>
- +getColumnIds() : List<String>
- +getValue(rowElement : Element, columnIdOrName : String) : Object
- +getValue(rowElement : Element, columnNumber : int) : Object
- +getValue(rowNumber : int, columnNumber : int) : Object
- +getValueAsString(rowElement : Element, columnIdOrName : String) : String
- +getValueAsString(rowElement : Element, columnNumber : int) : String
- +getValueAsString(rowNumber : int, columnNumber : int) : String
- +getVisibleColumnIds() : List<String>
- +getVisibleColumn(columnNumber : int) : String
- +getVisibleValue(rowElement : Element, columnNumber : int) : Object
- +getVisibleValue(rowNumber : int, columnNumber : int) : Object
- +getVisibleValueAsString(rowElement : Element, columnNumber : int) : String
- +getVisibleValueAsString(rowNumber : int, columnNumber : int) : String

Getting generic table data

A Generic Table is a special diagram. You can retrieve it by using the variable $Diagram. Thus, every method must accept the diagram instance or diagram name.

Use the following method to get the Generic Table instance. We use Generic Table instances to retrieve table information such as row elements and column names.

Getting generic table instances from diagram elements

To get a Generic Table instance from a specified diagram element, use the following code.

getTable(diagram : Diagram) : Table

```
#set($table = $generic.getTable($diagram))
```

Where the parameter is:

- diagram – the diagram element.

Return the instance of a Generic Table.

Example code:

```
#foreach($diagram in $project.getDiagrams("Generic Table"))
#set($table = $generic.getTable($diagram))
#end
```

Getting generic table instances from diagram names

To get a Generic Table instance from a specific diagram name, use the following code.

getTable(diagramName : String) : Table

```
#set($table = $generic.getTable($diagram))
```

Where the parameter is:

- diagram – the diagram name.

Return the instance of a Generic Table.

Example code:

```
#set($diagram = "diagram name")
#set($table = $generic.getTable($diagram))
```

Closing tables

Use the following method to close a table diagram.

Closing all diagram tables

CloseTables() : Void

```
$generic.closeTables()
```

Closing a specific table diagram

closeTable(diagram: Diagram) : Void

```
$generic.closeTable($diagram)
```

Where the parameter is:

- diagram – a table diagram you want to close.

Closing a table diagram with a specific diagram name

closeTable(diagramName: String) : Void

```
$generic.closeTable($diagramName)
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Where the parameter is:

- diagramName – the name of a table diagram you want to close.

Getting row elements

The Generic Table consists of a series of row elements and column names. Use the following methods to retrieve a list of row elements.

Getting all row elements

getRows() : List<Element>

```
$table.getRows()
```

The returned value is a list of Elements. The following shows an example of how to print all row element names.

```
#foreach($row in $table.rows)
$row.name
#end
```

Getting all filtered row elements

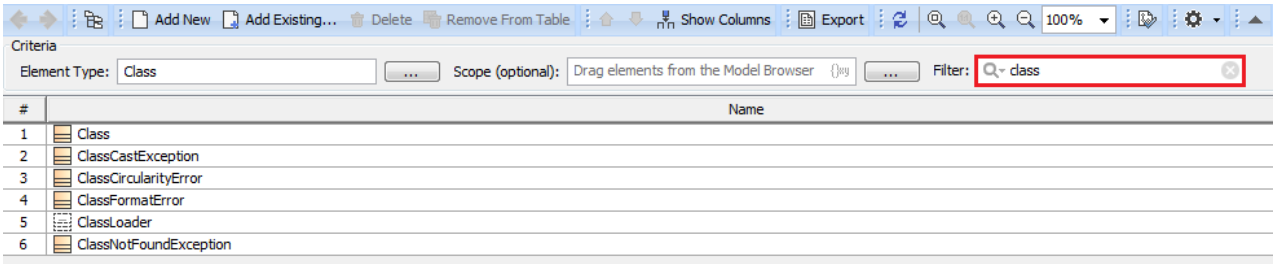
getFilteredRows() : List<Element>

```
$table.getFilteredRows()
```

The returned value is a list of Elements that have been filtered by the Filter field.

```
#foreach($row in $table.getFilteredRows())
$row.name
#end
```

The figure below shows the rows returned after applying the filter.



Rows returned after filtering.

Getting row elements in specific row numbers

getRow(rowNumber : int) : Element

```
$table.getRow($rowNumber)
```

Where the parameter is:

- rowNumber – the row number starts with 0.

The returned value is an Element.

Getting column names

Column names are referred by column IDs. A column is a MagicDraw QProperty ID that is not readable or accessible by the user. We need to create other methods that provide better usability than using QProperty ID. Use the following methods to retrieve column names.

> The methods in this section retrieves column names from the list returned by Show Columns menu item from generic table diagram. The methods in the preceding section, however, retrieves column names from the diagram itself and not from the list returned by Show Columns menu item.

Getting column names from column ID

getColumn(columnId : String) : String

```
$table.getColumn($columnId)
```

Where the parameter is:

- columnId – the ID of a column. Using column IDs has a benefit of consistency between different languages, for example, French and English. The column ID can be retrieved by the method getColumnIds().

The returned value is the name of a column.

Getting column names from column numbers

getColumn(columnNumber : int) : String

```
$table.getColumn($columnNumber)
```

Where the parameter is:

- columnNumber – a column number starts with 1. The column number 0 is the row number.

The returned value is the name of a column.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Getting all column names

getColumnNames() : List<String>

```
$table.getColumnNames()
```

The returned value is a list of column names.

Example code:

Prints all column names.

```
#foreach($colname in $table.getColumnNames())
$colname
#end
```

Getting all column IDs

getColumnIds() : List<String>

```
$table.getColumnIds()
```

The returned value is a list of column IDs.

Example code:

Print all columns IDs

```
#foreach($colid in $table.getColumnIds())
$table.getColumn($colid)
#end
```

Getting cell values

A cell value is the value in a row element with a column name. To get a cell value, you need to provide both the row element and column. A cell value is an Element or row number. Use the following methods to retrieve a cell value.

Getting values from row element and column ID

getValue(rowElement : Element, columnIdOrName : String) : Object

```
$table.getValue($rowElement, $columnId)
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Where the parameters are:

- rowElement – a row element.
- columnId – a column ID.

The returned value is the value in a cell. If the cell contains an element, the value is the Element.
Example code:

Prints the row, column, and value.

```
#foreach($diagram in $project.getDiagrams("Generic Table"))
#set($table = $generic.getTable($diagram))
#foreach($row in $table.getRows())
<h1>$row.name</h1>
#foreach($col in $table.getColumnIds())
$table.getColumn$col) : $table.getValue($row, $col)
#end
#end
#end
```

Getting values from row element and column name

getValue(rowElement : Element, columnIdOrName : String) : Object

```
$table.getValue($rowElement, $columnName)
```

Where the parameters are:

- rowElement – a row element.
- columnName – a column name.

The returned value is the value in a cell. If the cell contains an element, the value is the Element.

Getting values from row element and column number

getValue(rowElement : Element, columnNumber : int) : Object

```
$table.getValue($rowElement, $columnNumber)
```

Where the parameters are:

- rowElement – a row element.
- columnName – a column number starts with 1. The column number 0 is the row number.

The returned value is the value in a cell. If the cell contains an element, the value is the Element.

Getting values from row and column numbers

getValue(rowNumber : int, columnNumber : int) : Object

```
$table.getValue($rowNumber, $columnNumber)
```

Where the parameters are:

- rowNumber – a row number starts with 0.
- columnNumber – a column number starts with 1. The column number 0 is the row number.

The returned value is the value in a cell. If the cell contains an element, the value is the Element.

Getting values from row element and column IDs as string

getValueAsString(rowElement : Element, columnIdOrName : String) : String

```
$table.getValueAsString($rowElement, $columnId)
```

Where the parameters are:

- rowElement – a row element.
- columnId – a column ID.

The returned value is the value in a cell and is converted into String. The String value is created by MagicDraw text representation API.

Getting values from row element and column name as string

getValueAsString(rowElement : Element, columnIdOrName : String) : String

```
$table.getValueAsString($rowElement, $columnName)
```

Where the parameters are:

- rowElement – a row element
- columnName – a column name

The returned value is the value in a cell and is converted into String. The String value is created by MagicDraw text representation API.

Getting values from row element and column number as string

getValueAsString(rowElement : Element, columnNumber : int) : String

```
$table.getValueAsString($rowElement, $columnNumber)
```

Where the parameters are:

- rowElement – a row element.
- columnName – a column number starts with 1. The column number 0 is the row number.

The returned value is the value in a cell and is converted into String. The String value is created by MagicDraw text representation API.

Getting values from row and column numbers as string

getValueAsString(rowNumber : int, columnNumber : int) : String

```
$table.getValueAsString($rowNumber, $columnNumber)
```

Where the parameters are:

- rowNumber – a row number starts with 0.
- columnNumber – a column number starts with 1. The column number 0 is the row number.

The returned value is the value in a cell and is converted into String. The String value is created by MagicDraw text representation API.

Getting cell value from returned object as string

getCellValueAsString(object : Object) : String

```
$table.getCellValueAsString($object)
```

Where the parameter is:

- object – an object returned from the getValue function.

The returned value is the object in cell value converted into String.

For example:

```
#set($result = $table.getValue($row, $id))
#if($list.isArray($result))
    #foreach($r in $result)
    - $table.getCellValueAsString($r)
    #end
#else
    $table.getCellValueAsString($result)
#end
```

Getting visible column and cell values

A visible cell value is the column which is visible on a diagram. Unlike the methods described in section 23.1.5 Getting Cell Values, the following methods will retrieve only the value which is visible on a diagram.

Getting all visible column IDs

getVisibleColumnIds() : List<String>

```
$table.getVisibleColumnIds()
```

The returned value is a list of visible column IDs.

Getting visible column names from column numbers

getVisibleColumn(columnNumber : int) : String

```
$table.getVisibleColumn($columnNumber)
```

Where the parameter is:

- columnNumber – a column number starts with 1. The column number 0 is the row number.

The returned value is the name of a visible column.

Getting visible values from row elements and column number

getVisibleValue(rowElement : Element, columnNumber : int) : Object

```
$table.getVisibleValue($rowElement, $columnNumber)
```

Where the parameters are:

- rowElement – a row element.
- columnNumber – a column number starts with 1. The column number 0 is the row number.

The returned value is the value in a visible cell. If the cell contains an element, the value is the Element.

Getting visible values from row number and column number

getVisibleValue(rowNumber : int, columnNumber : int) : Object

```
$table.getVisibleValue($rowNumber, $columnNumber)
```

Where the parameters are:

- rowNumber – a row number starts with 0.
- columnNumber – a column number starts with 1. The column number 0 is the row number.

The returned value is the value in a visible cell. If the cell contains an element, the value is the Element.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Getting visible values from row element and column number as string

getVisibleValueAsString(rowElement : Element, columnNumber : int) : String

```
$table.getVisibleValueAsString($rowElement, $columnNumber)
```

Where the parameters are:

- rowElement – a row element.
- columnNumber – a column number starts with 1. The column number 0 is the row number.

The returned value is the value in a visible cell and is converted into String. The String value is created by MagicDraw text representation API.

Getting visible values from row and column numbers as string

getVisibleValueAsString(rowNumber : int, columnNumber : int) : String

```
$table.getVisibleValueAsString($rowNumber, $columnNumber)
```

Where the parameters are:

- rowNumber – a row number starts with 0.
- columnNumber – a column number starts with 1. The column number 0 is the row number.

The returned value is the value in a visible cell and is converted into String. The String value is created by MagicDraw text representation API.

Code examples for generic table tool

The code in this section demonstrates some scenarios with a Generic Table tool. The code may not be executable or may not return the correct result. This is only the idea of how to use the Generic Table tool and how it works.

The code shown below prints the row, column, and its value in the form of an HTML table.

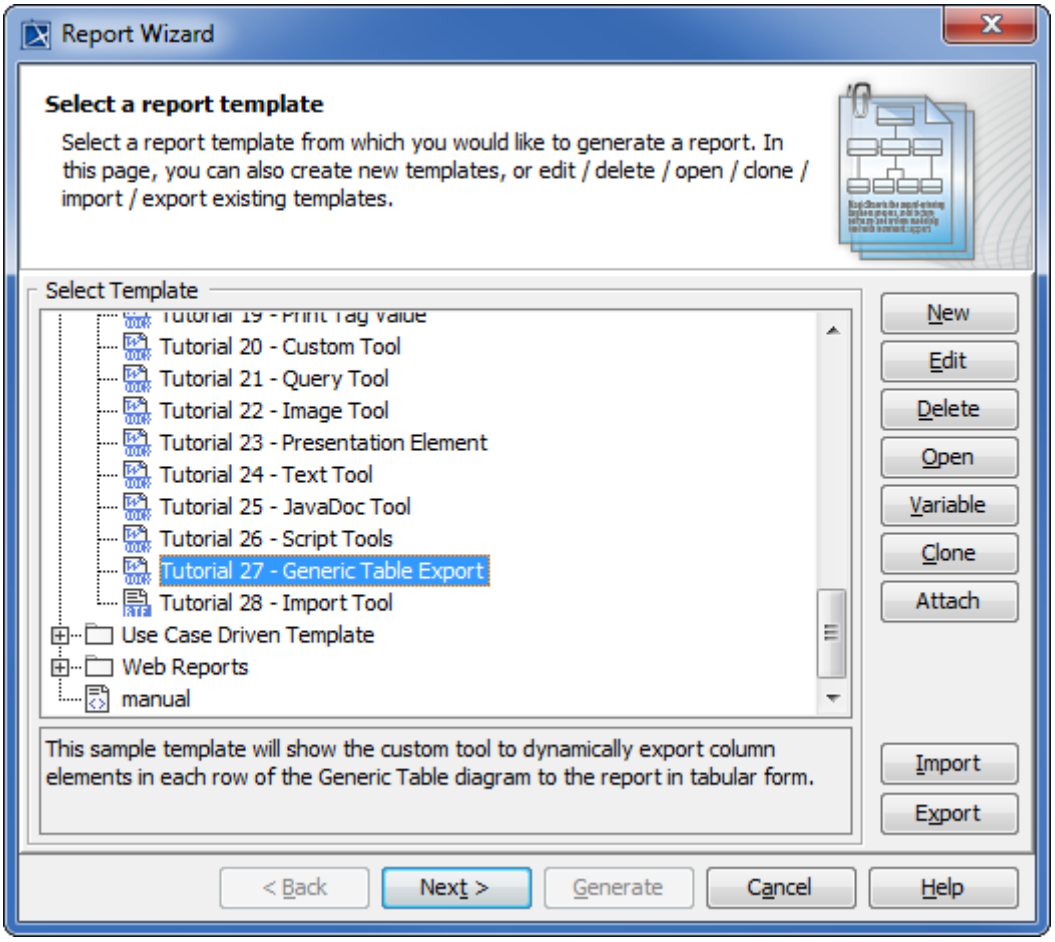
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

```
#foreach($diagram in $project.getDiagrams("Generic Table"))
      <table>
      <tr>
            #set($table = $generic.getTable($diagram))
                <td>#</td>
            #foreach($colName in $table.columnNames)
                    <td>$colName</td>
      #end
      </tr>
      #foreach($row in $table.rows)
            <tr>
                    <td>$velocityCount</td>
            #foreach($col in $table.columnIds)
                    <td>$table.getValueAsString($row, $col)</td>
            #end
            </tr>
        #end
        </table>
    #end
```

How to learn more about the GenericTableTool

1. From MagicDraw main menu choose **Tools** > **Report Wizard**.
2. In the Select Template pane, choose **Tutorial** > **Tutorial 27 - Generic Table Export**.
3. Click **Next**, and choose **Built-in** for Select Report Data.
4. Click **Next** again, Select Element Scope, and then click **Next** again. Select output options and click **Generate**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

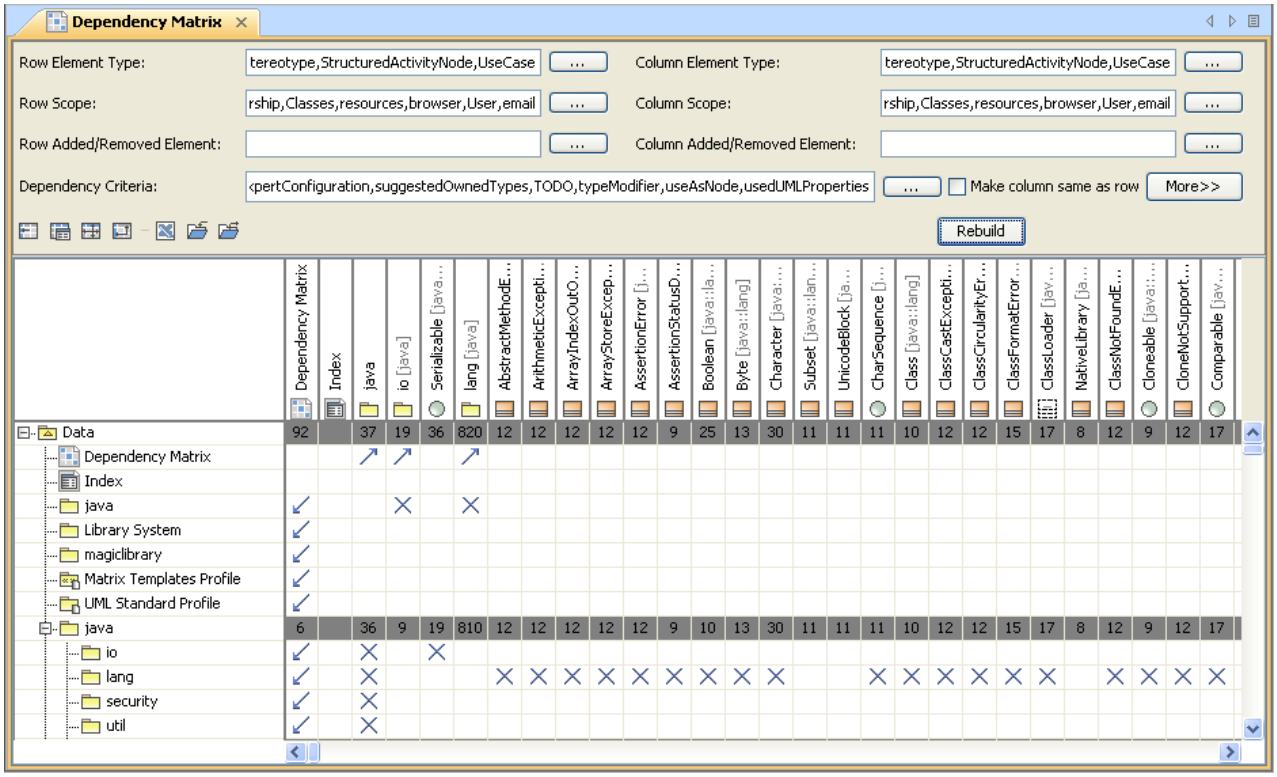


The Generic Table Export dialog.

Dependency Matrix tool

The Dependency Matrix tool allows templates to:

- Access data from Dependency Matrix.
- Use diagrams to get data from Dependency Matrix.
- Get row nodes and elements.
- Get column nodes and elements.
- Get dependencies in a cell (between specific row and column nodes).



The Dependency Matrix diagram.

Like other Custom tools, the Generic Table tool '*dependencymatrixtool.jar*' must be presented in the 'extensions' folder of the Report Wizard plugin. For further information on Custom tools and the installation, see section Custom tool(see page 1952).

To import the Dependency Matrix tool to a template, type the following code in the template:

```
#import("depmatrix","com.nomagic.reportwizard.tools.DependencyMatrixTool")
```

Related pages

Dependency matrix tool API

On this page

- Using Diagrams to Get Data from Dependency Matrix(see page 1900)
- Getting Dependency Matrix Instances from Diagram Elements(see page 1900)
- Getting Dependency Matrix Instances from Diagram Names(see page 1901)
- Getting Rows and Columns(see page 1901)
- Getting All Rows(see page 1901)
- Getting all Columns(see page 1902)
- Getting Dependencies between Rows and Columns(see page 1902)

Dependency Matrix tool allows templates to access data from a Dependency Matrix, use diagrams to get data, row elements, column names, or relations between row and column elements from the matrix.

You can use the following Dependency Matrix tool API in the template.

Class DependencyMatrixTool

Use the following methods to use diagrams to get data.

- +getMatrix(diagram : Diagram) : Matrix
- +getMatrix(diagramName : String) : Matrix

Class Matrix

Use the following methods to get matrix row elements or nodes, column elements or nodes, and dependencies between these nodes/elements.

- +getRowNodes() : List<ElementNode>
- +getColumnNodes() : List<ElementNode>
- +getCellDependencies(rowNode: ElementNode, columnNode: ElementNode)
- +getRowElements() : List<Element>
- +getColumnElements() : List<Element>
- +getRelation(row: Element, column : Element) : List<Relation>

Class Relation

Describes dependency in a Dependency Matrix cell. Use the following methods for class relations.

- +getName() : String
- +getElement() : Element
- +getSemanticType() : String
- +getDirection() : String

Using Diagrams to Get Data from Dependency Matrix

A Dependency Matrix is a special diagram. You can retrieve it using the $Diagram variable. Thus, every method must accept a diagram instance or a diagram's name. Use the following methods to return a Dependency Matrix instance. You can retrieve rows and columns from the Dependency Matrix instances.

Getting Dependency Matrix Instances from Diagram Elements

getMatrix(diagram : Diagram) : Matrix
To get a Dependency Matrix instance from a specified diagram element, use the following code.

```
#set($matrix = $depmatrix.getMatrix($diagram))
```

Where the parameter is:

- diagram – a diagram element

Return an instance of Dependency Matrix.

For example:

```
#foreach($diagram in $project.getDiagrams("Dependency Matrix"))
#set($matrix = $depmatrix.getMatrix($diagram))
#end
```

Getting Dependency Matrix Instances from Diagram Names

getMatrix(diagramName : String) : Matrix
To get a Dependency Matrix instance from a specified diagram's name.

```
#set($matrix = $depmatrix.getMatrix($diagram))
```

Where the parameter is:

- diagram – a diagram's name

Return a Dependency Matrix instance.

Example code:

```
#set($diagram = "diagram name")
#set($matrix = $depmatrix.getMatrix($diagram))
```

Getting Rows and Columns

The Matrix consists of rows and columns. Each row or column is an instance of ElementNode object. Each ElementNode represents a single model Element. There might be multiple ElementNodes for the same model Element in a Dependency Matrix diagram, since the same model Element can be represented multiple times in a diagram.

Getting All Rows

getRowNodes() : List<ElementNode>

Use this method to retrieve a list of rows, ordered in the same way as actually displayed in the diagram. Nodes representing grouping packages, that do not belong to the actual matrix data, are excluded.

```
$matrix.getRowNodes()
```

The returned value is a list of ElementNodes. Method **getElement() : Element** can then be used to retrieve the model element represented by a row.

To print all row elements' names, for example, type the following code:

```
#foreach($rowNode in $matrix.rowNodes)
$rowNode.element.name
#end
```

Getting all Columns

getColumnNodes() : List<ElementNode>

Use this method to retrieve a list of columns, ordered in the same way as actually displayed in the diagram. Nodes representing grouping packages, that do not belong to the actual matrix data, are excluded.

```
$matrix.getRowNodes()
```

The returned value is a list of ElementNodes. Method **getElement() : Element** can then be used to retrieve the model element represented by a column.

To print all column elements' names, for example, type the following code:

```
#foreach($col in $matrix.columns)
$colNode.element.name
#end
```

Getting Dependencies between Rows and Columns

getCellDependencies(rowNode: ElementNode, columnNode: ElementNode)

Use this method to retrieve the relations in a specific matrix cell, which is identified by provided row and column nodes.

```
$matrix.getCellDependencies($rowNode, $columnNode)
```

Where the parameter is:

- rowNode – ElementNode of a row
- columnNode – ElementNode of a column

The returned value is a list of Relations.

The Relation class contains the following methods:

- **getSemanticType** : String
  Return the semantic type
- **getElement** : Element
  Return a relationship element or null if the relationship is not an element, for example, tag name.
- **getName** : String
  Return a relationship name. If the relationship is a tag name, it will return the tag name. If

the relationship is NamedElement, it will return the element name; otherwise, return a human name.

- **getDirection** : String
  Return the direction.

To print the row, column, and its relationship name, for example, type the following code:

```
#foreach($row in $matrix.rowNodes)
   #foreach($col in $matrix.columnNodes)
      #foreach($rel in $matrix.getCellDependencies($row, $col))
         $row.element.name has $rel.name with $col.element.name
      #end
   #end
#end
```

To print rows, columns, and relations in a spreadsheet file format, for example, type the following code:

#import('depmatrix','com.nomagic.reportwizard.tools.DependencyMatrixTool') #forpage ($diagram in $report.filterDiagram($Diagram, ["Dependency Matrix"])) #set($matrix=$depmatrix.getMatrix($diagram)) $bookmark.create($diagram.name)		
	#forcol ($col in $matrix.columnNodes) $col.element.name#endcol	
#forrow($row in $matrix.rowNodes) $row.element.name	#forcol ($col in $matrix.columns)#if(!$report.isEmpty($matrix.getCellDependencies($row,$col))) ->#end#endcol	#endrow
#endpage		

Examples of dependency matrix tool

The code in this section demonstrates some scenarios with using the Dependency Matrix tool.

For not empty cells, prints the row element name, column element name and the name of dependency inside the intersecting cell

```
#foreach ($diagram in $project.getDiagrams("Dependency Matrix"))
 #set ($matrix = $depmatrix.getMatrix($diagram))
 #foreach ($row in $matrix.getRowNodes())
 #foreach ($col in $matrix.getColumnNodes())
 #foreach ($rel in $matrix.getCellDependencies($row, $col))
 $row.element.name has $rel.name with $col.element.name
 #end
 #end
 #end
#end
```

Counts the number of relations for each column.

```
#foreach ($diagram in $project.getDiagrams("Dependency Matrix"))
#set ($matrix = $depmatrix.getMatrix($diagram))
#foreach ($col in $matrix.getColumnNodes())
#set ($count = 0)
#foreach ($row in $matrix.getRowNodes())
#set ($numberOfrelation = $matrix.getCellDependencies($row, $col).size())
#set ($count = $count + $numberOfrelation)
#end
$col.element.name has $count relations
#end
#end
```

Query tool

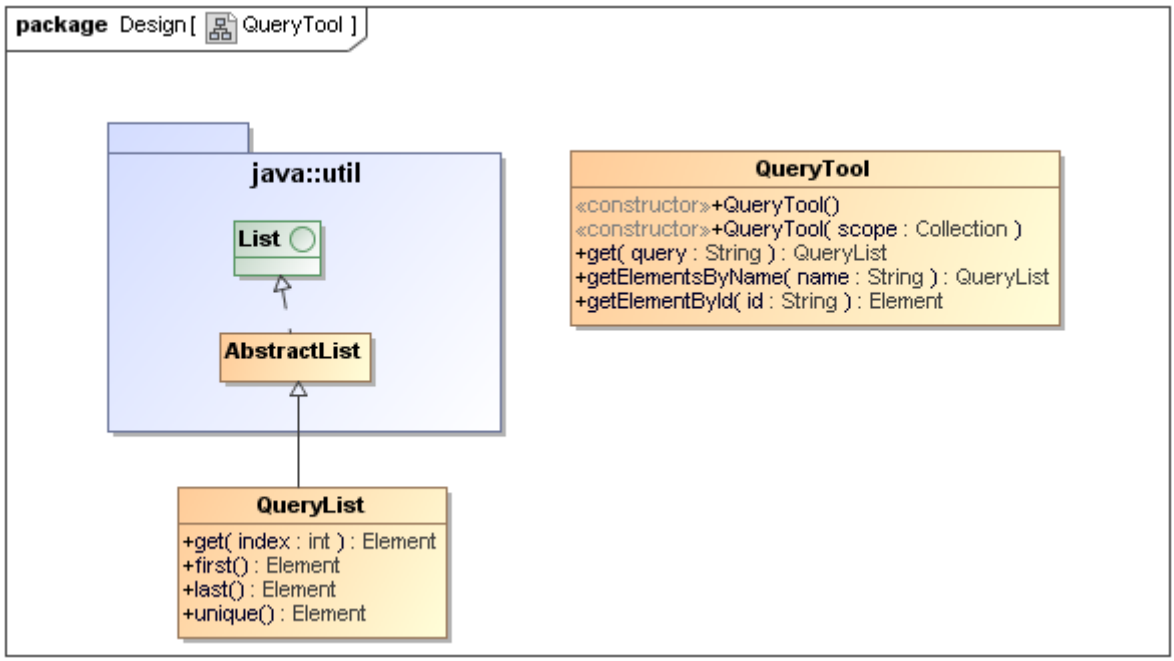
The Query tool provides extra functions to retrieve MagicDraw elements by using a query language. Like other Custom tools, the Query tool '*querytool.jar*' must be presented in the 'extensions' folder of the Report Wizard plugin. For further information on Custom tools and the installation, see Custom tool(see page 1952).

Type the following code in a template to import the Query tool.

```
#import('query', 'com.nomagic.reportwizard.tools.QueryTool')
```

Related pages

Query tool API



The Query Tool Class diagram.

Class QueryTool

Constructor

- QueryTool()
  Creates the Query tool from your selected element scope. This method will use a default scope from the variable $elements.
- QueryTool(scope : Collection<Element>)
  Creates the Query tool from a specific element scope.

Methods

- +get(query :String) : QueryList
- +getElementsByName(name : String) : QueryList
- +getElementById(name : String) : Element

Class QueryList

Methods

- +get(index : int) : Element
- +first() : Element
- +last() : Element
- +unique () : Element

Recognizable query patterns

Query patterns are described as a CSS3 selector pattern (http://www.w3.org/TR/css3-selectors). The following table shows all available query patterns of the query tool.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Pattern	Meaning
E	An element of type E.
E[foo]	An E element with a “foo” attribute.
E[foo=bar]	An E element whose “foo” attribute value exactly equals “bar”.
E[foo^=bar]	An E element whose “foo” attribute value begins exactly with the string “bar”.
E[foo$=bar]	An E element whose “foo” attribute value ends exactly with the string “bar”.
E[foo*=bar]	An E element whose “foo” attribute value contains the substring string “bar”.
E:empty	An E element that has no children.
#myid	An element with an ID equals “myid”.
E F	An F element descendant of an E element.
E > F	An F element child of an E element.

Query by Type

A pattern allows a template to query elements from a given type name, for example:

```
class
Retrieve all classes from a selected package scope.
```

A Type name is generated from the $element.elementType property.

Query by Attribute

A pattern allows a template to query elements from given attribute name and value. When a pattern is used as an expression to match an element, the attribute pattern must be considered to match the element and that element must have an attribute that matches the attribute represented by the attribute pattern.

[attr]
: Matches an element with an attr attribute regardless of the value of the attribute.

[attr=value]
: Matches an element with an attr attribute whose value is exactly val.

The Query tool matches against case sensitivity of attribute names and values in a pattern. A *VTL* property syntax (Camel case without space) will be used for the attribute names. You can use spaces in writing values.

When the value of an attribute is NamedElement, the element's name will be used as a default value. An asterisk (*) can be used to match any element type when combined with the other patterns, for example:

**[name=foo]*
Retrieve any element whose name attribute is exactly foo. Use an asterisk to match any element.

actor[name=foo]
Retrieve all actor elements whose name attribute is exactly foo.

actor[name=]
Retrieve all actor elements whose name attribute is empty.

class[owner=foo]
Retrieve class elements whose owner attribute name is exactly foo.

class[owner=foo][name=bar]
Retrieve class elements whose owner attribute name is exactly foo and name attribute is exactly bar.

Query by Substring Matching Attribute

This topic provides some additional attribute patterns for matching substrings in the value of an attribute:

[attr^=val]
Matches an element with an attr attribute whose value begins with the prefix val. If val is an empty string, the query returns nothing.

[attr$=val]
Matches an element with an attr attribute whose value ends with the suffix val. If val is an empty string, the query returns nothing.

[attr*=val]
Matches an element with an attr attribute whose value contains at least one instance of the substring val. If val is an empty string, the query returns nothing.

The Query tool matches against case sensitivity of attribute names and values in a pattern. A *VTL* property syntax (Camel case without space) will be used for attribute name. You can use spaces in writing values.

When value of attribute is NamedElement, the element name will be used as default value, for example:

```
*[name^=foo]
Retrieve any element whose name attributes begins with foo. Use an asterisk to match
any element.
actor[name$=foo]
Retrieve all actor elements whose name attribute is ends with foo.
class[qualifiedName*=nomagic]
Retrieve class elements whose qualifiedName attribute contains string nomagic.
```

Query Element without Children

You can use :empty to query an element that has no children at all. If an element does not any children, its “Owned Element” attribute value has a zero size of items, for example:

```
package:empty
Retrieve any package whose ownedElement attribute is empty.
package[name=foo]:empty
Retrieve any package whose name attribute is exactly foo and ownedElement attribute
is empty.
```

Query by Element ID

A pattern allows a template to query elements from a given element’s ID. An element’s ID is unique in a project, for example:

```
#17_0_3_8e9027a_1325220298609_477646
Retrieve an element whose ID is exactly 17_0_3_8e9027a_1325220298609_477646.
```

An element’s ID is generated from the $element.elementID property.

Query by Descendent

A pattern allows a template to query an element that is the descendant of another element, for example, an Actor that is contained within a Package. A descendant query is one or more white-spaces that separate(s) two sequences of patterns.

You can use an asterisk (*) to match any element type when combined with the other patterns, for example:

```
package class
Retrieve all classes that are the descendants of a package element.
package foo
 - class bar1
 - class bar2
 - class bar3
The classes bar1, bar2, and bar3 are the results.
package class *[name=foo]
Retrieve any element that (i) has a name value exactly foo and (ii) is inside a class
that is inside a package.
package * property
Retrieve a property element that is the grandchild or a later descendant of a package
element.
```

Query by Child Element

A pattern allows a template to query elements that are the children of another element (describes a parent-child relationship between two elements). An optional white-space around a greater sign can be ignored.

You can use an asterisk (*) to match any element type when combined with the other patterns, for example:

```
package > class
Retrieve all classes that are children of a package.
package foo
 - class bar1
 - class bar2
 - class bar3
The classes bar1 and bar2 are the results.
```

Additional Conditions for Query Patterns

When you are writing queries, be sure you observe the following conditions.

- A Query pattern cannot contain only spaces, for example:

```
$query.get('') or $query.get(' ')
An invalid query pattern.
```

- An element type cannot contain [,], :, *, >, #, and white space.
- Characters #, >, and white-space are the end statements except in an attribute name or value, for example:

```
$query.get("class#17_0_3_8e9027a") equivalent to
$query.get("class #17_0_3_8e9027a")
Retrieve an element's ID 17_0_3_8e9027a that is a grandchild or a later descendant of
a class element.
$query.get("class>property") equivalent to
$query.get("class > property")
Retrieve a property element that is the child of a class element.
$query.get("class property")
Retrieve a property element that is a grandchild or a later descendant of a class
element.
```

An end statement is required to separate an element and its child, or grandchild, for example:

```
$query.get("class[attr] property")
Retrieve a property element that is a grandchild or a later descendant of a class
element that contains an attr attribute.
$query.get("class[attr]property")
An invalid query pattern.
```

- Characters = , ^=, $=, and *= are keywords and all strings except] that are written next to these keywords mean the attribute values.
- Attribute names are mandatory.
- An attribute name cannot contain white-space, =, ^=, $=, *=, and], for example:

```
$query.get("*[ name =bean]") is equivalent to
$query.get("*[name=bean]")
Retrieve all elements whose name attribute is bean.
$query.get("*[ element type = bean ] ")
An invalid query pattern.
```

If an attribute value is not in any double quotes or single quotes, all strings, except], after a keyword mean an attribute value, for example:

```
$query.get("*[name=bean]")
 Retrieve all elements whose name attribute is bean
```

- Character # must contain at least one string.
- An ID cannot contain [, : , >, #, and white-space, for example:

```
$query.get("#17_0_3_8e9027a#17_0_3_8e9027b") is equivalent to
$query.get("#17_0_3_8e9027a #17_0_3_8e9027b")
Retrieve an element ID #17_0_3_8e9027b that is a grandchild or a later descendant of
an element ID 17_0_3_8e9027a.
$query.get("#17_0_3_8e9027a > *")
Retrieve all elements that are the children of an element ID #17_0_3_8e9027a.
$query.get("#") or
$query.get("#[name=a]#17_0_3_8e9027a")
An invalid query pattern.
```

- A : must be followed by empty and ":empty" must be followed by [, #, >, or white-space, except in an attribute, for example:

```
$query.get("*:empty") or
$query.get("*:empty#17_0_3_8e9027a") or
$query.get("*:empty > class") or
$query.get("*:empty[name*=class]")
A valid query pattern but the result of some patterns will be an empty list.
$query.get("*[name=:emptyx]")
Retrieve all elements whose name attribute is :emptyx
$query.get("*:emptyx") or
$query.get("*:em")
An invalid query pattern.
```

- If an attribute value of a MagicDraw element is null, it is equivalent to an empty attribute value.

Query methods

On this page

- Retrieving an Array of Element by Query Pattern(see page 1911)
- Getting a Single Result from Query Functions(see page 1912)
  - Getting the First Element from a Collection(see page 1912)
  - Getting the Last Element from a Collection(see page 1912)
  - Getting the nth Element from a Collection(see page 1912)
  - Getting the Unique Element from a Collection(see page 1913)
- Retrieving an Array of Elements by Name(see page 1913)
- Retrieving Elements by ID(see page 1913)

This section provides query methods that you can use to retrieve an array of elements by query pattern.

Retrieving an Array of Element by Query Pattern

```
$query.get(query)
```

- **Parameters**: *query* : String (the query string as described in section **Recognizable Query Patterns**.
- **Returns**: *java.util.List<Element>* (a list or a collection of MagicDraw Elements or an empty List if there is no matching element).
- **The sample code** to print all Use Case elements under a packagable element named *foo* is as follows:

```
#foreach ($uc in $query.get("usecase[owner=foo]"))
 $uc.name in $uc.owner.name
#end
```

Getting a Single Result from Query Functions

Since a MagicDraw model element does not have a unique identified attribute so all methods will be returned in a Collection. Thus we will need convenient methods to get a single result from the Collection. Additional methods for getting a single result from query methods are as follows:

- first() – returns the first element from a collection.
- last() – returns the last element from a collection.
- get(int) – returns the nth element of a collection.
- unique() – returns the first element from a collection (equivalent to first).

Getting the First Element from a Collection

```
$query.get(query).first()
```

- **Returns**: Element (the first element from a query result collection or a null value if there is no matching element).
- **A Sample code** to print the first package is as follows:

```
$query.get("package").first()
```

Getting the Last Element from a Collection

```
$query.get(query).last()
```

- **Returns**: Element (the last element from a query result collection or a null value if there is no matching element).
- **A sample code** to print the last package is as follows:

```
$query.get("package").last()
```

Getting the nth Element from a Collection

```
$query.get(query).get(index)
```

- Parameters: index : int (a zero-based integer indicating which element to retrieve).
- Returns: Element (an element from a specified index of a query result collection or if the index is out of bounds, it will print a warning message to console/log and returns a null value)
- A sample code to print the third package is as follows:

```
$query.get("package").get(2)
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

```
$query.get(query).unique()
```

This method is equivalent to the one in section **Getting the First Element from a Collection**.

- Returns: Element (a unique element from a query result collection or a null value if there is no matching element).
- A sample code to print a single package is as follows:

```
$query.get("package").unique()
```

Retrieving an Array of Elements by Name

```
$query.getElementsByName(name)
```

Retrieve an element from a selected package scope whose element name exactly equals a given parameter. This method is equivalent to query pattern *[name=var].

- **Parameters**: *name* : String (the element name string).
- **Returns**: *java.util.List<Element>* (a List or a collection of MagicDraw Elements or an empty List if there is no matching element).
- **A sample code** to print the first element whose name exactly equals to *foo* is as follows:

```
$query.getElementsByName("foo").unique()
```

Retrieving Elements by ID

```
$query.getElementById(id)
```

Retrieve an element from a selected package scope whose element ID exactly equals a given parameter. This method is equivalent to query pattern #id.

- **Parameters**: id : String (an element's ID).
- **Returns**: Element (a MagicDraw Element or a null value if there is no matching element).
- **A sample code** to print an element that contains ID 17_0_3_8e9027a_1325220298609_477646 is as follows:

```
$query.getElementById("17_0_3_8e9027a_1325220298609_477646").unique()
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Examples

This section provides three sets of sample code to demonstrate some scenarios with the query tool. The codes may not be executable or return a correct result. They are just examples to show you how to use the Query tool and how it works.

(i) Example 1

Print any classes whose containing in package com.nomagic.reportwizard.

Without QueryTool:

```
#foreach ($c in $Class)
  #if ($c.qualifiedName.startsWith("com.nomagic.reportwizard"))
    $c.name
  #end
#end
```

With QueryTool:

```
#import("query", "com.nomagic.reportwizard.tools.QueryTool")
#foreach ($c in $query.get("class[qualifiedName^=com.nomagic.reportwizard]"))
  $c.name
#end
```

(ii) Example 2

Print any classes whose containing in model Design and name ends with Bean.

Without QueryTool:

```
#set ($match = false)
#macro (recursiveCheck $e)
  #if ($e.elementType == "model" && e.name == "Design")
    #set ($match = true)
  #else
    #recursiveCheck($e.owner)
  #end
#end

#foreach ($c in $Class)
  #set ($match = false)
  #recursiveCheck($c)
  #if ($check)
    #if ($c.name.endsWith("Bean"))
      $c.name
    #end
  #end
#end
```

With QueryTool:

```
#import("query", "com.nomagic.reportwizard.tools.QueryTool")

#foreach ($c in $query.get("model[name=Design] class[name$=Bean]"))
  $c.name
#end
```

(iii) Example 3

Matches any property element which owner element is class whose name attributes value is exactly equal to foo.

Without QueryTool:

```
#foreach ($p in $Property)
  #set ($owner = $p.owner)
  #if ($owner.elementType == "class" && $owner.name == "foo")
      $p.name
  #end
#end
```

With QueryTool:

```
#import("query", "com.nomagic.reportwizard.tools.QueryTool")

#foreach ($p in $query.get("class[name=foo] > property"))
  $p.name
#end
```

Template Creation Tutorial

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 2010-2019 by No Magic, Inc. All Rights Reserved.

- Introduction(see page 1916)
- Creating Your First Template(see page 1916)
- Velocity templating language(see page 1920)
- Report Wizard Directives(see page 1928)
- Linking report templates to MagicDraw(see page 1932)
- Exercise: Building a simple class diagram report(see page 1946)

Introduction

This user guide demonstrates how to create a report template using Report Wizard and use it in MagicDraw. To make the most of Report Wizard, you need to understand Apache Velocity and MagicDraw elements. Report Wizard uses Apache Velocity, this means that you will need to know how to create a Velocity template and that every rule that applies to Velocity will also apply to MagicDraw Report Wizard.

Velocity is a Java-based template engine that processes templates and references Java objects to produce output documents. A basic Velocity template can contain static text, layouts, conditional statements, and placeholders for each referenced Java object. When a template is being read by Velocity, conditional statements will be processed and placeholders will be replaced with the value from the referenced Java objects. You can use Velocity, for example, to generate web pages, emails, and read *XML* documents, but with Report Wizard it is now possible to use Velocity to read other types of templates, such as Rich Text Format documents. This user guide covers some basic aspects of Velocity, MagicDraw elements, and Report Wizard directives.

Related page

What is a Template?

A template, in Velocity, is a text file that tells Velocity what the output should look like. It contains a document page style, layout, header, footer, and static text just like any other template that comes with most of the Word Processor programs. However, a Velocity template also contains specific placeholders for Java objects and Velocity Template Language scripts which will tell the Velocity Engine what and how to print the information in the output.

Creating Your First Template

This section describes two main things that you need to do to create a template as follows:

i. Getting the value from the MagicDraw element inside a template.
ii. Importing the template you have created to MagicDraw.

Related pages

Getting a value from a MagicDraw element to a template

On this page

- Getting element values to a template(see page 1917)
- Glossary(see page 1918)
- More information about working with values in Report Wizard(see page 1918)

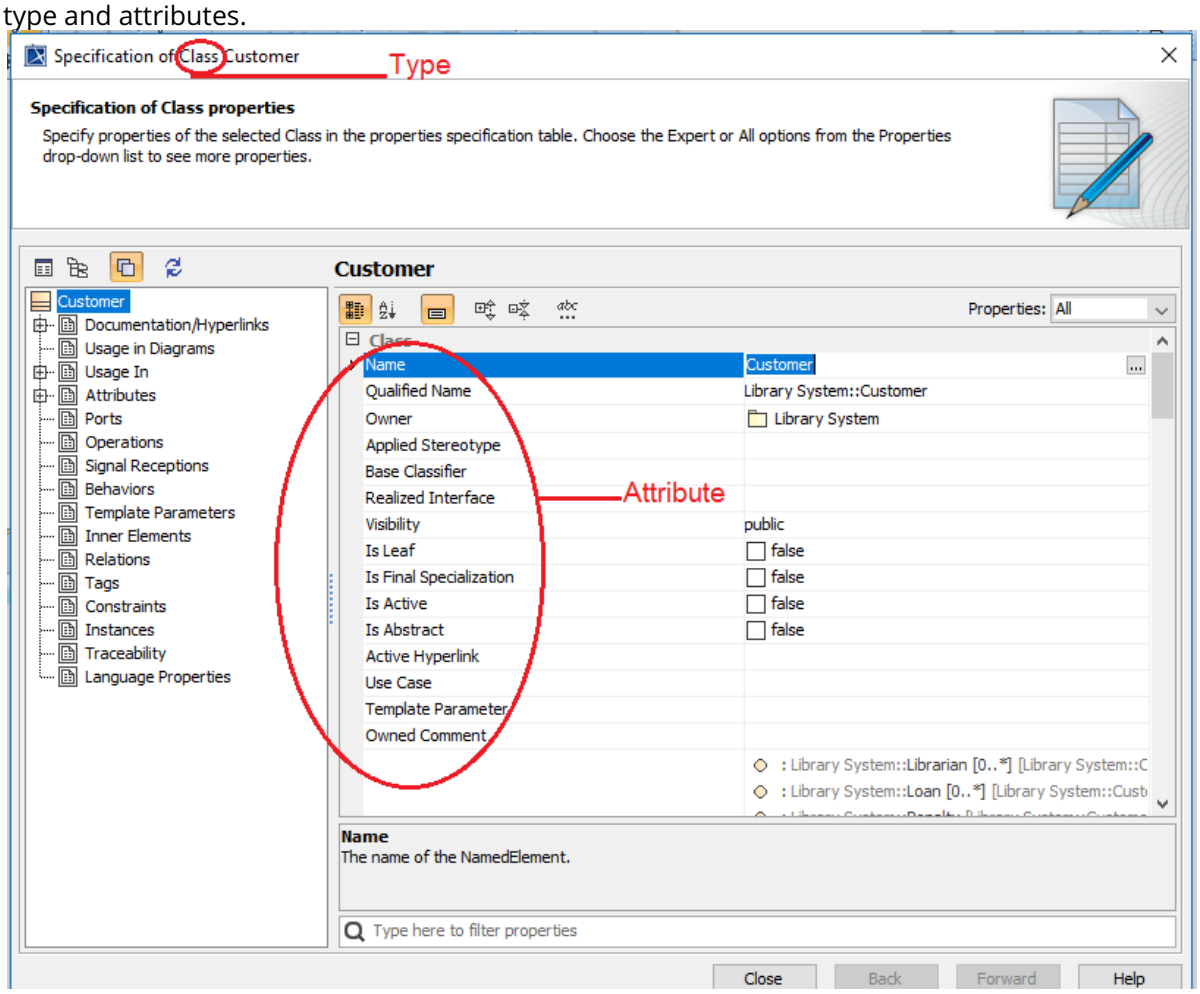
Getting element values to a template

You can access the value of an element, and include it in a report in Report Wizard.

To get a value from a modeling tool element to a template

---

1. First you need to know the element type and attributes. Open the Specification dialog to see the type and attributes.



The Specification dialog showing the element type on the title bar and the attributes in the properties specification table.

2. Open Microsoft Word and type, for example:

```
#foreach ($class in $Class)
 Name: $class.name
 Owner: $class.owner.name
 Visibility: $class.visibility
 Is Abstract: $class.isAbstract
#end
```

The example above shows that **$Class** is an array that contains all the Class elements in a **Selected Element Scope** (see Report Wizard user guide[352] for more information). **$class** is the individual Class element inside the array. To access the value of a property in **$class**, type: **$class** followed by **"."** and the attribute's name ("**.name**", "**.visibility**", and "**.isAbstract**" are the names of the **$class** attribute). The syntax for accessing the attribute value can be represented by: **$[Referenced object].[Attribute's name]**.

When you generate the output of the template for a project with a single Class element, the result will be shown as follows:

```
Name: Customer
 Owner: com
 Visibility: public
 Is Abstract: false
```

Glossary

- **Array**
  An array is a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created. After it is created, its length is fixed.
- **Attribute**
  Attribute defines values that can be attached to instances of the class or interface. For example, an attribute of a class represents the characteristics of the class.
- **Class**
  Class is the concrete realization of EncapsulatedClassifier and BehavioredClassifier. The purpose of a class is to specify a classification of objects, and to specify the Features that characterize the structure and behavior of those objects.
- **Element**
  Element is a constituent of a model. Every element has the inherent capability of owning other elements. When an element is removed from a model, all its ownedElements are also removed from the model. The abstract syntax for each kind of element specifies what other kind of elements it may own. Every element in a model must be owned by exactly one other element of that model, with the exception of the top-level Packages of the model.

More information about working with values in Report Wizard

To learn more about working with values in Report Wizard, visit the following pages:

Retrieving Slot Values(see page 1613)

352 https://docs.nomagic.com/display/MD190SP1/Report+Wizard

Getting property value(see page 1612)

Importing a template to MagicDraw using Report Wizard

On this page

- Importing a template to MagicDraw(see page 1919)
- Glossary(see page 1920)
- More information about working with templates and reports(see page 1920)

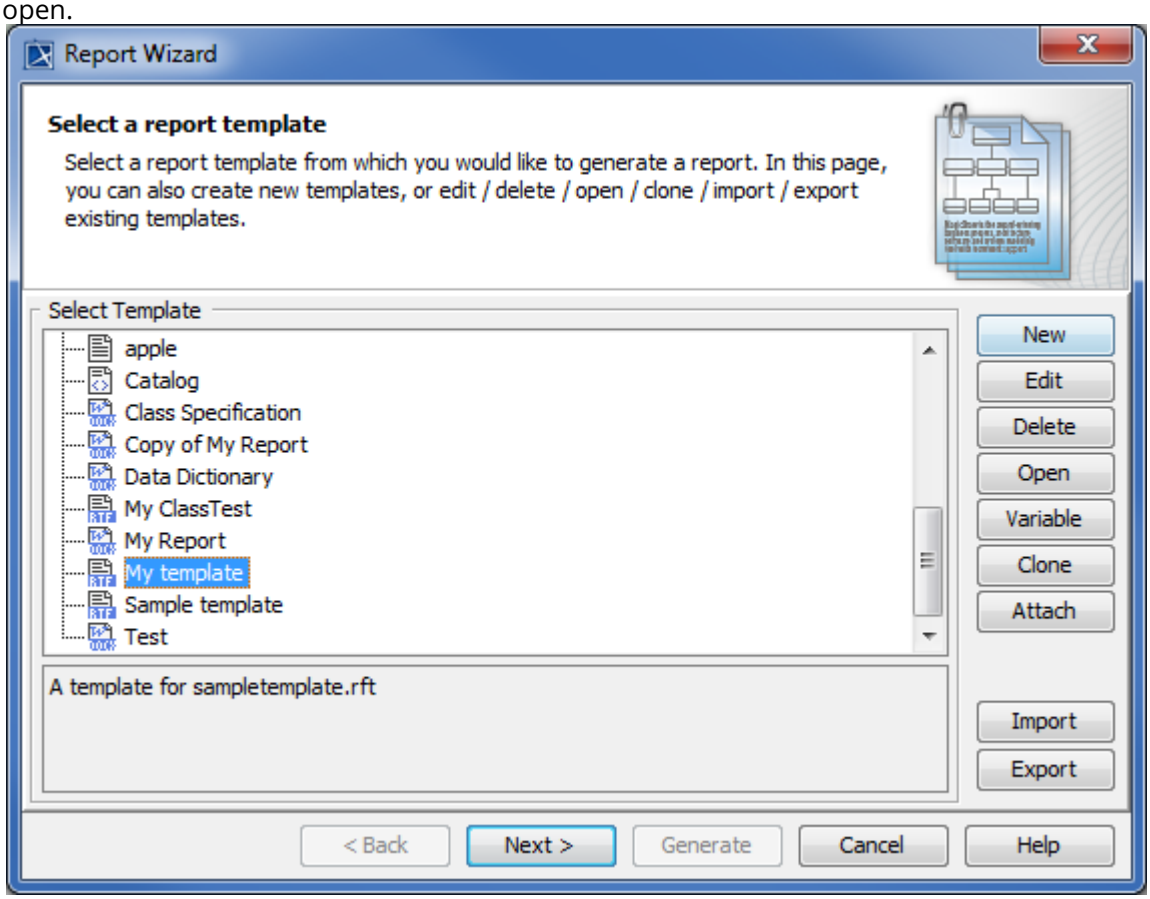
Importing a template to MagicDraw

Once you have created a template, you will need to import it into MagicDraw by using Report Wizard.

To import a template into a modeling tool using Report Wizard

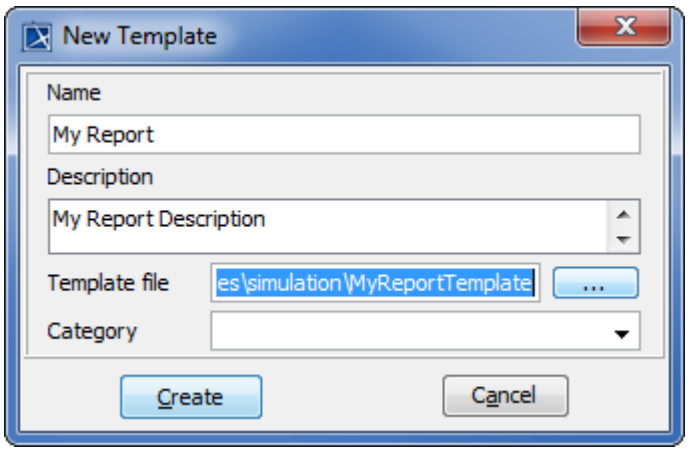
---

1. Click **Tools** > **Report Wizard** on the MagicDraw main menu.
2. Click the **New** button in the **Select a report template** wizard. The **New Template** dialog will open.



Opening the Report Wizard dialog and creating a new custom template by clicking the New button.

3. In the **New Template** dialog, type the name and description. Select a category, and then click ... .



Entering the new template name and description, and then selecting a template file in the New Template dialog.

4. Find the template file that you have created, and click Select.
5. Click Create to import the template.

To generate a report based on a template, select the template (in this example, **My template)** from the template list, and follow the steps as instructed by the wizard. See Report Wizard User Guide(see page 1547) for more information.

Glossary

- **Report template**
  Ready-made report configurations that you can use to quickly create a specific type of report in Report Wizard.

More information about working with templates and reports

For more information about working with templates and reports, visit the following pages:

Template Management Pane(see page 1549)

Default templates(see page 1735)

Report Data Management Pane(see page 1566)

Generating Reports from Report Wizard(see page 1734)

Velocity templating language

Velocity Template Language is a scripting language used only by Velocity Engine to determine what the output should look like. This section is divided into three parts:

- Velocity Variable
- Velocity Directive

- Comment and unparsing code

For more information on the Velocity Template Language, visit http://velocity.apache.org/engine/1.6.2/user-guide.html.[353]

Related pages

Velocity variable

A Velocity variable can be either a referenced Java object or a declared variable inside a template. A Velocity variable begins with **$** followed by the name of the variable. Depending on what is added to the Velocity Context (MagicDraw automatically adds its element to the Velocity Context) the variable can either be a local variable or a reference to a Java object.

To declare a local variable inside a template, type: **$** followed by a string beginning with a letter. A referenced Java object variable is provided by Report Wizard as shown in the following table (see the Linking Report Wizard template to MagicDraw(see page 1932) section for a complete list of variables).

The table shows a partial list of MagicDraw specific variables.

Variable Name	Return Value
$Class	Contains a list of class elements.
$UseCase	Contains a list of use case elements.
$Diagram	Contains a list of diagram elements.

If a Velocity variable is a reference to a Java object, you can call its methods and value by:

- Directly calling the object method with the Velocity variable, for example, **$object.method()**.This is useful if you want to use a Java object's method to process data and print it out.
- Calling the object properties, for example, **$object.name**.

Velocity allows you to call the referenced Java object's method directly like you would in Java. This is very useful as you can create methods in Java that can help you create a more flexible Velocity template. For instance, if you need to perform a complex operation, such as sorting a list according to certain variables, you can create a sorting method in Java, and then call that method from a Java object.

Report Wizard provides a collection of complex statements in standard tool methods. You can find more information on the standard tool in the Helper Modules(see page 1621) section, Report Wizard User Guide. You can also create your own custom tool, deploy, and use it inside a template. For more information on the custom tool, see Appendix A: Report Extensions(see page 1951), Report Wizard User Guide.

353 http://velocity.apache.org/engine/releases/velocity-1.6.2/user-guide.html+

Note

- Report Wizard does not support calling variables in double quotes.

  The following calls are not valid:

  ```
  #set($newVTL = "$vtl")
  #set($newVTL = "text $vtl")
  #aMacro("text $vtl")
  #aMacro("$vtl")
  #if("$text.name" == "class")
  ```

- If you want to concatenate string, use the following style:

  ```
  #set($newVTL = "text " + $vtl)
  ```

Velocity Directives

A Velocity directive is a keyword used by Velocity to perform certain tasks such as looping and controlling the output of the document based on values from MagicDraw.

With directives you can make dynamic templates and you can do more than just print out MagicDraw elements and attributes. A Velocity directive begins with **#** and followed by the directive name. Some of the frequently used Velocity directives are as follows:

1. **#if**, **#elseif**, and **#else**: these directives are used to decide whether or not text should be included in the output based on the conditional statement.
2. **#set**: this directive is used to assign a value to a variable.
3. **#foreach**: this directive is used to iterate through a list of variables.
4. **#macro**: this directive is used as a means to create a reusable script. It is especially useful if you need to call a certain line of scripts repeatedly.

Related pages

#if, #elseif, and #else statement

The **#if** directive allows you to include text when generating a document, on condition that the if-statement is true, for example:

```
#if ($condition)
 Hello World
 #end
```

The variable: ***$condition*** is evaluated to determine whether it is true, which will happen under certain circumstances:

1. ***$condition*** is a Boolean (true/false) that has a true value
2. ***$condition*** is not 'null'.
3. ***$condition*** is a comparison that is evaluated and returns 'true'.
4. ***$condition*** is a compound logical statement that is evaluated and returns 'true'.

If **$condition** returns 'true', the content between the ***#if*** and ***#end*** statements becomes the output. In this case, if ***$condition*** is true, the output will be: "Hello World". Conversely, if ***$condition*** returns 'false', there will be no output.

The ***#elseif*** or ***#else*** element can be used with the ***#if*** element. Note that Velocity Engine will stop at the first expression that is found to be true. The following example shows you how to add **#elseif** and **#else** to the **#if** statement:

```
#if( $condition )
 Content 1
 #elseif( $condition2 )
 Content 2
 #elseif( $condition3 )
 Content 3
 #else
 Content 4
 #end
```

From the above example, let us assume that ***$condition1*** is false, ***$condition2*** is true, and ***$condition3*** is true. The output for this conditional block will be ***Content 2*** because ***$condition2*** comes before ***$condition3*** even though both of them are true.

Comparing Values and Logical Operators

So far, *$condition* in the #if directive is assumed to be a Boolean value, however just like Java, Velocity supports the syntax to compare (greater than (>), less than (<), and is equal to (==)) two variables and the Logical Operators logical AND (&&), logical OR (||), and logical NOT (!) which will return a Boolean value.

In Velocity, the equivalent operator ("==") can be used to compare string, value, and objects. Note that the semantics of the equivalent operator ("==") are slightly different than those of Java where the equivalent operator ("==") can only be used to test object equality. In Velocity the equivalent operator ("==") can be used to directly compare numbers, strings, or objects. When comparing two objects with different classes, the string that represents the objects is compared. The following example compares two variables to see whether or not they are equal:

```
#set ($var1 = "cat")
#set($var2 = "dog")
#if ($var1 == $var2)
Var1 equals Var2.
#else
Var1 does not equal Var2
#end
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

The comparison operators ("<" and ">") is the same as the ones used in Java. The following example shows how to use the comparison operators to compare two values in which the **#if** statement will be evaluated 'true' and "**Var1 is less than Var2**" will be printed out in the generated report:

```
#set ( $var1 = 6 )
#set ( $var2 = 7 )
#if ( $var1 < $var2 )
Var 1 is less than Var2
#end
```

The logical AND, in Velocity, is represented by && and it must have at least two arguments. To write a conditional statement with the logical AND type, for example:

```
#if( $var1 < $var3 && $var3 > $var2 )
 Content 1
 #end
```

To better understand the above example, let us assume that **$var1 < $var3** is **argument 1** and **$var3 > $var2** is **argument 2** (**argument 1** and **argument 2** will be evaluated separately and will return true/false). The ***#if()*** directive will be evaluated 'true' if both **argument 1** and **argument 2** are true. If **argument 1** is false, the expression will be evaluated false and **argument 2** will not be evaluated.

If **argument 1** is true, Velocity Engine will then check the value of **argument 2**. If the value is true, the entire expression is true and *Content 1* becomes the output. If the value is false then there will be no output as the entire expression is false.

Logical OR operators work the same way as Logical AND, except for one of the references that needs to be evaluated 'true' for the entire expression to be considered true, for example:

#set ($var1 = 1) #set ($var2 = 2) #set ($var3 = 3) #if($var1 < $var3	**$var3 < $var2)** **Content 1** **#end**

To better understand the above example, let us assume that $var1 < $var3 is argument 1 and $var3 < $var2 is argument 2. According to the example, argument 2 is false because $var2 is less than $var3 not the other way around. Since argument 1 is true, Velocity Engine does not need to look at argument 2, whether argument 2 is true or false, the expression will be true, and Content 1 will be the output. Basically, a Logical OR operator requires only one argument to be true (either argument 1 or argument 2) to make an expression true. In the case that both argument 1 and argument 2 are false, there will be no output because the expression is false.

With Logical NOT operators, there is only one argument:

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

```
#set ( $bool = false )
 #if( !$bool )
 Content 1
 #end
```

As shown in the above example, ***!$bool*** is evaluated true and Content 1 is the output. But if ***$bool*** is true, then **!$bool** is false and there will be no output.

To include some text immediately following the **#else** directive, put curly brackets around the directive followed by the text to include, as shown in the example below:

```
#if( $foo == $bar)it is true!#{else}it is not!#end</li>
```

#set statement

In Velocity, you can declare a variable and set its value by using the **#set** directive, for example:

```
##String
#set ($var = "abc")
##Boolean
#set($var2 = true)
##Value
#set($var3 = 10)
```

The above example shows that **$var** has a string value, **$var2** has a Boolean value, and **$var3** has a numerical value. The **#set** directive can also set an array to a declared Velocity variable. To assign an array to a variable type, for example:

```
#set($var = ["a", "b", "c"])
```

#foreach statement

The **#foreach** directive will iterate through a list of variables and assign it to a local variable, for example:

```
#foreach ($localClass in $Class)
 $localClass.name
 #end
```

The above example shows that **$Class** is a list of class elements in MagicDraw. The **#foreach** directive will iterate through the **$Class** list and assign the value in the list to **$localClass** in each iteration. The code between **#foreach** and **#end** will be processed and the result will print the class name in each iteration. For example, if **$Class** contains class elements named "a", "b", and "c", the output from the above example will be:

```
a
b
c
```

#macro statement

Under certain circumstances, you might find that you often repeat several lines of a *VTL* code in multiple areas inside your template. To solve this problem, Velocity provides the **#macro** directive that allows you to repeat a segment of *VTL* codes. The first thing you have to do is to declare or create the macro itself, for example:

```
#macro (HelloWorld)
Hello World! It is such a beautiful world!
#end
```

Then within the template, you can call this macro and it will print "Hello World! It is such a beautiful world!"

To call this macro, for example, type:

```
#HelloWorld()
```

When the macro in the above example is passed on to the Velocity Template Engine, "Hello World! It is such a beautiful world!" will be printed out as a result. You may wonder why this is important, but imagine having to print a note with 10 lines of text in different parts of the document. The Velocity macro helps you save the space and time. It also allows you to pass the value from a variable on to a method parameter as you would using a Java method. To do this you have to create a macro, for example:

```
#macro( myMacro $color $somelist )
#foreach( $something in $somelist )
$something is $color.
#end
#end
```

In the above example, you have created a macro called "**myMacro**" that accepts two variables, the first variable is supposed to be a string and the second variable called "**somelist**" is supposed to be a list (note that when creating a macro that accepts variables always remember what types of variables it accepts, otherwise an error will occur during report generation). To call **myMacro** type, for example:

```
#set ($list = ["A Rose", "Blood", "Strawberry"])
#set ($color = "red")
#myMacro($color $list)
```

The above example shows that "myMacro" has been called and there are two variables in the brackets, **$color** and **$list**, separated by a space. The result of the example:

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

```
A Rose is red.
Blood is red.
Strawberry is red.
```

Comment and unparsing code

Comment

Comment is used for explaining or making notes regarding the code. You can include text comments in the report. The report engine will not parse comments.

To make a single line comment

- Type **##** before the comment. The report engine will skip all the text in the line, as follows:

```
##Comment here
```

To make multi-line comments

- Type **#*** followed by the multi-line comments and end them with ***#**. The report engine will skip all the lines between these symbols, as follows:

```
#*
    Comment line1
    Comment line2
    Comment line3
*#
```

Unparsing code

You can declare the code in Velocity template without parsing it into the report. The report engine will not parse the code.

To unparse code

- Type **#[[** followed by the code to unparse and end it with **]]#**, as follows:

```
<html>
    <body>
        #[[
            <?php
            session_start();
            YOUR CODE HERE
            ?>
        ]]#
    </body>
</html>
```

Report Wizard Directives

Report Wizard introduces three directives: **#forrow**, **#forcol**, and **#forpage**. These directives allow you to use Velocity to generate outputs other than web documents. With these directives, you can eliminate problems that usually occur when using Velocity to generate an rtf document, for example, **#foreach** generates broken rows in a table.

#forrow
This directive is used in the Rich Text Format, OpenDocument text, and OpenDocument spreadsheet, Microsoft Word Document (DOCX), and Microsoft Excel spreadsheet (XLSX) documents. Whenever #forrow iterates through a list, it creates a row in a table.

#forcol
This directive is used only in Microsoft Word document (DOCX), Microsoft PowerPoint presentation (PPTX), OpenDocument spreadsheet, and Microsoft Excel spreadsheet (XLSX). Whenever #forcol iterates through a list, it creates a column in a table.

#forpage
This directive is used in Rich Text Format document, OpenDocument Text, OpenDocument spreadsheet, OpenDocument presentation, Microsoft Word document (DOCX), Microsoft Excel spreadsheet (XLSX), and Microsoft PowerPoint presentation (PPTX). Whenever #forpage iterates through a list, it creates a new page in Rich Text Format, OpenDocument text or Microsoft Word (DOCX). It also creates a new sheet in OpenDocument spreadsheet or Microsoft Excel spreadsheet (XLSX), or a new slide in OpenDocument presentation or Microsoft PowerPoint presentation (PPTX).

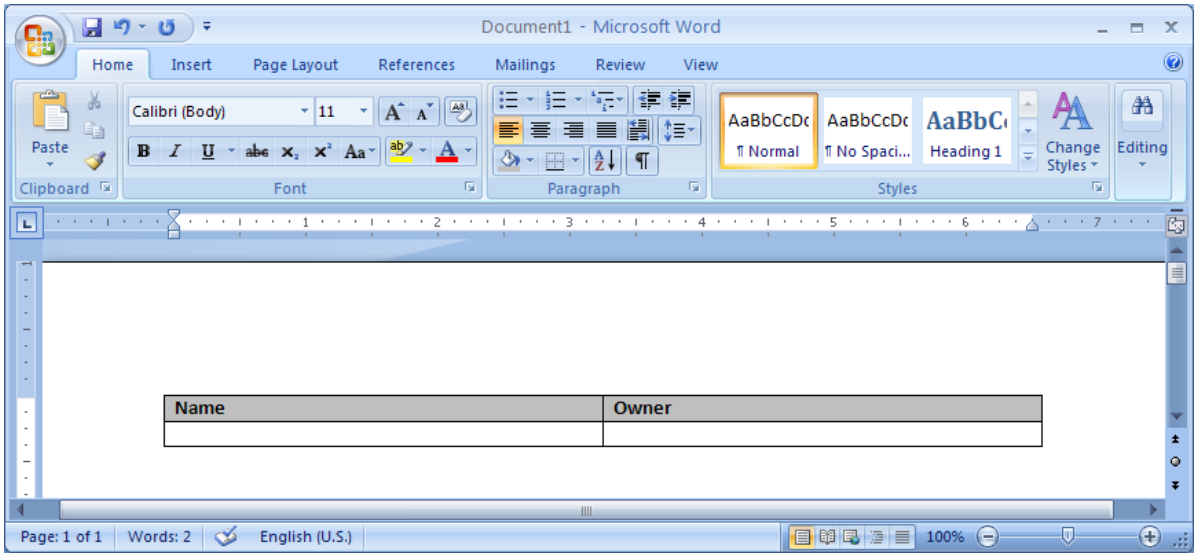
Related pages

#forrow directive

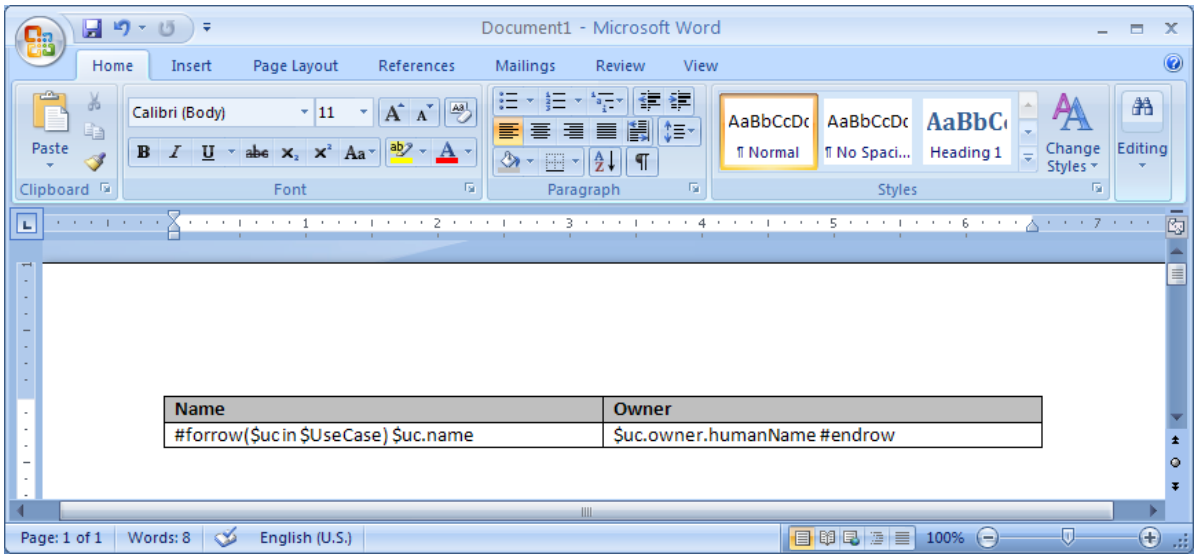
The Velocity Template Language does not support loops inside the table structure. However, the Report Wizard engine introduces a new custom syntax to allow loops in the table structure in order to clone the table rows. In the following example, we will create a table that will print out the name of a Use Case element and its owner in a table.

Related page

1. Open Microsoft Word or OpenOffice.org Writer.
2. Create a two-column and two-row table (see figure below).



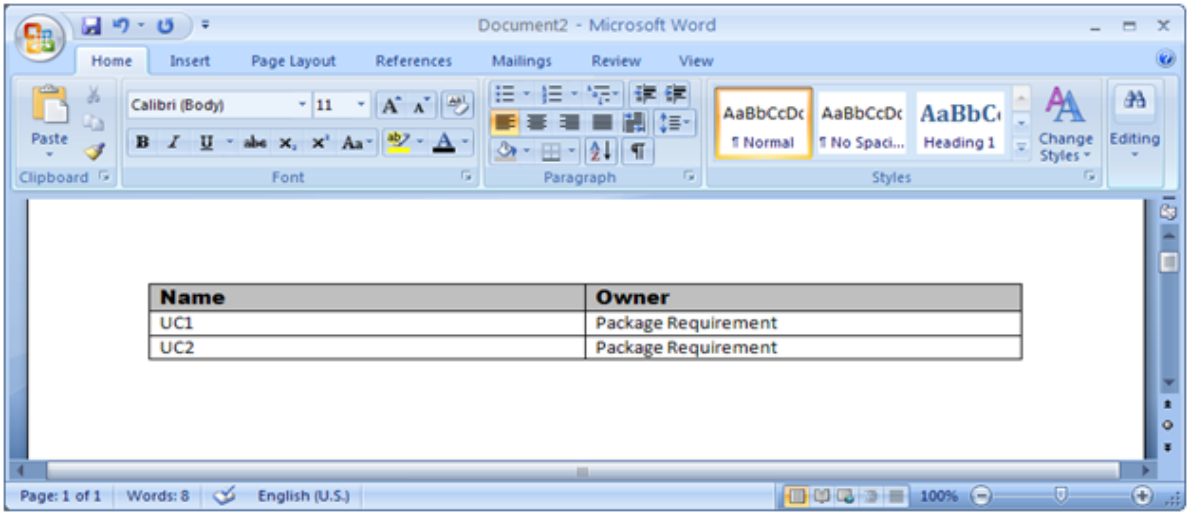
3. In the second row, first column type: "**#forrow ($uc in $UseCase) $uc.name**", and in the second row, second column type: "**$uc.owner.humanName #endrow**" (see figure below).



4. Save the template, and import it to Report Wizard.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

5. Generate a Use Case report. The report will appear as shown in the figure below.



#forcol directive

This directive is designed only for the OpenDocument Spreadsheet, Microsoft Word document (DOCX), Microsoft PowerPoint presentation (PPTX), and Microsoft Excel Spreadsheet (*XLSX*) templates. This directive allows loops over the column.

	A	B	C
1	#forcol($uc in $UseCase)	$uc.name	#endcol
2			

Based on the example shown in figure above, the engine will generate a report with different columns for each Use Case name. The output from this example will appear as shown in the figure below:

	A	B	C	D
1	Use Case A	Use Case B	Use Case C	Use Case D
2				

You can combine both **#forrow** and **#forcol,** and produce a more complex output report (see figure below).

	A	B	C
1	#forrow($p in $Package)	#forcol($e in $p.ownedElement)$e.name#endcol	#endrow

In the figure below, the output generated from the Magic Library sample project is shown.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

	A	B	C	D	E	F
1	Domain Analysis	Diagram Loan Collaboration	Diagram Loa	Diagram Obj	Dependency	Dependency
2	conceptual perspective					
3	Implementation	Node Remote Web Client	Node Remot	Artifact Interr	Artifact Magi	Artifact Magi
4	High Level Domain Analysis	Diagram conceptualPerspective1	Diagram con	Class Penalt	Association i	Association i
5	collaboration	Collaboration Loan Library Item				
6	Analysis and Design	Diagram Analysis and Design	Package Hig	Package Dor	Package Dor	Package Tech
7	object diagram	Instance Specification	Instance Spe	Instance Spe	Instance Spe	Instance Spe
8	Technical Design	Diagram domain.user	Diagram dom	Diagram dom	Diagram Mag	Package Magi
9	MagicLibrary	Instance Specification	Class Library	Class Books	Class VideoP	Class AudioP
10	Domain Design	Diagram Package Dependencies	Diagram Imp	Dependency	Dependency	Package Impl
11	composite structure	Collaboration Loan	Class Partici	Collaboration	ReadingItem	Loan
12						

#forpage directive

The **#forpage** directive is used to:

1. Create a new slide in an OpenOffice Presentation (*ODP*) or a Microsoft PowerPoint Presentation (*PPTX*) document.
2. Create a new page in an *RTF*, an OpenDocument Text (*ODT*), or a Microsoft Word (*DOCX*) document.
3. Create a new sheet in OpenDocument Spreadsheet (*ODS*) and Microsoft Excel Spreadsheet (*XLSX*).

The implementation of this directive is the same as that of **#forrow**:

```
#forpage($dia in $Diagram)
$uc.name
#endpage
```

The example above will create many pages each containing a Use Case element's name. The number of pages created depends on the number of Use Case elements in the **$UseCase** list.

In *ODP* and *PPTX*, **#forpage** creates a new slide.

Related page

Creating a new slide for a presentation

To create a new slide for a presentation

---

1. Open OpenOffice.org[354] Impress or Microsoft PowerPoint.
2. Create two text boxes.
3. Type in:
   (i) one of the text boxes:

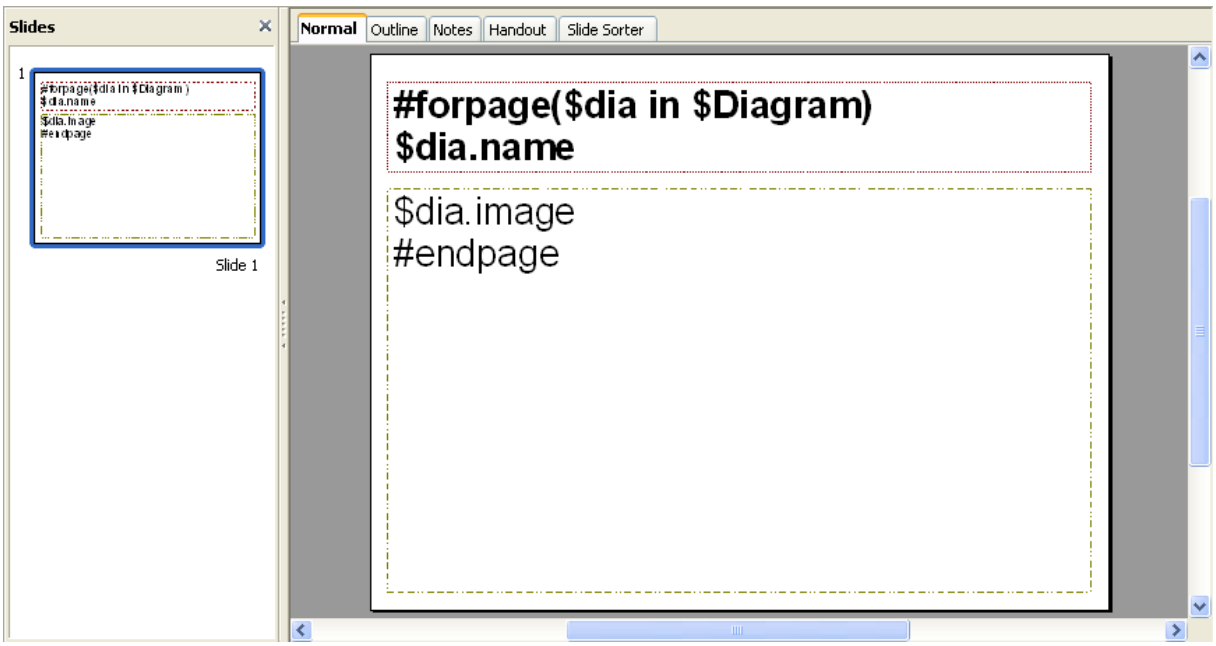
   ```
   #forpage($dia in $Diagram)
   $dia.name
   ```

354 http://OpenOffice.org

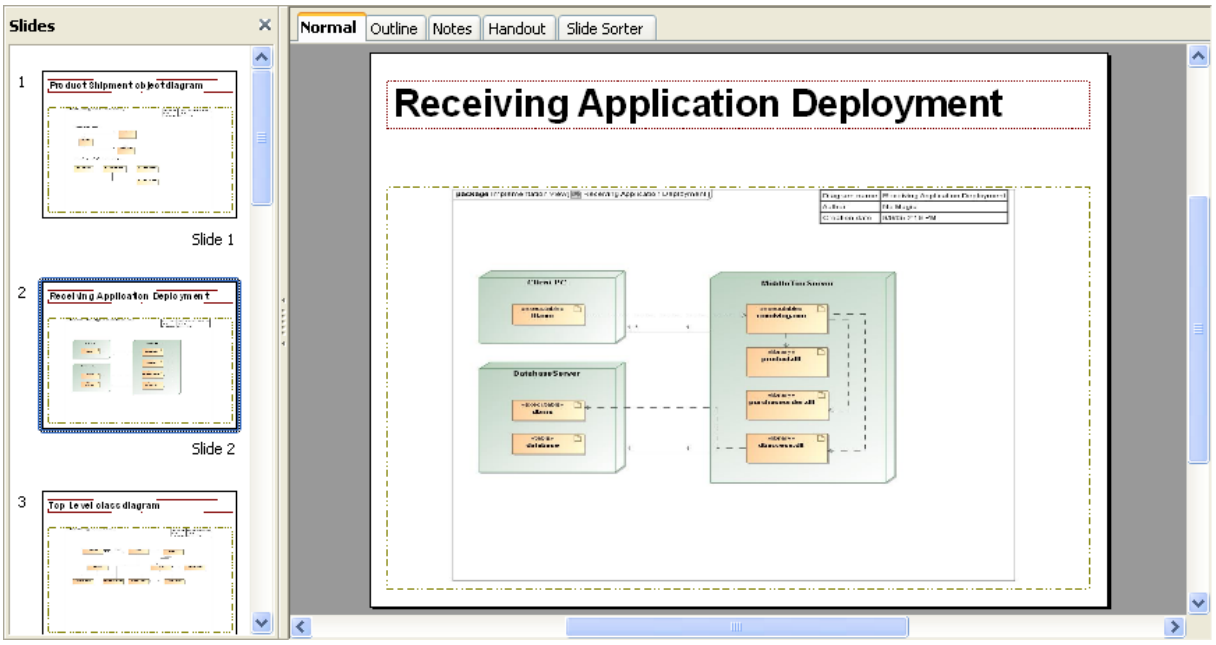
(ii) and type in the other text box:

```
$dia.image
#endpage
```

The result will appear as shown in the figure below.



The result of the generated report will appear shown in the figure below.



Linking report templates to MagicDraw

By now you should know how to write a basic template that makes a simple reference to a MagicDraw element. But how will you know if the element is referenced or not? This section shows you how elements are referenced and how to call them.

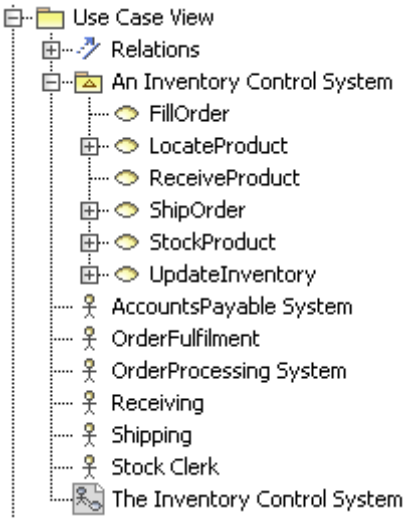
Related pages

How MagicDraw adds its elements to velocity context

In Velocity, whenever you want to reference to a Java object you need to manually add the object to the Velocity template and name it. In Report Wizard, however, this process is done automatically. What you need to understand is how it is done as it will help you to call the MagicDraw elements.

All MagicDraw elements in MagicDraw such as Class, Use Case, and Activity are the sub-classes of elements. During report generation, the report engine will get all the elements from a project within a selected scope that you have defined in Report Wizard, and then classify and add them to a list of the same type. The list will then be added to the Velocity Context with its type as its name. For instance, in a project where there are two types of Element, Class, and Use Case, all the elements that are identified as Classes will be kept on the same list and the list will be added to the Velocity Context and labeled as "Class" during report generation. This process also applies to the Use Case elements. To reference to a specific Class element, first you will have to make a reference to **$Class** (list of Class elements), and then iterate through the list to find that particular Class element by using the **#foreach**, **#forrow**, **#forcol**, or **#forpage** directive, followed by the **#if** directive to search for the name of the class.

This section will use part of the "*Inventory Control System.mdzip*" file in *<install.root>/samples/casestudies* as an example to show you how to reference to MagicDraw elements.



Use Case View.

You need to know the elements inside the "Use Case View" package before using it, see figure above. To do this, follow the instructions in the previous section, Creating your first template(see page 1916).

The "Use Case View" package contains three element types: (i) Actor, (ii) UseCase, and (iii) Diagram that you can call in a template by typing **"$Actor**," **"$UseCase**," and **"$Diagram**" respectively.

In this example you will create a report that will print the names of the Actors, UseCases, and diagram image.

Related pages

Creating a report including the Actor, Use Cases, and diagram image names

On this page

- Creating a report including the names of actors, use cases, and diagram images(see page 1934)
- Glossary(see page 1937)
- Sample model(see page 1937)
- More information about creating report templates(see page 1938)

Creating a report including the names of actors, use cases, and diagram images
This section will use part of the Inventory Control System.mdzip file as an example to illustrate how to reference to MagicDraw elements in a report. This example contains several packages. For instance, if you want to use the 'Use Case View' package, you need to know the elements inside the package before using it. To do this, follow the instructions in the previous section, Creating your first template(see page 1916).

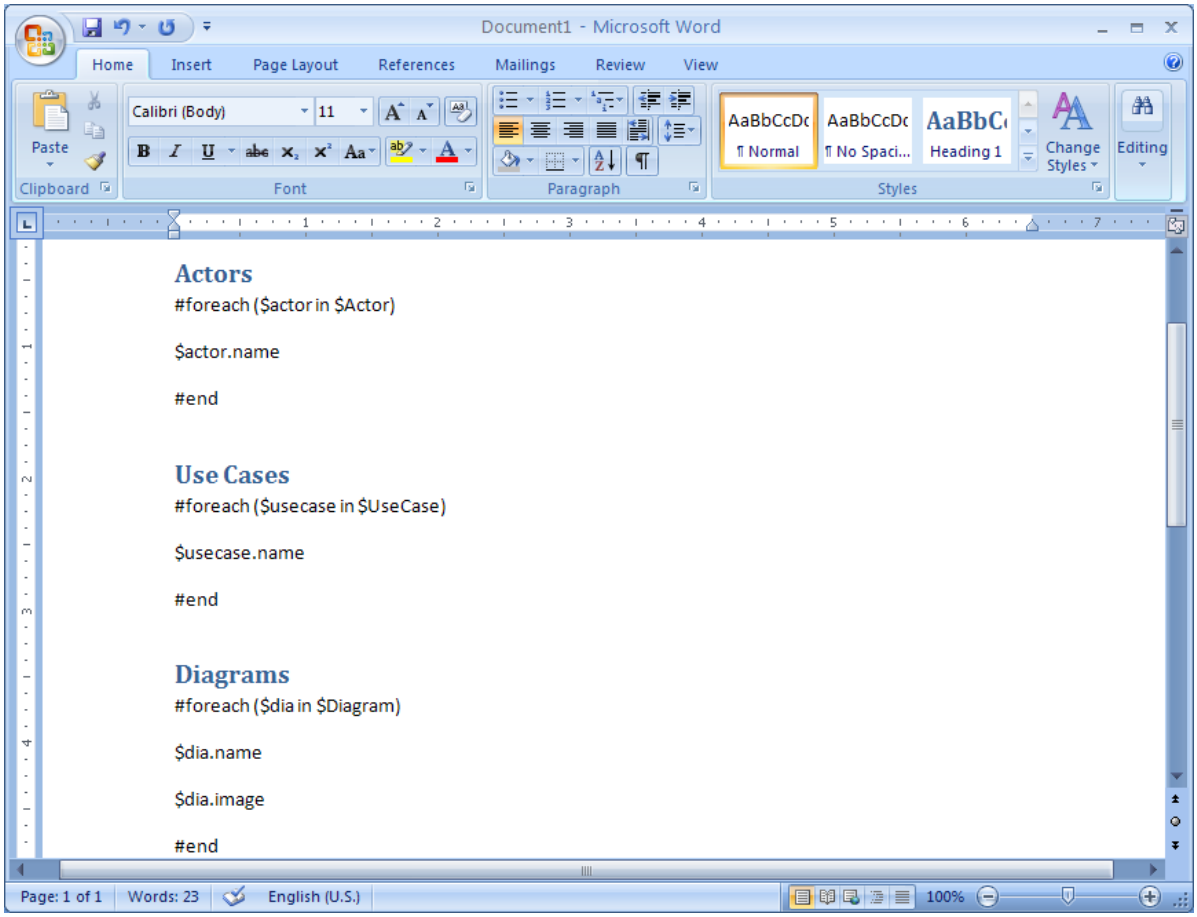
The 'Use Case View' package contains three element types: Actor, UseCase, and Diagram that you can name in a template by typing **"$Actor**," **"$UseCase**," and **"$Diagram**" respectively. The following example shows you how to create a report that will print the names of the Actors, UseCases, and diagram images.

To create a report that includes the names of Actor, Use Cases, and diagram images

---

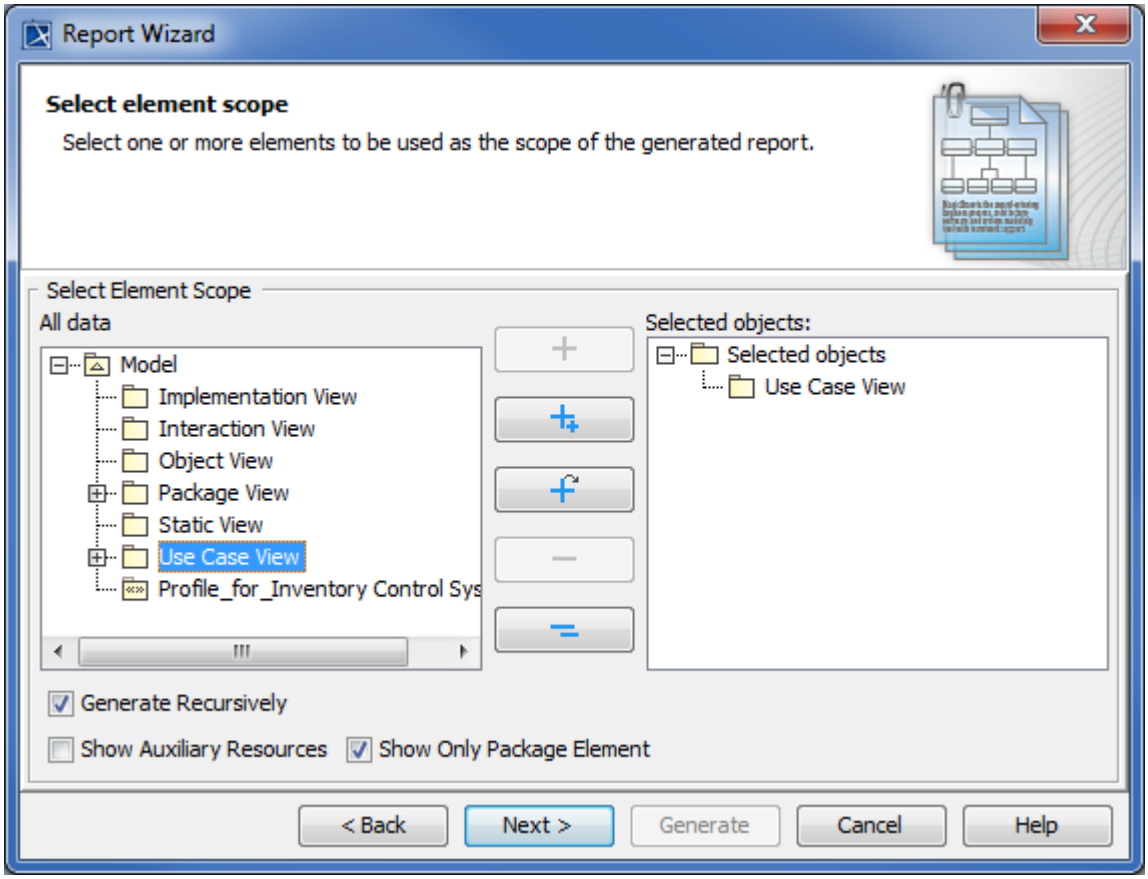
1. Open Microsoft Word or OpenOffice.org[355] Writer.
2. Use the **#foreach** directive to iterate through **$Actor**, **$UseCase**, and **$Diagram**.

355 http://OpenOffice.org



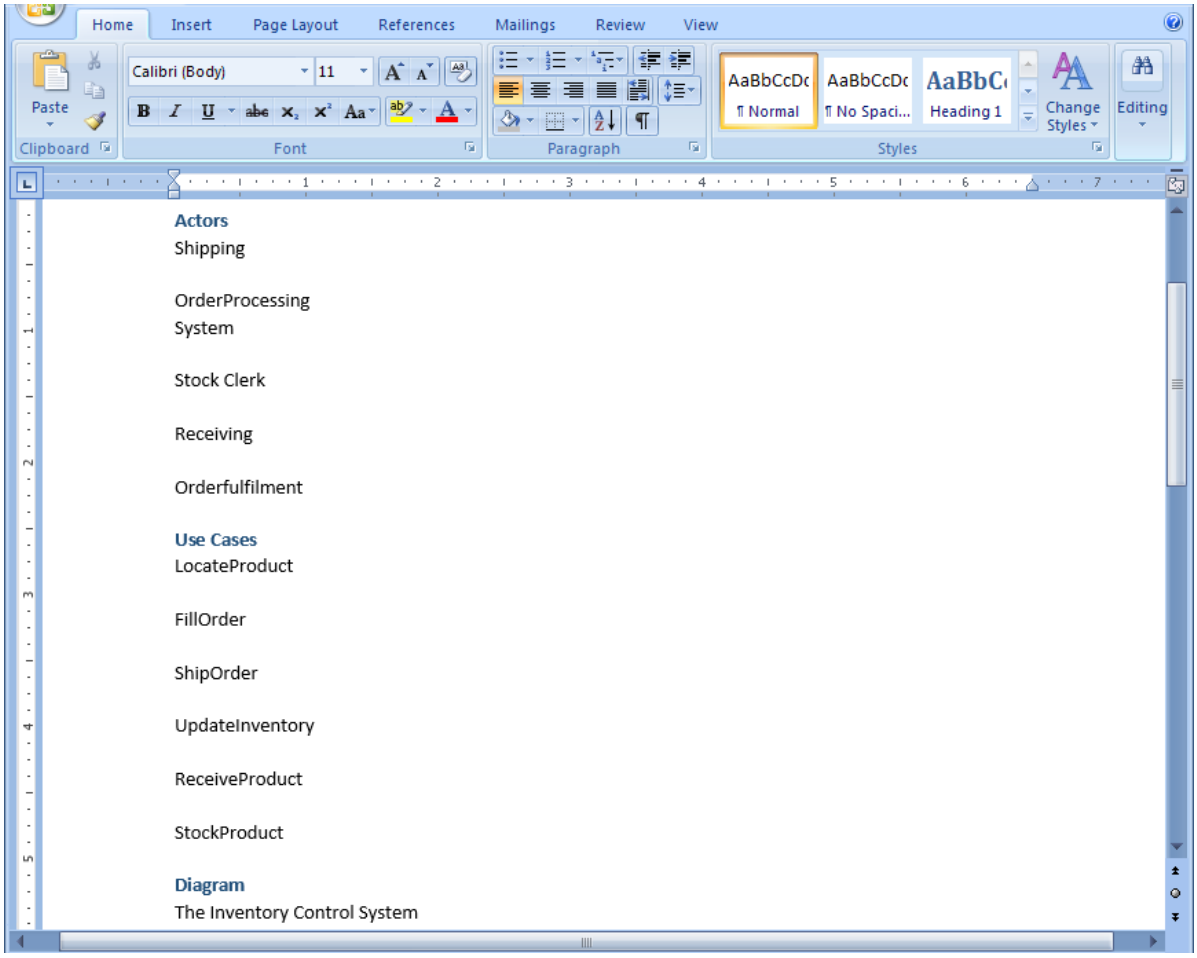
Using #foreach directive to iterate through $Actor, $UseCase, and $Diagram in a Rich Text Format template.

3. Use Report Wizard to add the template to MagicDraw.
4. With the "*Inventory Control System.mdzip*" project open, open the **Report Wizard** dialog and select your template.
5. Select the element scope so it covers only the "Use Case View" package.



Selecting the Use Case View package from the model as the element scope to be included in the report.

6. Complete all steps in the **Report Wizard** dialog, and generate the report. The output will be shown as in the following figure.



Shows a partial report output. A full report will print all the names of the Actors, Use Cases, and the Use Case diagrams.

Glossary

- **Actor**
  An actor models a type of role played by an entity that interacts with the subjects of its associated use cases, e.g., by exchanging signal and data. Actors may represent roles played by human users, external hardware, or other systems.
- **Element**
  The abstract root UML metaclass. It has no superclass in the hierarchy of UML elements. It is the superclass for all metaclasses in the UMLinfrastructure library. There is no general notation for an element. The specific subclasses of an element define their own notation. Each UML element has a composition relationship to itself to support the capability for elements to own other elements.
- **Use case**
  A specification of behavior. A use case is a kind of BehavioredClassifier that represents a declaration of a set of offered Behaviors. Each use case specifies some behavior that a subject can perform in collaboration with one or more actors. A UseCase may be owned either by a Package or by a Classifier. Use cases are a means to capture the requirements of systems,(i.e. what systems are supposed to do).

Sample model

Inventory Control System.mdzip

More information about creating report templates
To learn more about working with report templates, visit the following pages:

Creating Your First Template(see page 1916)

Linking report templates to MagicDraw(see page 1932)

Exercise: Building a simple class diagram report(see page 1946)

Supported file types

Besides generating Rich Text Format templates and Rich Text Format reports, Report Wizard also generates other types of document:

1. **.html* – Hyper Text Markup Language
2. **.rtf* – Rich Text Format
3. **.odt* – OpenDocument Text
4. **.ods* – OpenDocument Spreadsheet
5. **.odp* – OpenDocument Presentation
6. **.docx* – Microsoft Word Document
7. **.xlsx* – Microsoft Excel Spreadsheet
8. **.pptx* – Microsoft PowerPoint Presentation
9. **.xml* – Extensible Markup Language
10. **.txt* – Text
11. **.docbook* or **.dbk* DocBook

To create a template for a specific type of document, simply open the editor designed for that particular type of document, and then save the template making sure that the filename extension is the same as that of the type of document.

Related pages

Creating a template for an HTML, a text, an XML, or a DocBook document

To create a template for an *HTML*, a *text*, an *XML*, or a *DocBook* document

1. Open a text editor.
2. Add the code to the text file.
3. Save the template file (with a correct filename extension).
4. Add the template to Report Wizard.

- When you generate a report from a particular template, Report Wizard will automatically create an *HTML*, *text*, an *XML*, or a *DocBook* document depending on a given template filename extension.
- The **#forrow**, **#forcol**, and **#forpage** directives do not support *HTML*, *text, XML*, and *DocBook* format files.

Creating a template for an OpenDocument spreadsheet or Microsoft Excel spreadsheet

To create a template for an OpenDocument spreadsheet or Microsoft Excel spreadsheet

---

1. Open OpenOffice.org Calc or Microsoft Excel.
2. Add the code to the template file.
3. Save the template file.
4. Add the template to Report Wizard.

All multiline directives such as **#if, #else, #elseif**, **#foreach**, and **#macro** must be used under the following conditions:

1. The beginning and end of a statement must be declared within a single cell. A **#macro** statement must be declared in a single cell.

	A	B	C
1			
2		**Use Case**	
3		#foreach ($uc in $UseCase) $name #end	
4			

2. You can use **#foreach** in a single cell record only. To create data for multiple rows, use the **#forrow** directive instead, see figure below.

	A	B	C
1	#forrow($uc in $UseCase)	$uc.name	#endrow

3. You can use **#forcol** to create data for multiple columns, see figure below. You can use the statement in conjunction with **#forrow**.

	A	B
1	#forcol($uc in $UseCase)$uc.name#endcol	
2		

Creating a template for an OpenDocument Presentation (ODP) or Microsoft PowerPoint presentation (PPTX)

On this page

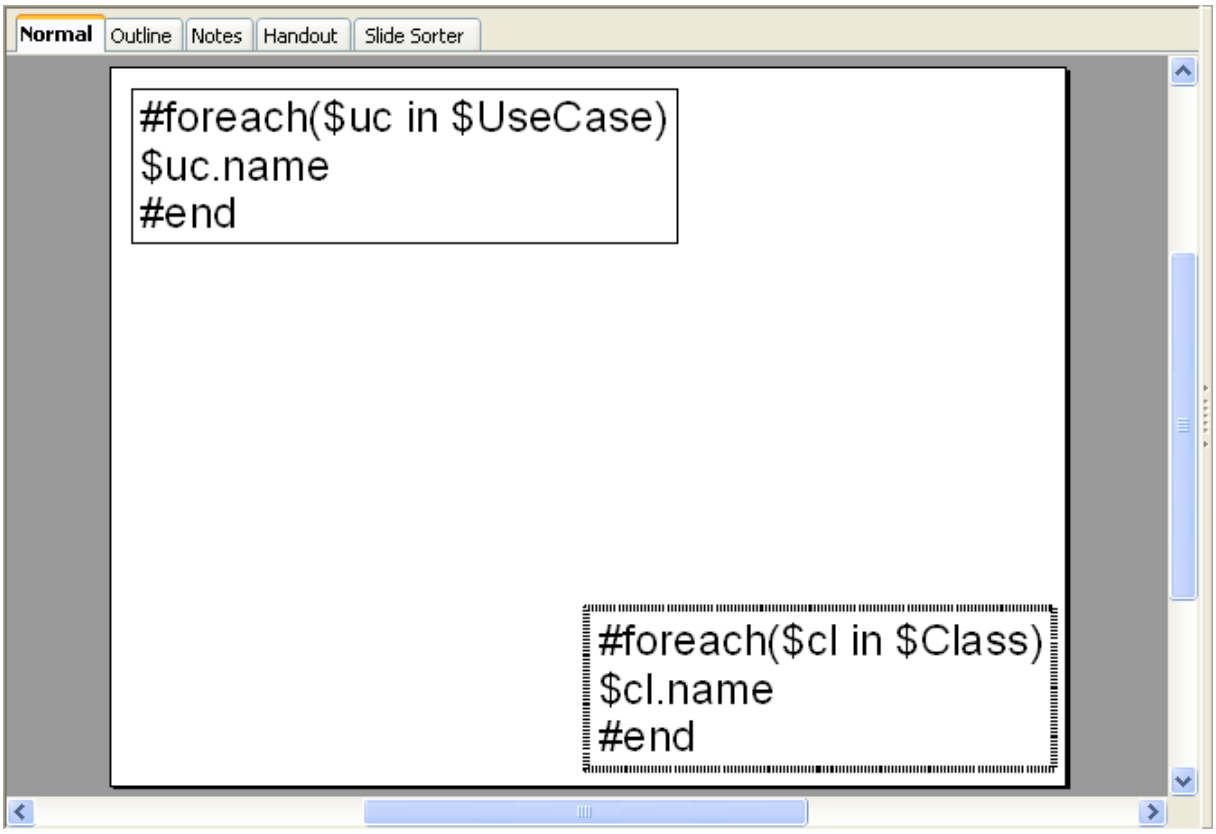
- Glossary(see page 1944)
- More information about working with Velocity Template Language (VTL)(see page 1944)

To create a template for an OpenDocument Presentation (ODP) or Microsoft PowerPoint presentation (*PPTX*)

---

1. Open OpenOffice.org[356] Impress or Microsoft PowerPoint.
2. Add the code to the template file.
3. Save the template file.
4. Add the template to Report Wizard.

A presentation document is a special document template that does not contain a content order. It uses a text box to insert text as the content of the document. A text box is an image structure (an image structure keeps the position of each image in x, y coordinates). You can change the position of a text box, or place it in the same position as others.



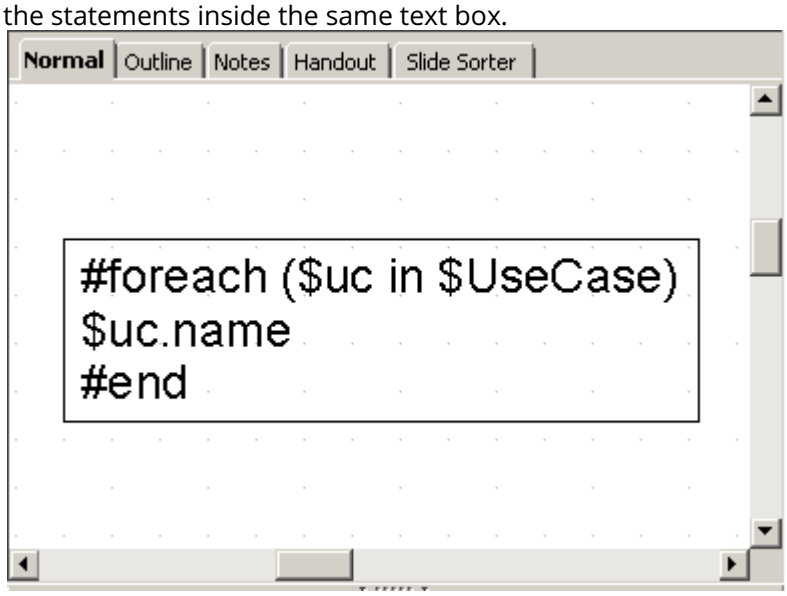
Writing VTL in a text box in OpenOffice.org[357] Impress.
The following are the rules that you need to follow when using the **#if, #elseif**, **#else**, **#foreach**, **#forpage**, and **#macro** directives in **.odp* and *.pptx* documents:

---

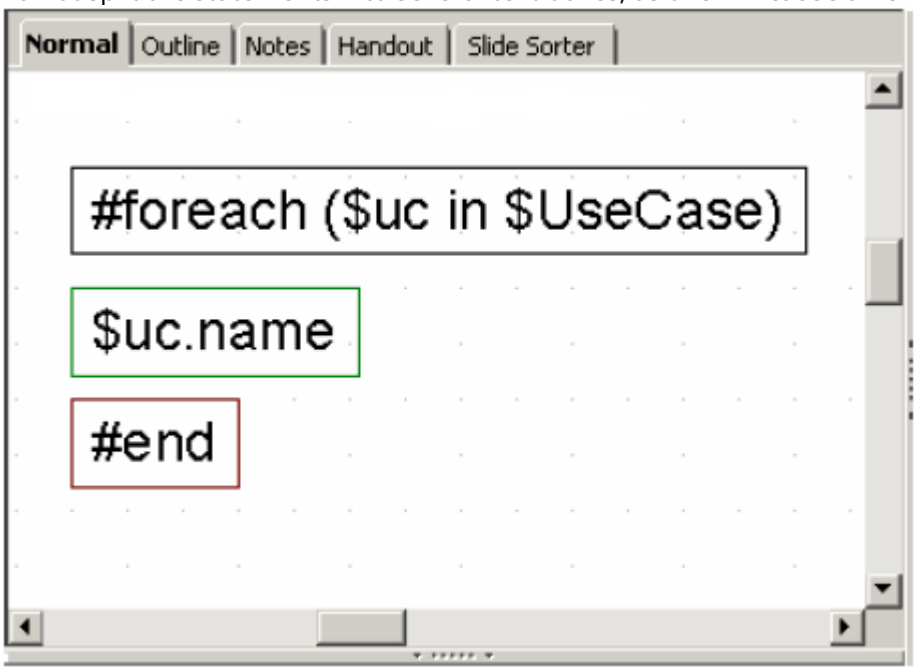
356 http://OpenOffice.org
357 http://OpenOffice.org

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Statement
  VTL statements that maintain a sequence order, such as **#if** and **#foreach**, must begin and end the statements inside the same text box.



Valid use of multi-line statement in ODP.
Do not split the statements into several text boxes, as this will cause an error.



Invalid use of a multi-line statement in ODP.

- #macro statement
  A #macro statement must be declared in the same text box.

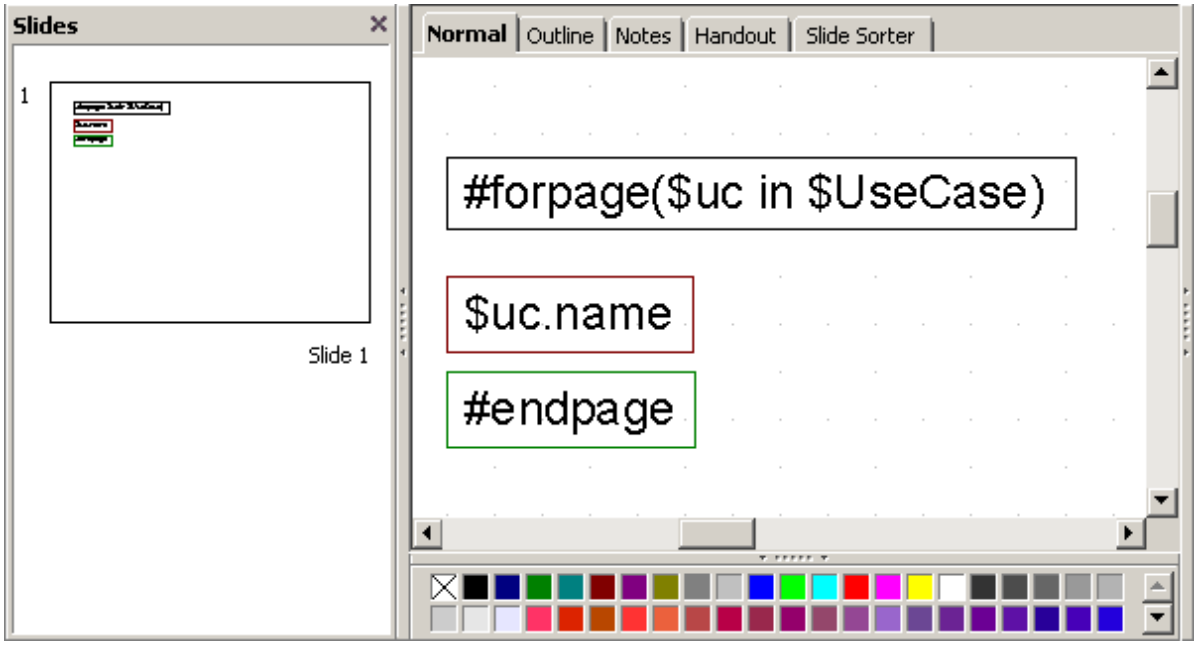
```
#foreach($c in $Class)
    #printName($c)
#end
```

```
#macro (printName $class)
    $class.name
#end
```

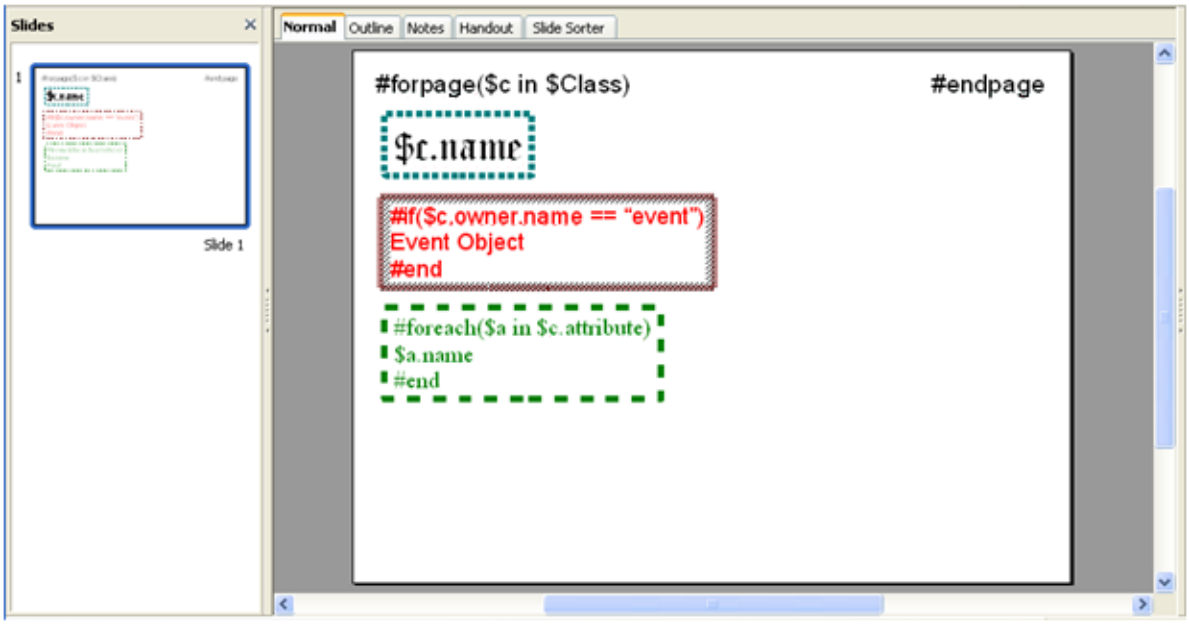
Valid use of macro statement in ODP.

- #forpage statement
  The **#forpage** statement is different from the others, as it does not contain any order. It can begin and end the statement in different text boxes. Its value can also be used in different text boxes.

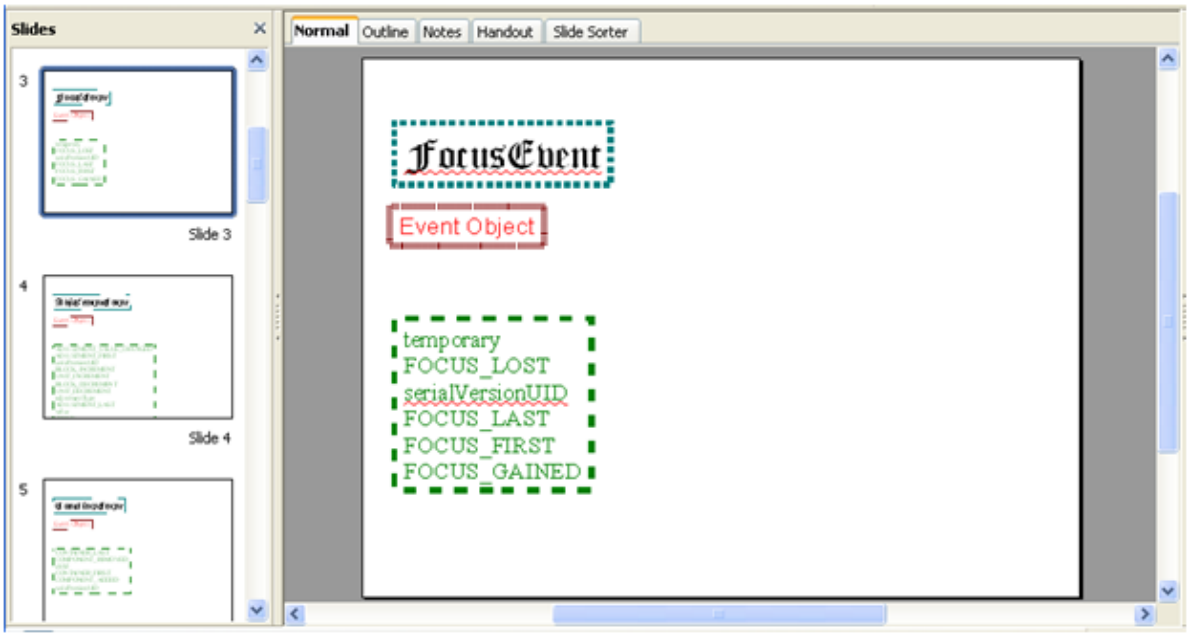
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Valid use of the #forpage statement in ODP.
To use a structured statement such as **#if**, **#macro**, or **#foreach** with the **#forpage** statement, ensure that the statement is in the same text box.



Valid use of #if and #foreach with #forpage statement.
Using an example from the previous figure, the generated result will be as follows:



The end result of using #if and #foreach with #forpage statement.

Glossary

- **VTL**
  The Velocity Template Language (VTL) is meant to provide an easy, simple, and clean way to incorporate dynamic content in a web page. A VTL statement begins with the `#` character and contains a directive: `set`.

More information about working with Velocity Template Language (VTL)
To learn more about working with Velocity Template Language (VTL), visit the following pages:

Velocity template language(see page 1920)

Velocity variable(see page 1921)

Velocity Directives(see page 1922)

Creating a template for a Rich Text Format or OpenDocument Text document, or Microsoft Word Document (DOCX)

On this page

- Creating a template for a Rich Text Format or OpenDocument Text document, or Microsoft Word Document (DOCX)(see page 1944)
- Glossary(see page 1945)
- More information about working with report templates in Report Wizard(see page 1945)

Creating a template for a Rich Text Format or OpenDocument Text document, or Microsoft Word Document (DOCX)
You can create a document in .rtf, .odt, or .docx file format and add it to MagicDraw as a report template.

To create a template for a Rich Text Format or OpenDocument Text document, or Microsoft Word Document (*DOCX*)

---

1. Open OpenOffice.org Writer or Microsoft Word for an **.odt*, **.rtf*, or **.docx* file. (To save a file as an **.odt* file in Microsoft Word, an add-on is required. You can get the add-on at http://odf-converter.sourceforge.net/.)
2. Add the Velocity Template Language codes to the template file.
3. Save the template file.
4. Add the template to MagicDraw.

When generating a document, Report Wizard will apply the text format used in writing the Velocity code.

For instance, when typing:

```
#set($var = "Hello World")
$var
$var
        $var
```

the generated output will be:

```
Hello World
Hello World
        Hello World
```

Glossary

- **Velocity**
  A Java-based template engine. It permits anyone to use a simple, yet powerful, template language to reference objects defined in Java code. Velocity is a project of the Apache Software Foundation[358], charged with the creation and maintenance of open-source software related to the Apache Velocity Engine[359]. All software created at the Velocity project is available under the Apache Software License[360] and free of charge for the public.

More information about working with report templates in Report Wizard
To learn more about working with report templates, visit the following pages:

Creating a template for an HTML, a text, an XML, or a DocBook document(see page 1938)

Creating a template for a Rich Text Format or OpenDocument Text document, or Microsoft Word Document (DOCX)(see page 1944)

---

358 http://www.apache.org/
359 http://velocity.apache.org/engine/index.html
360 http://www.apache.org/licenses/LICENSE-2.0.txt

Creating a template for an OpenDocument Presentation (ODP) or Microsoft PowerPoint presentation (PPTX)(see page 1939)

Creating a template for an OpenDocument spreadsheet or Microsoft Excel spreadsheet(see page 1939)

Exercise: Building a simple class diagram report

On this page

- Building a Class Diagram Report(see page 1946)
- Glossary(see page 1951)
- More information about working with report templates using Report Wizard(see page 1951)

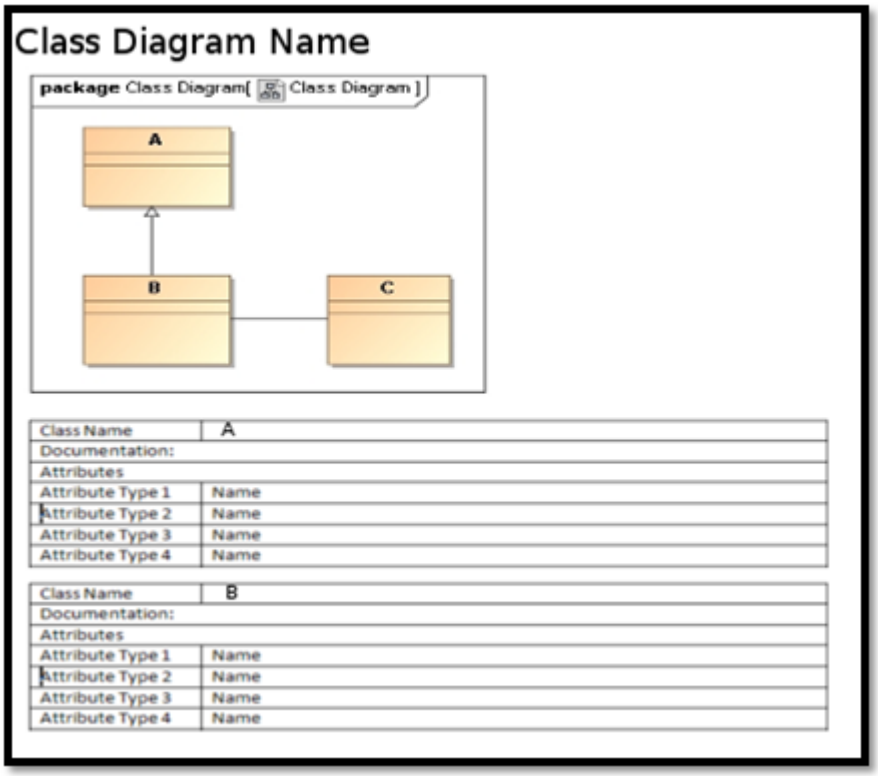
Building a Class Diagram Report

In this exercise, you will learn how to create a simple Class Diagram report. A Class Diagram report must contain the following information:

- A class diagram's name.
- A diagram.
- A table that describes the class (we will call it a class description table) based on the diagram shown below.
- The table must have the following structure:

Class Name	
Documentation:	
Attributes	
Attribute Type 1	Name
Attribute Type 2	Name
Attribute Type 3	Name
Attribute Type 4	Name

- The layout of the report should look like the example below.

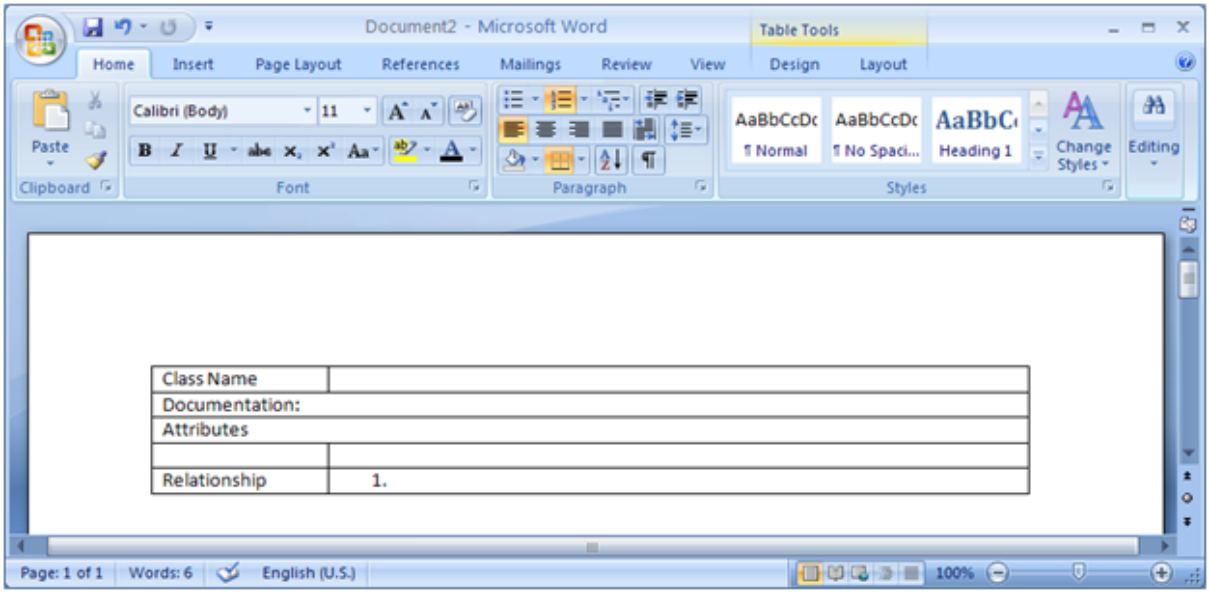


The layout of a basic class diagram on the diagram pane.

Instructions

The first and most important step in creating a report is setting up the document and its layout.

1. Open Microsoft Word.
2. Create a table that will be used as the basic layout for a class description table.



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

The layout for a class description table in a Word document containing the name of the class, the attributes, and the relationships.
As shown in the preceding figure, a diagram will appear at the top of the report followed by a class description table. Accordingly, each diagram on the Class diagrams list has a class description table describing each component in the diagram. Follow the next steps to create the document template:

Iterates through a list of diagrams.

1. Print the diagram and its name.
2. Get a list of class elements that are displayed in the diagram.
3. Iterate through the list of class elements.
4. Create a class description table for each class element in the list.

   The steps listed above contain two **#foreach** loops: the outer loop that iterates through a list of diagrams, and the inner loop that iterates through a list of Class elements. The list of diagrams in a MagicDraw project is represented by **$Diagram**.

5. At the top of the page of the word document type:

```
#foreach ($diagram in $Diagram)
 **Codes in between **
 #end
```

   $Diagram is a list of all the diagrams in a MagicDraw project. This means that a project will be in the list if it contains more than one type of diagram. To filter out other types of diagrams, add the **#if** directive with a conditional statement to test whether or not **$diagram** is a class diagram:

```
# foreach ($diagram in $Diagram)
#if($diagram.diagramType == "Class Diagram")
$diagram.name
$diagram.image
**Codes in between **
#end
#end
```

6. Next, you need to get all the elements contained in the diagram by using **$report**. **$report** is a standard tool containing helpful methods such as **getDiagramElements()** that returns a list of elements inside a diagram.

```
#foreach ($diagram in $Diagram)
#if($diagram.diagramType == "Class Diagram")
$diagram.name
$diagram.image
#set($eList = $report.getDiagramElements($diagram))
** Codes in between **
#end
#end
```

7. In the example above, **$eList** has been assigned with a list of Elements inside **$diagram**. Now, you need to create an inner loop that will cover the table that you created earlier, and then filter out the elements inside the diagram whose type is not Class element.

```
#foreach ($diagram in $Diagram)
#if($diagram.diagramType == "Class Diagram")
$diagram.name
$diagram.image
#set($eList = $report.getDiagramElements($diagram))
#foreach($e in $eList)
#if($e.elementType == "class")
| Class Name | |
|---|---|
| Documentation: | |
| Attributes | |
| | |
#end
#end
#end
#end
```

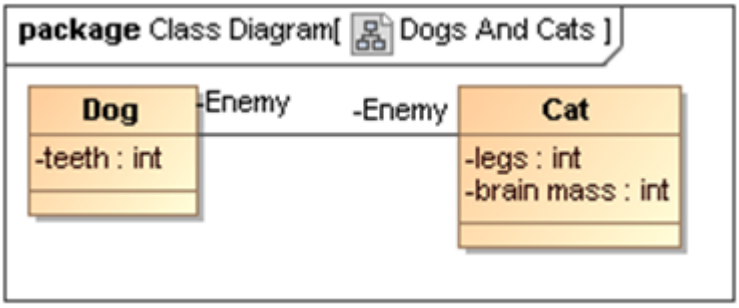
8. Enter the information of the class in the table by calling its attributes, etc.:

```
#foreach ($diagram in $Diagram)
#if($diagram.diagramType == "Class Diagram")
$diagram.name
$diagram.image
#set($eList = $report.getDiagramElements($diagram))
#foreach($e in $eList)
#if($e.elementType == "class")
| Class Name | $e.name |
|---|---|
| Documentation: $e.documentation | |
| Attributes | |
| #forrow($att in $e.attribute)$att.type.name | $att.name#endrow |
#end
#end
#end
#end
```

The Class Name and Documentation values are obtained by referencing the name and documentation attributes of **$e**.

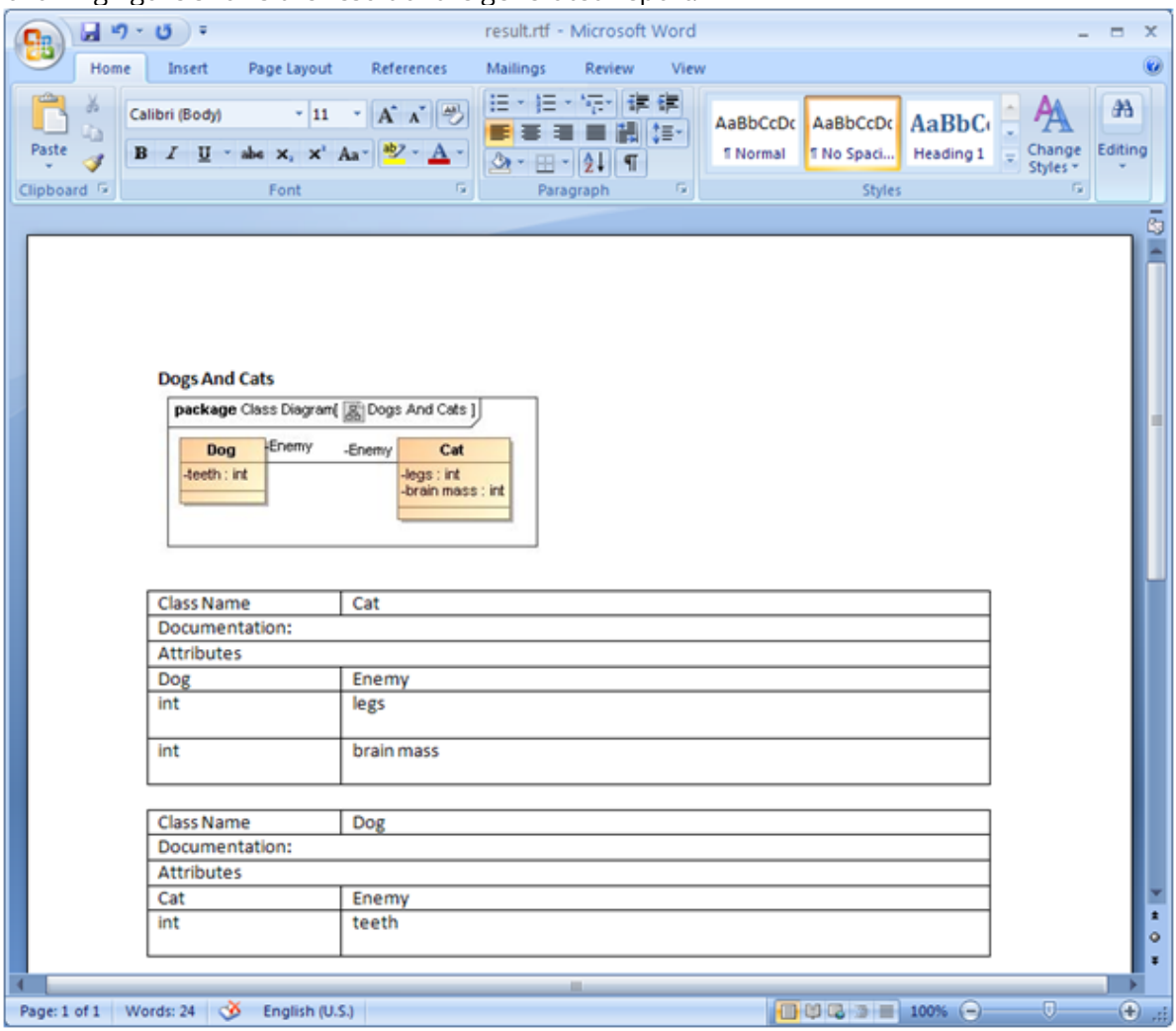
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

9. Use the **#forrow** directive as the attribute to print rows of the class attribute type (**$**att.type.name[361]) and the class attribute name (**$**att.name[362]).
10. Save the file as Rich Text Format. Open MagicDraw and draw a simple class diagram.



A simple class diagram.

11. Add the template to Report Wizard, and generate a class diagram report using this template. The following figure shows the result of the generated report.



The generated report of the class diagram in the Word document.

361 http://att.type.name
362 http://att.name

Glossary

- **Class diagram**
  A class diagram is a diagram in which the primary symbols in the contents area are class symbols.

More information about working with report templates using Report Wizard

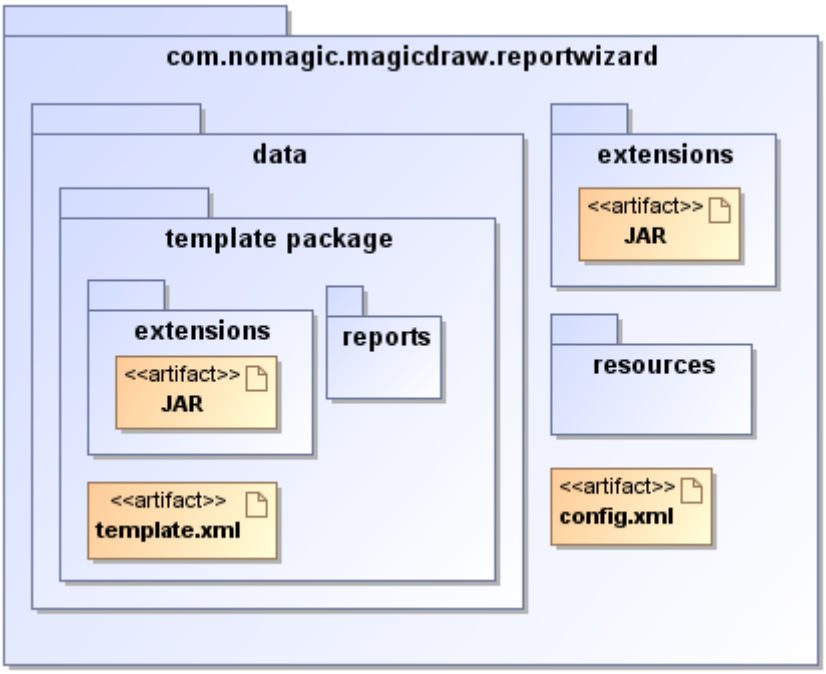
To learn more about working with report templates, visit the following pages:

Creating a report including the Actor, Use Cases, and diagram image names(see page 1934)

Appendix A: report extensions

Report Wizard has been enhanced with an extension that provides additional functionalities (for example, a Custom Tool that encapsulates Java functions in context). The report extensions are Java class and related resources collected into JAR archives, which are merely ZIP files that contain a specially formatted “manifest” file describing the contents of the archive.

To install the extension, copy the *.jar* file into the extensions directory under the Report Wizard plugin directory or template directory. The archives stored within subdirectories will not be loaded in Report Wizard, see figure below.



Report Wizard Plug-in Directory Structure.

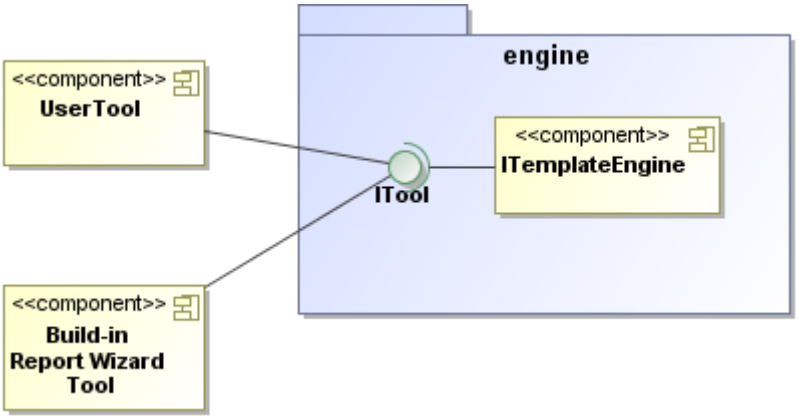
The extension deployed under the global extensions directory will affect all templates, which means you can use the deployed Custom Tool in all templates. Extensions deployed under a template package will affect only the specific template.

Related pages

Custom tool

A Custom tool is a template library written in Java to encapsulate functions in context. The Custom tool, see figure below, is extended from the report API. The tool can be used to develop custom context properties that are presentation centric or that can take advantage of the existing 'tool' extensions or by document authors familiar with Java.

Report Wizard uses the word 'tool' meaning an object encapsulating dynamic functions on the JavaBean. A tool is a "carrier" of data between the Java and template layers. In other words, the tool simplifies template development and maintenance.



Custom tool API Structure.

Context Name

A Context name is a name for variables or references in the template. Report Wizard uses the Velocity Template Language (*VTL*) as a standard syntax for the template. *VTL* uses "$" as a leading character followed by an identifier for the variable.

For example:

```
$UseCase
$foo.name
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Context Object

A Context object is a Java Object representing a variable. A Context object is modeled on the JavaBean specifications defined by Sun Microsystems.

For example, if a String represents the variable $foo.name in context, the value of the String will be printed out in the output report as follows.

```
Foo Name
```

Tool interface

Custom Tool is written in the Java language. The tool implements a specialized interface called ITool. The Report API provides both an interface and a class to support interface realization and class generalization. As mentioned earlier, a Tool is modeled on the JavaBean specifications. Functions implemented in this class must be defined in a series of setter or getter methods. The following sample shows the source code for the ITool interface that you must implement:

```
public interface ITool extends IBean
{
   // attributes
   Void VOID = new Void();
   // inner classes
   public class Void{}
   public class RetainedString implements CharSequence{}
   // methods
   void setContext(IContext context);
   String getContext();
   void setProperties(Properties properties);
   Properties getProperties();
}
```

All tools must implement the ITool interface (or one of its subclasses) as it defines the methods the template engine calls to execute. The table below provides a description of the methods in the ITool interface.

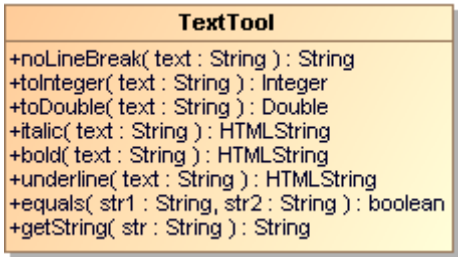
The table below lists Description of Methods in ITool interface.

Method	Description
setContext(Context context)	Once the class has been initialized, this method is called by the template engine runtime to set the template context. Overriding the current context will affect the code after this tool.
getContext()	Return the template context that is assigned to the current runtime.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Method	Description
setProperties(Properties properties)	This method is called by the template engine runtime, once the class has been initialized, to set the template properties. Overriding the current properties will not affect the engine.
getProperties()	Return the current template properties.
VOID	Represents the Void class. VOID is used to make sure that returning data from the setter method is absolutely nothing. In general, Velocity considers the return of void from the setter method as a 'null' value. This causes the word 'null' to be printed out in the report. To avoid this problem, the setter method may use VOID as a return object.
class Void	A void class is used as a return in the Tool when you want to be sure that nothing is returned to the context.
class RetainedString	A direct command report formatter to keep the referenced String format. The formatter and other reference insertion handlers should maintain Strings directed by this class.

The other classes that can be extended are Tool and ConcurrentTool.



ITool Interface And Related Class.

Class Tool

A Class Tool is a base class realizing the interface ITool. This class provides the default implementation for ITool.

This class also provides methods derived from the java.util.Observable class, which can be used to notify the observers. An observer class, such as a template engine or a graphical user interface class, can receive the notification message from the tool and manage to display or interact with the user.

Concurrent Tool

A Concurrent Tool provides concurrent tasks running in the template engine. This class implements an unbounded thread-safe queue, which arranges the elements in a first-in-first-out (FIFO) order. New tasks are inserted at the tail of the queue, and the queue retrieval operations obtain elements at the head of the queue.

The tool extends from this class and is ideal for processing a long task that does not want other tasks to wait until the process is complete. The following code shows the sample usage of Concurrent Tool.

```
import com.nomagic.magicreport.engine.ConcurrentTool;
public class LongTaskTool extends ConcurrentTool
{
   public String longTask() {
      // enqueue object
      offer(new ConsumeObject(referent));
   }
   public void consume(ConsumeObject consumeObject) {
      Object referent = consumeObject.get();
      // process long task
   }
}
```

An example of Concurrent Tool is *FileTool*. The root template that calls the **file tool** method will create a subprocess, offer the subprocess into a queue, and continue the root template until finished. The template engine will later call the **consume** method to complete the subprocess once the consumer queue is available. The number of concurrent processes available for the execution is declared in the template engine property (See **TemplateConstants.TEMPLATE_POOL_SIZE** for detail.).

Creating custom tool

There are a few steps involved in developing a custom tool. These steps can be summarized as follows:

1. Developing a Tool Class
2. Creating an Extension Package

Developing a Tool Class

Developing a Tool class requires setting a class path to *magicreport.jar*. You can find *magicreport.jar* from the library folder under the Report Wizard plugin directory. The following sample shows the source code for *Hello.java*.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

```
package mytool;
import com.nomagic.magicreport.engine.Tool;
public class HelloTool extends Tool
{
   public String getHello() {
      return "Hello World";
   }
   public String getHello(String name) {
      return "Hello " + name;
   }
}
```

The sample shows two methods following the getter concept defined by the JavaBean specification.

Creating an Extension Package

The extension package is delivered in a *JAR* file. *JAR* (Java ARchive) is a file format based on the popular *ZIP* file format and is used for aggregating many files into one. To create a *JAR* file, you can store *.class with the Java package folder structure in a *ZIP* file format or creating from a *JAR* tool.

To combine files into a *JAR* file (in this case), here is the sample code fragment:

```
jar cf MyTool.jar *.class
```

In this example, all the class files in the current directory are placed in the file named "*MyTool.jar*". A manifest file entry named *META-INF/MANIFEST.MF* is automatically generated by the *JAR* tool. The manifest file is the place where any meta-information about the archive is stored as named: Value pairs. Please refer to the *JAR* file specification for more details.

```
jar cf MyTool.jar mytool
```

The example above shows that all the class files in the directory mytool are placed in the file named "*My Tool.jar*".

A complete *JAR* tool tutorial can be found at:

- http://java.sun.com/docs/books/tutorial/deployment/jar/
- http://java.sun.com/javase/6/docs/technotes/tools/windows/jar.html

Installing custom tool

To install the extension, copy the *JAR* file containing a custom tool class into the extensions directory under the Report Wizard plugin directory or template directory. Report Wizard will automatically load a new *JAR* file when a new report is generated.

Importing custom tool to template

Report Wizard uses the import directive for importing a Custom Tool to a template.

An example of a directive:

```
#import('prefix', 'name')
```

The import directive declares that the template uses the custom tool, names the tool that defines it, and specifies its tag prefix before the custom tool is used in a template page. You can use more than one import directive in a template, but the prefix defined in each template page must be unique.

Attributes

name (type:String)
The uniform or full qualified class name that locates the Tool class, for example, mytool.HelloTool

prefix (type:String)
The prefix that precedes the custom tool name such as a string in $hello.getHello(). Empty prefixes are not allowed. When developing or using custom tools, do not use tag prefixes such as bookmark, sorter, template, file, array, group, map, iterator, list, date, report, exporter, profiling, and project, as they are reserved by Report Wizard.

The code fragment indicated below declares a custom tool.

```
#import('hello', 'mytool.HelloTool')
Get Hello
$hello.getHello()
Get Hello Foo
$hello.getHello('foo')
```

The output from the above template is:

```
Get Hello
Hello World
Get Hello Foo
Hello foo
```

Auto importing template tool

Report Wizard can automatically import a Template Tool to a template without having to declare the #import statement in a template code manually (see Importing Custom Tool to Template(see page 1956)).

To enable Report Wizard to automatically import a Template Tool, you have to use the existing template variable inside "*template.xml*" and "*report.xml*".

The current template variable syntax:

```
<variable>
  <name>Variable Name</name>
  <description>Variable Description</description>
  <type/>
  <value xml:space="preserve">Variable Value</value>
</variable>
```

Add "com.nomagic.magicreport.engine.Tool" to a type value where the name is a variable name and the value is a full-qualified template tool class, for example:

To automatically import TextTool

---

1. Edit the built-in.xml file in the *...\data\reports\Tutorial 24 - Text Tool\reports*.
2. Add a variable as shown in the following example.

```
<variable>
  <name>text</name>
  <description>A text tool</description>
  <type>com.nomagic.magicreport.engine.Tool</type>
  <value xml:space="preserve">com.nomagic.reportwizard.tools.TextTool</value>
</variable>
```

The result is a template code that can access com.nomagic.reportwizard.tools.TextTool from $text.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

```
$text.noLineBreak($diagram.name)
```

Name

Once you have edited a template variable, you have to restart MagicDraw prior to generating a report through MagicDraw.

Appendix B: office open XML format template

Office Open XML (also referred to as OOXML or Open XML) refers to the standardized file formats of spreadsheets, charts, presentations, and word processing documents for Microsoft Office. These standardized file formats become free and open as ECMA-376 Office Open XML File Formats - 2nd edition (December 2008) and ISO/IEC 29500:2008.

The most common Open XML file formats that Report Wizard supports include:

- DOCX - for word processing (text) documents
- XLSX - for spreadsheets

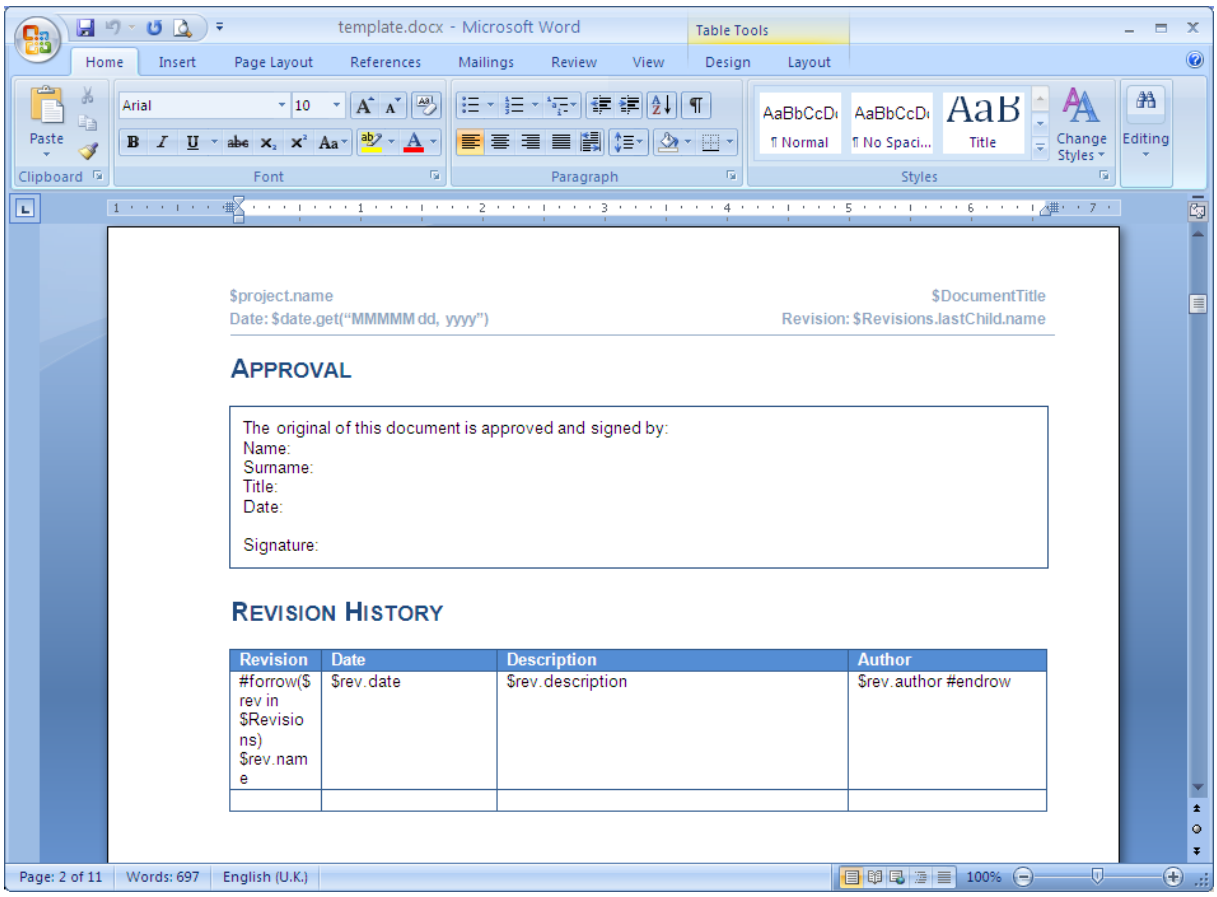
- PPTX - for presentations

> **Note**
> Support Office Open XML (DOCX, XLSX, and PPTX) and Microsoft Office 2007 and newer versions.

Related pages

Microsoft Office Word document (DOCX)

Report Wizard supports most *DOCX* features. You can place the *VTL* codes inside core (properties) and content of any *DOCX* file. All syntax usable in *RTF* can also be used in *DOCX*.

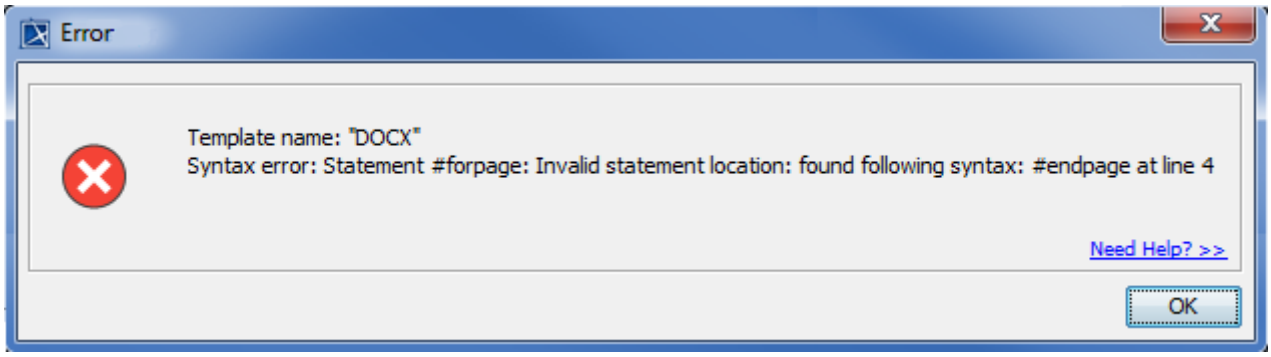


A Sample of DOCX Converted from an RTF Document.

Limitations when used in Microsoft Office Word document

You cannot use multi-line statements in different objects. If you try to use them in *DOCX*, an error message will open. See the following example.

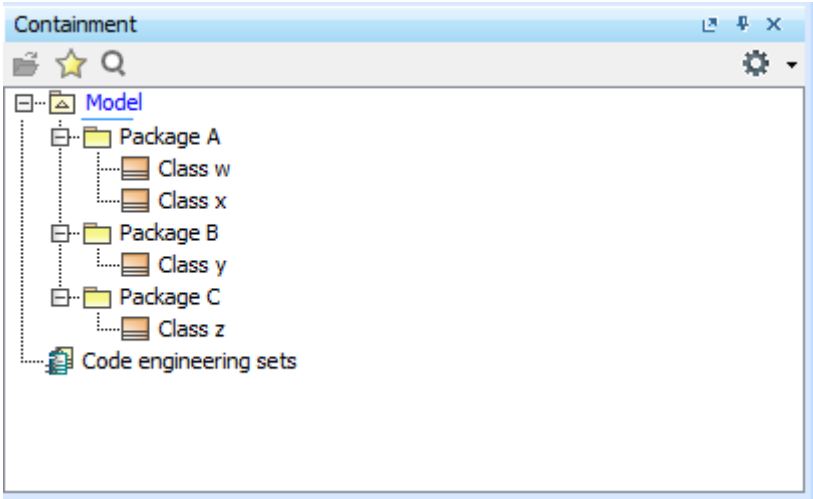
```
1   Using directive in different object
2   #forpage($c in $Class)
        $c.name
        #endpage
3
```



The Error Message of Invalid Usage of the Multi-line Statements in DOCX.

Creating data for multiple columns

#forcol is used for creating data for multiple columns in a row. This statement must be defined in table and can be used in conjunction with the **#forrow** statement. For example:



Sample elements in the MagicDraw Containment tree.

Template Code:

Name	`#forcol($p in $Package)$p.name#endcol`

Name	Package A	Package C	Package B

#forcol may create different number of columns in each row. See the example below.

Template Code:

#forrow($p in $sPackage) $p.name	#forcol($o in $sorter.sort($p.ownedElement)) $o.name#endcol#endrow

Output:

Package A	Class w	Class x
Package C	Class z	
Package B	Class y	

#forcol must be defined as the first statement in a column because it is parsed and processed before other statements in the column except **#forrow**. See the example below.

Template Code

Name	#set($p = $Package) #forcol($ap in $p) $ap.name#endcol

Above code will be parsed to

Name	#forcol($ap in $p) #set($p = $Package) $ap.name#endcol

Output:

Name

So, the template code should be:

Template Code:

Name#set($p = $Package)	#forcol($ap in $p) $ap.name#endcol

Output:

Name	Package A	Package C	Package B

#forcol will create dynamic columns in a row. This directive does not create columns for the whole table. See the example below.

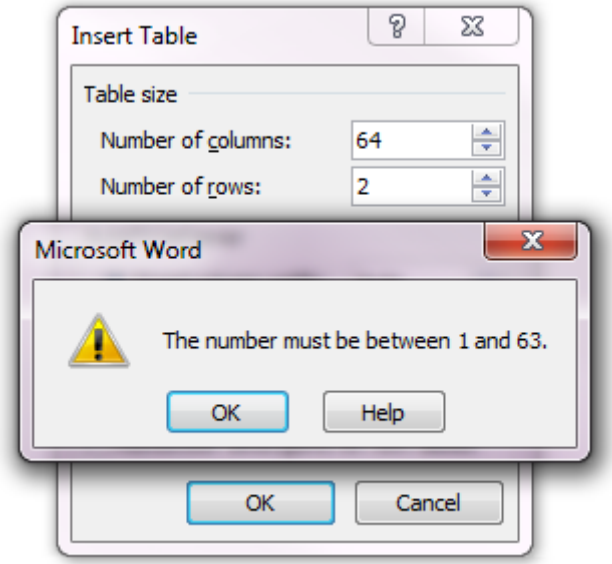
Template Code:

Name	#forcol(pinPackage)$p.name#endcol

Output:

Name	Package A	Package C	Package B

Docx supports at most 63 columns per row (see the following figure).



Docx supports at most 63 columns per row.

Creating merged column horizontally for DOCX

ReportWizard provides utility functions for creating table properties. The functions of this module are accessible from templates through **$tableprop**. In this version, ReportWizard provides only **$tableprop.mergeColumns()** to create merged columns horizontally in *DOCX* template.

This statement must be defined in table.

1. $tableprop.mergeColumns(int number)

Merging columns in specified number

Where the parameter is:

- number - the number of columns to merge. This value has to be of type integer and start from 2.

For example:

Template Code:

Project	$tableprop.mergeColumns($Package.size())Packages
	#forcol($p in $Package)$p.name#endcol

Project	Packages		
	Package A	Package C	Package B

2. $tableprop.mergeColumns(String stringNumber)

Merging columns in specified number as String

Where the parameter is:

- stringNumber - the string value of number of columns to be merged. This value has to be of type integer string and start from “2”.

For example:

Template Code:

Project	$tableprop.mergeColumns(“3”) Packages
	#forcol($p in $Package)$p.name#endcol

Output:

Project	Packages		
	Package A	Package C	Package B

> **$table.mergeColumns()** must be defined as the first statement in the column because it is parsed and processed before other statements in column expect **#forrow** and **#forcol.** See the example below.

Template Code:

Project	#set($gValue = $Package.size() + 1)$tableprop.mergeColumns($gValue)Packages	
	#forcol($p in $Package) $p.name#endcol	No Package

Above code will be parsed to

Project	$tableprop.mergeColumns($gValue) #set($gValue = $Package.size() + 1)Packages	
	#forcol($p in $Package) $p.name#endcol	No Package

Output:

Project	Packages			
	Package A	Package C	Package B	No Package

So, the template code should be:

Template Code:

Project#set($gValue = $Package.size() + 1)	$tableprop.mergeColumns($gValue)Packages	
	#forcol($p in $Package)$p.name#endcol	No Package

Project	Packages			
	Package A	Package C	Package B	No Package

$tableprop.mergecolumns() will make a column to be a merged column, it does not change the number of columns in a row. See the following example.

Template Code:

$tableprop.mergeColumns("5")Packages		
#forcol($p in $Package)$p.name#endcol		

Output:

Packages						
Package A	Package C	Package B				

The number of columns in the first row is still 3.

Microsoft Office Excel worksheet (XLSX)

Multi-Line Statements in *XLSX*

All multi-line directives such as #if, #else, #elseif, #foreach, and #macro must be used under the following conditions.

The beginning and ending statements must be declared within a single cell. The figures below show samples of invalid usage of the **#if** and **#foreach** statements between cells respectively.

	A	B
1	#if($e.elementType == "usecase")	
2	$e.name	
3	#end	
4		

Executing multi-line statements.

	A	B	C	D	E
1					
2		**Use Case**			
3		#foreach($uc in $UseCase)	$uc.name	#end	
4					

Invalid usage of #foreach statements.

In the first figure, since the body of the #if statement ($e.name) resides in the A2 cell, not in the A1 cell, the body will not be generated when generating a report, regardless of the evaluation of the #if statement. The code shown in the second figure will break the structure of a spreadsheet document.

The following two figures demonstrate samples of valid usage of the #if and #foreach statements respectively.

	A	B
1	#if($e.elementType == "usecase")$e.name#end	
2		

Valid usage of an #if statement.

	A	B	C
1			
2		**Use Case**	
3		#foreach($uc in $UseCase) $uc.name #end	
4			

Valid usage of a #foreach statement.

- A VTL Macro must be declared within a single cell. Do not insert the multi-cell recorded macros in a single cell, see figure below.

	A	B	C
1			
2		**Macro**	
3		#macro(insertCell $e)	
4		#if($e=="red")$e.name	
5		#else $e.name	
6		#end	
7		#end	
8			
9		**Use Macro**	
10		#foreach($e in $elements)	
11		#insertCell($e)	
12		#end	
13			

Invalid usage of a VTL Macro.

The macro will copy all contents between #macro and #end. Cells and rows will be included in the macro as well. Once this record has been inserted, the macro content will break the document structure.

The following figure demonstrates a sample of valid usage of the #macro statement.

	A	B	C
1			
2		**Macro**	
3		#macro(insert $e) #if($e=="red") $e.name #else $e.name #end #end	
4			
5		**Use Macro**	
6		#foreach($e in $elements) #insertCell($e) #end	
7			

Valid usage of a VTL Macro.

Creating data for multiple rows

The **#foreach** directive can only be used in a single cell record. To create data for multiple rows, use the **#forrow** directive instead (see the following figure).

	A	B	C	D
1	#forrow($uc in $UseCase)	$uc.name	#endrow	
2				

The forrow directive.

The figure below demonstrates the output of the above code.

	A	B	C
1		Use Case A	
2		Use Case B	
3		Use Case C	
4		Use Case D	
5			

Results of running the code from figure 7.

Creating data for multiple columns

#forcol is used for creating data for multiple columns (see the figure below). This statement can be used in conjunction with the **#forrow** statement.

	A	B
1	#forcol($uc in $UseCase)$us.name#endcol	
2		

The forcol directive.

The following figure demonstrates the output of the above code.

	A	B	C	D	E
1	Use Case A	Use Case B	Use Case C	Use Case E	
2					

Results of running the code from the figure above.

Displaying content in a cell

Texts in any generated report are always wrapped. Also, the cells' width in the generated report depends on the cells' width in the report template used. For example, an *XLSX* report template in the first figure below will generate an output report as shown in the second figure below.

	A	B	C	D
1	#forrow($uc in $UseCase)	$uc.name	#endrow	
2				

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

The forrow directive.

	A	B	C
1		this_is_u secase1	
2		this_is_u secase2	
3		this_is_u secase3	
4			

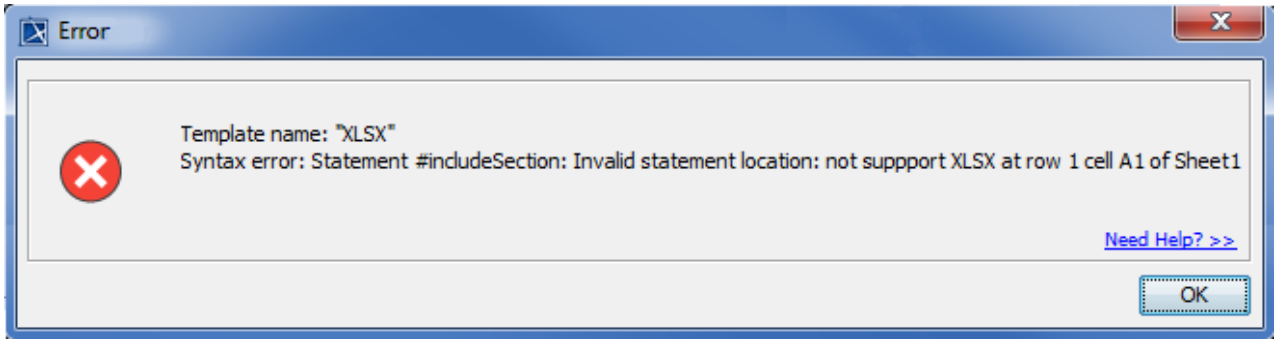
Results of running the code from the figure above.

Limitation when used in Microsoft Office Excel worksheet

You cannot use **#sectionBegin** and **#includeSection** in *XLSX*. If you try to use **#sectionBegin** or **#includeSection** in an *XLSX* report template (the first figure below) an error message will open (the second figure below).

	A	B
1		
2		
3		
4	#sectionBegin(SectionA)	
5		
6	This is section begin	
7		
8		
9	#sectionEnd	
10		

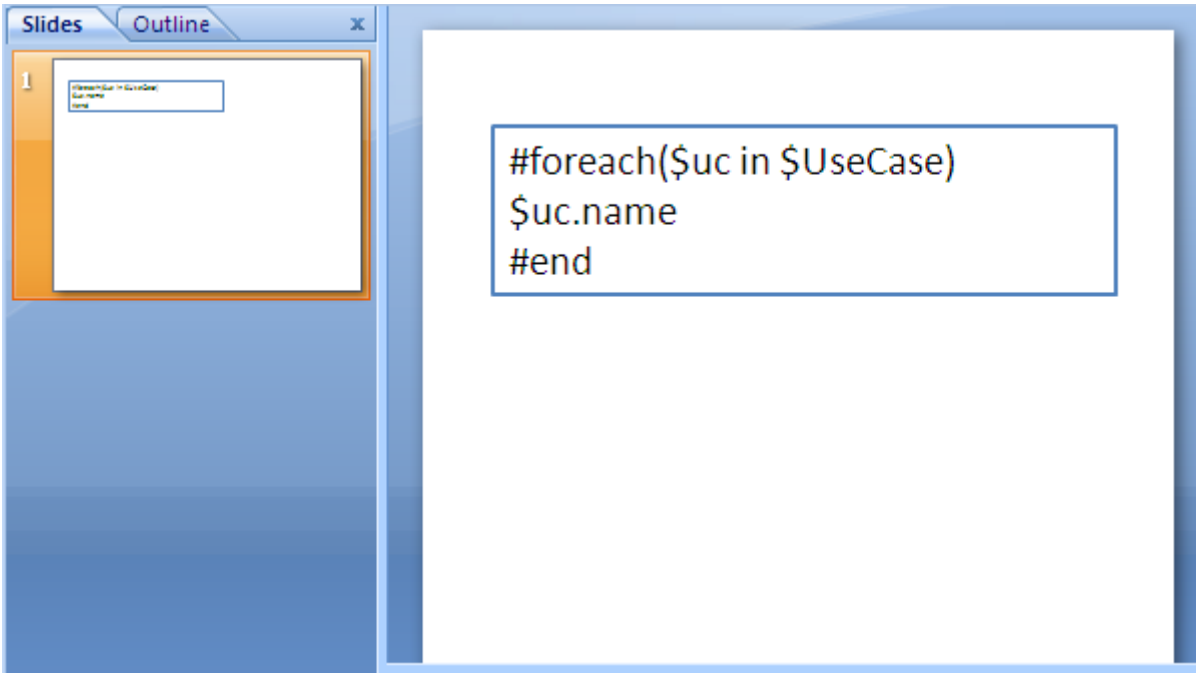
Invalid usage of a sectionBegin directive.



The error message from running the code in the figure above.

Microsoft Office PowerPoint presentation (PPTX)

A presentation document requires a special document template. This template does not contain any content order, and all text content is always placed inside a text box. A text box is an image structure (an image structure keeps the position of each image in x, y coordinates). You can change a Text box position. You can also place a Text box in the same positions as others (see the figure below).



A sample of PPTX template.

Multi-line statements in PPTX

Similar to *XLSX*, all multi-line directives such as #if, #else, #elseif, #foreach, and #macro must be used under the following conditions.

1. The beginning and ending statements must be declared within a single text box. The following code shows a sample of invalid usage of the#foreach statement between the text boxes.

```
#foreach($uc in $UseCase)
```

```
$uc.name
```

```
#end
```

Since the PPTX template does not provide the statement order, the template will not be interpreted as the order of the displayed images (text boxes). For example, $uc.name may not be processed after #foreach($uc in $UseCase) has been completely processed. The following code demonstrates a sample of valid usage of the #foreach statement.

```
#foreach($uc in $UseCase)
$uc.name
#end
```

2. A VTL Macro must be declared within a single text box. Do not insert the multi-cell recorded macros in a single text box. See the following code example.

```
#macro (insertText $e)
```

```
$e.name
```

```
#end
```

Since each text box does not have any sequence order, the macro cannot record any content between the text boxes. The following code example demonstrates a valid usage of the #macro statement.

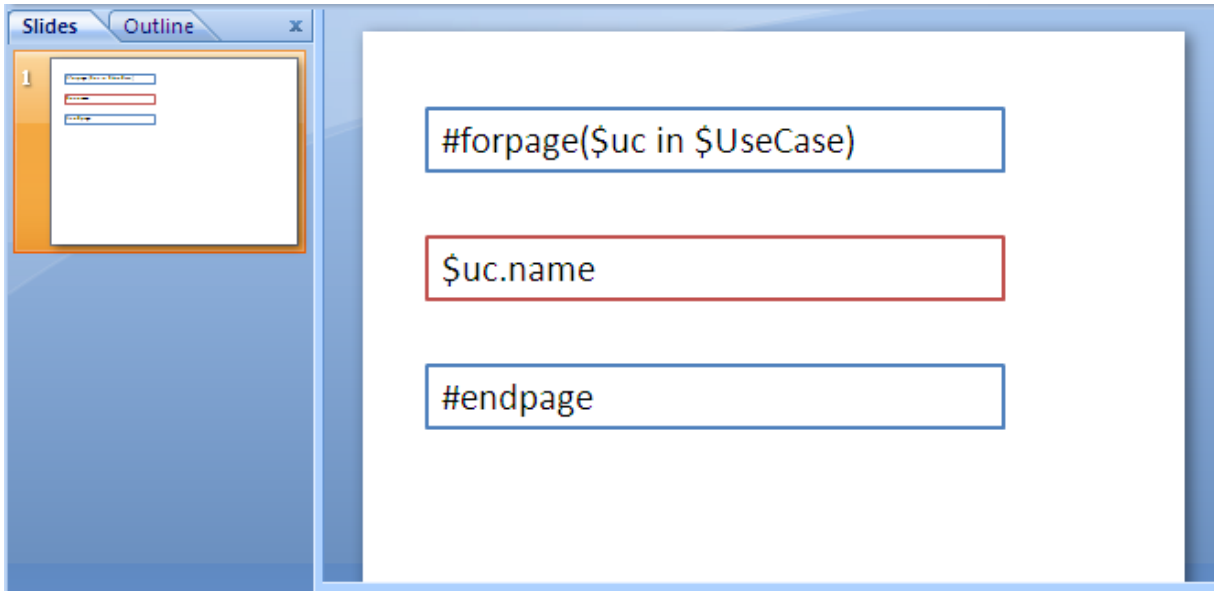
```
#macro(insertText $e)
$e.name
#end
```

```
#foreach($uc in $UseCase)
#insertText($uc)
#end
```

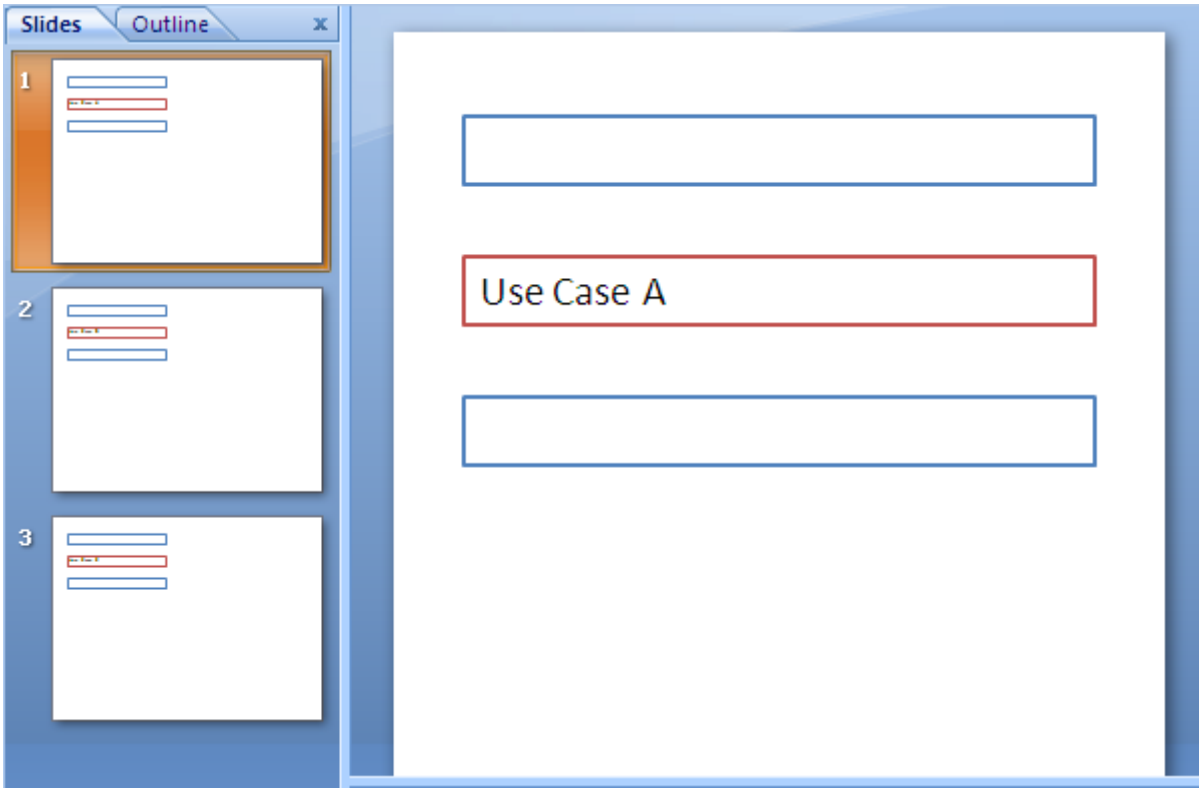
Creating data for multiple slides

In PPTX report templates, you can use the #forpage directive to create additional slide(s) in your presentation. You can use #forpage and #endpage directives in any text box. However, the #endpage directive must be on same slide as the #forpage directive or on the following slide, but not before the slide which contains the #forpage directive. All directives on the slide(s) between 1) the slide the #forpage directive appears and 2) the slide the #endpage directive appears will be included within the #forpage statement. For example, the template in the first figure below will produce the output as

shown in the second figure below.



Creating data for multiple pages using the forpage directive.



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Results of running the code from the figure above.

Creating a page with conditions

Since a directive in the *PPTX* report template does not provide any statement order (unless residing in the same text box), you cannot used the #if directive together with the #forpage directive. See figure below.

```
#forpage($e in $elements)
#if($e.elementType == "usecase")

$e.name

#end
#endpage
```

The code above will not produce the output report as expected, because the #forpage directive automatically covers all directives in the current page regardless of the statement order. Consequently, the #if directive may not be interpreted after the #forpage directive.

To avoid this problem, you can use the $report.filterElement($elements, $types) method. This helper method provides the element filter for the specified type. In this case, use the following code.

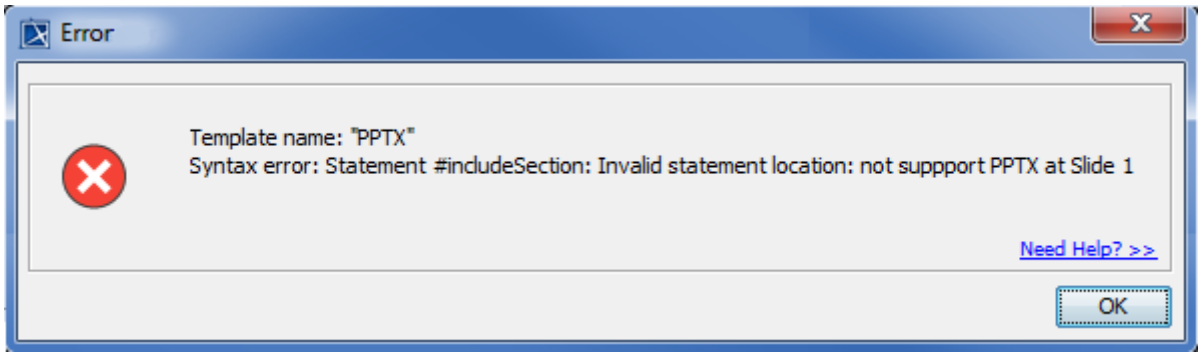
```
#forpage($e in $report.filterElement($elements,
["usecase"]))

$e.name

#endpage
```

For more details on $report.filterElement($elements, $types), see Section Helper Modules(see page 1621).

Limitation when used in Microsoft Office PowerPoint presentation

#sectionBegin, **#includeSection** cannot be used in any *PPTX* report template. If you try, for example, to use **#sectionBegin** in a *PPTX* report template, an error message (the figure below) will open.

```
#sectionBegin (sectionA)
Section A
#sectionEnd
```



The error message dialog that appears from running an invalid sectionBegin directive.

Appendix C: OpenDocument format template

The OpenDocument format (*ODF*) is an open file format for office documents, such as spreadsheets, presentations, and word processing documents. The standard was developed by the Open Office *XML* technical committee of the Organization for the Advancement of Structured Information Standards (OASIS) consortium.

The version 1.0 manifestation was published as an ISO/IEC International Standard, ISO/IEC 26300:2006 Open Document Format for Office Applications (OpenDocument) v1.0.

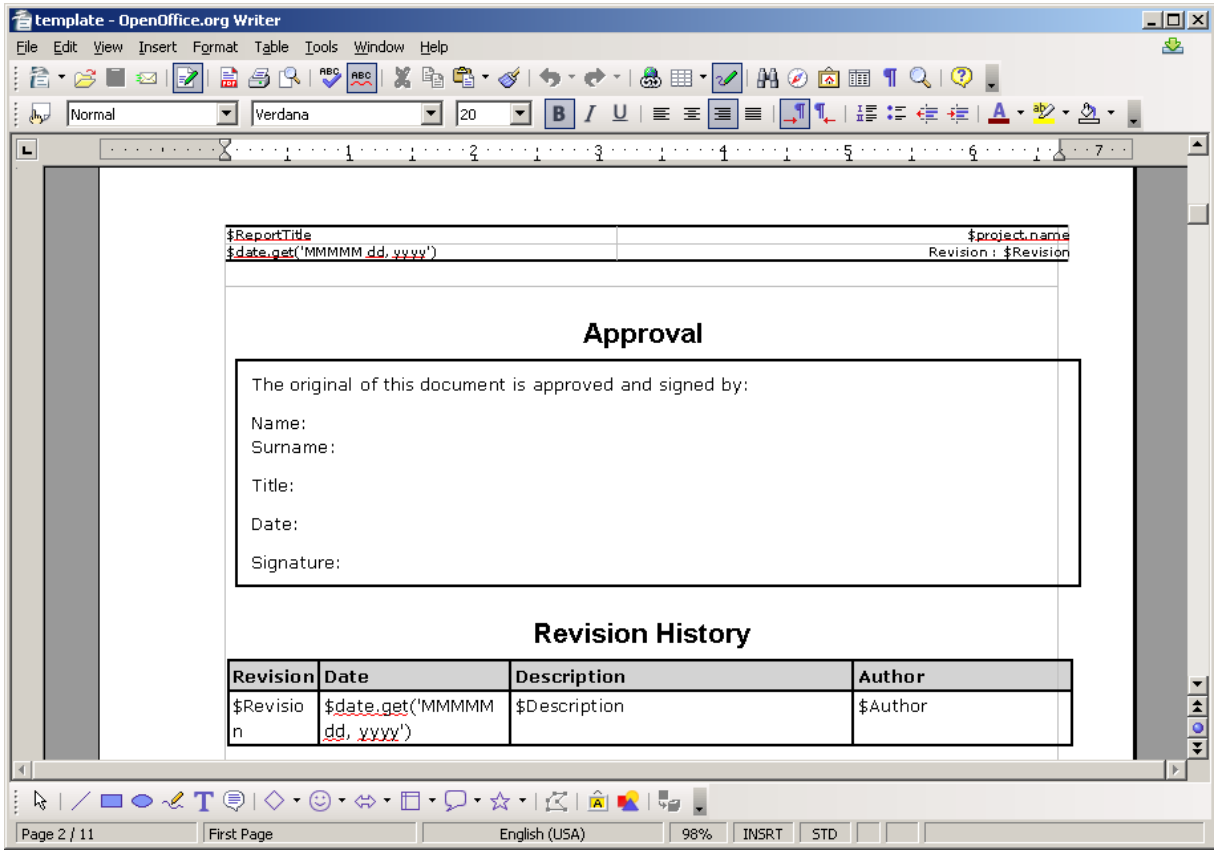
The most common files supported by Report Wizard are:

- *ODT*
  for word processing (text) documents
- *ODS*
  for spreadsheets
- *ODP*
  for presentations

Related pages

OpenDocument text

Report Wizard supports most OpenDocument Text (*ODT*) features. It enables you to input the *VTL* codes inside meta-data, styles, and text content. All syntax usable inside *RTF* can be used with *ODT*.



A sample of OpenDocument Text Converted From RTF Document.

OpenDocument spreadsheet

All multi-line directives such as #if, #else, #elseif, #foreach, and #macro must be used under the following conditions:

1. The beginning and ending statements must be declared within a single cell. The two figures below show two samples of invalid usage of the multi-line statements between cells.

	A	B
1	#if ($e.elementType == "usecase")	
2	$e.name	
3	#end	
4		

A sample of an invalid multi-line statement.

	A	B	C	D
1				
2		**Use Case**		
3		#foreach ($uc in $UseCase)	$name	#end
4				

Another sample of an invalid multi-line statement.

Since the body of the **#if** statement is contained in the cell A2, this cell will not be generated if the condition is not true (the element type is not “usecase”). Due to the constraints of spreadsheet document structure, the number of cells in a column must be equal to the number of cells in all columns, and the number of cells in a row must be equal to the number of cells in all rows.

The codes in the figure above will break the structure of a spreadsheet document. The two figures below demonstrate two samples of valid usage of the multi-line statements.

	A	B
1	#if ($e.elementType == "usecase")$e.name#end	
2		

A sample of a valid usage of multi-line statements.

	A	B	C
1			
2		**Use Case**	
3		#foreach ($uc in $UseCase) $name #end	
4			

Another sample of a valid usage of multi-line statements.

2. *VTL* Macro must be declared within a single cell. Do not insert the multi-cell recorded macros in a single cell, see figure below.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

	A	B	C
1			
2		**Macro**	
3		#macro(insertCell $e)	
4		#if($e=="red")$e.name	
5		#else $e.name	
6		#end	
7		#end	
8			
9		**Use Macro**	
10		#foreach($e in $elements)	
11		#insertCell($e)	
12		#end	
13			

A sample of an invalid usage of a VTL macro.

The macro will copy all contents between **#macro** and **#end**. Cells and rows will be included in the macro as well. Once this record has been inserted, the macro content will break the document's structure.

Creating Data for Multiple Rows

The **#foreach** directive can only be used in a single cell record. To create data for multiple rows, use the **#forrow** directive instead, see figure below.

	A	B	C
1	#forrow($uc in $UseCase)	$uc.name	#endrow

The #forrow directive being used to create data for multiple rows.

As shown in figure below, the output will be:

	A	B	C
1		Use Case A	
2		Use Case B	
3		Use Case C	
4		Use Case D	
5			

The results of running the code from the figure above.

Creating Data for Multiple Columns

#forcol is used to create data for multiple columns, see figure below. This statement can be used with **#forrow** (see the sample from the "Other Document" template).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

	A	B
1	#forcol($uc in $UseCase)$uc.name#endcol	
2		

The #forcol directive being used to create data for multiple columns.

From the usage of #forcol shown in figure below, the output will be:

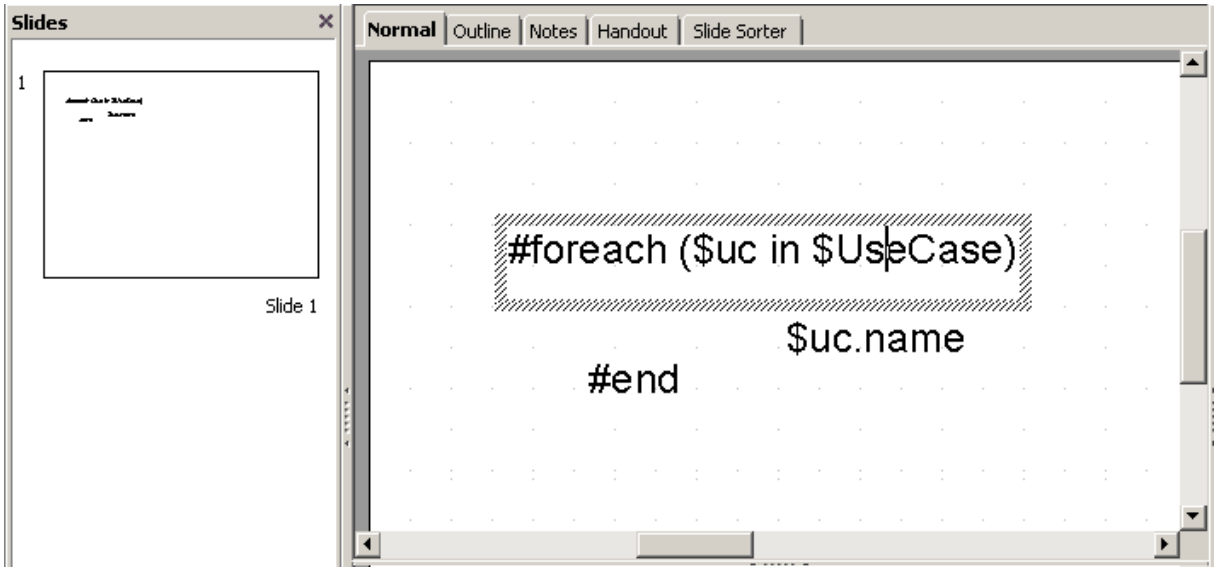
	A	B	C	D	E
1	Use Case A	Use Case B	Use Case C	Use Case D	
2					

The results of running the code from the figure above.

OpenDocument presentation

A presentation document is a special document template. This template does not contain a content order. The text content used within this document is inserted inside a text box. A text box is an image structure (An image structure keeps the position of each image in x, y coordinates).

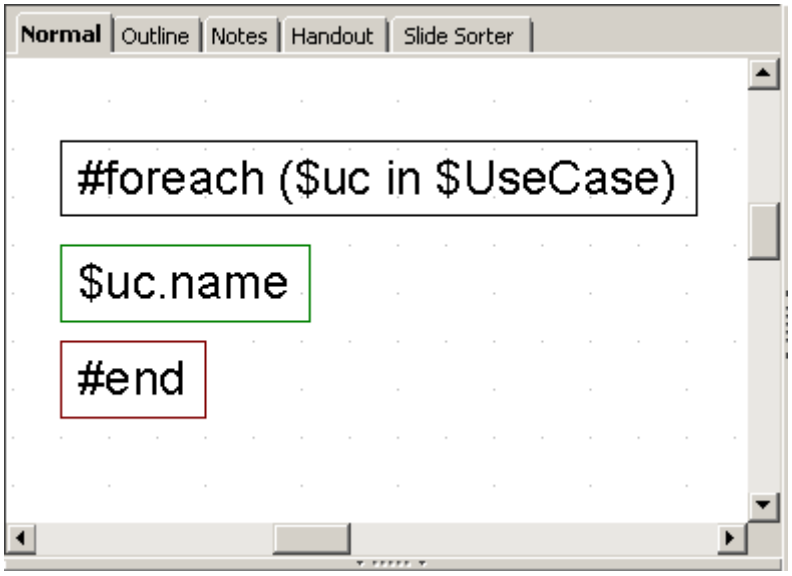
You can change the position of a Text box. You can also place Text boxes in the same positions as others, see figure below.



A sample of ODP Template.

Using the same concept as *ODS*, all multi-line directives such as #if-#else-#elseif, #foreach, and #macro must be used under conditions.

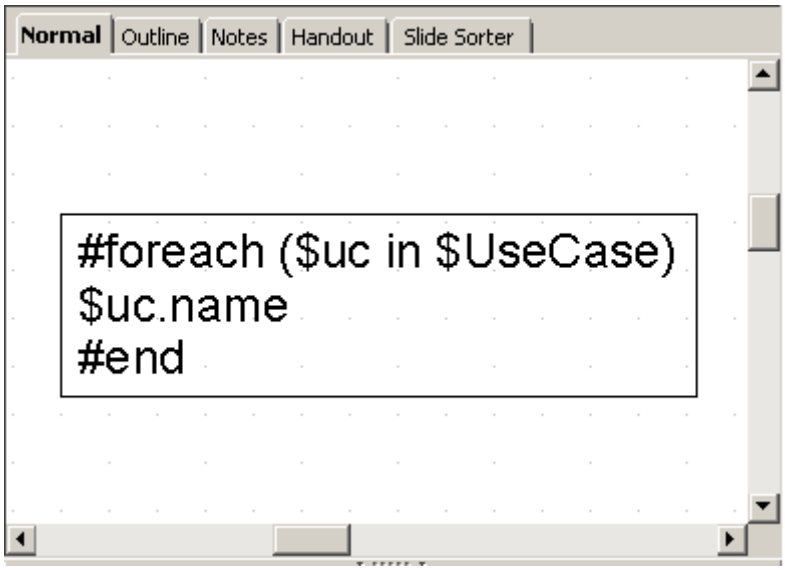
- The beginning and ending statements must be declared within a single text box. The figure below shows the sample of invalid usage of the multi-line #foreach statement between the text boxes.



Invalid Usage of Multiline Statement in ODP.

Since the *ODP* template does not provide the statement order, the template will not be interpreted in the order of the displayed images. For example, $uc.name[363] may not be processed after #foreach($uc in $UseCase) has been completed.

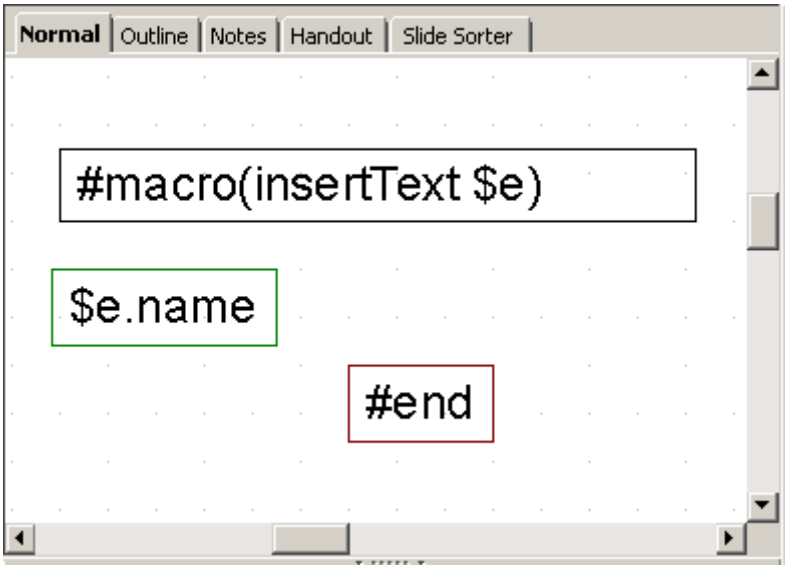
The figure below shows the sample of valid usage of the **#foreach** statement.



A sample of valid usage of the foreach statement.

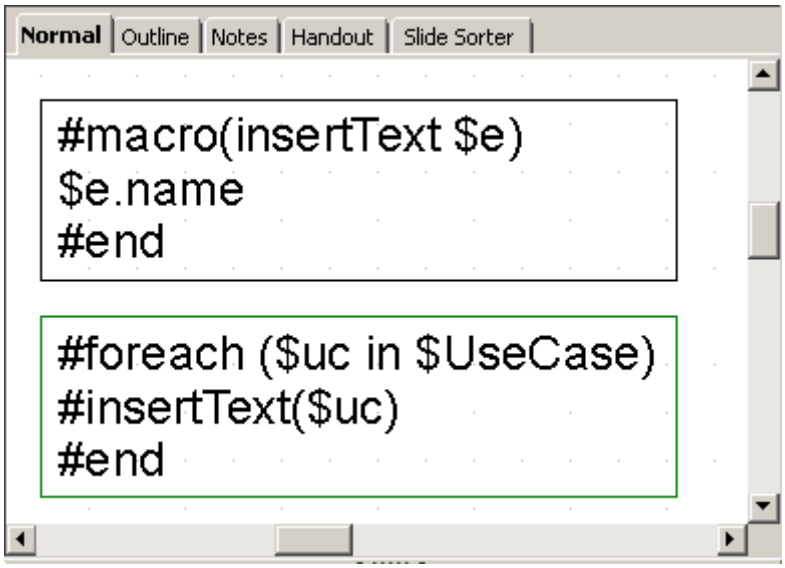
- A *VTL* Macro must be declared within a single text box. Do not insert the multi-cell recorded macros in a single text box as shown in figure below.

363 http://uc.name



Invalid Usage of Macro Statement in ODP.

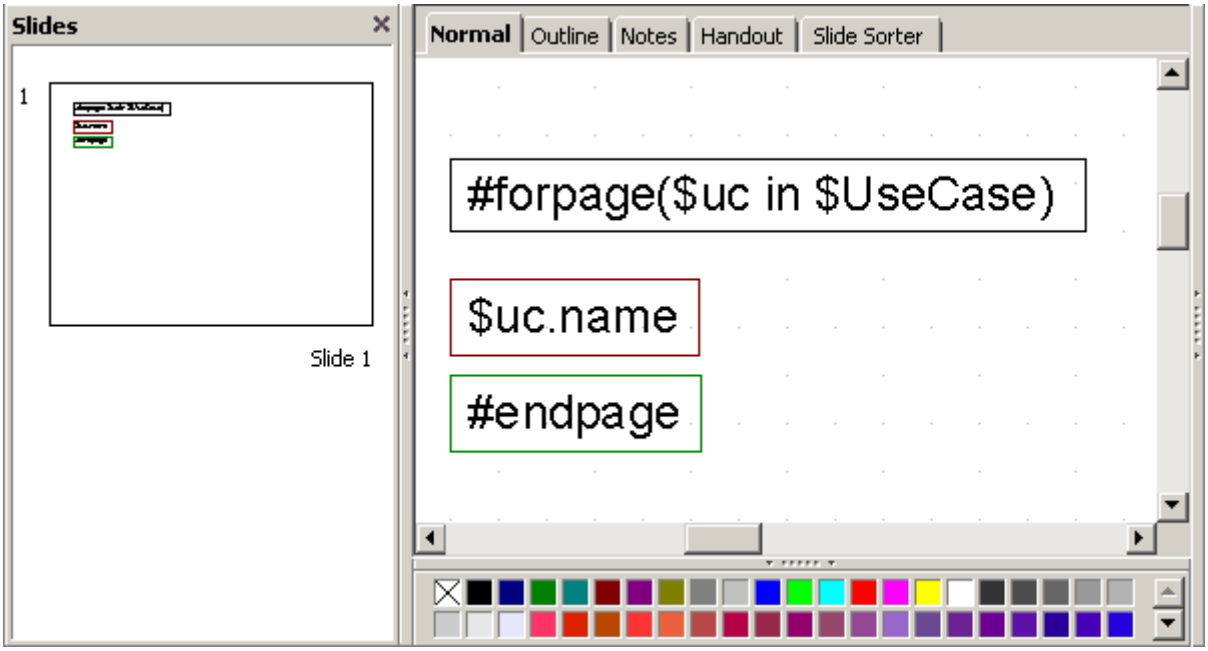
The text box does not have a sequence order; therefore, macros cannot record any content between the text boxes, see figure below.



Valid Usage of Macro Statement in ODP.

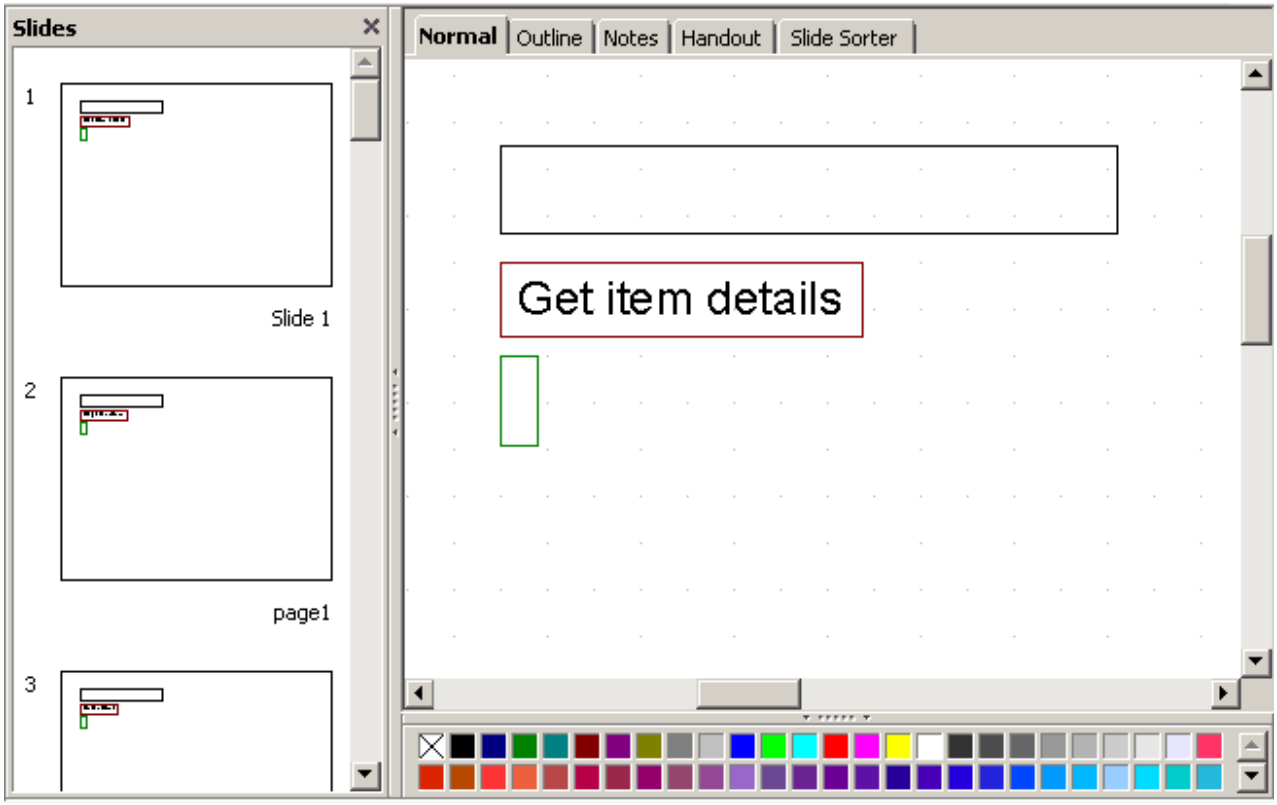
Creating Data for Multiple Slides

ODP uses the **#forpage** directive to create a slide for each data. The **#forpage** directive does not contain any order. You can use **#forpage** and **#endpage** in any text boxes. All directives on the slide will be included within the **#forpage** statement, see figure below.



A sample usage of the forpage directive.

The output from the code, shown in figure below, is an *ODP* with a single use case name for each slide.



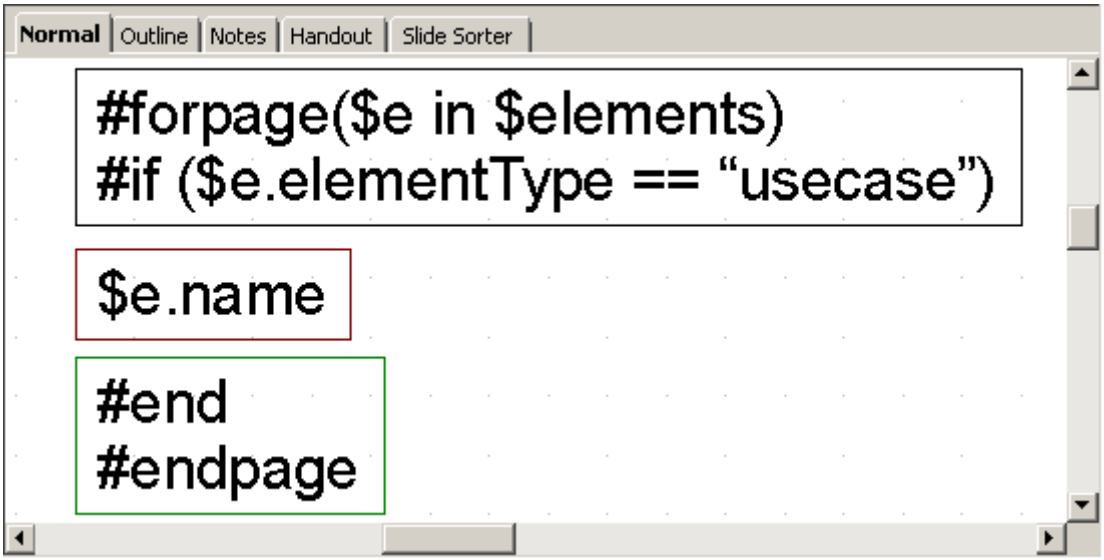
The result of running the code from the figure above.

For more samples of *ODP* reports, see the “Other Document” template.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Creating Page with Conditions

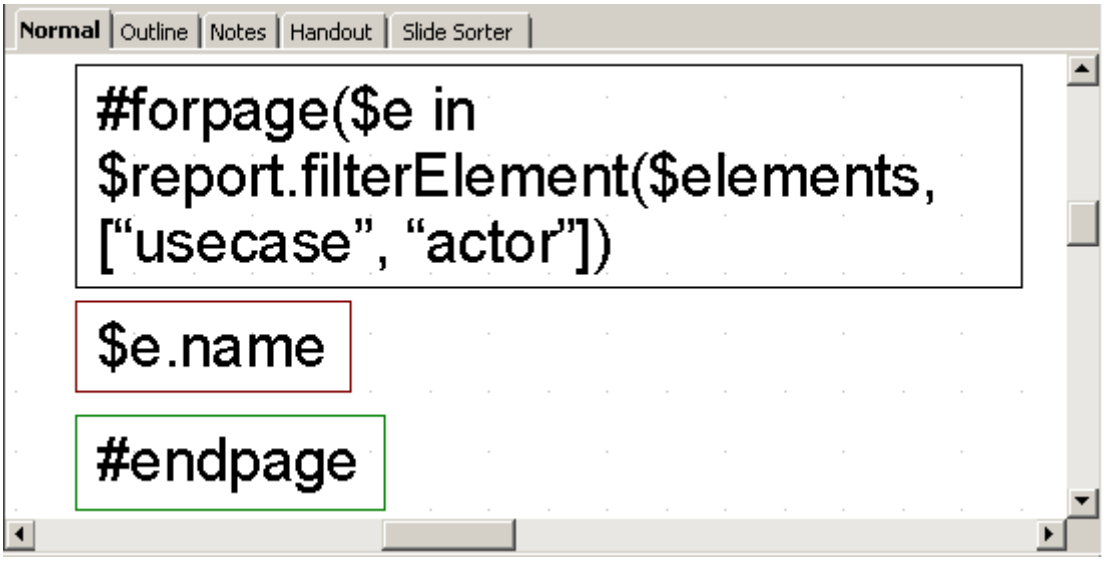
The *ODP* directive does not provide any statement order; therefore, the #if directive will not be attached to the #forpage statement, as shown in figure below.



A sample of Looping with Condition in General Style.

The codes in the figure above may not produce the report exactly as expected. #forpage automatically covers all directives in the current page without any statement order. Therefore, the #if directive may not be interpreted after #forpage.

To solve this problem you can use the $report.filterElement($elements, $types) method. This helper method provides the element filter for the specified type. The codes are shown in the figure below:



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

A sample of Looping with Conditional Filter.

For more details on $report.filterElement($elements, $types), see Section Helper tools(see page 1621).

OpenDocument conversion tool

The following tools can convert *RTF* documents to *ODF* documents.

Microsoft Office ODF Extensions

This Microsoft Office add-on allows Microsoft Office to open *ODF* documents and save Microsoft Office documents in the OpenDocument format.

Download Microsoft Office ODF Extensions from http://odf-converter.sourceforge.net/

OpenOffice.org[364]

OpenOffice.org[365] is an office suite that can open and save documents in many formats. This tool can also open *RTF* documents and save them as *ODF* documents.

Download OpenOffice.org[366] from http://www.openoffice.org.

OpenDocument references

ISO/IEC 26300:2006 Information Technology - Open Document Format for Office Applications (OpenDocument) v1.0.

- http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=43485

Application Supports for the OpenDocument Format

- http://opendocumentfellowship.com/applications

Microsoft expands List of Formats Supported in Microsoft Office

- https://news.microsoft.com/2008/05/21/microsoft-expands-list-of-formats-supported-in-microsoft-office/#sm.001yabsye3akd17114x1uf5dn16oa

OpenOffice.org[367]

- http://www.openoffice.org/

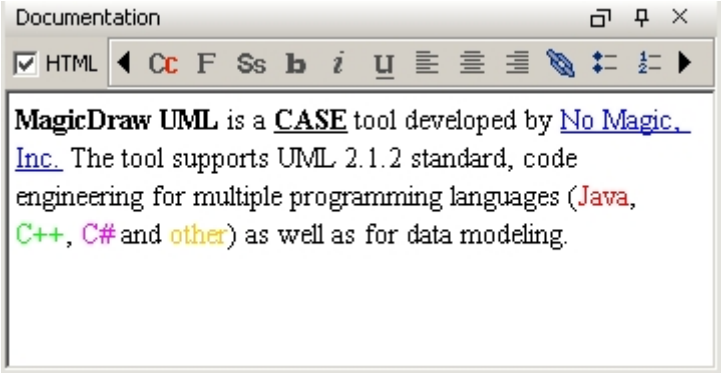
Microsoft Office ODF Extensions

- http://odf-converter.sourceforge.net/

364 http://OpenOffice.org
365 http://OpenOffice.org
366 http://OpenOffice.org
367 http://OpenOffice.org

Appendix D: HTML tag support

Report Wizard supports *HTML* code conversion to *RTF, ODF,* and *OOXML* file formats. The *HTML* code is created from the Element Documentation pane, see figure below.



The MagicDraw Documentation Pane.

The element documentation can also be retrieved by the following *VTL* code:

```
#foreach ($e in $elements)
$e.documentation
#end
```

Whenever the report engine encounters *HTML* content, it will automatically convert the content into a valid output format style.

Supported HTML tags

On this page

- Font tags(see page 1986)
  - Size(see page 1986)
  - Face(see page 1986)
  - Color(see page 1987)
- Font style tag(see page 1988)
- Phrase elements(see page 1989)
- Ordered and unordered lists and list Item tags(see page 1990)
  - Ordered lists(see page 1990)
  - Nested ordered lists(see page 1990)
  - Unordered lists(see page 1991)
  - Nested unordered lists(see page 1992)
- Definition list tags(see page 1992)
- Line and paragraph tags(see page 1993)
- Preformatted text(see page 1994)

- Heading tags(see page 1994)
- Link tags(see page 1995)
- Table tags(see page 1996)
  - Table elements(see page 1996)
  - Row elements(see page 1998)
  - Cell elements(see page 1999)
  - Column width (see page 2001)
  - Header elements(see page 2002)
- Image tags(see page 2004)
  - src(see page 2004)
  - width(see page 2004)
  - height(see page 2004)
  - alt(see page 2004)
- Superscript and subscript tags(see page 2005)
  - Superscript(see page 2005)
  - Subscript(see page 2006)

Font tags

A font tag consists of three attributes: **Size**, **Face**, and **Color**.

Size

The Size attribute determines the font size. Possible values are integers from 1 to 7. The default base font size is 3. The greater the value is, the larger the size becomes.

- The base font size for *RTF* documents is 24 dot (equivalent to 12 pt).
- The base font size for *ODF* documents is 12 pt.
- The base font size for *OOXML* documents is 12 pt.
- The base font size for *HTML* documents is determined by the web browser.

Each value will be multiplied by two. Shown below is an example of the size attribute:

```
<font size="5">
It will be rendered as font size 16 pt
<font size="3">
It will be rendered as font size 12 pt
<font size="1">
It will be rendered as font size 8 pt
```

If the size attribute is specified without the face attribute, the default font will be determined by the template or document editor, unless the font tag is covered by other HTML elements, such as <code> or <tt>.

Face

The Face attribute defines the font name. If the face attribute is specified without the size attribute, the default size will be determined by the template or the document editor.

Color

The Color attribute specifies the text color. A color value can be either a hexadecimal number (prefixed with a hash mark) or one of the following sixteen colors. Colors are case-insensitive.

The table below lists several Font Colors:

Color	Hexadecimal code
Black	#000000
Silver	#C0C0C0
Gray	#808080
White	#FFFFFF
Maroon	#800000
Red	#FF0000
Purple	#800080
Fuchsia	#FF00FF
Green	#008000
Lime	#00FF00
Olive	#808000
Yellow	#FFFF00
Navy	#000080
Blue	#0000FF
Teal	#008080
Aqua	#00FFFF

For example:

```
<font face="Arial"> This is Arial text</font><br>
<font face="Comic Sans MS">This is Comic Sans text</font><br>
<font face="Arial" size="1">This is small Arial</font><br>
<font face="Arial" size="7">This is large Arial</font><br>
<font face="Arial" color="Red">This is red Arial</font><br>
<font face="Arial" color="#FF0000">and this is red Arial too</font><br>
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

As shown in the figure below, the outputs in *RTF, ODF,* or *HTML* will be:

This is Arial text

This is Comic Sans MS text

This is small Arial

This is large Arial

This is red Arial

and this is red Arial too

Font style tag

The table below lists various Font Style Elements:

Tag name	Description
TT	Renders teletyped or monospaced text.
I	Renders italic text.
B	Renders bold text.
BIG	Renders text in large font.
SMALL	Renders text in small font.
STRIKE and S	Renders strikethrough text.
U	Renders underlined text.

- TT
  This tag will be rendered as <font face="Courier New">
- I
  This tag is supported by the existing HTML conversion component.
- B
  This tag is supported by the existing HTML conversion component.
- BIG
  This tag will be rendered as <font size="5">
- SMALL
  This tag will be rendered as <font size="1">

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- STRIKE and S
  This tag will be rendered in a strikethrough text.
- U
  This tag is supported by the existing HTML conversion component.

Phrase elements

The table below lists assorted Phrase elements:

Tag name	Function
EM	Indicates emphasis.
STRONG	Indicates stronger emphasis.
CITE	Contains a citation or a reference to other sources.
DFN	Indicates that this is the defining instance of the enclosed term.
CODE	Designates a fragment of computer code.
SAMP	Designates a sample output from programs, scripts, etc.
KBD	Indicates the text to be entered by the user.
VAR	Indicates an instance of a variable or program argument.
ABBR	Indicates an abbreviated form such as WWW, HTTP, URI, and Mass.
ACRONYM	Indicates an acronym such as WAC and radar.

- *EM*
  This tag will be rendered as <i>
- *STRONG*
  This tag will be rendered as <b>
- *CITE*
  This tag will be rendered as <i>
- *DFN*
  This tag will be rendered as <i>
- *CODE*
  This tag will be rendered as <font face="Courier New">
- *SAMP*
  This tag will be rendered as <font face="Courier New">

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- *KBD*
  This tag will be rendered as <font face="Courier New">
- *VAR*
  This tag will be rendered as <i>
- *ABBR*
  This tag will be rendered as normal text.
- *ACRONYM*
  This tag will be rendered as normal text.

Ordered and unordered lists and list Item tags

Ordered and unordered lists are rendered in an identical manner; however, ordered list items are numbered.

The report engine supports both unordered and ordered lists without attributes. The list tag attributes will be ignored in the report output. The list tag attributes are type, start, value, and compact.

Neither unordered nor ordered lists are supported in *XLSX* and *ODS* templates.

Ordered lists

An Ordered List is defined by the <OL> element. This element contains one or more <LI> elements that define the actual items of the list.

Unlike unordered lists (UL), items in an ordered list have a definite sequence. A conversion will render each item in the list with a number. All <OL> attributes will be ignored in the report output. An example of an Ordered List tag is shown in the figure below.

```
<OL>
   <LI>Apple</LI>
   <LI>Orange</LI>
   <LI>Banana</LI>
</OL>
```

As shown in the figure below, the outputs in *RTF, ODF,* or *HTML* will be:

1. Apple
2. Orange
3. Banana

Nested ordered lists

HTML conversion will indent nested lists with respect to the current level of nesting. Each level will begin at 1. An example of the Nested Ordered List tag is shown in the figure below.

```
<OL>
  <LI>Apple</LI>
  <LI>Orange</LI>
  <OL>
    <LI>Banana</LI>
    <OL>
      <LI>Grape</LI>
      <OL>
        <LI>Mango</LI>
      </OL>
    </OL>
  </OL>
</OL>
```

As shown in the figure below, the outputs in *RTF, ODF,* or *HTML* will be:

1. Apple
2. Orange
    1. Banana
        1. Grape
            1. Mango

Unordered lists

An Unordered List is defined by the UL element. This element contains one or more LI elements that define the actual list items.

A conversion will render the UL element with a bullet preceding each list item. All UL attributes will be ignored in the report output. The figure below shows an example of an Unordered List tag:

```
<UL>
  <LI>Apple</LI>
  <LI>Orange</LI>
  <LI>Banana</LI>
</UL>
```

As shown in the figure below, the outputs in *RTF, ODF*, or *HTML* will be:

- Apple
- Orange
- Banana

Nested unordered lists

Lists can also be nested. *HTML* conversion will indent nested lists with respect to the current level of nesting.

HTML conversion should attempt to present a small filled-in circle to the first level, a small circle outline to the second level, and a filled-in square to the third level. Bullets after the third level are filled-in squares. An example of the Nested Unordered List tag is shown in the figure below.

```
<UL>
  <LI>Apple</LI>
  <LI>Orange</LI>
  <UL>
    <LI>Banana</LI>
    <UL>
      <LI>Grape</LI>
      <UL>
        <LI>Mango</LI>
      </UL>
    </UL>
  </UL>
</UL>
```

As shown in the figure below, the outputs in *RTF, ODF,* or *HTML* will be:

- Apple
- Orange
  - Banana
    - Grape
      - Mango

Definition list tags

A Definition List is defined by the DL element. An entry in the list is created using the DT element for the term being defined and the DD element for the definition of the term.

A definition list can have multiple terms for a given definition, as well as multiple definitions for a given term. Authors can also give a term without a corresponding definition, and vice versa, but such a structure rarely makes sense.

A conversion will render DT as a non-indent item and DD as a single indent item. The Definition List tag is not supported in the *XLSX* template. The figure below shows an example of a Definition List tag:

```
<DL>
  <DT>Dweeb</DT>
  <DD>young excitable person who may mature into a Nerd or Geek</DD>
  <DT>Hacker</DT>
  <DD>a clever programmer</DD>
  <DT>Nerd</DT>
  <DD>technically bright but socially inept person</DD>
</DL>
```

As shown in the figure below, the outputs in *RTF, ODF* or *HTML* will be:

```
Dweeb
        young excitable person who may mature into a Nerd or Geek
Hacker
        a clever programmer
Nerd
        technically bright but socially inept person
```

Line and paragraph tags

A line break is defined by the <BR> element. This element inserts a single line break. It is an empty tag, meaning that it has no end tag. The line attributes will be ignored in the report output.

A paragraph is defined by the <P> element. The element automatically creates some space before and after itself. The paragraph attributes will be ignored in the report output. The figure below shows an example of a Line Break and Paragraph tag:

```
This is first text
This is second text <br>
This is third text
<p> This is paragraph </p>
```

As shown in the figure below, the outputs in *RTF, ODF,* or *HTML* will be:

```
This is first text This is second text

This is third text


This is paragraph
```

Preformatted text

A preformatted text is defined by the <PRE> element. All the space and carriage returns are rendered exactly as you type them. The preformatted attributes will be ignored in the report output. The figure below shows an example of the preformatted text:

```
<p>
  The PRE element tells visual
     user agents that
        the enclosed text
      is "preformatted"
</p>
<pre>
  The PRE element tells visual
     user agents that
        the enclosed text
      is "preformatted"
</pre>
```

As shown in the following two figures, the outputs in *RTF/ ODF* and *HTML* will be, respectively:

```
The PRE element tells visual user agents that the enclosed text is "preformatted"

    The PRE element tells visual
       user agents that
          the enclosed text
        is "preformatted"
```

```
This is the non pre

        This
                is
                the pre
```

Heading tags

A heading is defined by the <H1>, <H2>, <H3>, <H4>, <H5>, or <H6> element. In this report, all heading tags will be rendered as <b> for *ODT, RTF,* and *OOXML* document outputs. The heading attributes will be ignored in the report output. The figure below shows an example of Heading tags.

```
<h1>Heading 1</h1>
<h2>Heading 2</h2>
<h3>Heading 3</h3>
<h4>Heading 4</h4>
<h5>Heading 5</h5>
<h6>Heading 6</h6>
```

As shown in the following two figures, the outputs in *RTF / ODF* and *HTML* will be, respectively:

```
Heading 1
Heading 2
Heading 3
Heading 4
Heading 5
Heading 6
```

This is H1

This is H2

This is H3

This is H4

This is H5

This is H6

Link tags

A Link tag is defined by the <A> element. This element is used to create a link to another document with the href attribute. The href attribute specifies the destination of the link. Link tags are not supported in the *XLSX* template. The figure below shows an example of a Link tag:

```
<A href="http://www.google.co.th">
This is a link to www.google.com
</A>
```

As shown in the following three figures, the outputs in *RTF, ODF*, and *HTML* will be, respectively:

This is a link to www.google.com

This is a link to www.google.com

This is a link to www.google.com

Table tags

A table is defined by the <TABLE> element. A table consists of multi-dimensional data arranged in rows and columns.

Table elements

The <TABLE> element takes a number of optional attributes to provide presentational alternatives in a document. The table attributes will be ignored in the report output, except for the following attributes:

- border

  Specifies the width in unit of the border of a table.

- bgcolor

  Specifies the table background color. Applying the background color here will affect the whole table.

Table elements are not supported in *XLSX, PPTX, ODS,* and *ODP* templates.

The figure below shows an example of table tags:

```
<TABLE border="1" bgcolor="Silver">
    <TR>
        <TH>Abbreviation</TH>
        <TH>Long Form</TH>
    </TR>
    <TR>
        <TD>AFAIK</TD>
        <TD>As Far As I Know</TD>
    </TR>
    <TR>
        <TD>IMHO</TD>
        <TD>In My Humble Opinion</TD>
    </TR>
    <TR>
        <TD>OTOH</TD>
        <TD>On The Other Hand</TD>
    </TR>
</TABLE>
```

As shown in the following two figures, the outputs in *RTF / ODF* and *HTML* will be, respectively:

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Abbreviation	Long Form
AFAIK	As Far As I Know
IMHO	In My Humble Opinion
OTOH	On The Other Hand

Abbreviation	Long Form
AFAIK	As Far As I Know
IMHO	In My Humble Opinion
OTOH	On The Other Hand

Border

Border width in *HTML* is specified in pixels. When the table attributes are converted into *RTF*, *ODF*, or *OOXML*, 1 pixel will be equal to 1 pt.

Color

The attribute value type "bgcolor" refers to color definitions as specified in [SRGB]. A color value may be either a hexadecimal number (prefixed by a hash mark) or one of the following sixteen colors. Colors are case-insensitive.

The following table lists common Table Element Colors.

Color	**Hexadecimal Code**
Black	#00000000
Silver	#C0C0C0
Gray	#808080
White	#FFFFFF
Maroon	#800000
Red	#FF0000
Purple	#800080
Fuchsia	#FF00FF
Green	#008000
Lime	#00FF00
Olive	#808000

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Color	Hexadecimal Code
Yellow	#FFFF00
Navy	#000080
Blue	#0000FF
Teal	#008080
Aqua	#00FFFF

Row elements

The <TR> elements act as a container for a row of table cells. The <TR> elements must be contained within a <TABLE>.

<TR> contains <TH> or <TD> elements, which in turn contain the actual data of the table. <TR> takes presentational attributes for specifying the alignment of cells within the row and the row's background color. The row attributes will be ignored in the report output, except for the following attributes:

- align - Specifies the horizontal alignment for each cell in a row.
- valign - Specifies the vertical position of a cell's content.
- bgcolor - Specifies the table background color. A background color will apply to rows only (see Color in Table Elements for more details).

Row elements are not supported in *XLSX, PPTX, ODS*, and *ODP* templates.

Align

This attribute specifies the alignment of data and the justification of text in a cell. Possible values are:

- left - Left-flushed data/Left-justified text. This is the default value for table data.
- center - Centered data/Center-justified text. This is the default value for table headers.
- right - Right-flushed data/Right-justified text.
- justify - Double -justified text.
- char - No text alignment set.

Valign

This attribute specifies the vertical position of data within a cell. Possible values are:

- top: Cell data is flush with the top of a cell.
- middle: Cell data is centered vertically within a cell. This is the default value.
- bottom: Cell data is flush with the bottom of a cell.
- baseline: No text alignment set.

The figure below shows an example of TR tags:

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

```
<TABLE border="1">
    <TR align="center">
        <TD>This is center tr</TD>
        <TD>This center tr</TD>
    </TR>
    <TR bgcolor="Gray">
        <TD>This is gray tr</TD>
        <TD>This gray tr</TD>
    </TR>
    <TR valign="bottom">
        <TD>This is bottom tr<br>This is bottom tr</TD>
        <TD>This bottom tr</TD>
    </TR>
    <TR>
        <TD>This is normal tr, This is normal tr</TD>
        <TD>This is normal tr, This is normal tr</TD>
    </TR>
</TABLE>
```

As shown in the following two figures, the outputs in *RTF / ODF* and *HTML* will be, respectively:

This is center tr	This center tr
This is gray tr	This gray tr
This is bottom tr This is bottom tr	This bottom tr
This is normal tr, This is normal tr	This is normal tr, This is normal tr

This is center tr	This center tr
This is gray tr	This gray tr
This is bottom tr This is bottom tr	This bottom tr
This is normal tr, This is normal tr	This is normal tr, This is normal tr

Cell elements

The <TD> elements define a data cell in a table. <TD> elements are contained within a <TR> element (a table row). The cell attributes will be ignored in the report output, except for the following attributes:

- align
  specifies the horizontal alignment for each cell in the row. See Align in Row Elements for more details.
- valign
  specifies the vertical position of a cell's contents. See Valign in Row Elements for more details.

- bgcolor
  specifies the table background color. A background color will apply only to cells. See Color in Table Elements for more details.
- rowspan
  rows spanned by the cell
- colspan
  columns spanned by the cell

Cell elements are not supported in *XLSX, PPTX, ODS,* and *ODP* templates.

Row span

This attribute specifies the number of rows spanned by the current cell. The default value of this attribute is one ("1"). For an *RTF* output, the result of row span (**.rtf*) is readable only in Word on Mac, and MS Word.

Column span

This attribute specifies the number of columns spanned by the current cell. The default value of this attribute is one ("1"). The figure below shows an example of a column span:

```
<TABLE border="1">
    <TR>
        <TD align="right">This is align right<br>This is align right</TD>
        <TD align="center">This is align center</TD>
        <TD valign="top">This is valign top</TD>
        <TD valign="bottom">This is valign bottom</TD>
        <TD>This is normal td</TD>
    </TR>
    <TR>
        <TD>This is normal td</TD>
        <TD colspan="2">This is colspan</TD>
        <TD rowspan="2">This is rowspan</TD>
        <TD>This is normal td</TD>
    </TR>
    <TR>
        <TD bgcolor="red">This is red td</TD>
        <TD>This is normal td, This is normal td</TD>
        <TD>This is normal td, This is normal td</TD>
        <TD>This is normal td</TD>
    </TR>
</TABLE>
```

As shown in the following two figures, the outputs in *RTF / ODF* and *HTML* will be, respectively:

This is align right This is align right	This is align center	This is valign top	This is valign bottom	This is normal td
This is normal td	This is colspan		This is rowspan	This is normal td
This is red td	This is normal td, This is normal td	This is normal td, This is normal td		This is normal td

This is align right This is align right	This is align center	This is valign top	This is valign bottom	This is normal td
This is normal td	This is colspan		This is rowspan	This is normal td
This is red td	This is normal td, This is normal td	This is normal td, This is normal td		This is normal td

Column width

Table column width can be adjusted via attributes and stylesheet. For example, a column's width can be set by its width attribute. Percentage or pixel units are allowed for each <TD> element. You may set table width in <table> tags by absolute pixel value or percentage of page ratio, then allocate width from the table to each column later.

For example:

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

```
Using attributes:
<table width="500px">
        <tr>
                <td width="25%">25%</td>
                <td width="75%">75%</td>
       </tr>
</table>
<table width="500px">
        <tr>
                <td width="400px">400px</td>
                <td width="100px">100px</td>
        </tr>
</table>


<table>
    <tr>
        <td width="30">30px</td>
        <td width="60">60px</td>
        <td width="90">90px</td>
    </tr>
</table>

<table>
    <tr>
        <td width="25%">25%</td>
        <td width="75%">75%</td>
    </tr>
</table>



Using Stylesheet:
<table style="width:50%">

    <tr>
        <td style="width:70%">70%</td>
        <td style="width:30%">30%</td>
    </tr>
</table>
```

DOCX, RTF and ODT templates retain their width values. The column width in the report is the same as defined in the Advanced HTML Editor dialog.

If the defined values exceed the paper margins, the table column widths will be converted to a percentage format so that the table appears within the page margins.

Header elements

The <TH> elements define a header cell in a table. <TH> elements are contained within a <TR> element (a table row). The header attributes will be ignored in the report output, except for the following attributes:

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- align
  specifies the horizontal alignment for each cell in the row. See Align in Row Elements for more details.
- valign
  specifies the vertical position of a cell's contents. See Valign in Row Elements for more details.
- bgcolor
  specifies the table background color. A background color will apply to the whole table. See Color in Table Elements for more details.
- rowspan
  rows spanned by the cell. See Row span in Cell Elements for more details.
- colspan
  columns spanned by the cell. See Column span in Cell Elements for more details.

The default alignment for <TH> is center and the default font style for <TH> is bold. Header elements are not supported in *XLSX, PPTX, ODS,* and *ODP* templates. The figure below shows an example of header elements:

```
<TABLE border="1" bgcolor="Silver">
    <TR>
        <TH>Name</TH>
        <TH>Cups</TH>
        <TH>Type of Coffee</TH>
        <TH>Sugar?</TH>
    </TR>
    <TR>
        <TD>T. Sexton</TD>
        <TD>10</TD>
        <TD>Espresso</TD>
        <TD>No</TD>
    </TR>
    <TR>
        <TD>J. Dinnen</TD>
        <TD>5</TD>
        <TD>Decaf</TD>
        <TD>Yes</TD>
    </TR>
</TABLE>
```

The following two figures show the outputs in *RTF / ODF* and *HTML,* respectively.

Name	Cups	Type of Coffee	Sugar?
T. Sexton	10	Espresso	No
J. Dinnen	5	Decaf	Yes

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Name	Cups	Type of Coffee	Sugar?
T. Sexton	10	Espresso	No
J. Dinnen	5	Decaf	Yes

Image tags

The <img> tag embeds an image in a document. The <img> tag consists of four attributes: **src**, **width**, **height**, and **alt**.

src

The src attribute specifies the location of an image resource. The value of this attribute can be one of the following types:

- A URL. The recognized scheme types are *HTTP, HTTPS*, and *FILE*
- An absolute path such as *c:/user/image.png*. The path separator can be either /(slash) or \ (blackslash)

The output image format will depend on the format of the image source.

width

The width attribute specifies the width of an image in pixel units. For example, width = “100” or width = “100px”

height

The **height** attribute specifies the height of an image in pixel units. For example, height = “100”, or height = “100px”

> **Note**
> If the width or height attribute of an image is not specified, the size of the image will be calculated according to the following rules:
> - For an image file that contains the width and height properties, such as *JPG, PNG,* and *GIF*, the size of the image output will be calculated from the size of the image.
> - For an image file that has no width and height properties, such as *SVG, EMF,* and *WMF*, the size of the image output will be calculated from the size of the paper.

alt

The **alt** attribute specifies the alternate text of an image in case the image cannot be displayed.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

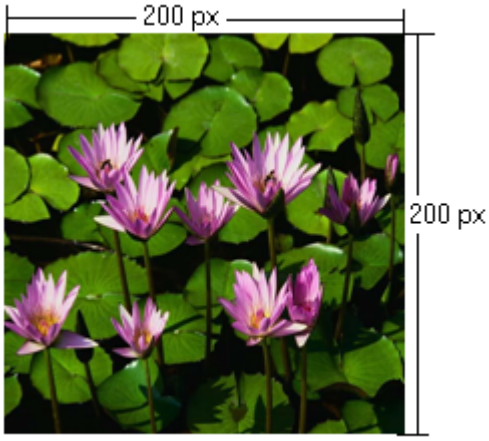
Note

- The alternate text of image tags in DOCX, XLSX, and PPTX is available only in Microsoft Office.
- The alternate text of image tags in RTF files is shown only in the Description field, rather than the Title field in Microsoft Word and OpenOffice Writer.
- The alternate text of image tags in RTF files is not supported in LibreOffice.

In the event that the HTML code is as follows:

```
<img src="file:/D:/Waterlilies.jpg" width="200" height="200" alt="Waterlilies"/>
```

The image will appear as shown in the figure below.



Superscript and subscript tags

Superscript

The <sup> tag defines a supersript, which is a very small letter, figure, or symbol, and printed above the line. Superscript characters can be used to write footnotes, like www[1], for example:

```
<html>
    <body>
        This is text contains <sup>superscript</sup> text.
    </body>
</html>
```

This is text contains ^{superscript} text.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Subscript

The <sub> tag defines a subscript, which is a very small letter, figure, or symbol, and printed below the line. For example, subscript characters can be used to write chemical formulas, such as H2O:

```
<html>
    <body>
        This is text contains <sub>subscript</sub> text.
    </body>
</html>
```

This is text contains $_{\text{subscript}}$ text.

Supported CSS

On this page

- Background(see page 2007)
  - Color Specification(see page 2007)
  - Supported Tags(see page 2008)
- Border(see page 2009)
  - Border Width(see page 2009)
  - Length Specification(see page 2009)
  - Border Color(see page 2009)
  - Border Style(see page 2010)
  - Supported Tags(see page 2010)
- Margin(see page 2011)
  - Supported Tags(see page 2011)
- Padding(see page 2013)
  - Supported Tags(see page 2013)
- Color(see page 2014)
  - Supported Tags(see page 2014)
- Display(see page 2014)
  - Supported Tags(see page 2014)
- Font(see page 2015)
  - Font Family(see page 2015)
  - Font Style(see page 2015)
  - Font Variant(see page 2015)
  - Font Weight(see page 2015)
  - Font size(see page 2015)
  - Supported Tags(see page 2015)
- Text Align(see page 2016)
  - Supported Tags(see page 2016)
- Text Transform(see page 2016)
  - Supported Tags(see page 2017)
- White-Space(see page 2017)
  - Supported Tags(see page 2017)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Width(see page 2018)
- Text Decoration(see page 2019)
    - Supported Tags(see page 2019)
- Vertical Align(see page 2020)

Background

The shorthand property for the background property is 'background', which is used for setting an individual background style. Such attribute can be used in *RTF, ODT,* and *DOCX* report templates.

Color Specification

Color attributes such as background-color, can be specified by either a keyword or a numerical RGB specification (hexadecimal, integer, and percentage value). The list of Color Keywords and Corresponding RGB Specification is displayed in the table below.

Color	RGB Specification Hexadecimal Code	RGB Specification Integer	RGB Specification Percentage
Maroon	#800000	RGB(128,0,0)	RGB(50%,0%,0%)
Red	#FF0000	RGB(255,0,0)	RGB(100%,0%,0%)
Orange	#FFA500	RGB(255,165,0)	RGB(100%,65%,0%)
Yellow	#FFFF00	RGB(255,255,0)	RGB(100%,100%,0%)
Olive	#808000	RGB(128,128,0)	RGB(50%,50%,0%)
Purple	#800080	RGB(128,0,128)	RGB(50%,0%,50%)
Fuchsia	#FF00FF	RGB(255,0,255)	RGB(100%,0%,100%)
White	#FFFFFF	RGB(255,255,255)	RGB(100%,100%,100%)
Lime	#00FF00	RGB(0,255,0)	RGB(0%,100%,0%)
Green	#008000	RGB(0,128,0)	RGB(0%,50%,0%)
Navy	#000080	RGB(0,0,128)	RGB(0%,0%,50%)
Blue	#0000FF	RGB(0,0,255)	RGB(0%,0%,100%)
Aqua	#00FFFF	RGB(0,255,255)	RGB(0%,100%,100%)
Teal	#008080	RGB(0,128,128)	RGB(0%,50%,50%)
Black	#000000	RGB(0,0,0)	RGB(0%,0%,0%)
Silver	#C0C0C0	RGB(192,192,192)	RGB(75%,75%,75%)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Color	RGB Specification Hexadecimal Code	RGB Specification Integer	RGB Specification Percentage
Gray	#808080	RGB(128,128,128)	RGB(50%,50%,50%)

The format of an RGB specification value in form of the hexadecimal notation is a '#' immediately followed by either three or six hexadecimal characters. The three-digit RGB notation (#rgb) is converted to the six-digit form (#rrggbb) by replicating digits. For example, #FB0 is converted to #FFBB00.

For example:

```
<p style = “background-color:#FB0”>
<p style = “background-color:#FFBB00”>
```

The RGB specification Integer and Percentage form is "rgb" followed by RGB values in the same format in brackets. You cannot use both integer and percentage at the same time. In the case that an RGB value is more than a maximum value (255 or 100%) or less than a minimum (0 or 0%), the value will be set to the maximum or minimum value instead, for example:

```
<p style = "background-color:rgb(-100,500,0)">

<p style = "background-color:rgb(0,255,0)">

<p style = "background-colr:rgb(-100%,200%,0%)">

<p style = "background-colr:rgb(0%,100%,0%)">
```

The color used in all of the above examples is lime.

Supported Tags

The ‘background-color’ property can be used in the following tags:

```
b, i, u, h1, h2, h3, h4, h5, h6, tt, code, samp, kbd, pre, big, small, strike,
s, em, cite, dfn, var, strong, font, a, dl, dt, dd, ul, ol, li, table, tr, td,
th, p, div, span, sup, and sub.
```

For example:

```
background color : <span style="background-color:red;">red </span>
<span style="background-color:#008000;">green </span>
<span style="background-color:blue;">blue </span>
<span style="background-color:#FF8040;">orange </span>
<span style="background-color:yellow;">yellow </span>
```

background color : red green blue orange yellow

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Border

The shorthand property for the border property is 'border', which is used for setting the same width, color, and style of all four borders on four sides of a box. Border can be used in *RTF, ODT*, and *DOCX* report templates.

Border Width

Border width properties specify the width of border areas. The value of the border property is length. The border width properties that can be used with a longhand property are as follows:

- `'border-top-width'`
- `'border-right-width'`
- `'border-bottom-width'`
- `'border-left-width'`

Length Specification

The format of a length value is a real number immediately followed by a unit identifier. There are two types of length units: relative and absolute. Relative length units specify a length relative to another length property of the displaying device.

Relative units are:

- **px**
  pixels, relative to the displaying device.

Absolute units are:

- **in**
  inches, 1 inch is equal to 2.54 centimeters.
- **cm**
  centimeters.
- **mm**
  millimeters.
- **pt**
  points - the points used by CSS 2.1 are equal to 1/72nd of an inch.
- **pc**
  picas - 1 pica is equal to 12 points.

Border Color

Border color properties specify the border colors of a box. The value of the border color has two meanings:

- **Color**
  Specifies a border color. See Color Specification(see page 2292) for more details on color.
- **Transparent**
  Referred to as a transparent border.

The border color properties that can be used with a longhand property are as follows:

- `'border-top-color'`

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- `'border-right-color'`
- `'border-bottom-color'`
- `'border-left-color'`

Border Style

The Border style properties specify the style of a box's border lines. The border style may take one of the following 10 values:

1. **none**
   This value will create no border because the computed border width is zero.
2. **hidden**
   The same as 'none'.
3. **dotted**
   This value will create a dotted border.
4. **dashed**
   This value will create a dashed border.
5. **solid**
   This value will create a single line segment border.
6. **double**
   This value will create a two solid line border.
7. **groove**
   This value will create a border that looks as though it were carved into the rendering surface.
8. **ridge**
   The opposite of 'groove'. This value will create a border that looks as though it were coming out of the rendering surface.
9. **inset**
   This value will create a border that looks as though it were embedded in the rendering surface.
10. **outset**
    The opposite of 'inset'. This value will create a border that looks as though it were coming out of the rendering surface.

- The supported border values for *ODF* documents are **none**, **hidden**, **solid**, and **double**.
- The supported border values for *RTF* documents are **none**, **hidden**,**dotted**, **dashed**, **solid**, and **double**.
- The supported border values for *DOCX* documents are **none**, **hidden**,**dotted**, **dashed**, **solid**, **double**, **outset**, and **inset** (**groove** and **ridge** will be rendered as solid).

The border style properties that can be used with a longhand property are as follows:

- `'border-top-style'`
- `'border-right-style'`
- `'border-bottom-style'`
- `'border-left-style'`

Supported Tags

Border properties can be used in the following tags:

`h1, h2, h3, h4, h5, h6, td, th, p, and div.`

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

For example, if you want to have a red border, type the following line of code, see the figure below.

```
<p style="border:2px solid red;">This is 2px solid red border.</p>
```

The result of using the above example will be as shown in the figure below.

This border is 2px solid red.

Margin

Margin properties specify the width of margin area within a box. The 'margin' property is a shorthand property for setting the margin of all four sides while the other margin properties only set their respective side. The value of the margin property is length (See Length Specification for more details). Margin can be used in *RTF, ODT, ODP, DOCX* and *PPTX* report templates.

The margin properties that can be used with a longhand property are as follows:

- `'margin-top'`
- `'margin-right'`
- `'margin-bottom'`
- `'margin-left'`

Supported Tags

The supported tags for margins are as follows:

`h1, h2, h3, h4, h5, h6, table (supported only in RTF, ODT, and DOCX), p, and div.`

For example, if you want to set a specific margin width for your cells, type the lines of code as shown in the figure below.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

```
<p>This is first paragraph.</p>

<table width="100%" cellpadding="0" style="margin-top:30px;margin-right:30px;
margin-left:30px;margin-bottom:30px;" cellspacing="0" border="1">
    <tr>
      <td>This is cell1</td>
      <td>This is cell2</td>
    </tr>
    <tr>
      <td>This is cell3</td>
      <td>This is cell4</td>
    </tr>
 </table>

 <p>This is second paragraph.</p>

 <table width="100%" cellpadding="0" style="margin-top:50px;margin-right:50px;
 margin-left:50px;margin-bottom:50px;" cellspacing="0" border="1">
    <tr>
      <td>This is cell1</td>
      <td>This is cell2</td>
    </tr>
    <tr>
      <td>This is cell3</td>
      <td>This is cell4</td>
    </tr>
 </table>

  <p>This is third paragraph.</p>
```

The result of using the above example will be as indicated in the figure below.

This is first paragraph.

This is cell1	This is cell2
This is cell3	This is cell4

This is second paragraph.

This is cell1	This is cell2
This is cell3	This is cell4

This is third paragraph.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company. 2012

Padding

Padding properties specify the width of padding area within a box. The 'padding' property is a shorthand property for setting the padding for all four sides while the other padding properties only set their respective side. The value of the padding property is length (See Length Specification for more details) adding can be used in *RTF, ODT* and *DOCX* report templates.

The padding properties that can be used with a longhand property are as follows:

- 'padding-top'
- 'padding-right'
- 'padding-bottom'
- 'padding-left'

Supported Tags

The supported tags for padding are as follows:

`h1, h2, h3, h4, h5, h6, td, th, p, and div.`

- The supported border values of table tags for *RTF* documents are margin-left and margin-right.for more details.
- The supported border values of table tags for *DOCX* documents are margin-left and margin-right.
- The supported border values of all tags for *PPTX* documents are margin-left, margin-top and margin-bottom.

For example, if you want to create specific padding styles for your cells, type the lines of code shown in the figure below.

```
<table width="100%"  border="1">
     <tr>
       <td  style="padding-bottom:20px;padding-top:20px;padding-left:20px;padding-right:20px;">This is cell1</td>
       <td>This is cell2</td>
    </tr>
    <tr>
       <td>This is cell3</td>
       <td>This is cell4</td>
    </tr>
</table>
```

The result of using the above example will be as shown in the figure below.

This is cell1	This is cell2
This is cell3	This is cell4

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Color

The 'color' property specifies the foreground color of an element's text contents. Only color values are allowed in this property (See Color Specification for more details on color).

Supported Tags

The supported tags for colors are as follows:

```
b, i, u, h1, h2, h3, h4, h5, h6, tt, code, samp, kbd, pre, big, small, strike,
s, em, cite, dfn, var, strong, font, a, dl, dt, dd, ul, ol, li, table, tr, td,
th, p, div, span, sup, and sub.
```

For example, if you want to have a green text color, type the lines of code indicated in figure.

```
<p style="color:green;">This text color is green.</p>
<span style="color:#800080;">This text color is purple. </span>
<span style="color:red;">This text color is red.</span>
```

The result of using the above example will be as shown in the figure below.

This text color is green.

This text color is purple. This text color is red.

Display

The 'display' property specifies how text will be displayed. The possible display values are:

- **block**: This value will make an element appear in the output report.
- **none**: This value will make an element disappear from the output report.

Supported Tags

The supported tags for the display property are:

```
br, b, i, u, h1, h2, h3, h4, h5, h6, tt, code, samp, kbd, pre, big, small,
strike, s, em, cite, dfn, var, strong, font, a, dl, dt, dd, ul, ol, li
(supported only in RTF, DOCX, and PPTX), table, tr, p, div, span, sup, and sub.
```

For example, if you want to display an element but do not want to display the other(s), type the lines of code as shown in the figure below.

```
<p style="display:none;">This paragraph is none display.</p>
<p style="display:block;">This paragraph is block display.</p>
```

The result of using the above example will be as indicated below.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company. 2014

This paragraph is block display.

Font

The 'font' property is a shorthand property for setting, except the following:

- `'font family'`
- `'font-style'`
- `'font-variant'`
- `'font-weight'`
- `'font-size'`

Font Family

Specify a prioritized list of font family names and/or generic family names.

Font Style

Specify either normal or italic face (within the specified font family).

Font Variant

Specify either normal or small-caps of variant (within the specified font family). Font variant is not supported in *XLSX* report templates.

Font Weight

Specifies the weight of the font. The following values are defined:

- **normal**, **lighter**, **100**, **200**, **300**, and **400** will be rendered as '**normal**'.
- **bold**, **bolder**, **500**, **600**, **700**, **800**, and **900** will be rendered as '**bold**'.

Font size

Specify the font size. Possible values include xx-small, x-small, small, medium, large, xx-large, and the integer number from 1 to 7.

Supported Tags

The supported tags for the font property are as follows:

`b, i, u, h1, h2, h3, h4, h5, h6, tt, code, samp, kbd, pre, big, small, strike, s, em, cite, dfn, var, strong, font, a, dl, dt, dd, ul, ol, li, table, tr, td, th, p, div, span, sup, and sub.`

For example, if you want to have bold fonts in small caps, type the lines of code as displayed in the figure below.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

```
<font style="font-variant:small-caps;font-weight:bold;">
This font is bold and small-caps</font>
```

The result of using the above example will be as indicated in the figure below.

THIS FONT IS BOLD AND SMALL-CAPS

Text Align

The 'text align' property describes how inline content of a block is aligned. Possible values of the text align property include: **left**, **right**, **center** and **justify**. The 'text align' property can be used in *RTF, ODT, ODP, DOCX,* and *PPTX* report templates.

Supported Tags

The supported tags for the text align property are as follows:

`table, tr, td, th, p, and div.`

For example, if you want to align your paragraph to the left, center, and right, type the lines of code as shown in the Figure below.

```
<p style="text-align:right;">This paragraph alignment is right.</p>
<p style="text-align:left;">This paragraph alignment is left.</p>
<p style="text-align:center;">This paragraph alignment is center.</p>
```

The result of using the above example will be as displayed in the figure below.

This paragraph alignment is right.

This paragraph alignment is left.

This paragraph alignment is center.

Text Transform

The 'text transform' property controls capitalization effects of an element's text. Possible values of the 'text transform' property include:

1. **capitalize**
   This will write the first character in an uppercase letter. Other characters will not be affected.
2. **uppercase**
   This will write each word in upper case letters.
3. **lowercase**
   This will write each word in lower case letters.
4. **none**
   This will not create any capitalization effects.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company. 2016

Supported Tags

The supported tags for the text transform property are as follows:

```
b, i, u, h1, h2, h3, h4, h5, h6, tt, code, samp, kbd, pre, big, small, strike,
s, em, cite, dfn, var, strong, font, a, dl, dt, dd, ul, ol, li, table, tr, td,
th, p, div, span, sup, and sub.
```

For example, if you want to make all letters in a paragraph become uppercases, type the line of code displayed in the figure below.

```
<p style="text-transform:uppercase;">This is paragraph uppercase.</p>
```

The result of using the above example will be as shown in the figure below.

THIS IS PARAGRAPH UPPERCASE.

White-Space

The 'white-space' property affects the vertical position of white-space inside a lined box. White-space can be used in *RTF, ODT, DOCX* and *PPTX* report templates. Possible values of the 'white-space' property include:

1. **normal**
   To collapse sequences of white-space and break lines as necessary to fill line boxes.
2. **pre**
   To prevent from collapsing sequences of white-space. Lines are only broken at newlines in the source or at occurrences of "\A" in the generated content. Report Wizard will render this value as the <pre> tag.
3. **nowrap**
   To collapse white-space as for 'normal' but suppressing line breaks within text.
4. **pre-wrap**
   To prevent user agents from collapsing sequences of white-space. Lines are broken at newlines in the source, at occurrences of "\A" in the generated content, and as necessary to fill line boxes. This value will not be rendered as pre.
5. **pre-line**
   To direct user agents to collapse sequences of white-space. Lines are broken at newlines in the source, at occurrences of "\A" in the generated content, and as necessary to fill line boxes. This value will not be rendered as pre.

Supported Tags

The supported tags for the 'white-space' property are as follows:

```
b, i, u, h1, h2, h3, h4, h5, h6, tt, code, samp, kbd, pre, big, small, strike,
s, em, cite, dfn, var, strong, font, a, table, tr, td, th, p, div, span, sup,
and sub.
```

For example, if you want to preserve some white-space, type these lines of code as shown in the figure below.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company. 2017

```
<p style="white-space:pre;">
 This    white space        is pre.
    This is        white space        pre.
</p>
```

The result of using the above example will be as indicated in the figure below.

This white space is pre.

This is white space pre.

Width

The 'width' property specifies the contents width of each box generated by a block-level. The value of the width property is length (See Length Specification(see page 2006) for more details). Width can be used in *RTF, ODT,* and *DOCX* report templates.

The supported tag for the width property includes table.

For example, if you want to have a specific table width, type the lines of code as described in the figure below.

```
<table border=1 cellpadding="0" style="width:300px;" cellspacing="0">
  <tr>
     <td>This is cell1</td>
     <td>This is cell2</td>
  </tr>
  <tr>
     <td>This is cell3</td>
     <td>This is cell4</td>
  </tr>
</table>
<br>
<table style="width:6in;" border="1" cellpadding="0" cellspacing="0">
  <tr>
     <td>This is cell1</td>
     <td>This is cell2</td>
  </tr>
  <tr>
     <td>This is cell3</td>
     <td>This is cell4</td>
  </tr>
</table>
```

The result of using the above example will be as shown in the figure below.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

This is cell1	This is cell2
This is cell3	This is cell4

This is cell1	This is cell2
This is cell3	This is cell4

Text Decoration

The 'text decoration' property describes decorations that are added to text. The possible values of the text decoration property are as follows:

- **none**
  To produce no text decoration.
- **underline**
  To underline each line of text.
- **overline**
  To add a line above each line of text.
- **line-through**
  To add a line that strikes through each line of text.
- **blink**
  To add the blinking effect to text (being visible and invisible alternatively). The blink text decoration supports only the *ODT* report template.

> Report Wizard does not currently support the overline text decoration.

Supported Tags

The supported tags for the text decoration property are:

```
b, i, u, h1, h2, h3, h4, h5, h6, tt, code, samp, kbd, pre, big, small, strike,
s, em, cite, dfn, var, strong, font, a, dl, dt, dd, ul, ol, li, table, tr, td,
th, p, div, span, sup, and sub.
```

For example, if you want to create a line-through text, type the line of code as displayed in the figure below.

```
This word is <span style="text-decoration:line-through;">line through.</span>
```

The result of using the above example will be as indicated in the figure below.

This word is ~~line through.~~

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company. 2019

Vertical Align

The 'vertical align' property affects the vertical position of an element inside a lined box. This property can be used in *RTF, ODT* and *DOCX* report templates. Possible values of the vertical align property include:

- **baseline**
  This value will align the baseline of a box with the baseline of the parent box. Leave the vertical alignment to be default when you encounter this value.
- **middle**
  This value will align the vertical midpoint of a box.
- **top**
  This value will align the top of the aligned sub-tree with the top of a line box.
- **bottom**
  This value will align the bottom of the aligned sub-tree with the bottom of a line box.

The supported tags for the vertical align property are: table, tr, td, and th, for example, see figure below:

```
<table width="100%" cellpadding="0" style="vertical-align:middle;" cellspacing="0" border=1>
  <tr>
    <td>
     This is cell1
     This is cell1
    </td>
    <td>
     This vertical align is middle.
    </td>
 </tr>
 <tr>
    <td>
     This is cell3
     This is cell3
    </td>
    <td>
     This vertical align is middle.
    </td>
 </tr>
</table>
```

The result of using the above example will be as shown in the figure below.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company. 2020

This is cell1 This is cell1	This vertical align is middle.
This is cell3 This is cell3	This vertical align is middle.

Appendix E: DocBook Support

DocBook was built as an application to write technical documents related to computer hardware and software. It is a semantic markup language that was extended from XML language. As an open source application, DocBook has been adopted as a standard to write other types of documentation as well. The current version of DocBook, published by O'Reilly Media and XML Press, is 5.1.

The Report Wizard's DocBook engine enables you to input the *VTL* codes inside DocBook's document content. It also allows you to convert HTML into DocBook tags, rearrange tags, and escape special characters. Report Wizard uses "*dbk*" and "*docbook*" file extensions to save DocBook files. The figure below shows the DocBook's document content.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company. 2021

```
<?xml version="1.0" encoding="UTF-8"?>
<book xmlns="http://docbook.org/ns/docbook"
    xmlns:xlink="http://www.w3.org/1999/xlink" version="5.0">
    <info>
        <title>Sample Book</title>
        <author>
            <orgname>No Magic Asia</orgname>
            <address>
                <city>Bangkok</city>
                <street>Rama 9</street>
                <postcode>10310</postcode>
                <country>Thailand</country>
            </address>
            <email>sample@nomagicasia.com</email>
        </author>
    </info>
    <part>
        <title>List of Classes</title>
        <subtitle>Display all classed in the project.</subtitle>
        <chapter>
            <title>Class name and documentation</title>
            <subtitle>Display name and documentation of classes</subtitle>
            #foreach($c in $Class)
            <sect1>
                <title>$c.name</title>
                <subtitle></subtitle>
                <para>$c.documentation</para>
            </sect1>
            #end
        </chapter>
    </part>
</book>
```

Related pages

The Supported HTML Tags

HTML Paragraph Elements

The HTML tags in the Report Wizard's engine for DocBook defines how a paragraph is arranged in a DocBook document. There are two categories of paragraph elements:

1.1.1 Paragraph
1.1.2 Preformatted Text

Paragraph

The Paragraph element (<p>) defines a paragraph. The <p> element shown in the first figure will be rendered in a DocBook format in the second figure below.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company. 2022

```
<p>This is my first paragraph.</p>
```

```
<para>This is my first paragraph.</para>
```

Preformatted Text

The Preformatted text element (<pre>) displays the text exactly as you expected it to. The <pre> tag preserves both spaces and line breaks. The <pre> element shown in the first figure below will be rendered in a DocBook format in the second figure below.

```
<pre>This is my first paragraph.
        This is my second paragraph.</pre>
```

```
<literallayout>This is my first paragraph.
        This is my second paragraph.</literallayout>
```

HTML Font Styles

Report Wizard specifies a group of HTML tags that you can use to define how fonts will be displayed on rendered text. They are called font style elements.

1. Teletype Text
2. Italics
3. Bold
4. Strikethrough
5. Underline

Teletype Text

The HTML Teletype Text element (<tt>) creates teletype or monospaced text. The <tt> element shown in the first figure below will be rendered in a DocBook format as in the second figure below.

```
<tt>teletype or monospaced</tt>
```

```
<code>teletype or monospaced</code>
```

Italics

The HTML Italics element (<i>) renders text slanted to the right. The <i> element shown in the first figure below will be rendered in a DocBook format as in the second figure below.

```
<i>italic</i>
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company. 2023

```
<emphasis role="italic">italic</emphasis>
```

Bold

The HTML Bold element (<b>) renders bold text. The <b> element shown in the first figure below will be rendered in a DocBook format as in the second figure below.

```
<b>bold</b>
```

```
<emphasis role="bold">bold</emphasis>
```

Strikethrough

The HTML Strikethrough element (<strike> or <s>) renders text with a horizontal strike or a line through the middle. The <strike> element shown in the first figure below will be rendered in a DocBook format as in the second figure below.

```
<strike>strikethrough</strike>
```

```
<emphasis role="strikethrough">strikethrough</emphasis>
```

Underline

The HTML Underline element (<u>) underlines text. The underline <u> element shown in the first figure below will be rendered in a DocBook as in second figure below.

```
<u>underline</u>
```

```
<emphasis role="underline">underline</emphasis>
```

HTML Phrase Elements

There is a group of HTML tags that allows you to specify how fonts will be displayed on rendered text. They are called HTML Phrase Tags or Elements. The following is a list of these font style elements:

1. Emphasis
2. Strong
3. Citation
4. Definition
5. A Fragment of Computer Code
6. Sample Text
7. Keyboard Input
8. Variable

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company. 2024

Emphasis

The Emphasis element () emphasizes text content and displays it in italics. The element shown in the first figure below will be rendered in a DocBook format as in the second figure below.

```
<em>italic emphasis</em>
```

```
<emphasis>italic emphasis</emphasis>
```

Strong

The Strong element () strongly emphasizes text content. It renders the text in bold. The element shown in the first figure below will be rendered in a DocBook format as in the second figure below.

```
<strong>strong</strong>
```

```
<emphasis role="bold">strong</emphasis>
```

Citation

The Citation element (<cite>) presents a citation or a reference to other sources. The <cite> element shown in the first figure below will be rendered in a DocBook format as in the second figure below.

```
<cite>citation</cite>
```

```
<emphasis>citation</emphasis>
```

Definition

The HTML Definition element (<dfn>) gives a definition for the term being defined. The Definition element <dfn> shown in the first figure below will be rendered in a DocBook format as in second figure below.

```
<dfn>defining instance of the enclosed term</dfn>
```

```
<emphasis>defining instance of the enclosed term</emphasis>
```

A Fragment of Computer Code

The HTML fragment of code element (<code>) designates part of computer code. The <code> element shown in the first figure below will be rendered in a DocBook format as in the second figure below.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company. 2025

```
<code>code</code>
```

```
<code>code</code>
```

Sample Text

The HTML Sample Text element (<samp>) designates sample output of programs or scripts. The <samp> element shown in the first figure below will be rendered in a DocBook format as in the second figure below.

```
<samp>SAMP</samp>
```

```
<code>SAMP</code>
```

Keyboard Input

The HTML Keyboard Input element (<kbd>) indicates user input or text to be entered by the user. The <kbd> element shown in the first figure below will be rendered in a DocBook format as in the second figure below.

```
<kbd>text to be entered by the user</kbd>
```

```
<code>text to be entered by the user</code>
```

Variable

The HTML Variable element (<var>) indicates an instance of a variable or program argument. The <var> element shown in the first figure below will be rendered in a DocBook format as in the second figure below.

```
<var>variable or program argument</var>
```

```
<emphasis>variable or program argument</emphasis>
```

HTML Link Element

The HTML Link element (<a>) defines a hyperlink that links one page to another. The <a> element supports the attribute href, which specifies the URL of a page the hyperlink goes to.

The <a> element shown in the first figure below will be rendered in a DocBook format as in the second figure below.

```
<a href="www.google.co.th">www.google.co.th</a>
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company. 2026

```
<link xlink:href="www.google.co.th">www.google.co.th</link>
```

HTML Image Element

The HTML Image element (<img>) defines an image in an HTML page. The <img> element supports the attribute src, which specifies the URL of an image.
The <img> element shown in the first figure below will be rendered in a DocBook format as in the second figure below.

```
<img src="files/pic.png"/>
```

```
<inlinemediaobject>
    <imageobject>
        <imagedata fileref="files/pic.png"/>
    </imageobject>
</inlinemediaobject>
```

The supported HTML lists

The DocBook engine of Report Wizard supports the following HTML lists:

1. Unordered Lists
2. Ordered Lists
3. Definition Lists

Unordered Lists

An unordered list is defined by the <ul> element. An unordered list element <ul> contains one or more <li> elements that define the actual items of the list in bullets. The Unordered List element shown in the first figure below will be rendered in a DocBook format as in the second figure below.

```
<ul>
    <li>Fruit</li>
    <ul>
        <li>Apple</li>
        <li>Orange</li>
    </ul>
    <li>Vegetable</li>
    <ul>
        <li>Cucumber</li>
        <li>Lettuce</li>
    </ul>
</ul>
```

```
<itemizedlist>
    <listitem>
        <para>Fruit</para>
        <itemizedlist>
            <listitem>
                <para>Apple</para>
            </listitem>
            <listitem>
                <para>Orange</para>
            </listitem>
        </itemizedlist>
    </listitem>
    <listitem>
        <para>Vegetable</para>
        <itemizedlist>
            <listitem>
                <para>Cucumber</para>
            </listitem>
            <listitem>
                <para>Lettuce</para>
            </listitem>
        </itemizedlist>
    </listitem>
</itemizedlist>
```

Ordered Lists

An ordered list is defined by the <ol> element. The ordered list element <ol> contains one or more <li> elements that define the actual items of the list preceded by numbers. The unordered list element shown in the first figure below will be rendered in a DocBook format as in the second figure below.

```
<ol>
    <li>Fruit</li>
    <ol>
        <li>Apple</li>
        <li>Orange</li>
    </ol>
    <li>Vegetable</li>
    <ol>
        <li>Cucumber</li>
        <li>Lettuce</li>
    </ol>
</ol>
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

```
<orderedlist>
    <listitem>
        <para>Fruit</para>
        <orderedlist>
            <listitem>
                <para>Apple</para>
            </listitem>
            <listitem>
                <para>Orange</para>
            </listitem>
        </orderedlist>
    </listitem>
    <listitem>
        <para>Vegetable</para>
        <orderedlist>
            <listitem>
                <para>Cucumber</para>
            </listitem>
            <listitem>
                <para>Lettuce</para>
            </listitem>
        </orderedlist>
    </listitem>
</orderedlist>
```

Definition Lists

A definition list is defined by <dl> element. A definition list element <dl> consists of two parts: (i) A term that is created using the <dt> element and (ii) a description that is created using the <dd> element. The definition list supports one to multiple terms and the corresponding definition(s). The term comes before the definition.
The definition list element shown in the first figure below will be rendered in a DocBook format as in the second figure below.

```
<dl>
    <dt>Coffee</dt>
        <dd>Cappuccino</dd>
</dl>
```

```
<variablelist>
    <title></title>
    <varlistentry>
        <term>Coffee</term>
        <listitem>
            <para>Cappuccino</para>
        </listitem>
    </varlistentry>
</variablelist>
```

The supported HTML table elements

The HTML table element (<table>) is divided into rows (<tr>), and each row may contain a number of table data cells (<td>) or (<th>).

The <table>, <tr>, <td>, and <th> elements support two attributes: (i) align and (ii) valign.

(i) align

You use align to horizontally arrange items within a table cell. The possible values are as follows:

- "left" to left-align the content in a cell.
- "center" to center-align the content in a cell.
- “right” to right-align the content in a cell.
- “justify” to stretch the content with equal width.
- "char" to align the content to a specific character.

(ii) valign

You use valign to vertically arrange items within a table cell. The possible values are as follows:

- "top" to top-align the content in a cell.
- "middle" to center-align the content in a cell.
- "bottom" to bottom-align the content in a cell.

The table element shown in the figure below will be rendered in a DocBook format as in figure below.

```
<table>
    <tr align = "center">
        <th>Name</th>
        <th>Cups</th>
        <th>Types of coffee</th>
        <th>Sugar?</th>
    </tr>
    <tr>
        <td align = "left">Jame</td>
        <td align = "left">10</td>
        <td align = "left">Espresso</td>
        <td align = "left">No</td>
    </tr>
</table>
```

```
<table>
    <title></title>
    <tgroup cols="4">
        <colspec colname="c1" colnum="1" colwidth="1.01*"/>
        <colspec colname="c2" colnum="2" colwidth="1.0*"/>
        <colspec colname="c3" colnum="3" colwidth="1.01*"/>
        <colspec colname="c4" colnum="4" colwidth="1.01*"/>
        <thead>
            <row>
                <entry align="center">Name</entry>
                <entry align="center">Cups</entry>
                <entry align="center">Types of coffee</entry>
                <entry align="center">Sugar?</entry>
            </row>
        </thead>
        <tbody>
            <row>
                <entry align="left">Jame</entry>
                <entry align="left">10</entry>
                <entry align="left">Espresso</entry>
                <entry align="left">No</entry>
            </row>
        </tbody>
    </tgroup>
</table>
```

The supported style sheet properties

The DocBook engine of Report Wizard supports specific style sheets within an HTML element. The supported style sheet properties are as follows:

1. Font
2. Text Align
3. Text Decoration
4. Vertical Align

Font

The “font” property is a shorthand property for setting the “font-style”, and “font-weight” properties.

Font Style

The “font-style” property defines that the font be displayed either in normal way or in italics within a specified font family. The font-style shown in the first figure below will be rendered in a DocBook format in the second figure below.

```
<p style="font-style:normal">This is normal font style paragraph.</p>
<p style="font-style:italic">This is italic font style paragraph.</p>
```

```
<para>This is normal font style paragraph.</para>
<para><emphasis role="italic">This is italic font style paragraph.</emphasis></para>
```

Font Weight

The “font-weight” property specifies the weight of the font. The values “100” to “900” form an ordered sequence where each value indicates the font weight. The keyword ‘normal’ is synonymous with “400”, and “bold” is synonymous with “700”.

The following values are defined:

- normal, lighter, 100, 200, 300, and 400 will be rendered as ‘normal’.
- bold, bolder, 500, 600, 700, 800, and 900 will be rendered as ‘bold’.

The font-weight property shown in the first figure below will be rendered in a DocBook format in the second figure below.

```
<p style="font-weight:normal">This is normal font weight paragraph.</p>
<p style="font-weight:lighter">This is lighter font weight paragraph.</p>
<p style="font-weight:300">This is 300 font weight paragraph.</p>

<p style="font-weight:bold">This is normal font weight paragraph.</p>
<p style="font-weight:700">This is 700 font weight paragraph.</p>
```

```
<para>This is normal font weight paragraph.</para>
<para>This is lighter font weight paragraph.</para>
<para>This is 300 font weight paragraph.</para>

<para><emphasis role="bold">This is normal font weight paragraph.</emphasis></para>
<para><emphasis role="bold">This is 700 font weight paragraph.</emphasis></para>
```

Text Align

The text-align property specifies the horizontal alignment of inline content of a block. The supported tags of the text-align property are <table>, <tr>, <td>, and <th>. The values of this property are:

- "left" to left-align the content.
- "center" to center-align the content.
- “right” to right-align the content.
- “justify” to stretch the content with equal width.
- "char" to align the content to a specific character.

The text-align property shown in the first figure below will be rendered in a DocBook format as in the second figure below.

```
<table style="text-align:center">
    <tr>
        <td>center</td>
        <td>center</td>
    </tr>
    <tr style="text-align:left">
        <td>left</td>
        <td style="text-align:right">right</td>
    </tr>
</table>
```

```
<p style="text-decoration:none">This is none text decoration paragraph.</p>
<p style="text-decoration:underline">This is underline text decoration paragraph.</p>
<p style="text-decoration:line-through">This is line-through text decoration paragraph.</p>
```

Text Decoration

The text-decoration property specifies decorations that are added to text. The values of this property are none, underline, or line-through. The text-decoration property shown in the first figure below will be rendered in a DocBook format as in the second figure below.

```
<table>
    <title></title>
    <tgroup cols="2">
        <colspec colname="c1" colnum="1" colwidth="1.0*"/>
        <colspec colname="c2" colnum="2" colwidth="1.0*"/>
        <tbody>
            <row>
                <entry align="center">center</entry>
                <entry align="center">center</entry>
            </row>
            <row>
                <entry align="left">left</entry>
                <entry align="right">right</entry>
            </row>
        </tbody>
    </tgroup>
</table>
```

```
<para>This is none text decoration paragraph.</para>
<para><emphasis role="underline">This is underline text decoration paragraph.</emphasis></para>
<para><emphasis role="strikethrough">This is line-through text decoration</emphasis></para>
```

Vertical Align

The vertical-align property specifies the vertical alignment of text or an element within a lined box. The supported tags of vertical-align are <table>, <tr>, <td>, and <th>. The values of the vertical-align property are top, middle, and bottom.

- "top" to top-align the content.
- "middle" to center-align the content.
- "bottom" to bottom-align the content.

The vertical-align property shown in the first figure below will be rendered in a DocBook format as in the second figure below.

```
<table style="vertical-align:top">
    <tr>
        <td>top</td>
        <td>top</td>
    </tr>
    <tr style="vertical-align:bottom">
        <td>bottom</td>
        <td style="vertical-align:middle">middle</td>
    </tr>
</table>
```

```
<table>
    <title></title>
    <tgroup cols="2">
        <colspec colname="c1" colnum="1" colwidth="1.0*"/>
        <colspec colname="c2" colnum="2" colwidth="1.0*"/>
        <tbody>
            <row>
                <entry valign="top">top</entry>
                <entry valign="top">top</entry>
            </row>
            <row>
                <entry valign="bottom">bottom</entry>
                <entry valign="middle">middle</entry>
            </row>
        </tbody>
    </tgroup>
</table>
```

UML Profiling and DSL Guide

UML is a general purpose visual modeling language for specifying, constructing and documenting the artifacts of systems to be used with all major application domains and implementation platforms. It has been widely adopted by both industry and academia as the standard language for describing software systems.

The MagicDraw UML tool provides the ability to use the DSL engine to adapt domain-specific profiles, enabling you to create your own custom diagrams, custom specification dialogs, custom real-time semantic rules, etc. In other words, you can create a low-budget specialized domain-specific tool and hide the UML underneath.

DSL customization is a model-driven approach, based on UML profiling.

Related pages

Basic Concepts

Concept	Description
DSL	Domain Specific Language
Stereotyped element	An element with no customized stereotype applied.
Customization element	An element in which the customization data is specified. Create the customization element by using a Profile diagram.
Customization data	A collection of properties and attributes of the Customization element.
Customization model	A collection of customization elements.
DSL element	An element with the applied customization data.

Related Pages

- UML Profiling and DSL Guide(see page 2034)
  - Basic Concepts(see page 2035)
  - Working with Profiles(see page 2036)
  - Customizing diagrams(see page 2045)
  - Domain Specific Language Customization(see page 2067)
  - Distributing Profiles(see page 2156)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Working with Profiles

Profile in UML is a special kind of Package(see page 800), used to contain collections of stereotypes(see page 856), domain specific data types, and libraries.

This chapter includes the following sections on working with profiles:

- Creating Profiles(see page 2036)
- Applying Stereotypes(see page 2043)
- Modifying Profile(see page 2044)

Related pages

- Package(see page 800)
- Stereotype(see page 856)

Creating Profiles

Profiles can be created in any project. However, often many projects use the same profiles. In order to reuse the same profile, it must be created as an independent file with shared data called a module.

To create a Profile as a module

1. Create a new project.
2. In the Model Browser(see page 29), right-click the root **Model** and create a new Profile.
3. Create a new Profile diagram(see page 324) inside the Profile.
4. Create stereotypes (see page 856)in the diagram.
5. Use Associations(see page 651) and Generalizations(see page 722) between them if needed.
6. In the Model Browser, from the created Profile shortcut menu, choose **Project Usages > Share Packages.**
7. Select the Profile as shared, enter its description, and click **OK**.
8. Save the project.

Now you may use this profile in other projects.

To use the created profile in other projects

- From the **File** menu, choose Use Project[368] and select the project file. Only shared packages of the module will be loaded into the project.

368 https://docs.nomagic.com/display/MD190/Managing+used+projects

Tip
If you don't want to see particular information, move elements from shared packages into the root model or other non-shared packages in your module.

Related Pages

- Project partitioning(see page 236)

Specifying Invisible Stereotypes, Tags, and Constraints

Normally, tags are used to hold additional specific information. This information helps interpret basic elements differentiated by some external tools, such as transformations, code engineering, etc.

In these cases, you only want to mark the model, not make an impact on existing diagrams.

Tag definitions are properties of stereotype in UML2. UML does not allow using tagged values without assigning stereotype. Your diagrams will be changed when some tags are added into the model.

A modeling tool enables you to make chosen stereotypes or tags in diagrams invisible.

To make a stereotype invisible

The stereotype you want to make invisible must be inherited from «InvisibleStereotype» found in the modeling tool (e.g., MagicDraw) profile.

1. Create a custom stereotype.
2. Create a Generalization between your stereotype and InvisibleStereotype. Your custom stereotype will be not displayed on symbols where it is applied. Note that all tags will also be invisible on symbols.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example
It is not desirable to see an author tag in diagrams. To accomplish this, follow these steps:

1. Create a stereotype «copyrighted» that extends the Element metaclass.
2. Create a property "author: String" for a «copyrighted» stereotype.
3. Inherit this stereotype from «InvisibleStereotype» (create generalization in the model or diagram).

Now you may enter the tagged value "author" for any element in your model.
The stereotype «copyrighted» will be automatically applied but will be invisible in diagrams.

To create an invisible tag

1. Apply «InvisibleStereotype» on your tag definition (a property of a stereotype).

Hidden Tags
All hidden tags will be invisible on the symbols of stereotyped elements.

Example:
The stereotype «identification» has two tags: ID and Description. The user would like to see a Description value on elements in diagrams, but ID should be hidden as it is used in some external model transformation tools or similar activities.
Apply «InvisibleStereotype» on the ID property of the «identification» stereotype and it will be not visible on stereotyped elements on diagrams.

To make a constraint invisible

1. Apply the «InvisibleStereotype» stereotype to the constraint.
2. Invisible constraints may be used while creating validation constraints, in DSL customization, or in other modeling cases.

Related pages

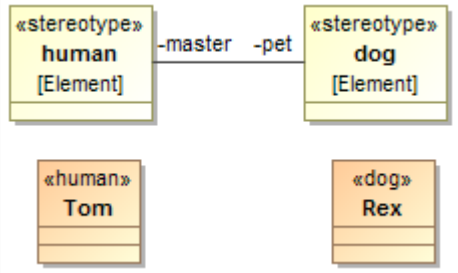
- Stereotype(see page 856)
- Tag(see page 864)
- Generalization(see page 722)

Initializing opposite association end

If two stereotypes are connected with an Association (see page 651)and there are roles at both ends, the role at one end will be automatically initialized when the opposite role is specified.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example:
When «dog» Rex is set as a pet for «human» Tom, Tom will be automatically set as “master” for Rex:



Related pages

- Stereotype(see page 856)
- Association(see page 651)
- Association End(see page 654)

Applying Different Icons for the Same Stereotype

A modeling tool provides the ability to create a stereotype, which uses different icons depending on a stereotype “kind." For example, PseudoState can be represented by different notations that depend on PseudoStateKind.

To apply different icons on the same stereotype

1. Create an Enumeration(see page 697) (in our example, *supercarKind*) to define all stereotype “kinds."
2. Apply the *«iconHolder»* stereotype to the enumeration.
3. Create as many EnumerationLiterals of as many different kinds as you need (in our example, it is “*ferrari*” and “*audi*”).
4. Apply stereotypes with different icons to every EnumerationLiteral (in our example, *«ferrari»* and *«audi»*).
5. Create a Stereotype (in our example, *supercar*) with a property (tag definition), with type enumeration (*supercarKind*). You could also select a default value.
6. Assign the created stereotype to an element and select one of the enumeration values in tags. A stereotyped element will change its icon to the enumeration value.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example of applying icons.

Related Pages

- Stereotype(see page 856)
- Enumeration(see page 697)

Grouping Tags inside Stereotype

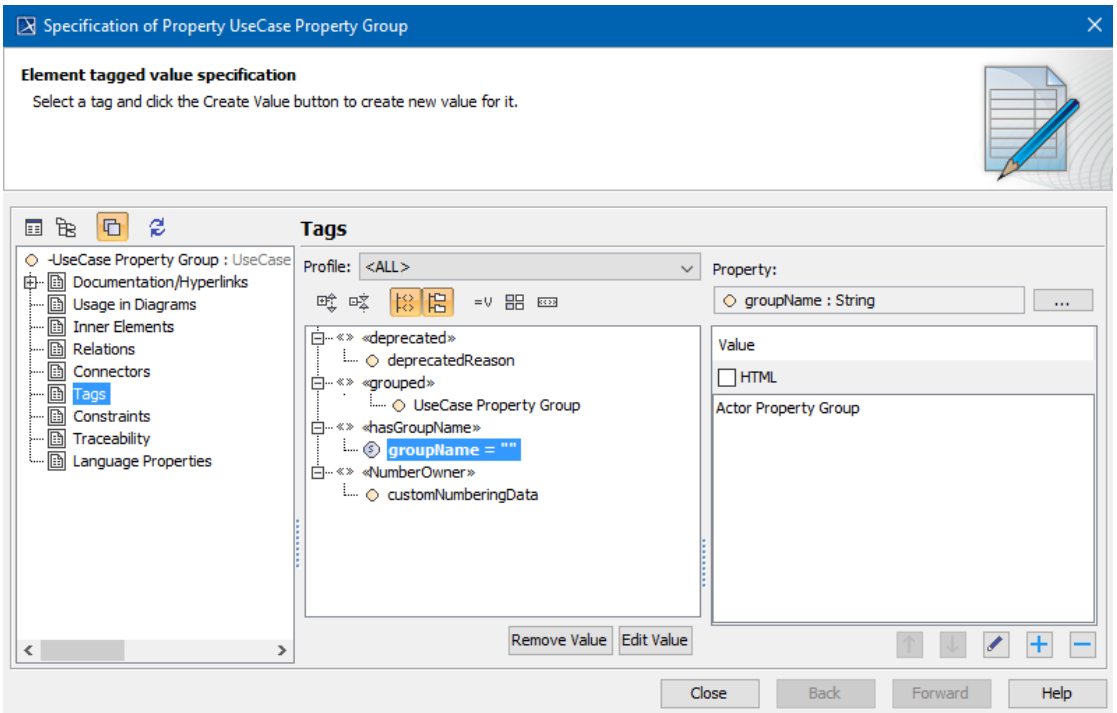
When a stereotype has many properties, it is very useful to group these properties into named groups. For this purpose, every property could have a tagged value with its group name.

To group tags

1. Open the Specification window(see page 913) of the stereotype property.
2. Click the **Tags** property group.
3. Select the **groupName** property and assign a value for it:
    a. Click the **Edit Value** button

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

b. In the **Value** area, type a group name.



4. Repeat these steps for every stereotype property.

Related Pages

- Applying Stereotypes(see page 2043)
- Stereotype(see page 856)
- Tag(see page 864)
- Specification window(see page 913)

Creating Tags with Default Values

To create a tag with a default value

1. Create a tag definition(see page 864) (Property).
2. Specify default value(see page 865) for this tag.
3. Specify multiplicity larger than 0 (1, 1..*, 2, etc.).

Multiplicity means that this tag is mandatory.

When the stereotype is applied, mandatory tags will be automatically created and default values will be assigned.

Related Pages

- Applying Stereotypes(see page 2043)
- Stereotype(see page 856)
- Tag(see page 864)

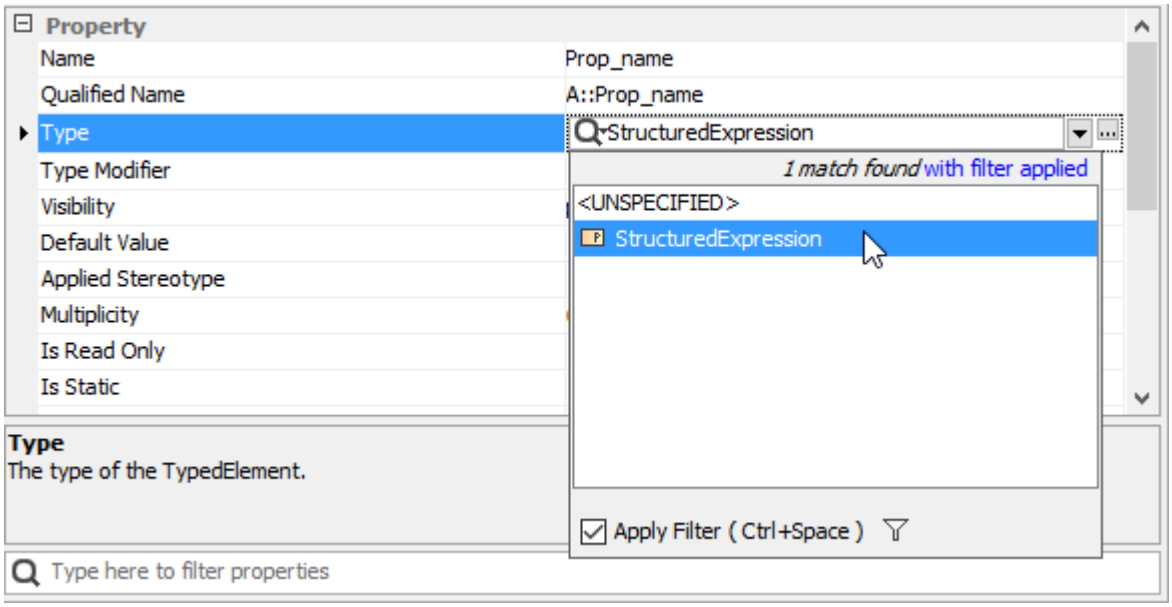
Creating Structured Expression Tags

If you need a tag value, as defined by using the dialog for specifying criteria(see page 1227), you must create a structured expression tag definition.

To create a structured expression tag definition

---

1. Create a new stereotype(see page 856).
2. Right-click the stereotype and select **Create Element > Property** to create a new tag definition.
3. Open the Specification window(see page 913) of the tag definition and select the **Type** property. Click the property specification cell and select the StructuredExpression value as a type.



Once the tag definition is created, you can apply the stereotype with this tag definition on an element in your model. Then, in the Specification window of the element, click the **Tags** property group, select the structured expression tag, and click the **Create Value** button to define the value.

> **Tags**
> The structured expression tag can also be represented as a regular property in the Specification window and **Properties tab** of the element. You must then create a customization data(see page 2068) for the stereotype with this tag definition.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related Pages

- Applying Stereotypes(see page 2043)
- Stereotype(see page 856)
- Tag(see page 864)
- Creating Customization Data(see page 2068)
- Specifying criteria for querying model(see page 1227)

Applying Stereotypes

A stereotype (see page 856)defines how an existing metaclass may be extended. The stereotype enables the use of the platform or domain-specific terminology or notation in place of, or in addition to, the ones used for the extended metaclass.

When a stereotype is applied to a model element, the values of the properties may be referred to as tagged values(see page 864).

To apply a stereotype, do one of the following:

- In the corresponding element's Specification window(see page 913), click the **Applied Stereotype** property specification cell, then click , and select stereotypes from the open list. After you have selected, click **Apply**.
- In the element's shortcut menu, select **Stereotype** and select a stereotype that you wish to apply. Click **Apply** when you are done.
- On the diagram pane, select an element and in that element's name area, type the stereotype name between brackets « ». The element's name comes after the stereotype.

  If you want to name the element Books and assign it the «table» stereotype, in the element's name area type the following: «table» Books. The name completion for the stereotypes works in the name editing mode, press Ctrl+Spacebar or Ctrl+Backspace to get a list of available stereotypes.

You can change the order of the stereotypes applied to the element. The symbol style of the first stereotype in the list will be applied to the element.

Related pages

- Stereotype(see page 856)
- Modifying Profile(see page 2044)
- Tag(see page 864)
- Working with projects(see page 104)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Modifying Profile

If you want to make changes in the profile, you can modify it by either opening the profile as a regular project or modify it in the read-write mode.

Opening Profile as Project

This is the safest way to modify your profiles. It helps to see all infrastructure (including diagrams, tests, examples, etc.) that is not shared.

To open a Profile as a project

- In the Browser, from the used Profile shortcut menu, choose **Project Usages > Open**.
- On the **File** menu, click **Open Project** to open the Profile as a simple Project.

Modifying Profile in Read-Write Mode

Using the profile in the read-write mode is the fastest way to add changes when you are not in teamwork and do not care if there are possible editing conflicts. It helps to perform basic refactoring of your shared projects by moving the elements directly from one project to the other. You may choose read-write usage mode when trying to use a profile as a project.

To use Profile in the read-write mode

---

1. Do one of the following:
    - If the profile is not in use yet, on the **File** menu, click **Use Project**. Select **Use Local Project** or **Use Server Project,** depending on where the project is located. Select a profile project and click **Next**.
    - If the profile is already in use, choose **Project Usages > Options** from its shortcut menu.
2. In the **Accessibility** area, click **Read-write**. By setting this option you will be able to change the profile directly in your project. Click **Finish** after you are done.

Related Pages

- Working with projects(see page 104)
- Project partitioning(see page 236)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Customizing diagrams

You can create your own diagram types for specific domains, platforms, technology, or other purposes using the **Customize Diagram Wizard**. This powerful engine enables the creation of custom elements in the diagram toolbar, custom symbol styles, and other customizations.

To open the **Customize Diagrams** dialog

- On the main menu, click **Diagrams** > **Customize**.

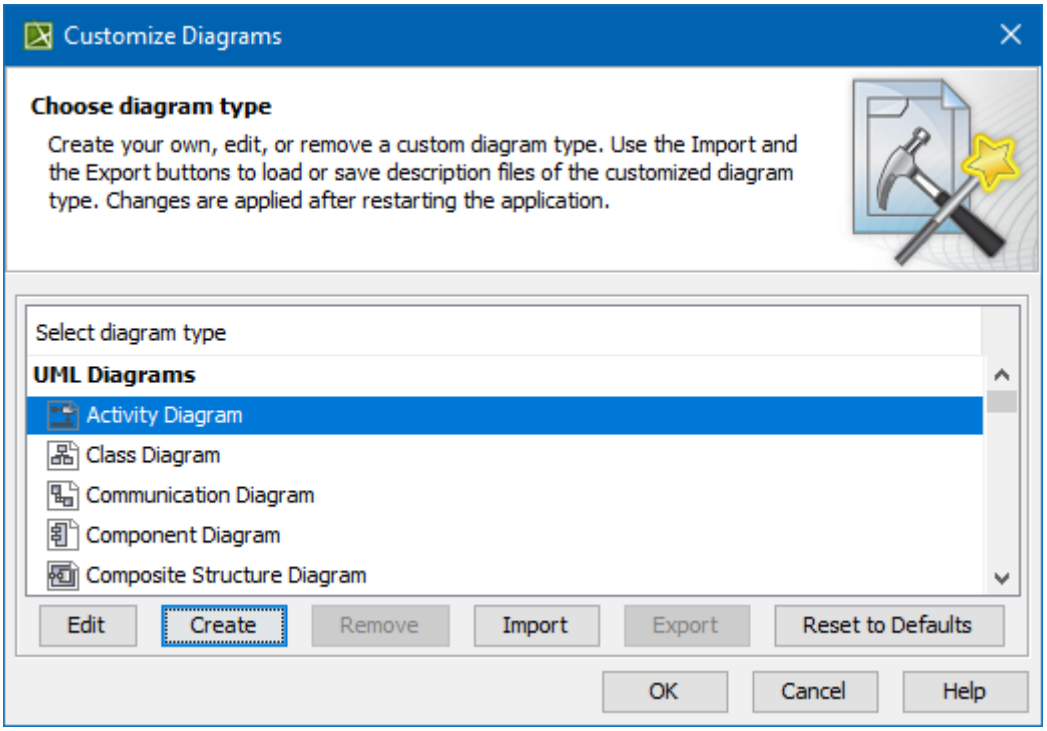
You can change the properties of existing diagrams (**Edit** function) or create your own brand new diagram type (**Create** function). Diagram customization descriptors are saved in a separate file for every diagram, so you are able to exchange these customizations with your partners or colleagues (use **Import** or **Export** function).

The **Reset to default** button in the **Customize Diagrams** dialog restores the default configuration for diagrams bundled with MagicDraw installation. It does not work with user-defined diagrams.

To open the **Customize Diagram Wizard**

Do either:

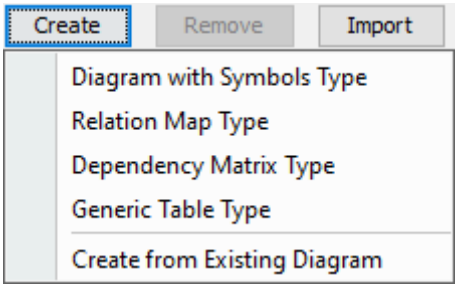
- In the **Customize Diagrams** dialog, select a diagram type and click **Edit**.



- Click **Create** and then select one of the following commands:

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- **Diagram with Symbols Type**, if you need to create a new diagram type.
- **Relation Map Type**, if you need to create a new relation map diagram.
- **Dependency Matrix Type**, if you need to create a new matrix type.
- **Generic Table Type**, if you need to create a new generic table diagram.
- **Create from Existing Diagram**, if you need to create custom diagrams from existing ones.



Related pages

- Creating new diagram type(see page 2046)
- Creating new dependency matrix type(see page 2052)
- Customizing diagram palette(see page 2057)

Creating new diagram type

The **Customize Diagram Wizard** contains the following steps for creating a new diagram type or modifying a chosen one.

- Step 1: Specify diagram type and icon (see page 2046)
- Step 2: Specify used projects(see page 2047)
- Step 3: Specify Toolbars(see page 2048)
- Step 4: Specify Toolbar Buttons(see page 2049)
- Step 5: Specify Symbol Properties(see page 2051)
- Step 6: Specify Smart Manipulators(see page 2051)

All Custom Diagrams are based on standard UML diagrams (like Class, Use Case, Sequence, and other).

Step 1: Specify diagram type and icon

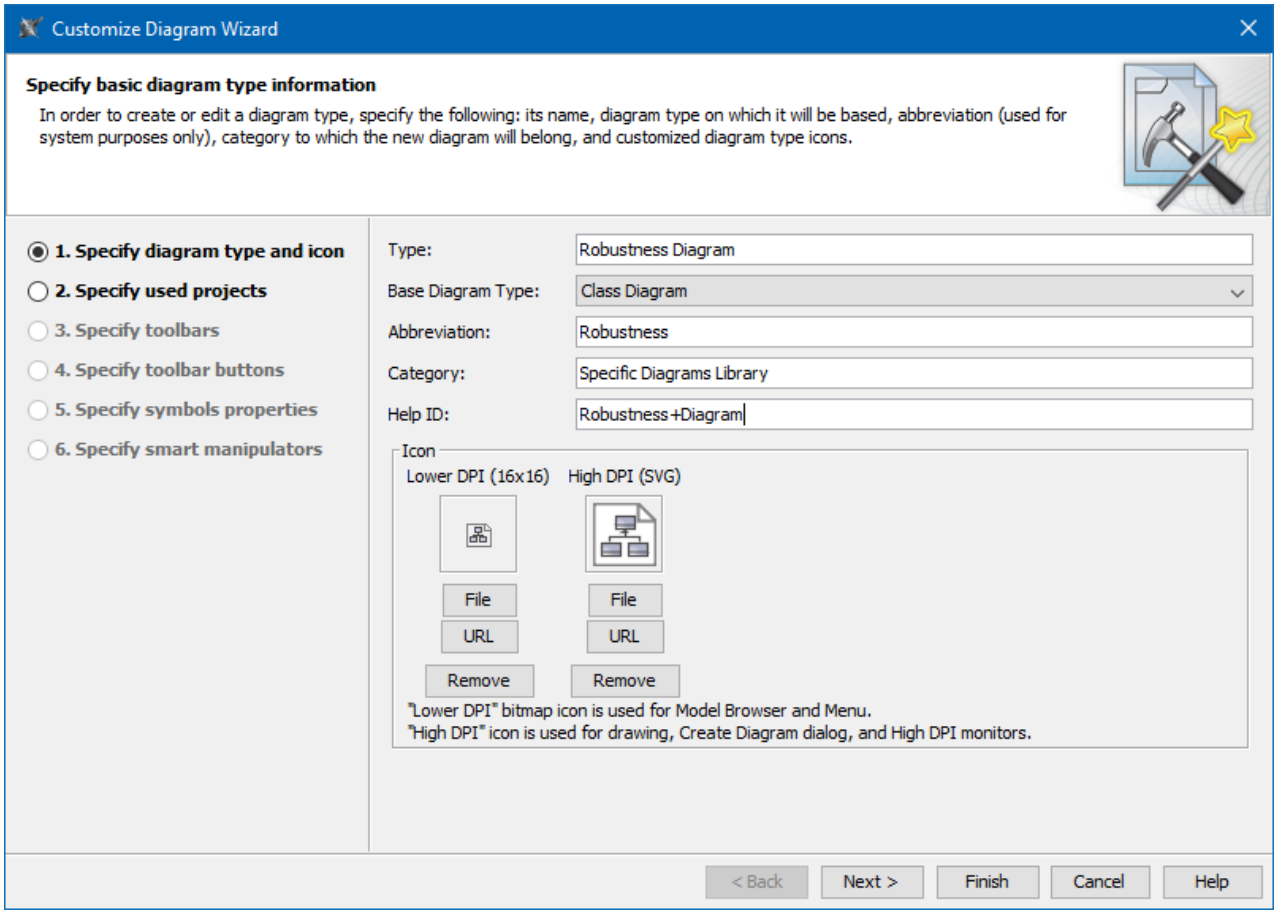
To create your own diagram, first specify the following properties

---

- Diagram type name (for example, Robustness Diagram).
- Base Diagram Type – standard UML diagram to be extended. All configurations, semantics, and other settings will be inherited from this diagram type.
- Abbreviation – a short form of the diagram name. It will be used in Diagram Frames header or Diagram shapes in Content diagrams.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

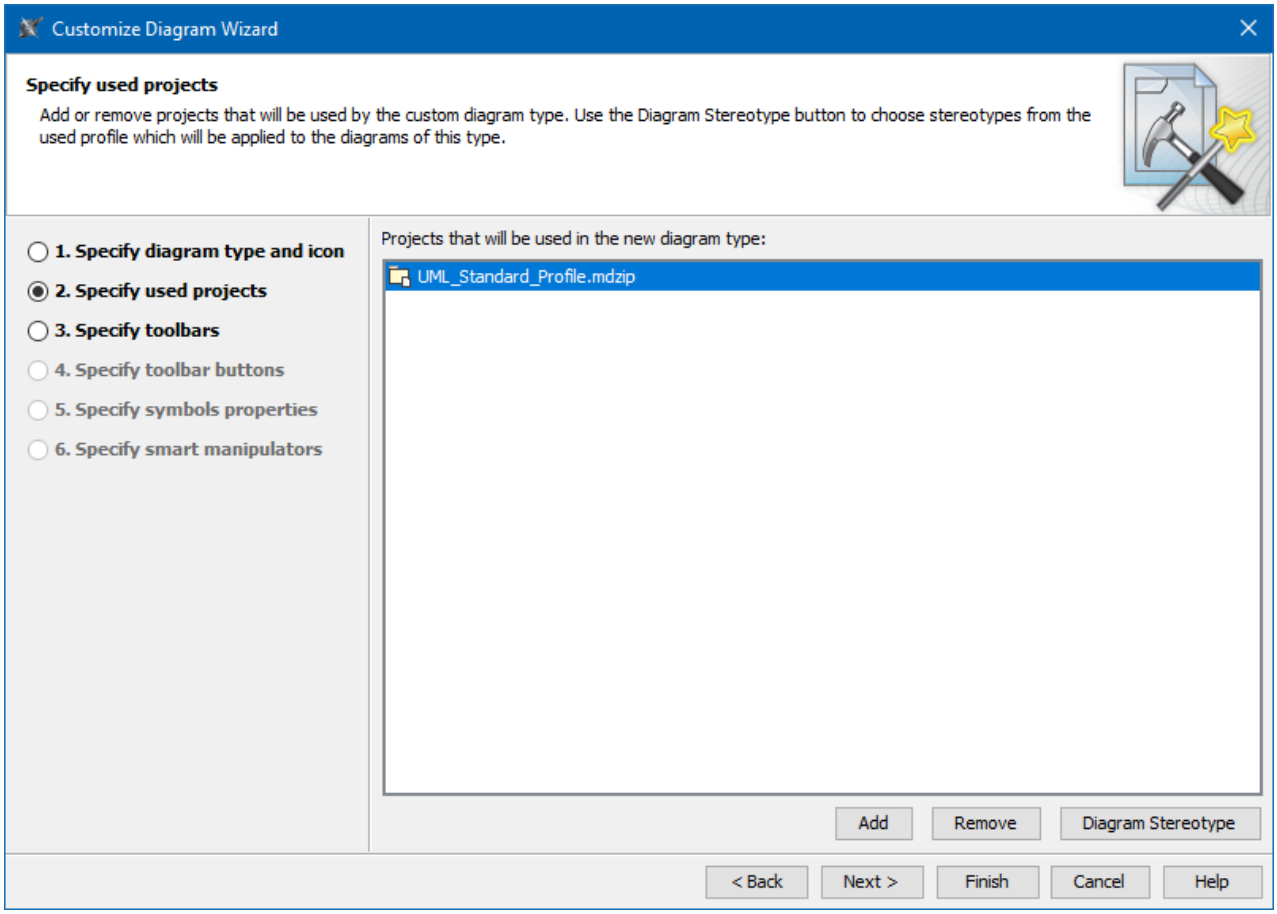
- Category – creates your specific category in the **Diagrams** menu or in the **Create Diagram** command list. You can store all your customized diagrams in this category. If you leave this field empty, the customized diagrams will be added to the **Custom Diagrams** category.
- Help ID – sets a specific Help ID to invoke help or documentation topics.
- Icons – several icons for your custom diagram representation in MagicDraw GUI.



Step 2: Specify used projects

Custom Diagrams are oriented to a new specific domain, technology or platform, and are often based on a custom profile.

Select the required used projects or local profiles. How to use a custom server profile see in customizing diagram palette.(see page 2057)



- Selected profiles must be local, the server profiles cannot be used.
- Do not remove the UML Standard Profile, which is selected by default, from the list. It must be used by any custom diagram type.

The custom diagram could use stereotyped elements. Profiles defining these stereotypes must be used by the custom diagram. The selected used projects or profiles load when a user creates a custom diagram in a project. You can choose a stereotype for the diagram by clicking the **Diagram Stereotype** button.

Step 3: Specify Toolbars

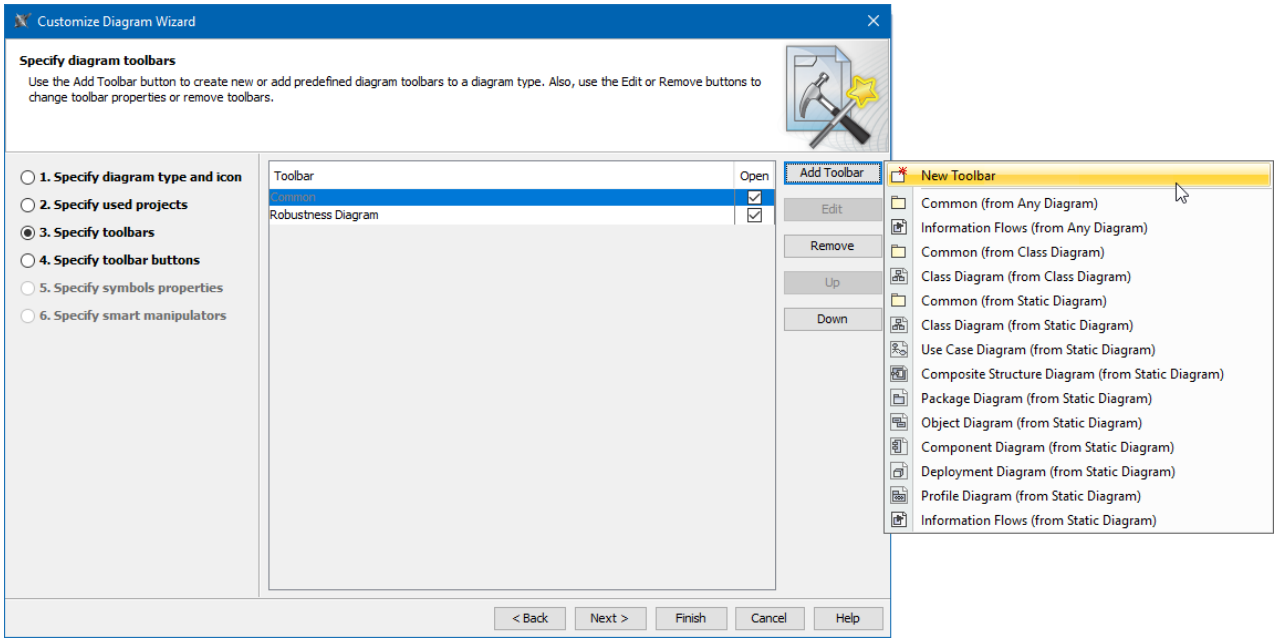
Every diagram differs by the elements used in them. In the **Specify toolbars** step, you can group standard or custom elements.

You can:

- Create your own custom diagram toolbar.
- Create your own toolbar, name it, and select an icon.
- Choose standard toolbars that will be visible in your diagram.
- Select existing toolbars inherited from the base diagram type.
- Arrange the order of toolbars by using "Up" and "Down" buttons.

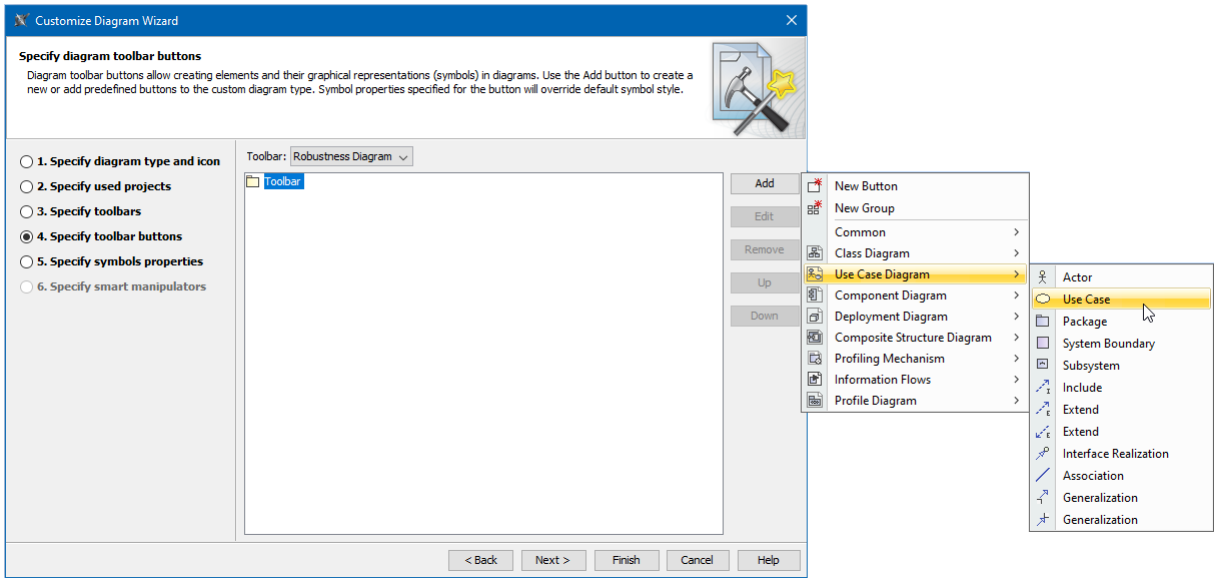
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Select which toolbars will expand or collapse by default (use the **Open** check box).



Step 4: Specify Toolbar Buttons

Select standard or stereotyped elements for your custom diagram toolbars.



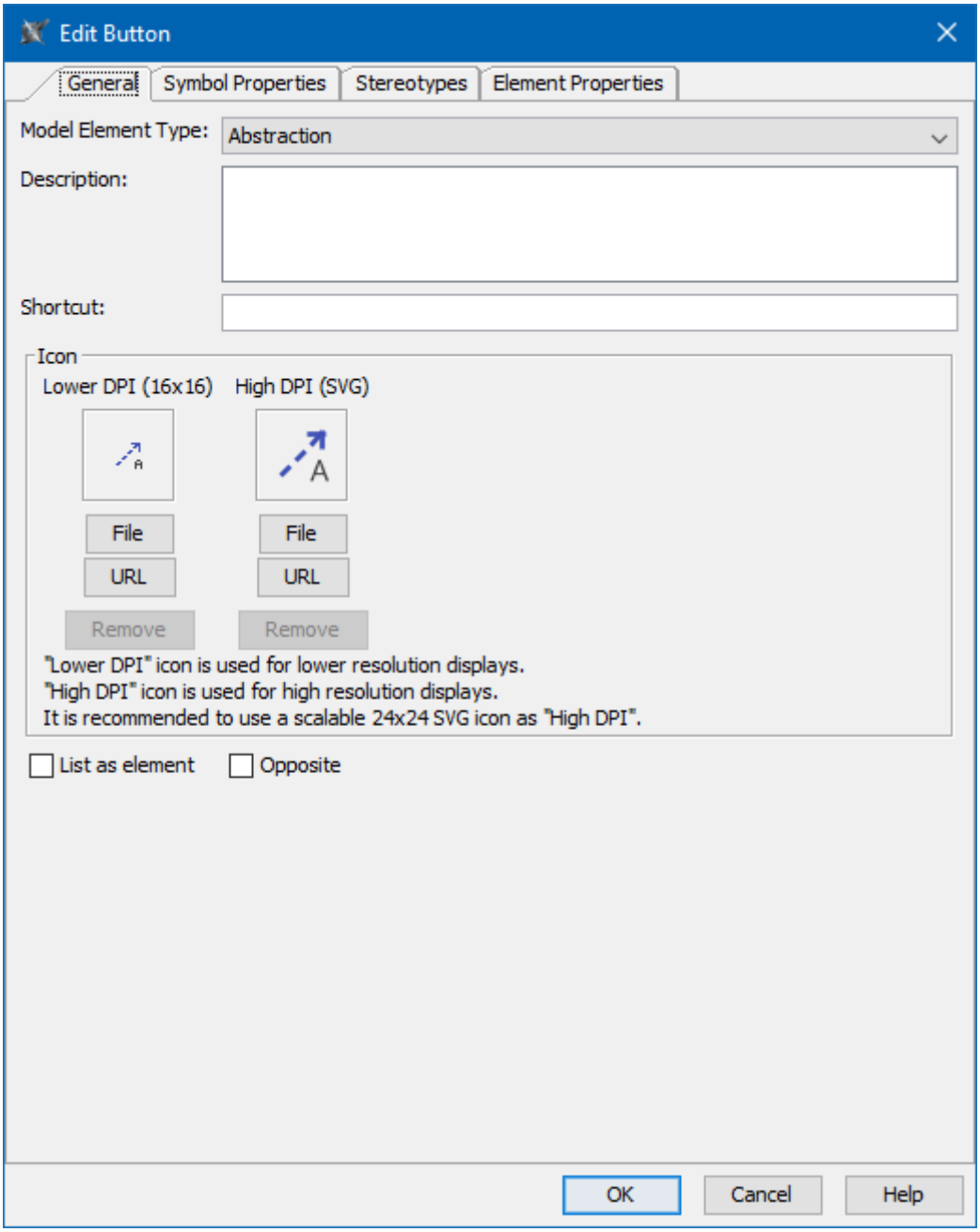
Click **Add** and select standard UML elements or click **Add** and then **New Button** to create your own buttons to create customized or stereotyped elements.

Customize the following properties when creating a new button:

- Model element type
- Custom icon for a button
- Shortcut that activates the button
- Tooltip description

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Stereotype(s) to apply to the created element
- Symbol style – selected symbol properties will be used only when the element is created using this button
- Default values for some primitive model element properties (like isAbstract = true or similar)



- List as element – select this check box to add the button as a command in the **Create Element** command list.
- Opposite – this check box is only available for relationship buttons. Select it to add the opposite relationship button.

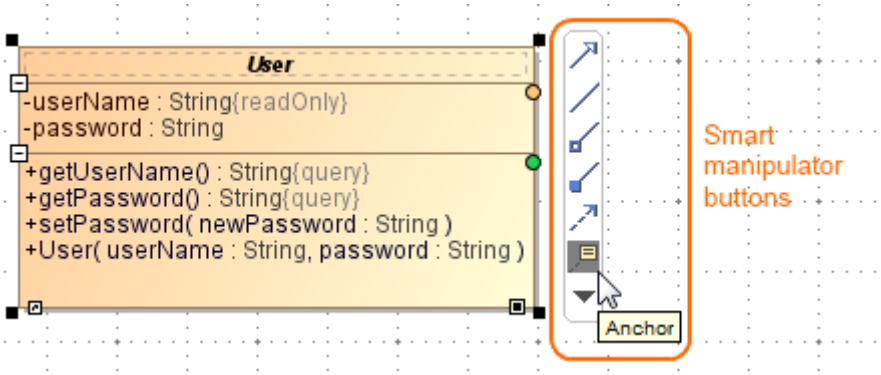
Step 5: Specify Symbol Properties

This step allows you to customize the style for any element appearing in your custom diagram (e.g. a class dropped in your diagram should be suppressed and red).

You can change the appearance of standard symbols, symbols of stereotyped elements, and the custom diagram itself. The customized element style will be used only in the appropriate custom diagram type.

Step 6: Specify Smart Manipulators

Smart manipulators are special buttons that appear in the pop-up window when a shape or path is selected on a diagram.



Example of smart manipulator

You can configure the kind of relationship suggested when a custom shape or path is selected on a diagram.

There are three main sub-steps:

1. Create a new configuration (or modify an existing one). Select the element to customize. The smart manipulators configuration can be related to:
    - Element, displayed as [Element name]
    - Element + stereotype(s)
    - Symbol, displayed as {Symbol name}
    - Symbol + stereotype(s)
    - Stereotype(s), displayed as «Stereotype name»
2. Select suggested relationships for the selected configuration.
3. Select target elements for the selected relationship.

If several configuration settings could be applied to the same selected element on the diagram, only one configuration will be used according to this priority**:**

#	Configuration	Comment

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

1	Symbol + stereotype(s)	When stereotype(s) for symbol applied
2	Stereotype	
3	Symbol	
4	Element + stereotype(s)	When stereotype(s) for element applied
5	Element	

If several configurations are created for a few stereotypes (for example, Stereotype1 and Stereotype2) in the same diagram, when both stereotypes are applied to one symbol on that diagram pane, the first configuration (in this case, Stereotype1) is used, unless you specify a symbol+ stereotype(s) configuration.

Inheritance of configurations

All diagrams inherit their configurations from a base diagram (example: in this hierarchy: Any diagram > Static diagram > Class diagram > Generic DDL diagram).

If you add a new configuration to a class diagram (for example, Symbol A + stereotype B), the configuration will be used in the generic DDL diagram, as the class diagram is its base diagram. Any static diagrams will not have such configurations.

To change the configuration from the base diagram type

- From the **Configurations** list, select the configuration by the element type and choose the **Specify own configuration** button. This overwrites the inherited configuration.

Creating new dependency matrix type

Creating new dependency matrix types is available in Standard, Professional, Architect and Enterprise editions.

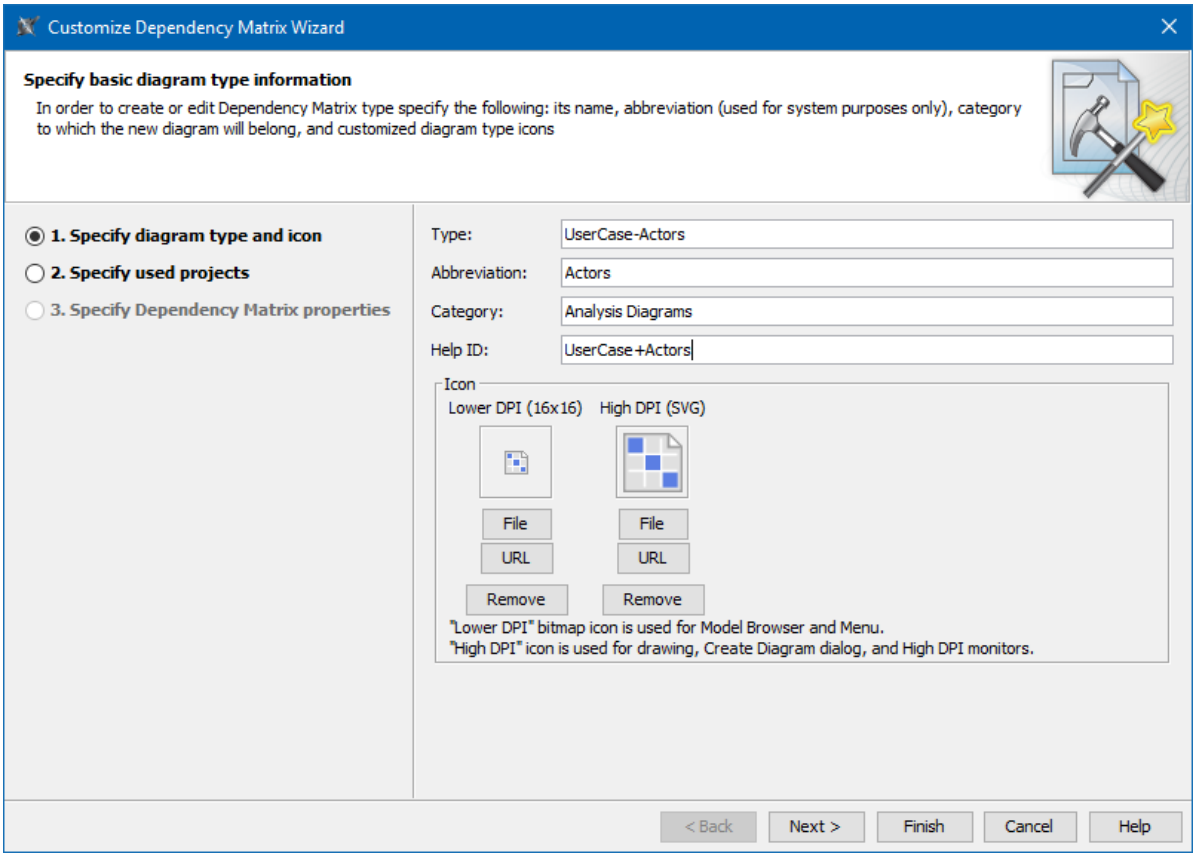
The **Customize Dependency Matrix Wizard** contains the following steps for creating a new dependency matrix type or modifying a chosen one.

- Step 1: Specify Name for a New Matrix Type and Icons(see page 2053)
- Step 2: Specify Used Projects(see page 2053)
- Step 3: Specify Dependency Matrix Properties(see page 2054)

Step 1: Specify Name for a New Matrix Type and Icons

To create your own custom matrix type, first define the following properties

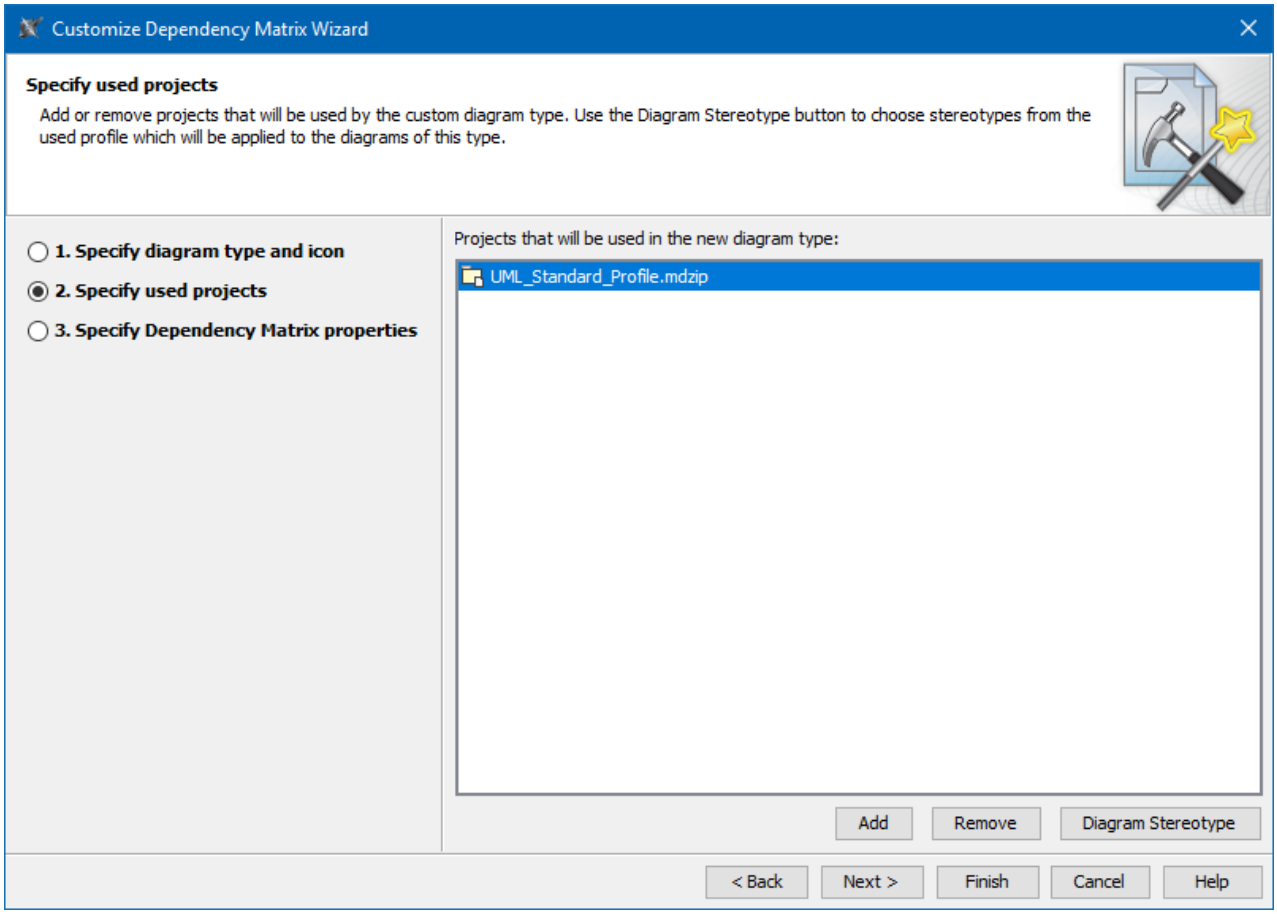
- Matrix type name (for example, UseCase-Actors).
- Abbreviation – a short form of the dependency matrix type. It is displayed on the diagram overview shape of the custom matrix, when the **Show Abbreviated Type** property is set to *true*.
- Category – your specific category where the new matrix type will be stored. The category is displayed on the main menu under **Diagrams** and in the command list of **Create Diagram** on the **Model Browser** shortcut menu. You can keep the default value, if you need the new matrix type to be stored in the **Analysis Diagrams** category.
- Help ID – sets a specific Help ID to invoke help or documentation topics.
- Icons – several icons for the new dependency matrix type representation in MagicDraw GUI.



Step 2: Specify Used Projects

Select the required used projects or profiles.

> Do not remove the UML Standard Profile, which is selected by default, from the list. It must be used by any custom dependency matrix type.



A custom dependency matrix can represent the relationships among stereotyped elements. Profiles that define these stereotypes must be used by the custom dependency matrix type.

All the selected used projects or profiles load when a dependency matrix of the custom matrix type is created in the project.

To choose the stereotype for the custom dependency matrix type

- Click the **Diagram Stereotype** button.

Step 3: Specify Dependency Matrix Properties

This step allows you to specify the appearance of the new dependency matrix type. For example, you can predefine the default column text direction, row and column scope, and so forth.

Property name	Description
Dependency Matrix	
Legend Location	Select to choose the location of the legend (**Top-Left Corner** or **Above Column Header**) or set the **Do Not Display** mode to keep it hidden.

Take Whole Model As Scope	Set to *true* to select the whole model as a scope for relation analysis. The root package *Model* will be selected by default in both the **Row Scope** and **Column Scope** boxes.
Direction	Select a direction of relationships for the relation analysis and representation in the dependency matrix. Be aware that new relationships will be created with the selected direction.
Dependency Criteria	Select an expression type and specify the expression to be used as the relation criteria.
Show Elements	Select to show only related (by a selected dependency criteria), only non- related, or all elements.
Show Inner Dependencies	Set to *true* to show the number of relationships in every owning element cell.
Suppress Criteria Area	Set to *true* to suppress the **Criteria** area toolbar.
Read Only	Set to *true* to make the matrix read-only. You will not be able to create or delete relationships.
Description Area	Type the description for the dependency matrix. The text will be displayed in the description area of the matrix.
Hide Types	Set to *true* to hide the **Row Element Type** and **Column Element Type** boxes from the **Criteria** area.
Hide Scope	Set to *true* to hide the **Row Element Scope** and **Column Element Scope** boxes from the **Criteria** area.
Hide Dependency Criteria	Set to *true* to hide the **Dependency Criteria** and **Direction** boxes from the **Criteria** area.
Column	

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Column Scope Defined	Set to *true* to take empty scope as a whole model.
Column Types Include Subtypes	Set to *true* to display subtypes of selected element types, for example, if a class is selected, then all its subtypes, such as component or custom subtypes, will be displayed.
Column Owner Display Mode	• Select the **Compact tree** mode to display elements with their direct and common owners in the column header. The data will be represented as a tree. • Select the **Complete tree** mode to display elements with all their owners in the column header. The data will be represented as a tree. • Select the **Hidden** mode to display elements without any owners in the column header. The data will be represented as a list. • Select the **Full qualified name** mode to display elements with their owners in the column header. The data will be represented as a list.
Column Text Direction	Specify the direction of the text in the column header. Be aware that this property value can be applied only when the **Column Owner Display Mode** property value is **Hidden** or **Full qualified name**.
Column Element Type	Specify the element types to show in the columns of the dependency matrix.
Column Query	Select **Containment** or **Custom** to specify the expression to collect column elements.
Column Property Filter	Select properties and their values to create more specific filters for column elements.
Column Sorting Mode	Select **Ascending** or **Descending** to sort elements alphabetically. You can also select the **Custom** sorting mode to use your own order.
Column Header Height	Specify the height of the column header in pixels.
Column Types Include Subtypes	Set to *true* to display subtypes of selected element types. For example, if a class is selected, then all its subtypes (such as component or custom subtypes like SysML block) and requirements will be displayed.
Row	

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Row Scope Defined	Set to *true* to take empty scope as a whole model.
Row Types Include Subtypes	Set to *true* to display the subtypes of selected element types. For example, if a class is selected, all its subtypes (such as component or custom subtypes like SysML block) and requirements will be displayed.
Row Owner Display Mode	• Select the **Compact tree** mode to display elements with their direct and common owners in the row header. The data will be represented as a tree. • Select the **Complete tree** mode to display elements with all their owners in the row header. The data will be represented as a tree. • Select the **Hidden** mode to display elements without any owners in the row header. The data will be represented as a list. • Select the **Full qualified name** mode to display elements with their owners in the row header. The data will be represented as a list.
Row Element Type	Specify the element types to show in the rows of the dependency matrix.
Row Query	Select **Containment** or **Custom** to specify the expression to collect row elements.
Row Property Filter	Select properties and their values to create more specific filters for row elements.
Row Sorting Mode	Select **Ascending** or **Descending** to sort elements alphabetically. You can also select the **Custom** sorting mode to use your own order.
Row Header Width	Specify the width of the row header in pixels.

Customizing diagram palette

You can simply add the appropriate button to the palette of any desired diagram type for a new DSL element type. To do this, you must make certain changes in the customization profile.

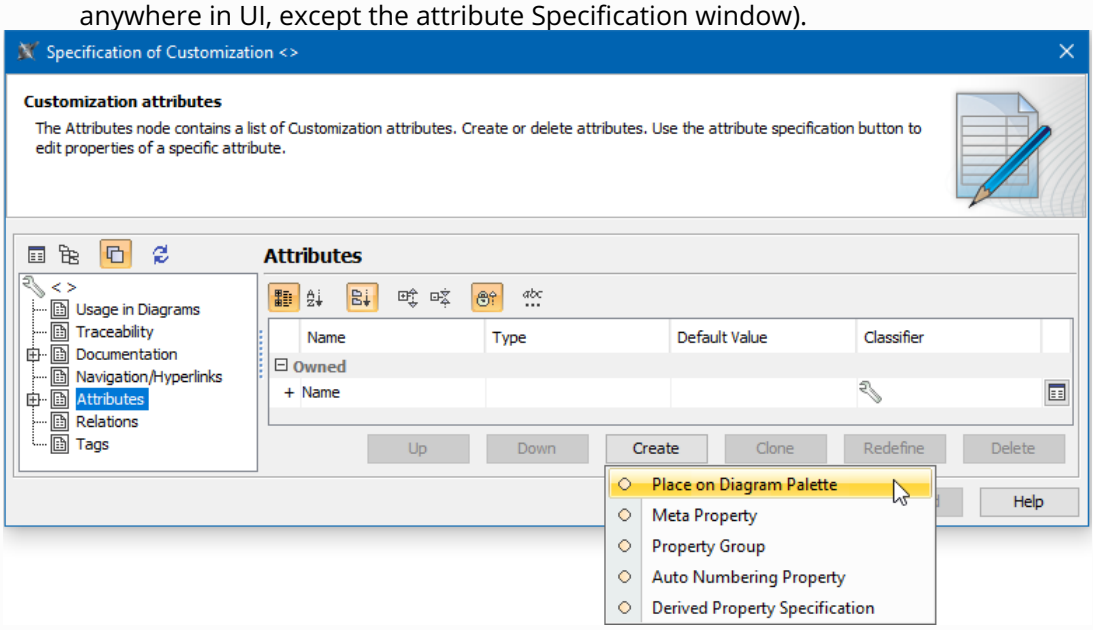
To add the button for creating a DSL element on the palette of a relevant diagram type

1. Open the customization profile to edit.
2. Create a Profile diagram or open an existing one.
3. Create a customization.
4. Set the DSL element type as the customization target.

> **How can I simply set the customization target?**
> Drag the DSL element type to the shape of the newly created customization element.

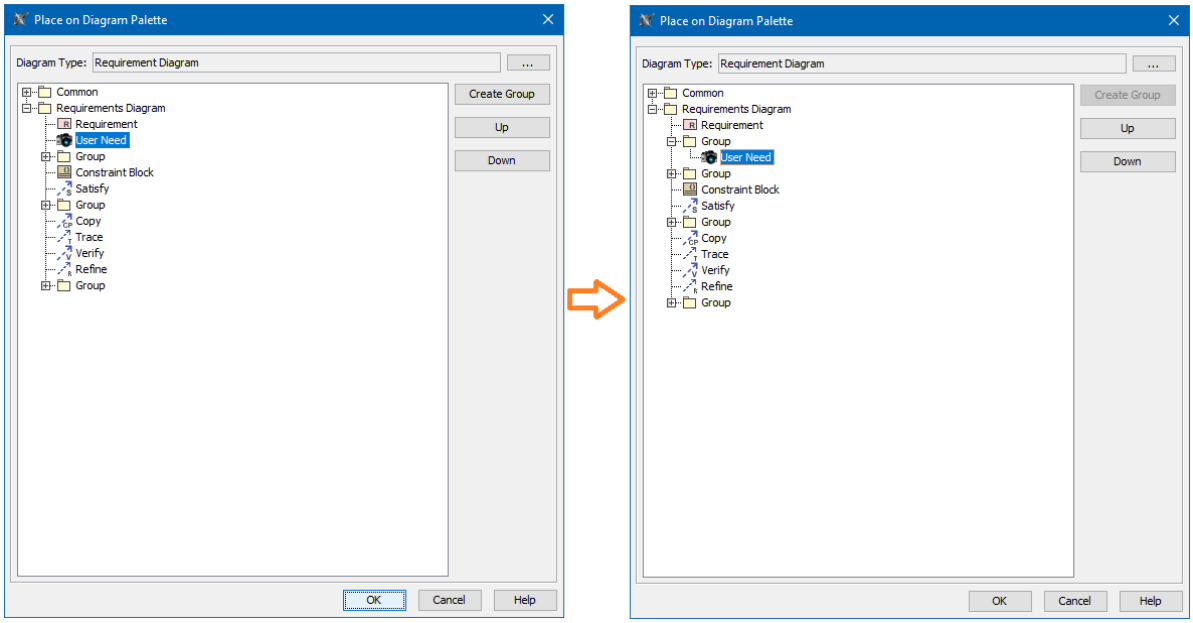
5. Create the **Place on Diagram Palette** attribute for the customization.

> **How can I create the attribute?**
> a. Open the Specification window of the customization element.
> b. Select **Attributes** on the left side of the window.
> c. Click the **Create** button and select **Place on Diagram Palette** (see the following figure).
> d. Type the new attribute name (this is optional, since the name is not visible anywhere in UI, except the attribute Specification window).



6. In the Specification window of the **Place on Diagram Palette** attribute, select the cell of the **Place** property value, and click .
7. In the open dialog, select the diagram type of the palette to customize.
8. In the same dialog, use the **Up** and **Down** buttons to change the position of the new button on the palette.
9. If you want to add several new buttons and organize them into a group on the palette, create a new Group package by clicking the **Create Group** button (shown below, on the left). The button of the DSL element type appears under the Group package (see the screenshot on the right).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

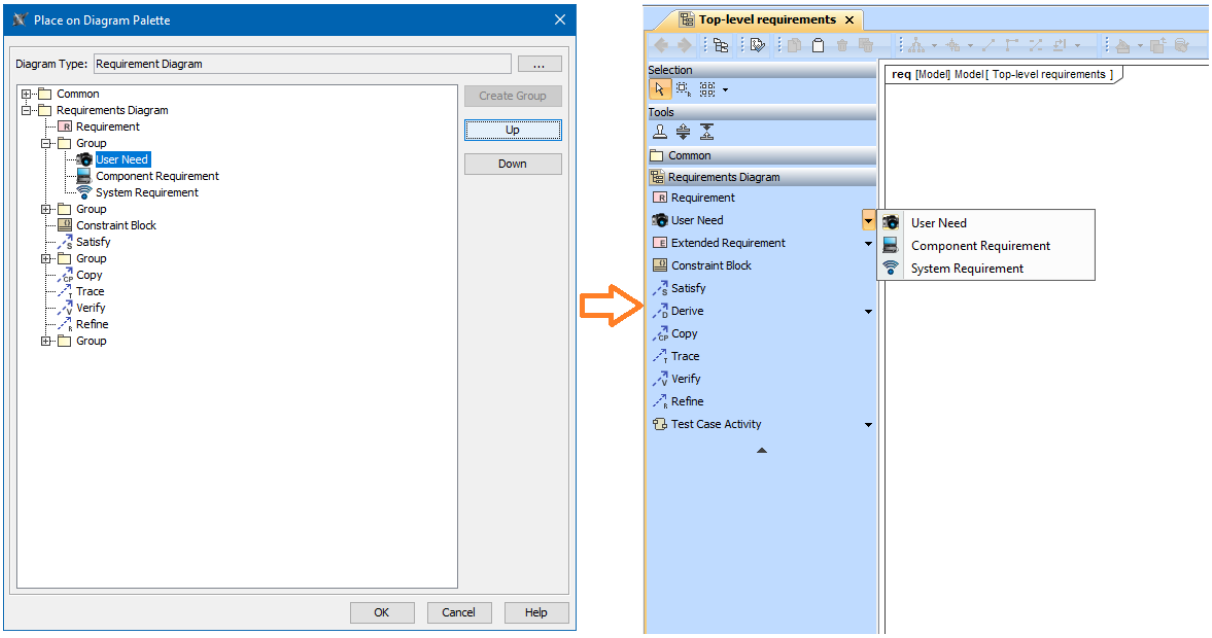


> ⓘ If you want to undo the group creation, simply click the **Up** or **Down** button.

10. Close the dialogs and save the changes.
11. Reopen the diagram with the customized palette.

As a result, you can see the new button on the diagram palette. If you work in a collaborative environment(see page 1360), other users who work with the same project or projects using the same customization profile can see the new button instantly after updating their projects.

If you repeat the procedure to add more buttons within the same group on the diagram palette, you can see the appropriate changes (shown in the following figure).



Related pages

- Customizing diagrams(see page 2045)

Creating new relation map type

The **Customize Relation Map Diagram Wizard** contains the following steps for creating a new diagram type or modifying a chosen one:

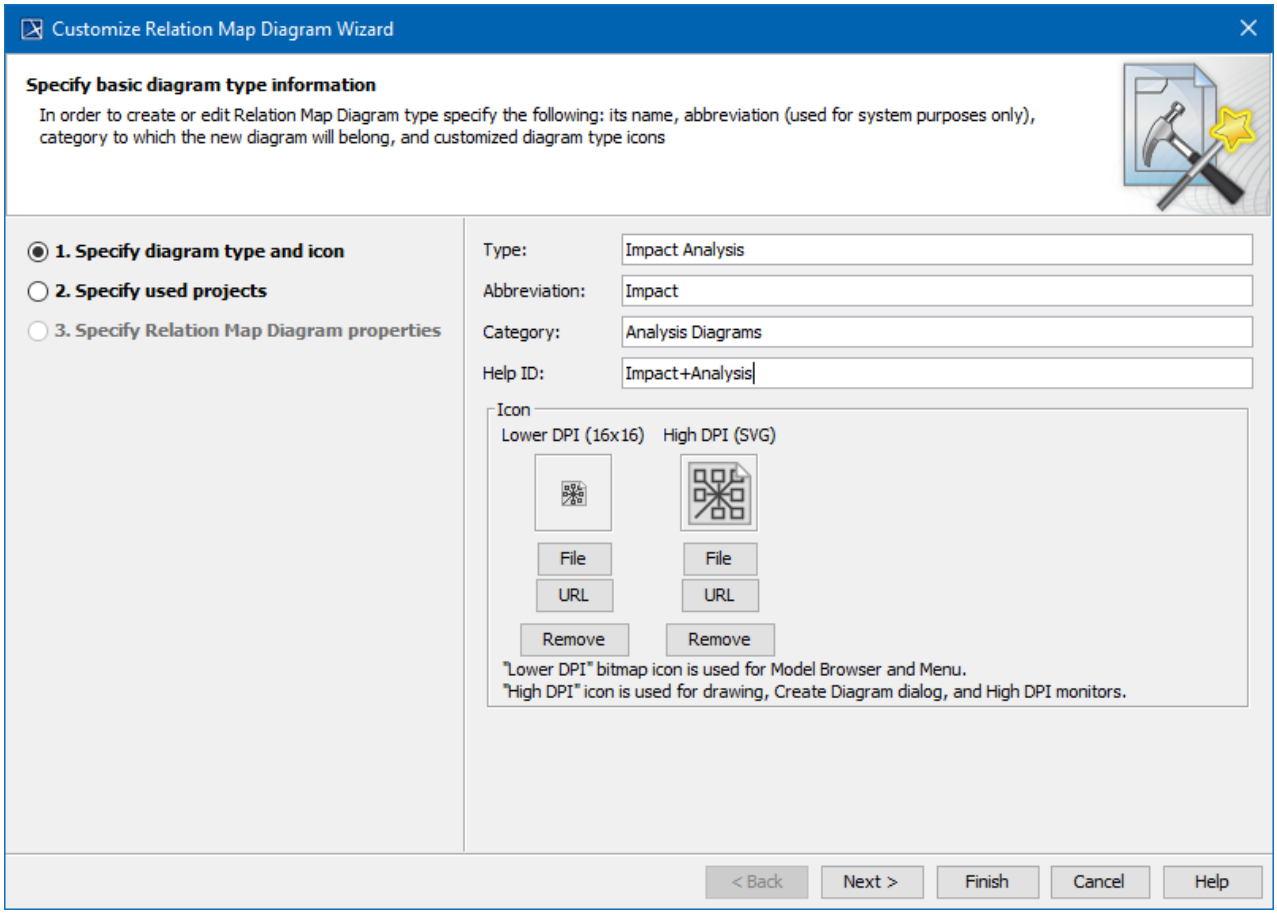
- Step 1: Specify diagram type and icon(see page 2060)
- Step 2: Specify used projects(see page 2061)
- Step 3: Specify Relation Map Diagram properties(see page 2062)

Step 1: Specify diagram type and icon

To create your own diagram, first specify the following properties:

- Diagram type name (for example, Impact Analysis).
- Abbreviation – a short form of the diagram name. It will be used in Diagram Frames header or Diagram shapes in Content diagrams.
- Category – creates your specific category in the **Diagrams** menu or in the **Create Diagram** command list. You can store all your customized diagrams in this category. If you keep this field empty, the customized diagrams will be added to the **Custom Diagrams** category.
- Help ID – sets a specific Help ID that invokes help or documentation topics.
- Icons – several icons for your custom diagram representation in MagicDraw GUI.

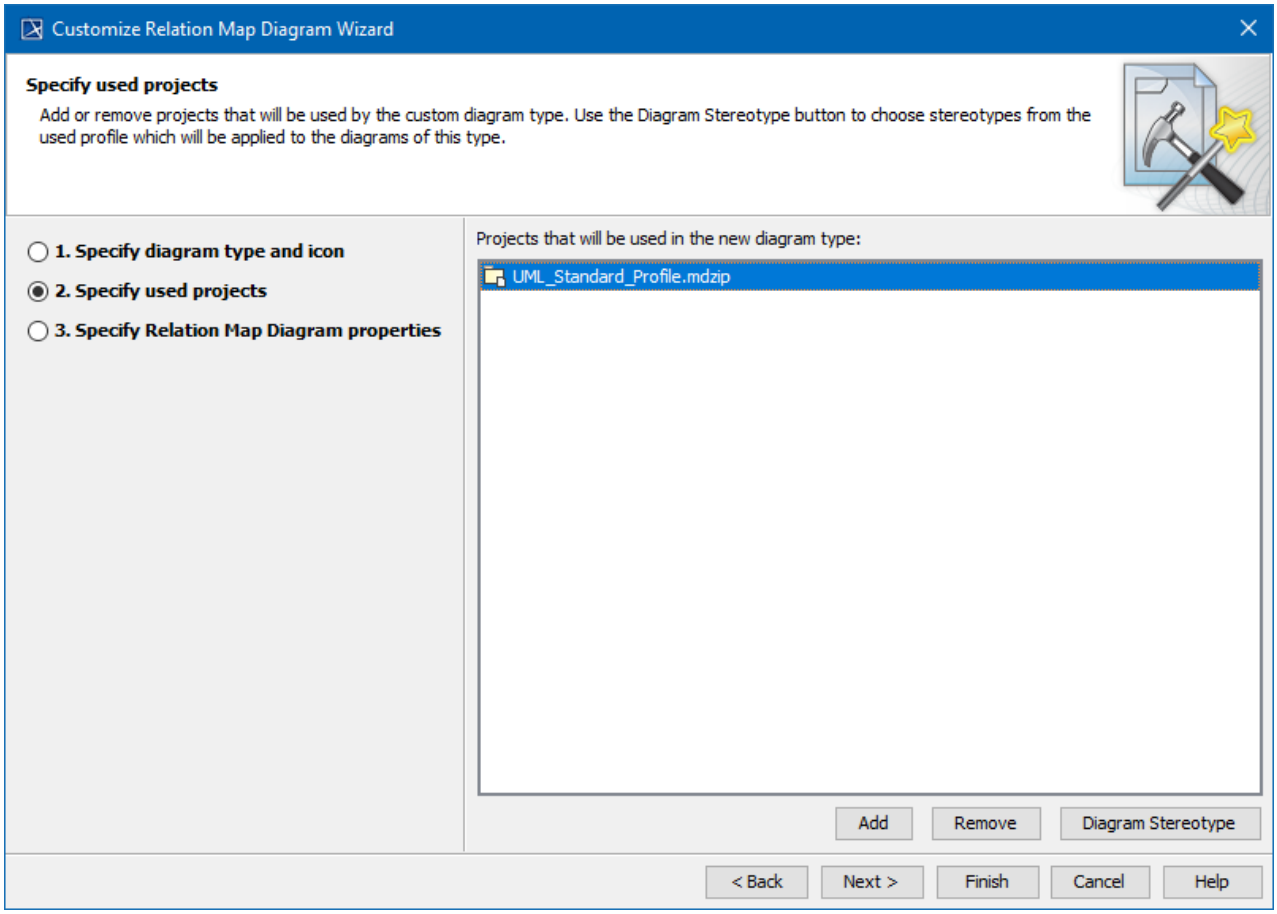
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Step 2: Specify used projects

Custom Diagrams are oriented to a new specific domain, technology or platform, and are often based on a custom profile.

Select the required used projects or profiles.



Do not remove the UML Standard Profile, which is selected by default, from the list. It must be used by any custom diagram type.

You can apply the stereotype for your Relation Map diagram by clicking the **Diagram Stereotype** button.

Step 3: Specify Relation Map Diagram properties

This step allows you to specify the appearance of the new relation map diagram.

Property Name	Description
Cut Element Names	Set to *true* to cut element names that are too long and make the Relation Map difficult to read.
Include Subtypes	Set to *true* to display subtypes of the selected elements. For example, if a class is selected, then all its subtypes (such as component or custom subtypes like SysML block) are displayed.
Show Element Numbers	Set to *true* to show the element number before its name.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Property Name	Description
Show Legend	Set to *true* to display the relationship legend.
Show Parameters	Set to *true* to show the parameters' signatures of operation and behavior on symbols.
Show Single Node Per Element	Set to *true* to show a single node per element when the graph has more than one representation of that element in the Relation Map.
Show Applied Stereotypes	Set to *true* to display stereotypes applied to the elements.
Relation Criterion	Click ... to select and specify criteria in order to represent relations between elements.
Element Types	Click ... to select specific element types to be displayed.
Layout	Select *Tree* or *Radial* from the drop-down menu to customize the layout of your Relation Map.
Depth	Type the number to specify the level of the relation map branches that will be expanded automatically.
Filter Area Expanded	Set to *true* to show the filter area expanded.
Make Selected As Context	Set to *true* to make the selected element the core of the Relation Map.
Show Full Types	Set to *true* to show full element types in the Relation Map.
Show Relation Styles	Set to *true* to show relation line styles.
Description Area	Specify the description of the Relation Map.
Possible Context Types	Click ... to specify the possible context types.
Show Scope	Set to *true* to display the Relation Map scope, that is, to see the fragment of the model (or whole model) from which the Relation Map is built.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Property Name	Description
Show Context	Set to *true* to show the Context field in the diagram toolbar.
Show Element Type	Set to *true* to show the element type specification box in the diagram toolbar.
Show Relation Criterion	Set to *true* to display relation criteria in the diagram toolbar.

Creating new generic table type

The **Customize Generic Table Wizard** contains the following steps for creating a new diagram type or modifying a chosen one:

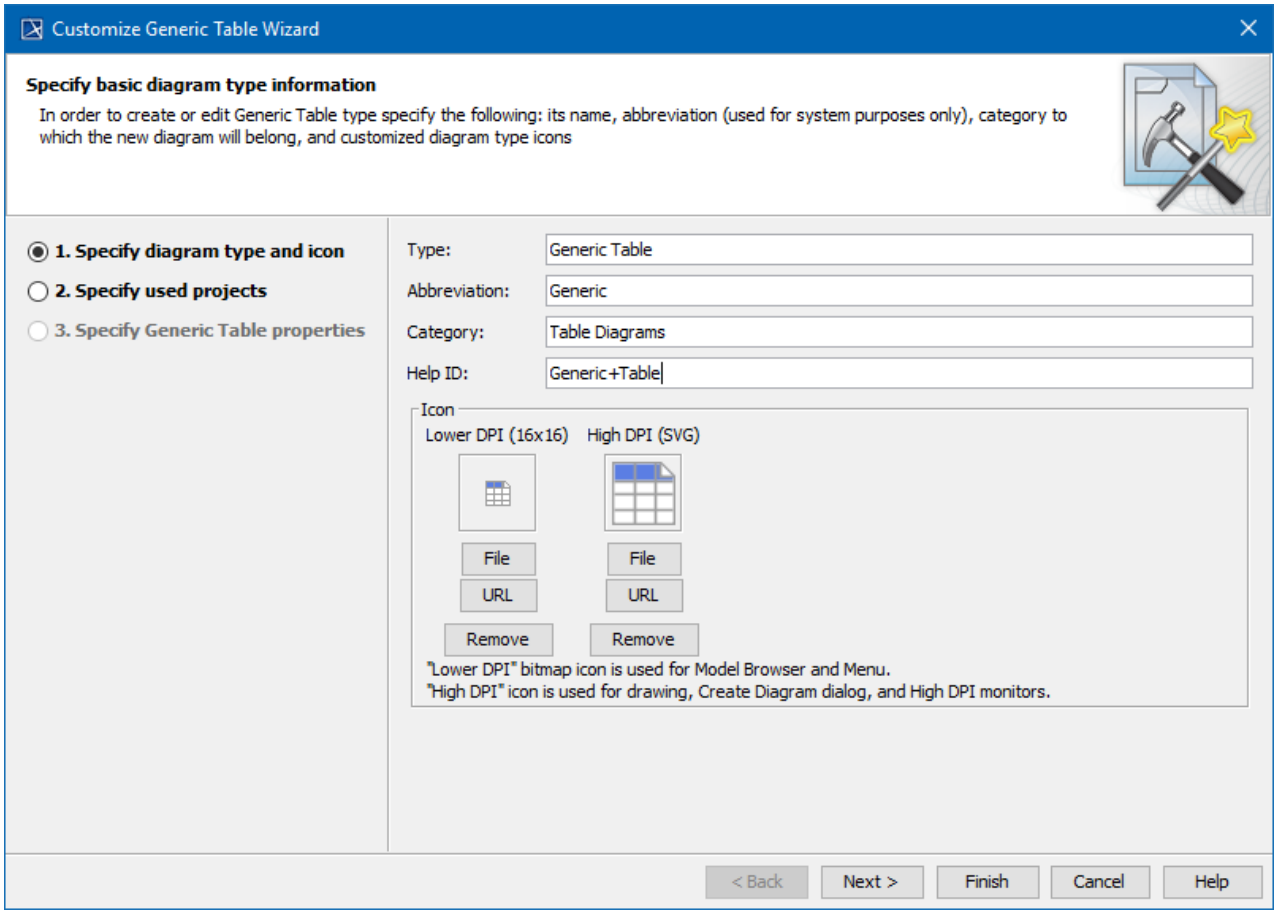
- Step 1: Specify diagram type and icon(see page 2064)
- Step 2: Specify used projects(see page 2065)
- Step 3: Specify Generic Table properties(see page 2066)

Step 1: Specify diagram type and icon

To create your own diagram, first specify the following properties:

- Diagram type name (for example, Generic Table).
- Abbreviation – a short form of the diagram name. It will be used in Diagram Frames header or Diagram shapes in Content diagrams.
- Category – creates your specific category in the **Diagrams** menu or in the **Create Diagram** command list. You can store all your customized diagrams in this category. If you keep this field empty, the customized diagrams will be added to the **Custom Diagrams** category.
- Help ID – sets a specific Help ID that invokes help or documentation topics.
- Icons – several icons for your custom diagram representation in MagicDraw GUI.

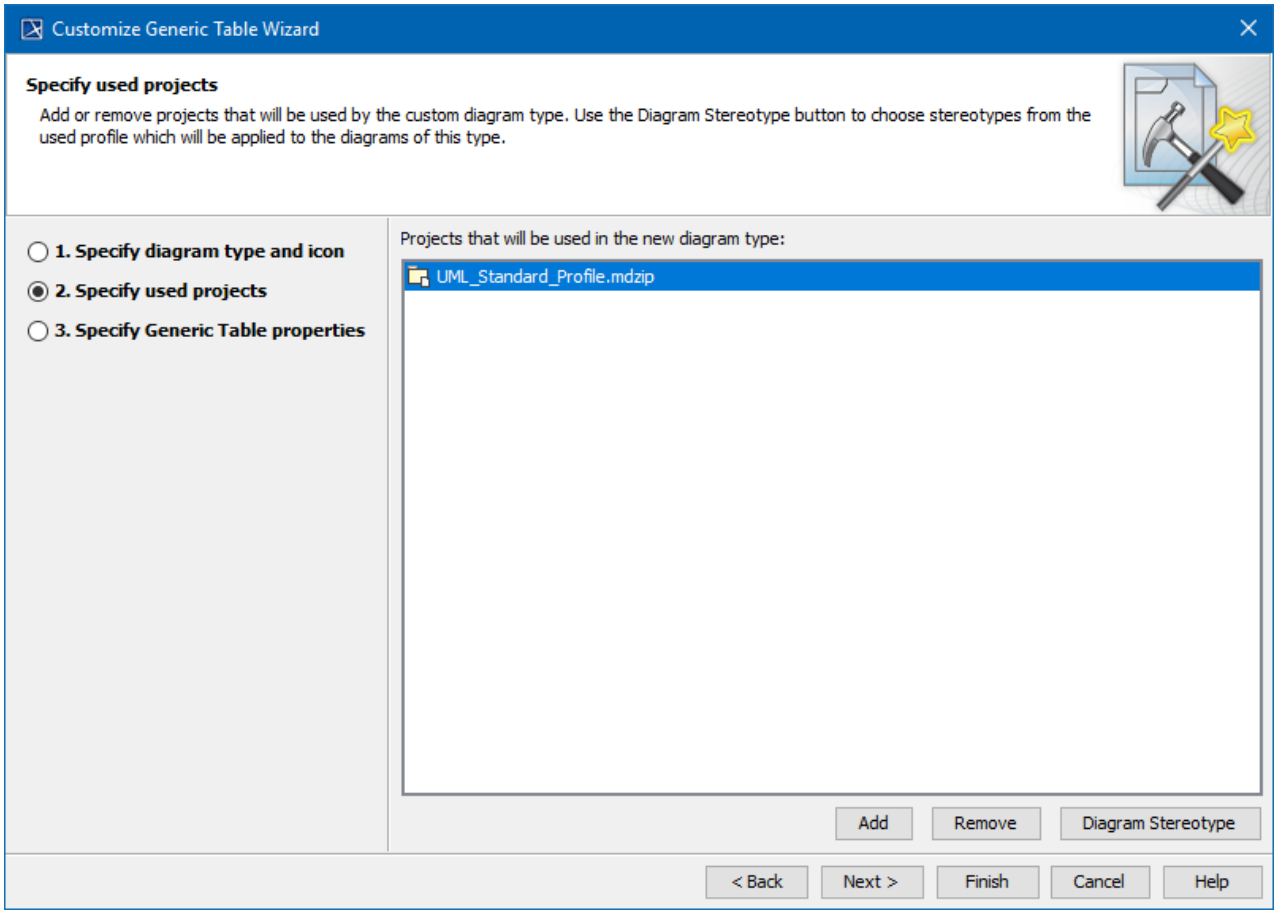
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Step 2: Specify used projects

Custom Diagrams are oriented to a new specific domain, technology or platform, and are often based on a custom profile.

Select the required used projects or profile.



> Do not remove the UML Standard Profile, which is selected by default, from the list. It must be used by any custom diagram type.

Click the **Diagram Stereotype** button to apply the specific stereotype to your Generic Table.

Step 3: Specify Generic Table properties

This step allows you to specify the appearance of a new Generic Table.

Property Name	Description
Show Column Icons	Set to true to show icons in column headers.
Custom Columns	Click ... to select and specify operations for calculating/gathering values of the custom column.
Display Mode	Click ... to select one of the visualization options: *Complete tree* (the table represents all elements with their owners), *Compact tree* (the table combines grouping elements), or *List* (selected model elements are organized into a plain list).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Property Name	Description
Show Scope As Root	Set to *true* to show scope as a root node in the tree display mode.
Show Element Number	Set to *true* to display the element number in the name column.
Row Element Type	Click ... to select the type of elements to be shown as row elements.
Query	Click ... to select and specify operations to query the model.
Types Include Subtypes	Set to *true* to include subtypes of the selected elements.
Columns	Click ... to select columns to appear in the Generic Table.
Row Filters	Set to *true* to display filtered rows in the Generic Table.
Show Detailed Column Name	Set to *true* to show classifiers next to properties in the column header.
Column Width	Click to adjust the width of columns.
Default Row Elements Owner	Click to select the default owner of row elements.
Hide Columns	Click ... to select columns that will not be displayed in the Generic Table.
Show Full Path	Set to *true* to show the full path of elements.
Show Filter	Set to *true* to show the filter applied in the Generic Table.
Show Element Type	Set to *true* to show the element type specification box in the criteria area of the Generic Table.
Show Scope	Set to *true* to display the scope in the criteria area of the Generic Table.
Legends	Click ... to select a legend for the Generic Table.
Description Area	Specify the description of the Generic Table.

Domain Specific Language Customization

DSL (Domain Specific Language) customization is a model-driven approach, based on UML profiling. Our modeling tool provides the ability to use DSL engine for adapting domain specific profiles to create your own custom specification windows, custom real-time semantic rules and others. In other words, you can create specialized domain specific tool and hide UML underneath.

The modeling tool's DSL customization engine is able to process user defined rules for DSL elements and reflect this in MagicDraw GUI and diagrams behavior.

The topics in this section show you how to create a new DSL element, how to customize its Specification window, shortcut menu, how to create customization rules for relationships drawing, how to customize possible owned elements, how to create numbering customization, how to extend metamodel with derived properties and others.

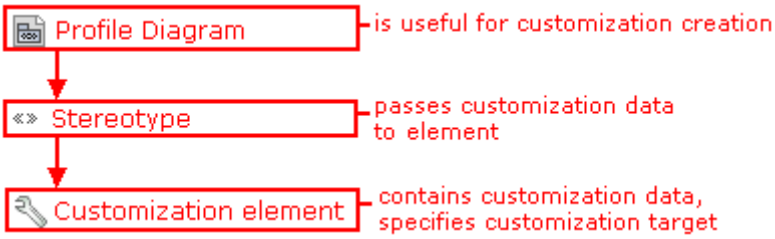
Related Pages

- Creating Customization Data(see page 2068)
- Using Customization Data(see page 2093)

Creating Customization Data

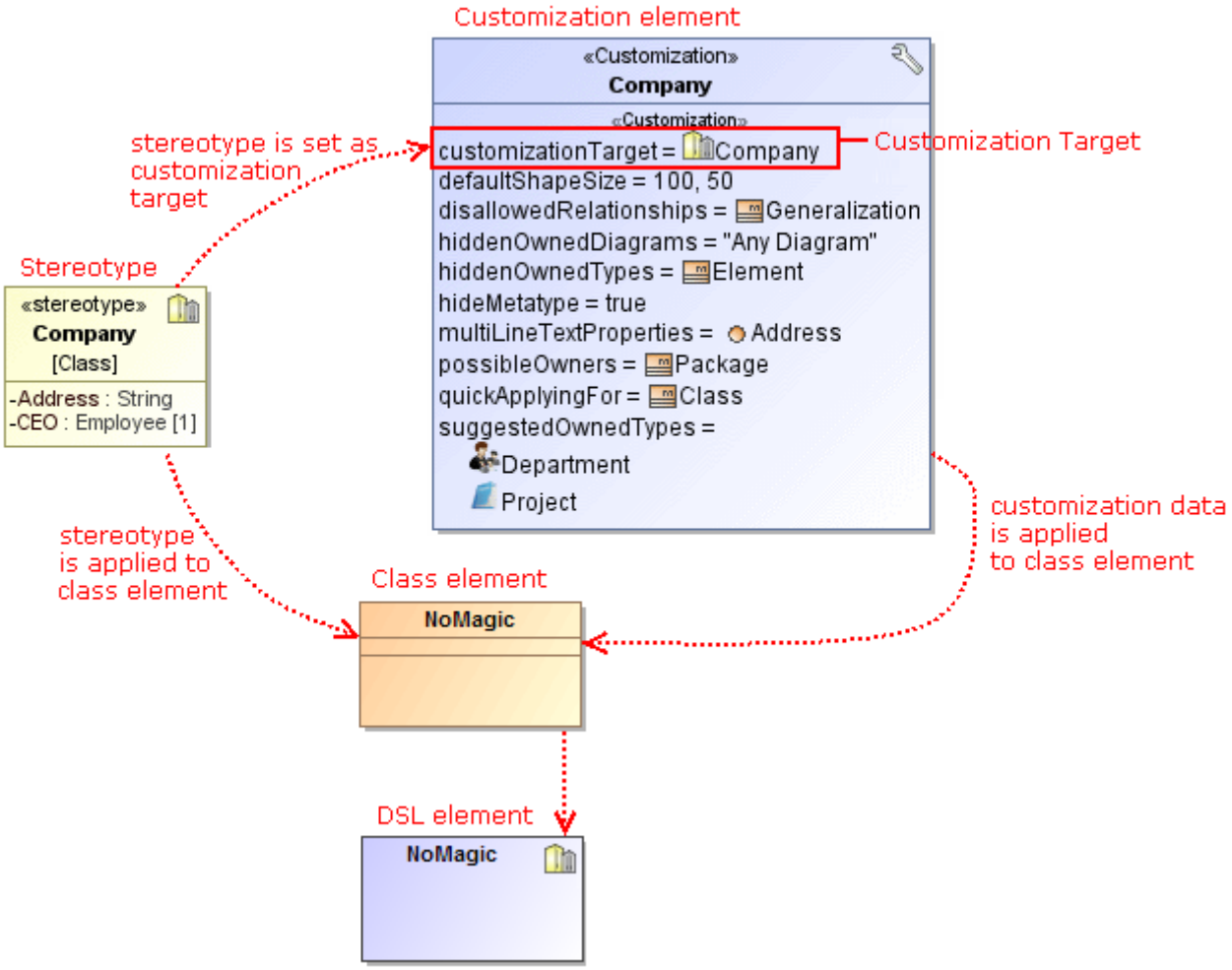
In our modeling tool, customization creation begins by creating the following components:

- Profile Diagram(see page 324)
- Stereotype(see page 856)
- Customization element.



Compounds of DSL customization creation

In the following figure, you can see an example of how customization data is passed to DSL elements through stereotypes.



Example of applying customization rules to DSL element

In the preceding figure, you can see the customization element, stereotype, class element, DSL element and relations between them. The *Company* stereotype element is set as customization target in the *Company* customization element. In addition, the *Company* stereotype is applied to the class element. The customization data from the *Company* customization element is passed to the class element. Thus, the class element becomes the DSL element. DSL element properties will appear in the Specification window(see page 913) and the Properties panel of the DSL element as regular properties.

Related pages

- Stereotype(see page 856)
- Profile diagram(see page 324)
- Specification window(see page 913)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Creating your First Customization

We will demonstrate the recommended step-by-step instructions on how to customize an element, how to begin to create customization, how to create customization data, and how the customization data will be represented on the DSL element.

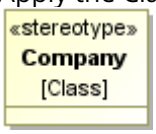
1. Create a new project(see page 105) and name it *Organization_Stereotypes.mdzip*.
2. Create a Profile element and name it *Stereotypes*.

   Stereotypes
   The purpose of the Profile named *Stereotypes* is to store all the stereotypes. Later we will share the package in order to use the stereotypes in the customization project.

3. Create a Profile Diagram(see page 483) and name it *Stereotypes*.

   The Profile Diagram pallet contains all the necessary for the customization creation buttons.

4. In the Profile Diagram, create a Stereotype element(see page 856) and name it *Company*.
5. Apply the Class metaclass to the *Company* Stereotype.

   - You can see how customization rules will be passed to the *DSL* element by using the stereotype.
   - We selected the *Class* metaclass, because the «Company» stereotype will be applied to a class element. In other words, the type of the DSL element will be the class.
   - You can also assign an icon to the stereotype. The icon will be represented on the DSL element.

6. Share the stereotypes package(see page 239). The *Organization_Stereotypes.mdzip* project becomes a shared project.

   Only the content of the shared *stereotypes* package will be visible in the project that will use the project.

7. Save the *Organization_Stereotypes.mdzip* project.
8. Create a new project(see page 105) and name it *Organization_Customizations.mdzip.*
9. Create a package(see page 800) and name it *customizations*.

   The purpose of the *customizations* package is to store all the customization data. Later we will share the package.

10. Create a profile diagram(see page 483) and name it *Customizations*.

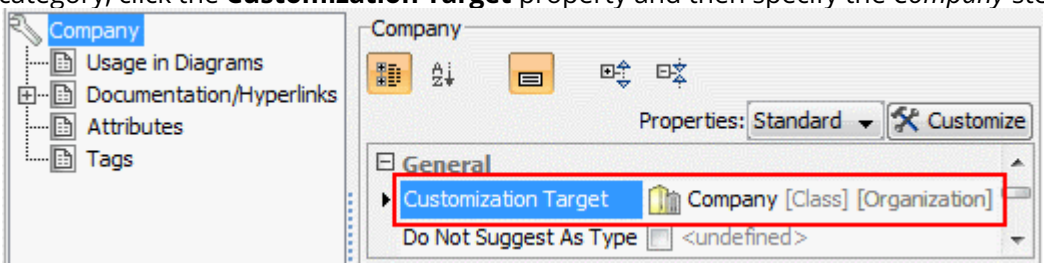
    The *Customizations* diagram is intended for customization elements creation.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

11. Create a customization element and name it Company.

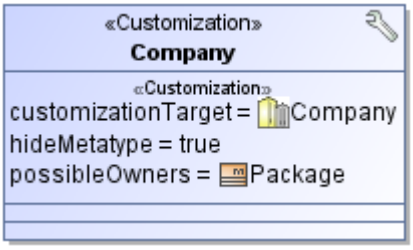
> On a diagram, you can see that the customization element is a class with the applied «*Cu stomization*» stereotype.

12. Use(see page 242) the *Organization_Stereotypes.mdzip* project in the current project.
13. Open the *Company* customization element Specification window(see page 913), under the **General** category, click the **Customization Target** property and then specify the *Company* stereotype.



> - To assign a customization target quickly, select the *Company* stereotype in the Model Browser and drag it to the shape of the *Company* customization element on the diagram pane.
> - In the **Customization** Specification window, you can specify customization data that will be passed to the DSL element.

14. In the *Company* customization element Specification window, under the **General** category, set the **Hide Metatype** check box to true. If the value is *true*, the element acts like a new standard element type in the modeling tool.
15. In the *Company* customization element Specification window, under the **Owned Elements** category, click the **Possible Owners** property and then select the *Package* metaclass.



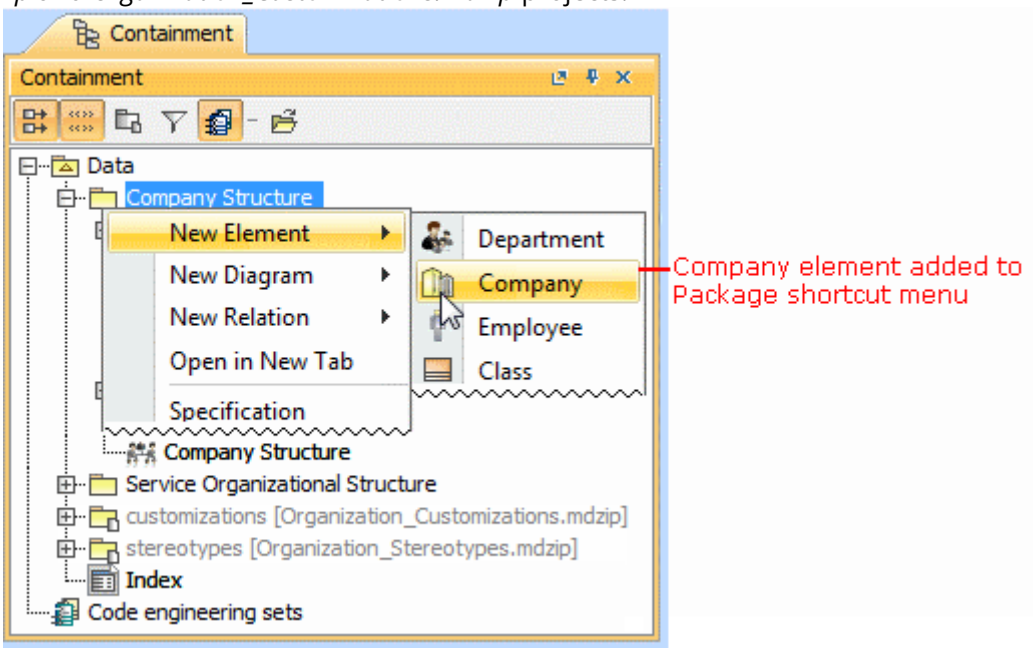
> - The DSL element will be added to the package shortcut menu, under the **Create Element** command.
> - After click on the **Possible Owner** property, the element Selection dialog appears. In this dialog, by default metaclasses are not displayed. To display the *Package* metaclass, click the *Include meta classes into search results* button.

16. Share the *customizations* package. The *Organization_Customizations.mdzip* project with the shared *customizations* package becomes a shared project.
17. Save the *Organization_Customizations.mdzip* project.
18. Create a project and name it *Organization*.
19. Use the *Organization_Customizations.mdzip* project in the *Organization* project.

> The *Organization_Stereotypes.mdzip* project will be used in the *Organization* project automatically, because the *Organization_Customizations.mdzip* project uses the *Organization_Stereotypes.mdzip* project.

20. In the *Organization* project, create a Package and in that package, create a class element and name it *NoMagic*.
21. Apply the *Company* stereotype to the *NoMagic* class element.
22. Save and reopen the project.

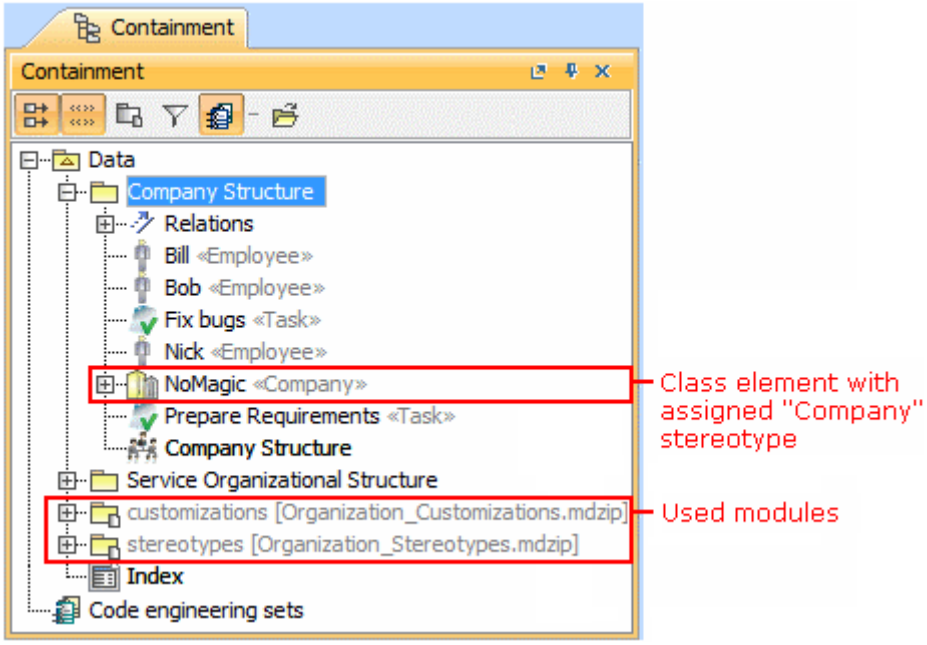
In the following figure, you can see the Containment tree of the project with the used *Organization_Stereotypes.mdzip* and *Organization_Customizations.mdzip* projects.



Structure of a project that use other projects

In this example, the «*Company*» stereotype is applied to the *NoMagic* class element. You can see that the *Company* element is added to the package shortcut menu.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Example of the "Possible Owner" property realization

> **Create Element**
> If you do not want the new DSL element to appear on the **Create Element** menu, you can hide the appropriate command when customizing your modeling tool perspective(see page 68).

Related pages

- Rules of Stereotypes that cannot be Allowed to Apply(see page 2092)
- Customization Elements(see page 2073)
- Customization Element Properties(see page 2090)
- Creating Customization Data(see page 2068)
- Project partitioning(see page 236)

Customization Elements

Use the customization element properties to customize:

- the DSL element Specification window(see page 913)
- the DSL element shortcut menu
- allowed to draw relationships
- allowed to drag-and-drop elements
- other

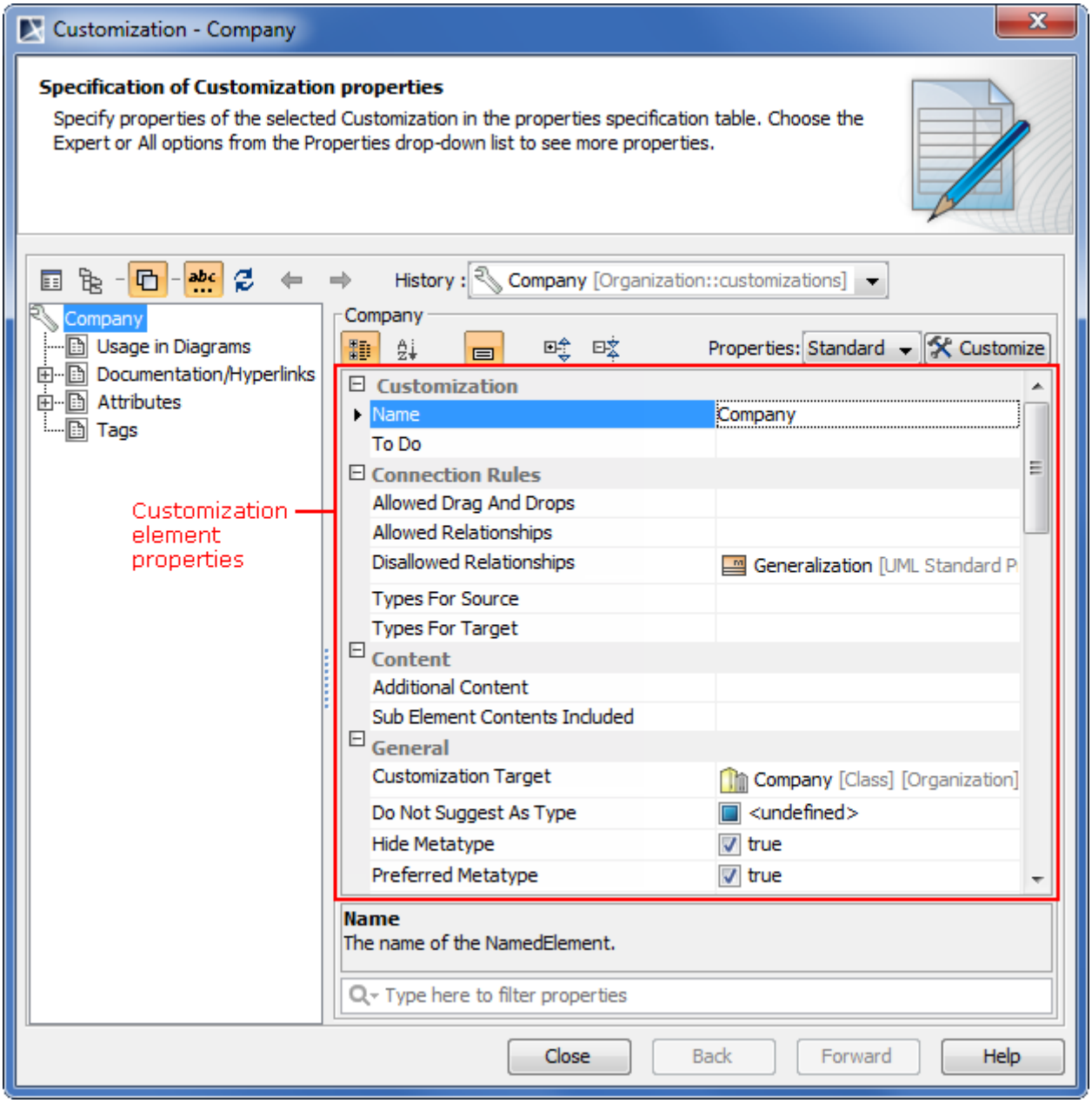
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To specify a new property

1. In a profile diagram(see page 324), create a customization element.
2. Open the **Customization** element Specification window.
3. In the right side of the window, select a property and modify it depending on its type.

> **Tip!**
> The properties in the **Customization** Specification window are listed in the following two locations:
> 1. General specification pane
> 2. **Tags** property group
>
> Specify properties in either location. Changes made in the general specification pane reflect in the **Tags** property group and vice versa.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Customization element Specification window, list of properties

Related Pages

- Customization Element Properties Description(see page 2075)

Customization Element Properties Description

Property	Description
Customization	
Name	Specifies the name of the Customization element.
To Do	Specifies the To Do text of the Customization element. Use the To Do property for adding notes about work to be done. Then, you can quickly search for the elements that have the To Do properties defined(see page 229).
Connection Rules	
Allowed Drag and Drop	Specifies elements that describe allowed drag-and-drop operations for customized elements(see page 2108).
Allowed Relationships	Specifies the types of relationships allowed to connect to the DSL element. For a more detailed description, see Creating Custom Rules for Relationships (see page 2113).
Disallowed Relationships	Specifies types of relationships not allowed to connect to the DSL element. For a more detailed description, see Creating Custom Rules for Relationships (see page 2113).
Types For Source	Specifies metaclasses or stereotypes allowed to connect as the source of the relationship. Types cannot conflict with UML permitted types for this relationship. The Types For Source property is intended for relationships only. The Customization Target property can either be the stereotype that extends the metaclass of the relationship, or the metaclass of the relationship itself. For a more detailed description, see Creating Custom Rules for Relationships (see page 2113).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Property	Description
Types For Target	Specifies metaclasses or stereotypes allowed to connect as targets of the relationship. Types cannot conflict with UML permitted types for this relationship. The Types For Source property is intended for relationships only. That is, the Customization Target property can either be the stereotype that extends the metaclass of the relationship, or the metaclass of the relationship itself. For a more detailed description, see Creating Custom Rules for Relationships (see page 2113).
Content	
Additional Content(see page 2085)	Stores a rule (in a form of derived property expression) for gathering the additional content of the DSL element.
Sub Element Contents Included(see page 2088)	Specifies whether the content of the DSL element includes only directly related elements or both directly and indirectly related ones.
General	
Customization Target(see page 2081)	Specifies a stereotype or a metaclass to customize. See a case study in Creating your First Customization (see page 2070).
Do No Suggest As Type (see page 2082)	Excludes customized elements from being suggested as a type in the list dialogs.
Hide Metatype(see page 2083)	Indicates whether the DSL element will be recognized as a new type.
Preferred Metatype(see page 2115)	Specifies a preferred metatype, if there is more than one Customization Target (see page 2081) defined in the Customization Specification window. For more information, see Creating Property Groups and Subgroups (see page 2126).
Quick Applying For	Specifies an element to which the created customization can be applied directly from the element shortcut menu. For more information, see Rules of Stereotypes that cannot be Allowed to Apply (see page 2092).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Property	Description
Customization Target(see page 2081)	Specifies whether or not the name auto completion list is displayed when entering name for the DSL element on the diagram. If the Do Not Suggest Name Auto Completion property value is true, then the auto completion drop-down list is not displayed.
Model Initialization	Specifies stereotypes that can be automatically applied to elements (source or target) after connecting those elements with customized relationships. The condition is that the customization target must be a stereotype for relationship. For more information, see Initializing Custom Model (see page 2116).
Apply to Source	Specifies stereotypes that will be applied on the source element of the relationship after connection. The customization target should be a stereotype of relationship. For more information, see Initializing Custom Model (see page 2116).
Apply to Target	Specifies stereotypes that will be applied to the target element of the relationship after connection. The customization target should be a stereotype of relationship. For more information, see Initializing Custom Model (see page 2116).
Super Types	Specifies the element that will be the super type of the DSL element. The generalization relationship will be created from the DSL element to the specified super type. For more information, see Required Generalization or Interface realization(see page 744).
Naming	
Category	Creates a category to group DSL elements in various shortcut menus, if the Hide Metatype property is true. For more information, see Creating a DSL Element from the Customized Category in the Shortcut Menu(see page 2118).
Keyword	Defines a keyword to be displayed instead of the applied stereotype name.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Property	Description
Representation Text	Defines the text that will be used in status bars, dialogs, logs, and others. See an example of the Representation Text property usage in a drag-and-drop operation customization(see page 2108).
Do Not Suggest Name Auto Completion (see page 2083)	Specifies a short name for the DSL element. When creating the DSL element, the short name will be added automatically.
Owned Elements	
Hidden Owned Diagrams	Specifies the diagram types that will hidden for the DSL element. For example, if the property value is Any Diagram, all diagrams will be hidden and unavailable from the DSL element shortcut menu, under the Create Diagram command in the Model Browser. For more information, see Customizing Possible Owned Elements (see page 2119).
Hidden Owned Types	Specifies the metaclass or the stereotype that will be hidden for the DSL element. For example, if the property value is the Element metaclass, then all standard UML elements will be hidden in the DSL element shortcut menu, under the Create Element command in the Model Browser. For more information, see Customizing Possible Owned Elements(see page 2119) .
Possible Owners	Specifies the metaclass or stereotype that can own the DSL element (specifies types of elements that can be owners of the DSL element). For example, to create a DSL element inside standard packages in the Model Browser, you must specify the Package metaclass as the Possible Owners property value. For more information, see an example in Creating Your First Customization(see page 2070) and for a more detailed description, see Customizing Possible Owned Elements(see page 2119) .
Suggested Owned Diagrams	Specifies types of diagrams that will appear in the DSL element shortcut menu when creating possible inner diagrams. For more information, see Customizing Possible Owned Elements(see page 2119).
Suggested Owned Types	Specifies types of elements that can be owned by the DSL element. For example, if the property value is Section, then the Section elements can be owned by the customized element. For more information, see Customizing Possible Owned Elements(see page 2119).
Properties	
Check Spelling	Specifies the properties for which the spell checker is enabled.

Property	Description
Help ID	Defines a string value referring to an element-specific help topic(see page 2121), which can be opened for the DSL element.
In Shortcut Menu	Specifies properties that will be displayed in the DSL element shortcut menu. For more information, see Quick property editor(see page 2308) .
Multi Line Text Properties	Specifies properties for which multi line text editor will be used. Note only properties of string type is listed in editor of this property. For more information about creating properties in the DSL element Specification window, see Customizing Specification Window.(see page 2093) For more information about editing property values in the Specification window, see the Editing Property Values(see page 921).
Show Properties When Not Applied	Customizes the stereotype properties to be visible in the element Specification window even if the stereotype is not yet applied on the element. For more information, see Always Visible Properties(see page 2095) .
Show Properties When Not Applied Limited By Element Type	Customize the stereotype properties to be visible in the DSL element Specification window dependent on another stereotype or metaclass. For more information, see Always Visible Properties(see page 2095) .
Show Properties When Not Applied Limited By Profile Application	Customize the stereotype properties to be visible in the DSL element Specification window dependent on the corresponding profile application. For more information, see Always Visible Properties(see page 2095) .
Standard Expert Configuration (see page 2089)	Specifies properties visibility mode in DSL element Specification window.
Used UML Properties	Allows to select UML element properties that will be visible in DSL element Specification window. For more information, see Using Standard UML Properties(see page 2094).
Symbol	
Default Shape Size	Defines default size of the shape when the symbol is created on a diagram. For more information, see Setting default symbol size(see page 2124)

Property	Description
Symbol Standard Expert Configuration (see page 2089)	Specifies properties visibility mode in DSL element Symbol Properties dialog(see page 975), in the symbol shortcut menu and in menus that open after clicking the Compartments or Create Element smart manipulators.
Properties Displayed in Compartments(see page 2090)	Allows to select the specific properties to be displayed in compartments on shapes.

Related pages

- Specification window(see page 913)

Customization Target

The **Customization Target** property is a mandatory property in customization creation. It specifies the stereotype or metaclass you want to customize.

To define a stereotype as a customization target

1. In the **Customization** Specification window(see page 913), click the **Customization Target** property.
2. In the property specification cell, click the button. The element Selection dialog(see page 1031) appears.
3. Select the stereotype.
4. Click **OK** to close the dialog.
5. Click **Close**.
6. Apply the selected stereotype to the desired class.
7. Reopen the project.

To define a metaclass as a customization target

1. In the **Customization** Specification window, click the **Customization Target** property.
2. In the property specification cell, click the button. The element Selection dialog(see page 1031) appears.
3. Clear the **Apply Filter** selection. All metaclasses will be included in the list.
4. Select the *Class* metaclass.

   Tip
   All standard metaclasses are stored in the UML Metamodel. You can also filter elements(see page 233) to find a desired element quicker.

5. Click **OK** to close the dialog.

6. Click **Close**.
7. Reopen the project.

Related Pages

- Creating Customization Data(see page 2068)
- Using Customization Data(see page 2093)
- Customization Elements(see page 2073)
- Creating your First Customization(see page 2070)
- Creating and specifying derived properties(see page 2148)
- Defining expressions(see page 2152)
- Hide Metatype(see page 2083)

Do Not Suggest As Type

Excludes elements from being suggested as a type in the list dialogs.

To exclude a DSL element from the list dialogs

---

1. In the **Customization** Specification window(see page 913) , click the **Customization Target** property.
2. Click the ... button. The element Selection(see page 2300) dialog appears.
3. Select a stereotype or a metaclass.
4. Click **OK** to close the dialog.
5. Set the **Hide Metatype** property to *true* .
6. Set the **Do Not Suggest As Type** property to *true*.
7. Click Close.
8. Reopen the project. Elements or metaclasses selected in the third step of this procedure are not listed in the list dialogs. For example, in the **Property** Specification window(see page 913), click the **Type** property and then the ... button. In the element Selection(see page 2300) dialog, the elements or metaclasses selected as **customization targets** will not be displayed.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related Pages

- Creating Customization Data(see page 2068)
- Using Customization Data(see page 2093)
- Customization Elements(see page 2073)
- Customization Element Properties Description(see page 2075)
- Customization Target(see page 2081)
- Creating your First Customization(see page 2070)
- Creating and specifying derived properties(see page 2148)
- Defining expressions(see page 2152)
- Hide Metatype(see page 2083)

Hide Metatype

Indicates whether the customized stereotype is recognized as a new type. If the **Hide Metatype** property value is *false*, the DSL element is *not* considered as a new type element and will not be displayed in the shortcut menus (see the **Possible Owners** property).

To define the hide metatype property

1. In the **Customization** Specification window(see page 2093), click the **Customization Target** (see page 2081) property and then select a customization target.
2. Set the **Hide Metatype** property to *true*.
3. Reopen the project.

Related Pages

- Creating Customization Data(see page 2068)
- Using Customization Data(see page 2093)
- Customization Elements(see page 2073)
- Customization Element Properties Description(see page 2075)
- Customization Target(see page 2081)
- Creating your First Customization(see page 2070)
- Creating and specifying derived properties(see page 2148)
- Defining expressions(see page 2152)

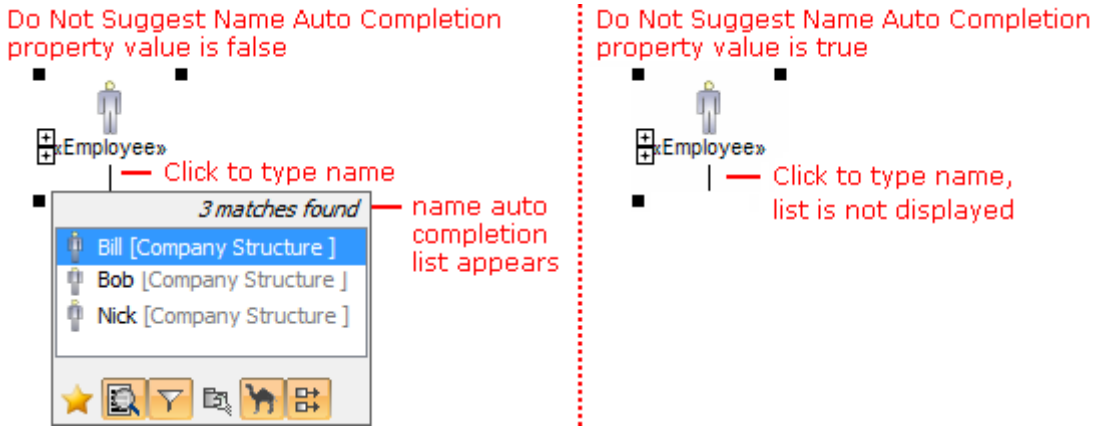
Do Not Suggest Name Auto Completion

The property is useful when you do not want to display the name auto completion list for DSL elements typically displayed only in one diagram, or if the names of the DSL elements are full sentences or fragments of sentences.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To hide the name auto completion list

1. In the Customization Specification window(see page 2093), click the Customization Target(see page 2081) property and then select a customization target.
2. Set the **Do Not Suggest Name Auto Completion** property to *true*.
3. Reopen the project.



Example of name auto completion list that appears when typing a DSL element name.

Related Pages

- Creating Customization Data(see page 2068)
- Using Customization Data(see page 2093)
- Customization Elements(see page 2073)
- Customization Element Properties Description(see page 2075)
- Customization Target(see page 2081)
- Creating your First Customization(see page 2070)
- Creating and specifying derived properties(see page 2148)
- Defining expressions(see page 2152)

Abbreviation

The **Abbreviation** property enables creation of a DSL element with an abbreviated name. You can enter the default name while specifying the customization element.

To define an abbreviation

1. In the **Customization** Specification window(see page 913), click the **Customization Target** property and then select a stereotype or metaclass for which the abbreviation will be used.
2. Click the **Abbreviation** property and then type the text.

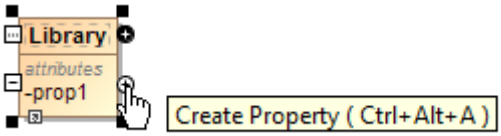
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

3. Reopen the project.

Case Study

We will demonstrate how to add an abbreviation for the class attribute.

1. Create a profile diagram(see page 324).
2. Create a customization element named *PropertyAbbreviation*.
3. In the *PropertyAbbreviation* customization Specification window(see page 2093), click the **Customization Target** property and then select the *Property* metaclass.
4. Under the **Naming** category, click the **Abbreviation** property and type a short name for the property, for example, *prop*.
5. Close the Specification window. Reopen the project.
6. Create a class(see page 297) in the class diagram.
7. Create an attribute in the class. The property with the predefined name (or abbreviation) is created.



Creating an attribute in a class.

Related Pages

- Creating Customization Data(see page 2068)
- Using Customization Data(see page 2093)
- Customization Elements(see page 2073)
- Customization Element Properties Description(see page 2075)
- Customization Target(see page 2081)
- Creating your First Customization(see page 2070)
- Creating and specifying derived properties(see page 2148)
- Defining expressions(see page 2152)

Additional Content

Use the **Additional Content** property to specify custom content for a desired element in addition to the standard one defined by the UML ownership. The custom content of an element is a collection of elements related to this element by some user-defined rule, such as some type of relationship.

To customize the content of a DSL element

1. Create a customization element for the DSL element or select an existing one.
2. Create a derived property(see page 2148) for this customization element.

3. Specify a rule for gathering the additional content of the DSL element by creating an expression (see page 2152)for the derived property.
4. Define the derived property as a value of the **Additional Content** property.
5. Find the property in the general property group of the Specification window(see page 913) of the customization element, under the **Content** category.

The custom content of the element will then be visible in the following places:

- Containment tree(see page 32)
- **Find** dialog(see page 217)
- Element Selection dialog(see page 1031)
- Rows and columns of a dependency matrix(see page 334)

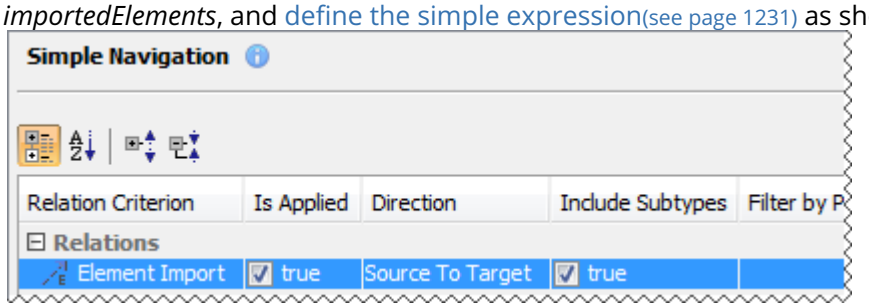
Case Study

Let's say we need UML Packages to display not only their owned elements, but also the elements related to the Packages by the Element Import relationship. In order to obtain this, we must specify the rule for gathering the custom content of the Package. Then we will analyze the contents of the customized Package in the Containment tree.

To specify the rule for gathering the custom content of the Package

---

1. Create a profile diagram(see page 324) and add a customization element name (e.g., *PackageCustomization)* to it.
2. Define the Package metaclass as the customization target(see page 2081) of the customization element.
3. Create a derived property specification(see page 2148) for the customization element, name it *importedElements*, and define the simple expression(see page 1231) as shown in the following figure.

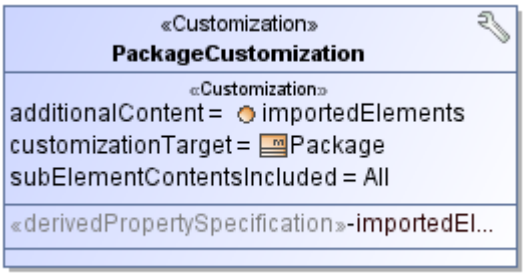


4. In the general property group of the Specification window of the customization element, under the **Content** category, find the **Additional Content** property and click the cell of the property value.
5. Define the *importedElements* derived property as a value of the **Additional Content** property.

Property

- The **Additional Content** property can store a single value which can be only a derived property specification.
- Select the derived property specification from either the same customization element or another one with more abstract customization target. In this case, for example, this could also be the customization element with the Element metaclass as a customization target.

When you are done, the shape of the *PackageCustomization* element should look like the one depicted in the following figure.

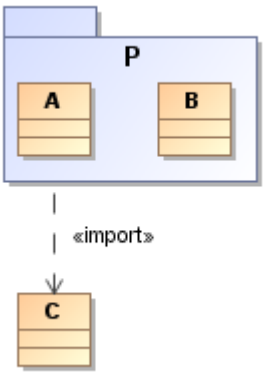


6. Reopen the project.

To see how the **Additional Content** property affects the contents of the Package

---

1. Create a new diagram and draw in it a package named, e.g., *P*.
2. Draw two classes named *A* and *B* in the package.
3. Outside the package, draw a class named *C* and connect it to the package with the Element Import relationship.



4. Now take a look at the Containment tree. The package *P* contains both owned classes *A* and *B* as well as the class *C*, which is not owned, but related to the package *P* by the Import Element relationship.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related Pages

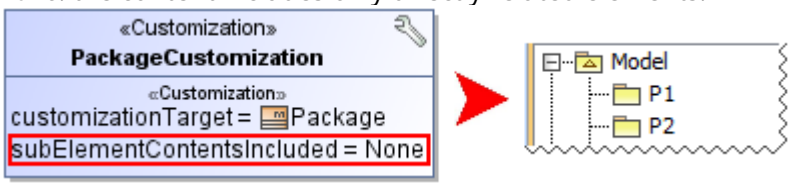
- Creating Customization Data(see page 2068)
- Using Customization Data(see page 2093)
- Customization Elements(see page 2073)
- Customization Element Properties Description(see page 2075)
- Customization Target(see page 2081)
- Creating your First Customization(see page 2070)
- Creating and specifying derived properties(see page 2148)
- Defining expressions(see page 2152)

Sub Element Contents Included

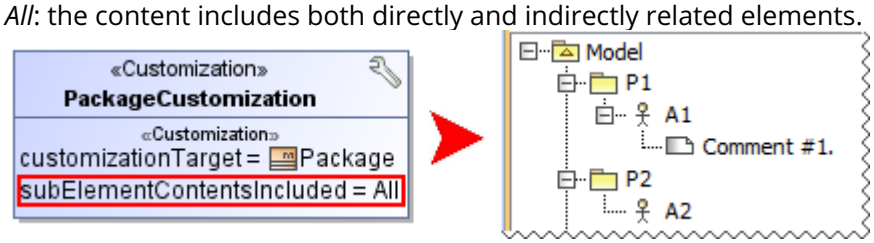
Use the **Sub Element Contents Included** property to specify whether the content of a desired element type includes only directly related elements or both directly and indirectly related ones.

The property value can be either:

- *None*: the content includes only directly related elements.



- *All*: the content includes both directly and indirectly related elements.



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related Pages

- Creating Customization Data(see page 2068)
- Using Customization Data(see page 2093)
- Customization Elements(see page 2073)
- Customization Element Properties Description(see page 2075)

Standard Expert Configuration

In the DSL element Specification window(see page 913), properties are hidden or displayed by using Standard, Expert and All modes.

To customize Standard Expert mode

1. Create a customization element(see page 2070) for the DSL element or select an existing one.
2. In the element Specification window, click the **Standard Expert Configuration** property and then click the ... button.
3. In the **Customize Properties** dialog(see page 72), click to select the mode in which the property will be displayed in the DSL element Specification window.

Tip!

- The **Customize Properties** dialog contains the list of used UML properties and custom properties. You can select which UML properties and DSL specific properties will be displayed in the DSL element Specification window.
- You can also customize property groups. In the **Customize Properties** dialog, property groups are displayed at the end of the list.

Related Pages

- Creating Customization Data(see page 2068)
- Using Customization Data(see page 2093)

Symbol Standard Expert Configuration

Using the **Symbol Standard Expert Configuration** property, you can specify properties visibility mode:

- in the DSL element Symbol Properties dialog(see page 975),
- in the symbol shortcut menu(see page 2106), and
- in menus that open after clicking the Compartments(see page 569) or Create Element smart manipulators.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To customize Standard Expert mode

1. Create a customization element for the DSL element or select an existing one.
2. In the element Specification window(see page 913), click the **Symbol Standard Expert Configuration** property and then click the button.
3. In the **Customize Properties** dialog(see page 72), click to select the mode to display the property in the DSL element Symbol Properties dialog and in other places.
4. Reopen the project.

Related Pages

- Creating Customization Data(see page 2068)
- Using Customization Data(see page 2093)

Properties Displayed in Compartments

Using Properties Displayed in Compartments, you can select the specific properties to be displayed on the element shape.

To customize the properties to be displayed in compartments

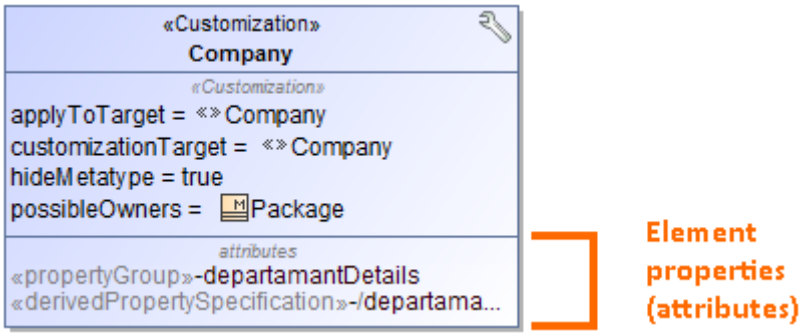
1. Create a customization for your DSL element or select an existing one.
2. In the element Specification window(see page 913), under Symbol, locate **Properties Displayed in Compartments**, and click the ... button.
3. In the **Select Properties** dialog, select the specific properties you want to see displayed in the compartment of your shape(s).
4. Reopen or refresh the project.

Customization Element Properties

Properties
There are two property meanings used in customization elements: properties specifying the element (in the Specification window(see page 913)), and properties as element attributes. The latter is described in this section.

You can create a property in the customization element, then apply the particular stereotype and create customization rules.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



To create a property in the customization element

1. In a profile diagram(see page 324), create a customization element.
2. Select the customization element on the diagram.
3. Click the ⊕ button. The menu with properties appears.

4. Select the desired property described in the following table:

Attribute	Applied Stereotype	Description
Place on Diagram Palette	«placeOnPalettePr operty»	
Meta Property	«metaProperty»	Changes the name of the default meta property.
Property Group	«propertyGroup»	Creates your own groups and subgroups to group the properties either of standard UML metaclasses or stereotypes.
Auto Numbering Property	«AutoNumber»	Creates property used to specify numbering rules for a DSL element.
Derived Property Specification	«derivedPropertyS pecification»	Creates a derived property that is the property whose values are calculated automatically from the other properties' values.

Related pages:

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Creating Customization Data(see page 2068)
- Using Customization Data(see page 2093)

Rules of Stereotypes that cannot be Allowed to Apply

When applying a stereotype to a DSL element, the list of the available stereotypes that may be applied is filtered according to the following rules:

- Only one DSL stereotype (with Hide Metatype=*true*) may be applied to an element. For example, if «Block» is already applied, the «Requirement» stereotype will not appear in the stereotype shortcut menu.
- In order to apply a new DSL stereotype (e.g. when changing from «Requirement» to «businessRequirement»), clear the currently applied stereotype and apply the change. The stereotype shortcut menu will then snow all DSL stereotypes.

  Tip!
  Quickly change the stereotype by converting one element type to other (in the DSL element shortcut menu, click **Refactor** and then **Convert**).

- DSL ownership rules are checked and it is not allowed to apply a stereotype not allowed in the parent element of DSL element.
- DSL relationship rules are checked and it is not allowed to apply a stereotype if a relationship exists and the source or target element violates DSL rules for the existing DSL relationship.
- DSL rules are also checked when drawing new relationships in a diagram.

Related Pages

- Customization Elements(see page 2073)
- Customization Element Properties(see page 2090)
- Creating Customization Data(see page 2068)
- Creating your First Customization(see page 2070)

Creating Inheritance of DSL Customization

DSL customization is inherited when a stereotype is inherited. It is not necessary to modify existing DSL customization when extending the (standard) profile.

For example, to create your own subtype of SysML «Requirement», named, let's say, *Performance Requirement*, it is enough to define a new stereotype «Performance Requirement» and add a generalization between the SysML «Requirement» and «Performance Requirement».

All predefined symbol properties (style) and semantics rules of SysML «Requirement» will be reused on elements stereotyped by «Performance Requirement».

The subtypes could have their own customization with some basic properties (for example, human name, etc.). Customizations will be merged, and only additions should be defined.

Try to avoid conflicting customizations, as results can be unpredictable.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Generalization(see page 722) between the two customization elements is also possible. In this case, if these rules are not redefined in the subclass (the particular properties are empty), the specific (subclass) customization inherits the customization rules. If both customizations have the same property filled, the rules are not merged; the customization from the subclass will be used.

Related Pages

- Creating Customization Data(see page 2068)
- Using Customization Data(see page 2093)

Using Customization Data

The following sections describe how to use customization data in particular cases while working with models and modeling tools developed by No Magic, Inc.

- Customizing Specification Window(see page 2093)
- Customizing Element Shortcut Menu(see page 2106)
- Customizing Drag-and-Drop(see page 2108)
- Creating Custom Rules for Relationships(see page 2113)
- Defining Preferred Metatype(see page 2115)
- Initializing a Custom Model(see page 2116)
- Creating a DSL Element from the Customized Category in the Shortcut Menu(see page 2118)
- Customizing Possible Owned Elements(see page 2119)
- Creating Element - Specific Help Topics(see page 2121)
- Creating a DSL Element from the Shortcut Menu and Other Locations(see page 2122)
- Customizing Symbols(see page 2124)
- Creating Property Groups and Subgroups(see page 2126)
- Creating Numbering Customizations(see page 2132)
- Extending Metamodel with Derived Properties(see page 2146)

Additional resources

Enabling Domain Specific Extensions to SysML[369]

Customizing Specification Window

The main purpose of DSL is to hide UML, so that DSL elements would look standard or as new type elements in models. DSL element properties are specified in the DSL element Specification window(see page 913) and the **Properties** panel (at the bottom of the Browser window(see page 29)) as regular properties.

There are several customizations and grouping cases for properties:

- Using Standard UML Properties(see page 2094)
- Always Visible Properties(see page 2095)

369 https://www.youtube.com/watch?v=3IBv3922a-A&feature=youtu.be

- Meta property substitution (changing name of UML property)(see page 2101)

Using Standard UML Properties

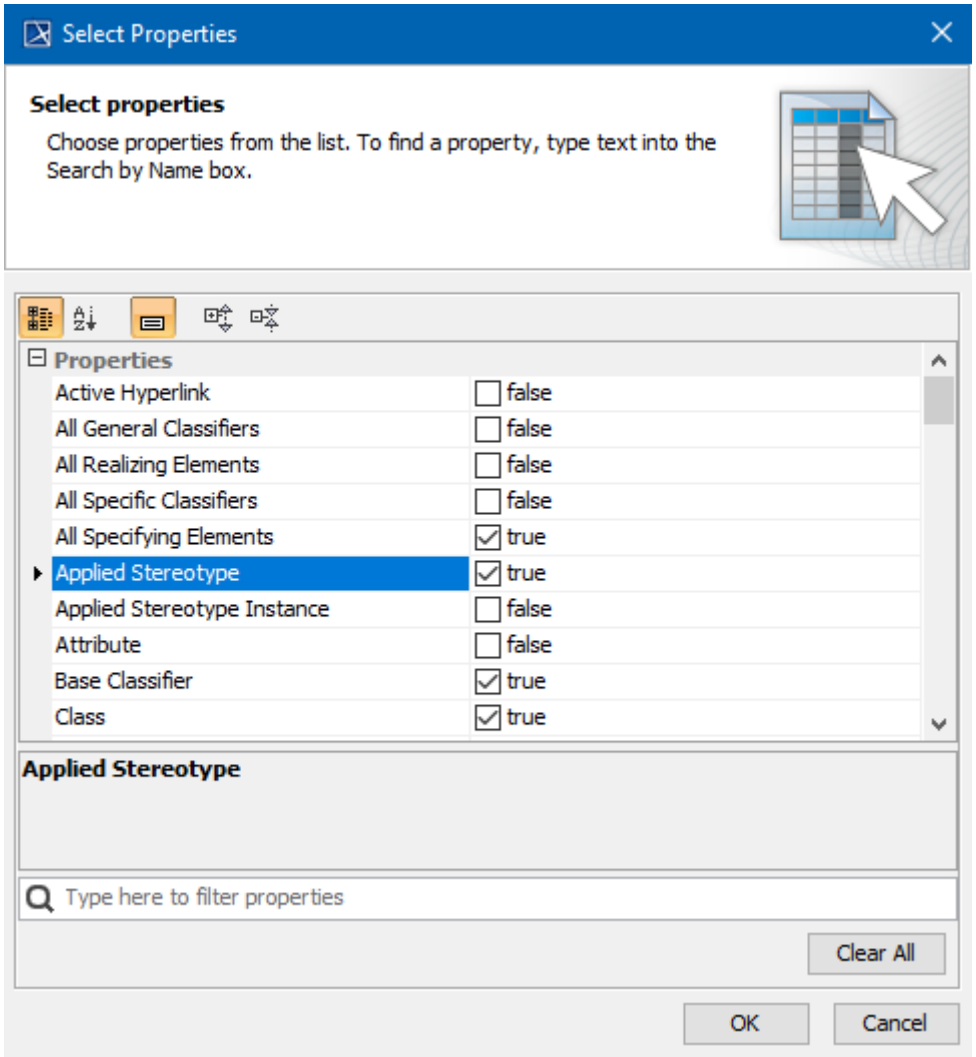
DSL element specification may contain custom properties defined in a stereotype. All DSL element properties are used by default, but you may select which UML properties should be used in the DSL element Specification window(see page 913).

To use standard UML properties

---

1. Open the customization element Specification window.
2. Click the **Used UML Properties** property value specification cell and then click the ... button.
3. In the **Select properties** dialog, set the selected standard UML properties that you want to be displayed in the DSL element Specification window value to *true*.
4. Click **OK** when you are done.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Select Properties dialog.

Related Pages

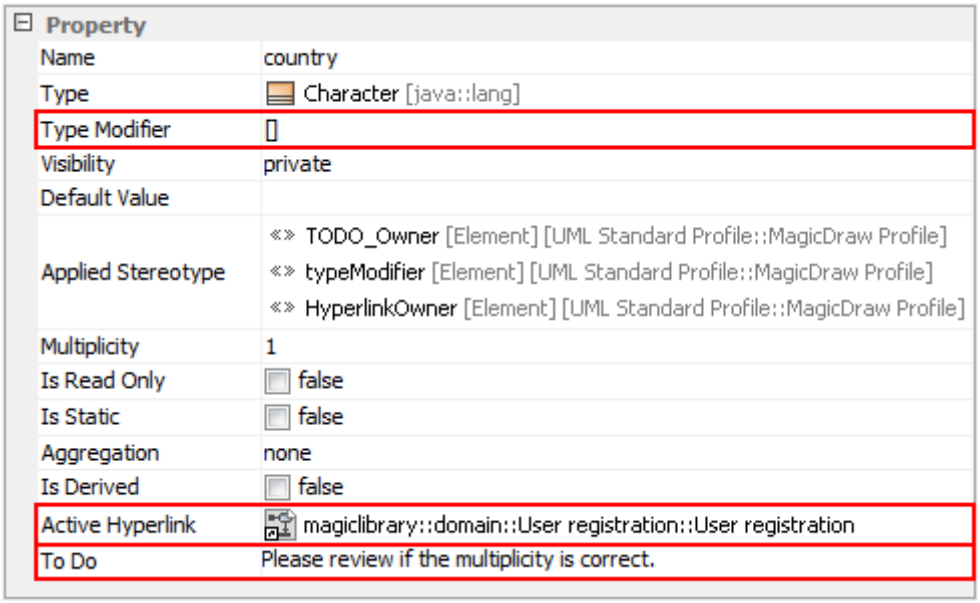
- Creating Customization Data(see page 2068)
- Using Customization Data(see page 2093)
- Customizing Specification Window(see page 2093)
- Always Visible Properties(see page 2095)
- Meta property substitution (changing name of UML property)(see page 2101)

Always Visible Properties

Generally, stereotype properties become visible in an element's Specification window(see page 913) only after the stereotype is applied on the element. However, you may customize stereotype properties to be visible in the element Specification window even if the stereotype is not yet applied to the element.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

This feature allows specifying domain-specific properties for standard UML elements. It is also used to make stereotype(see page 856) properties (**Type Modifier**, **Active Hyperlink**, and **To Do**, etc.) visible even if appropriate stereotypes are not yet applied to an element.



Always visible properties in UML property Specification Window.

To make stereotype properties visible when the stereotype is not yet applied to an element

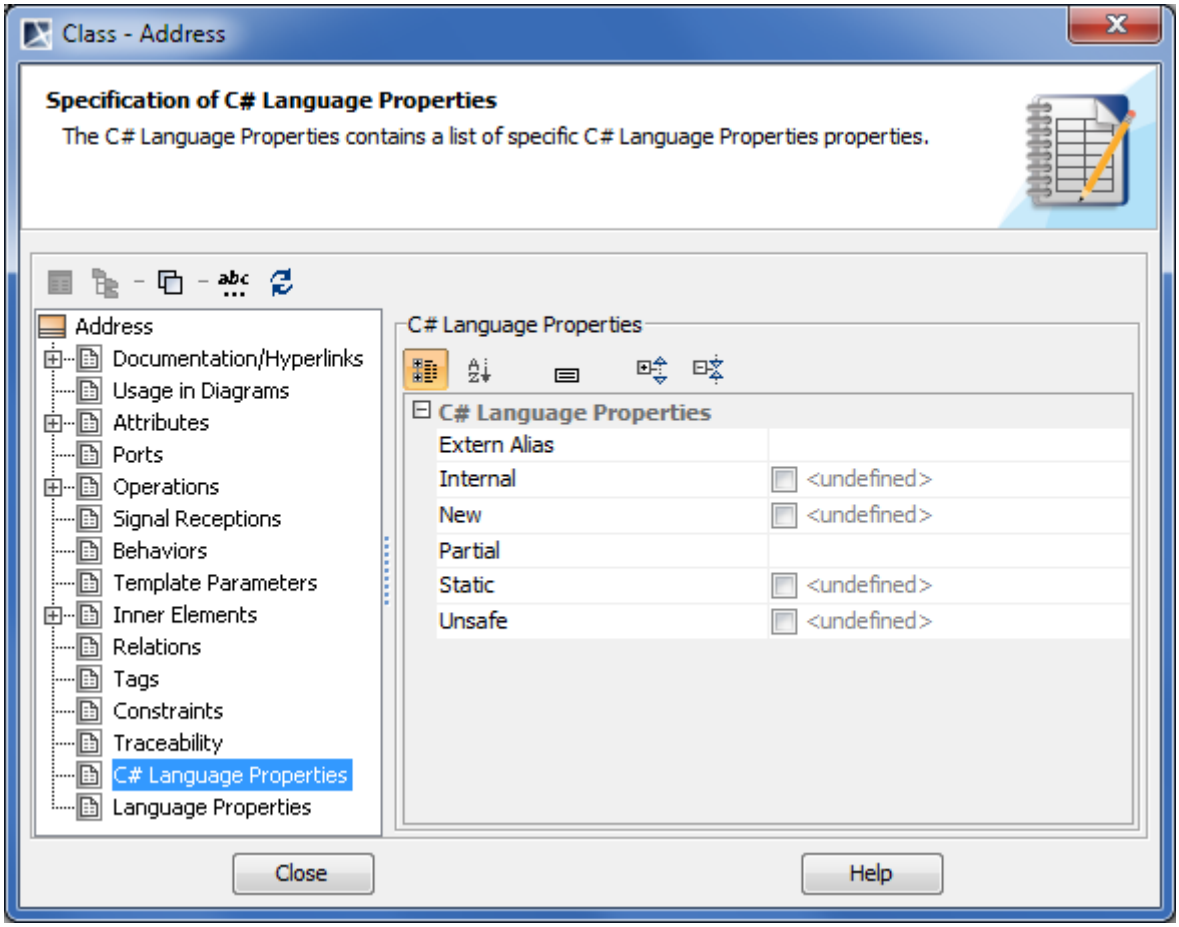
---

1. Open the customization element Specification window.
2. Click to change the **Show Properties When Not Applied** property to *true*. All properties of the customized stereotype will become visible in the DSL element Specification window even if the stereotype is not yet applied to the element.

Case Study

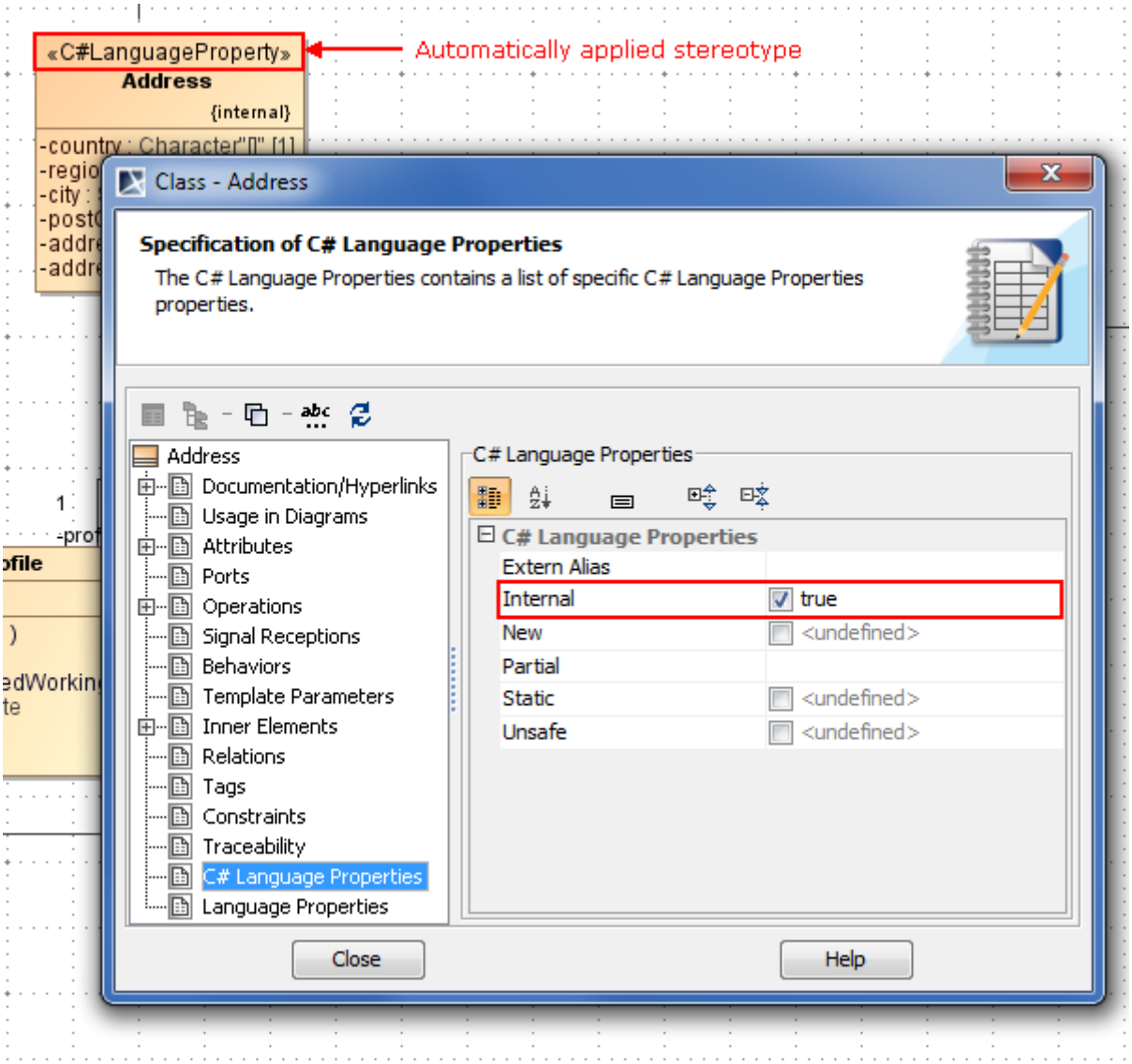
Use the *C# Profile*(see page 242) (*C#_Profile.mdzip*) in your project and you will see the **C# Language Properties** appear in a class Specification window(see page 913). C# properties will appear in the Specification window of a class even though no profile-specific stereotype is applied to the class.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



C# properties in class Specification window .

Specify a desired C# property value in the class Specification window and you will see the «C#LanguageProperty» stereotype automatically applied to the class.



Class with automatically applied «C#LanguageProperty» stereotype.

Related Pages

- Creating Customization Data(see page 2068)
- Using Customization Data(see page 2093)
- Customizing Specification Window(see page 2093)
- Advanced cases of properties visibility(see page 2098)

Advanced cases of properties visibility

There are two methods of displaying stereotype properties that have not yet been applied:

- **Profile application**. Properties will be visible only when the owning package of an element has a corresponding profile application.
- **Another stereotype and / or metaclass**. Properties will be visible only when:
  - Another stereotype is already applied to an element.

- An element's metaclass is one of the metaclasses extended by the stereotype.

To make the visibility of stereotype properties dependent on a corresponding profile application

1. Open the customization element Specification window(see page 913).
2. Click to change the **Show Properties When Not Applied** property to *true*.
3. Click to change the **Show Properties When Not Applied Limited By Profile Application** property to *true*.

To make the visibility stereotype properties dependent on another stereotype or a metaclass

1. Open the customization element Specification window(see page 913).
2. Click to change the **Show Properties When Not Applied** property value to *true*.
3. Click the **Show Properties When Not Applied Limited By Element Type** property and then select a stereotype or a metaclass.

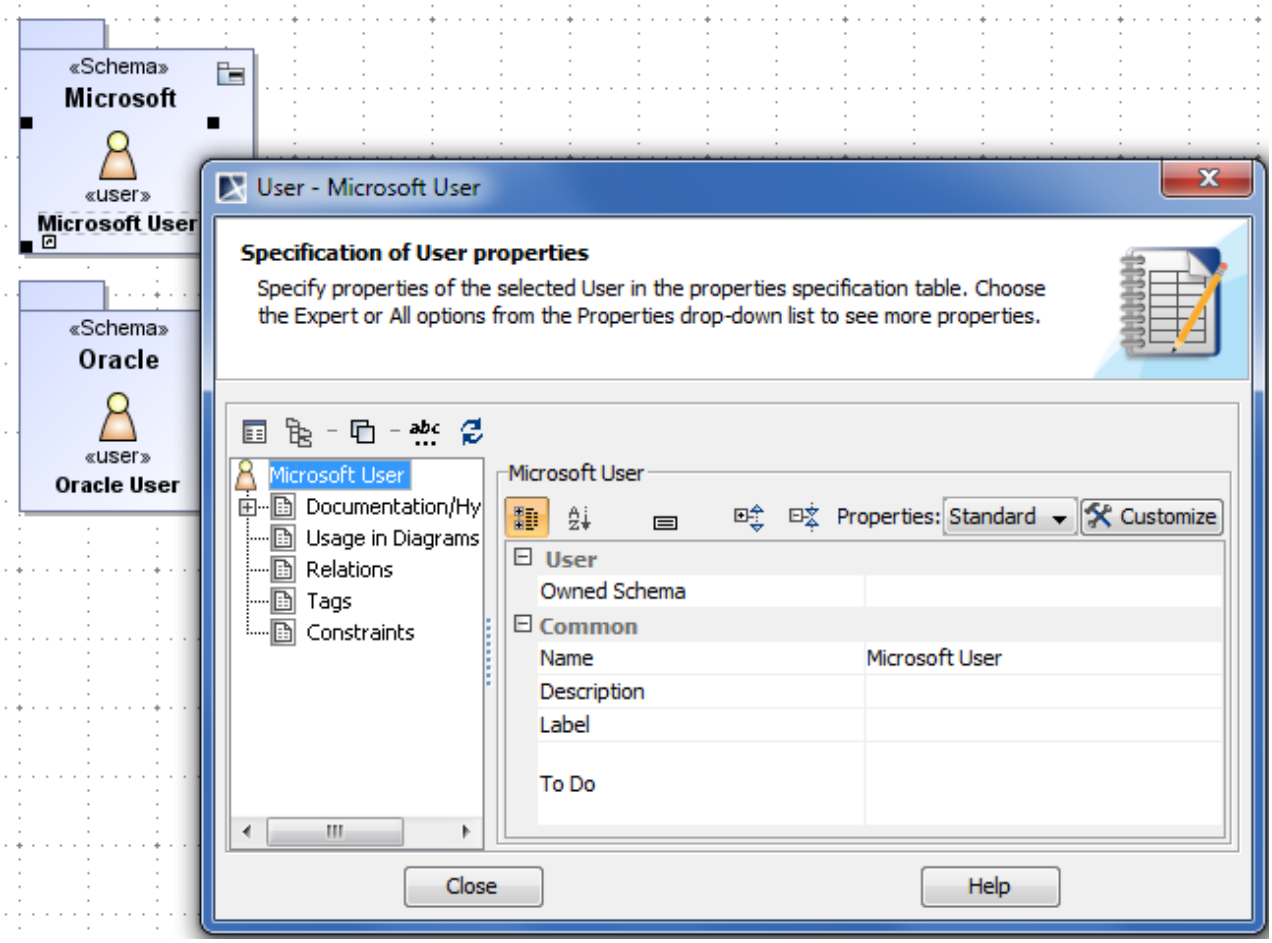
You can select more than one value.

Case Study
Let's analyze the customization element of the «OraUser» stereotype in the *Oracle Customization* profile (*Oracle_Profile.mdzip*). This profile belongs to the *Cameo Data Modeler* plugin and can be used for database modeling.

If the «User» stereotype is selected as the **Show Properties When Not Applied Limited By Element Type** property value, properties of the «OraUser» stereotype will be visible only for elements that have the «User» stereotype applied.

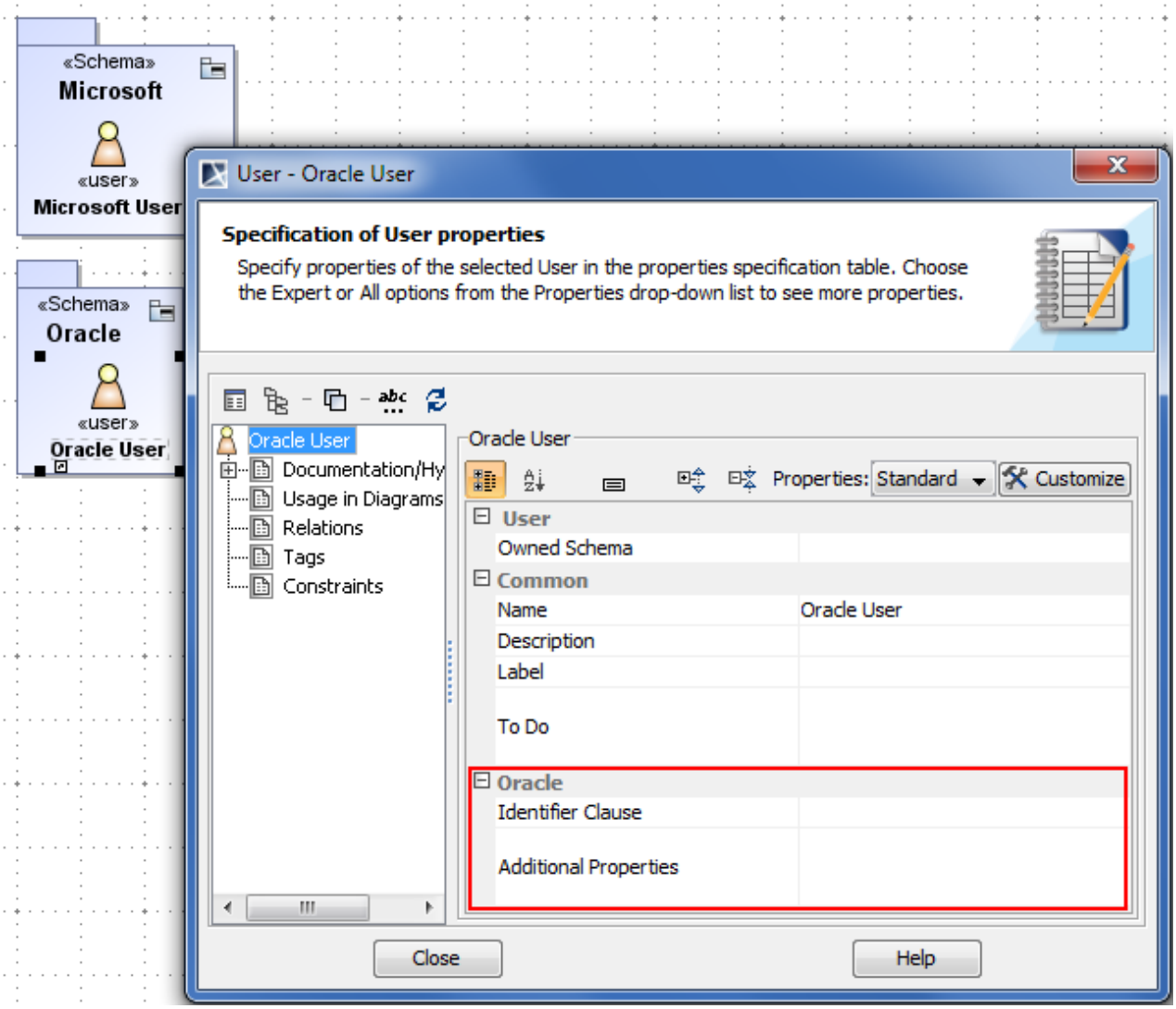
If the **Show Properties When Not Applied Limited By Profile Application** property value is set to *true*, properties of the «OraUser» stereotype will be visible only for elements whose owning package has the *Oracle Customization* profile application, such as elements inside the Oracle schema.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Specification window of user that is NOT INSIDE Oracle schema

In conclusion, Oracle-specific properties will be visible only for users inside the Oracle schema.



Specification window of user that is INSIDE Oracle schema

Related Pages

- Creating Customization Data(see page 2068)
- Using Customization Data(see page 2093)
- Customizing Specification Window(see page 2093)

Meta property substitution (changing name of UML property)

You can customize a standard UML property to represent a more restricted type. You can also rename the property.

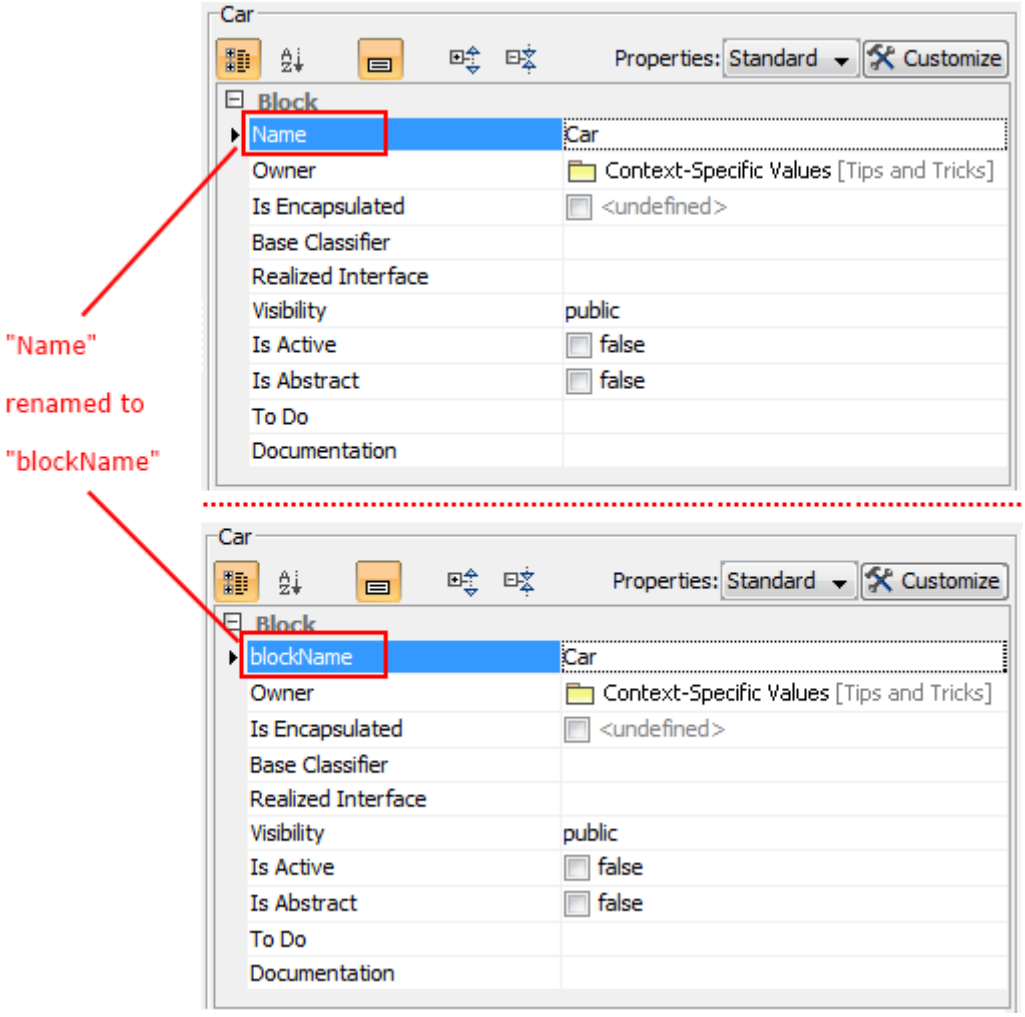
Property name substitution

The following case study presents an example of property name substitution.

Case study

In the DSL element Specification window(see page 913), rename the Name property to *blockName*.

1. Create a *name* attribute(see page 2090) in the customization element.
2. Apply the «*metaProperty*» stereotype(see page 2043) on the attribute.
3. In the «*metaProperty*» attribute Specification window, click the **New Name** property, and then type a new name *blockName*.
4. To apply the changes, reopen the project.

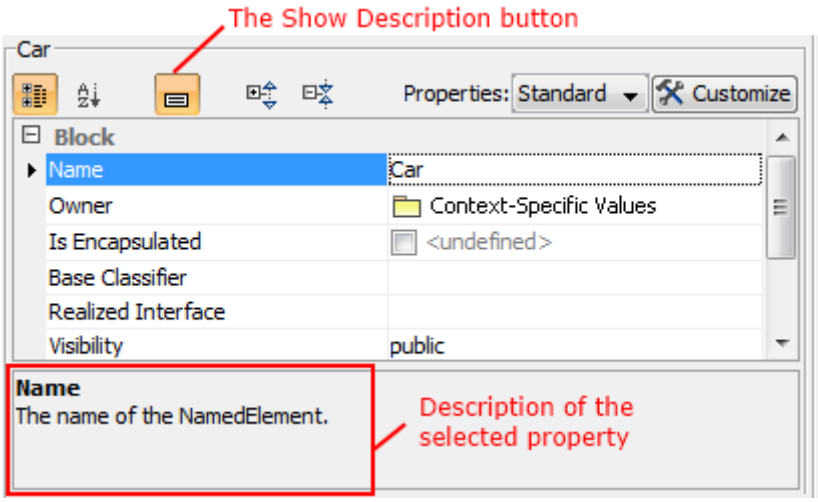


Example of property name substitution.

Property description substitution

In the element Specification window(see page 913), the description of the selected property is displayed below the properties list (see the preceding figure). You can customize (change) the property description. For example, after changing the property name (see the Case Study above), you could also change the property description.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Example of property description area in element Specification window .

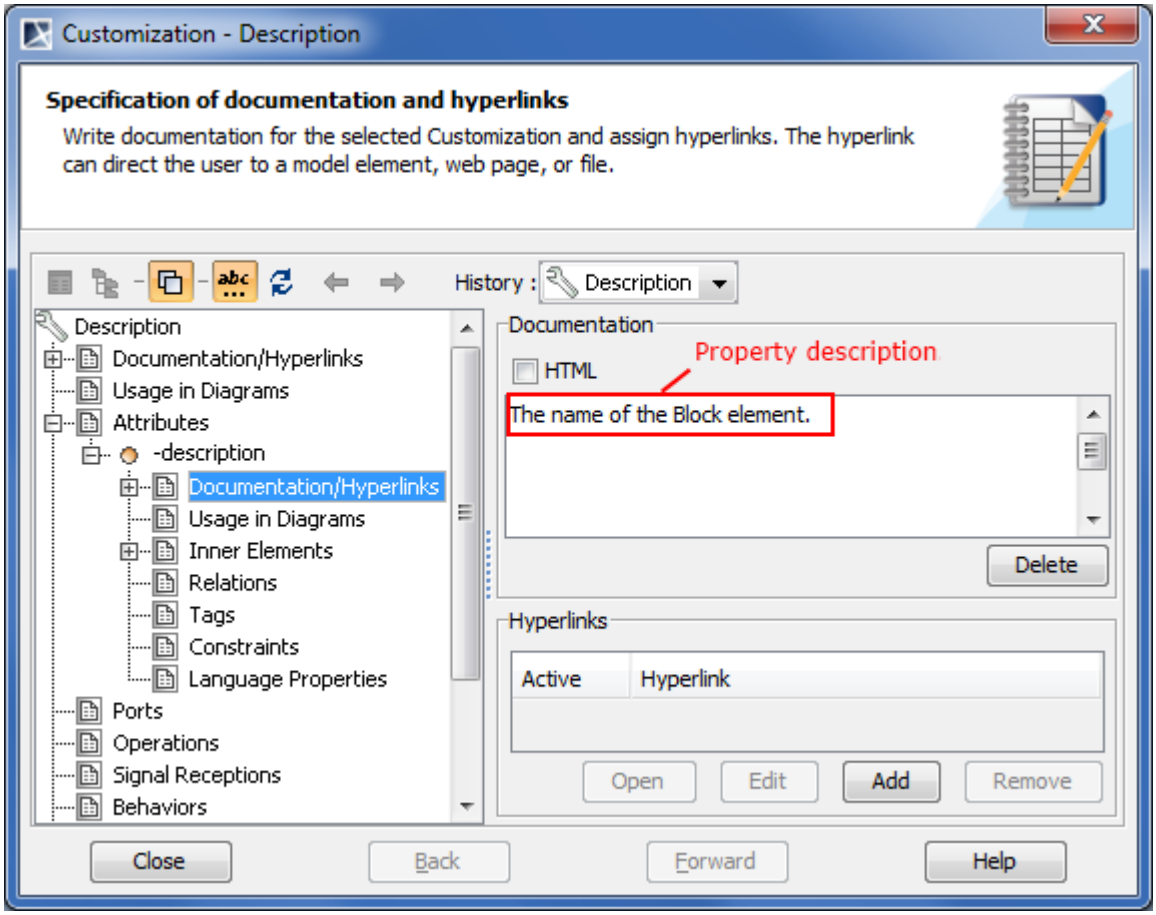
Case study

To change the *Name* property description

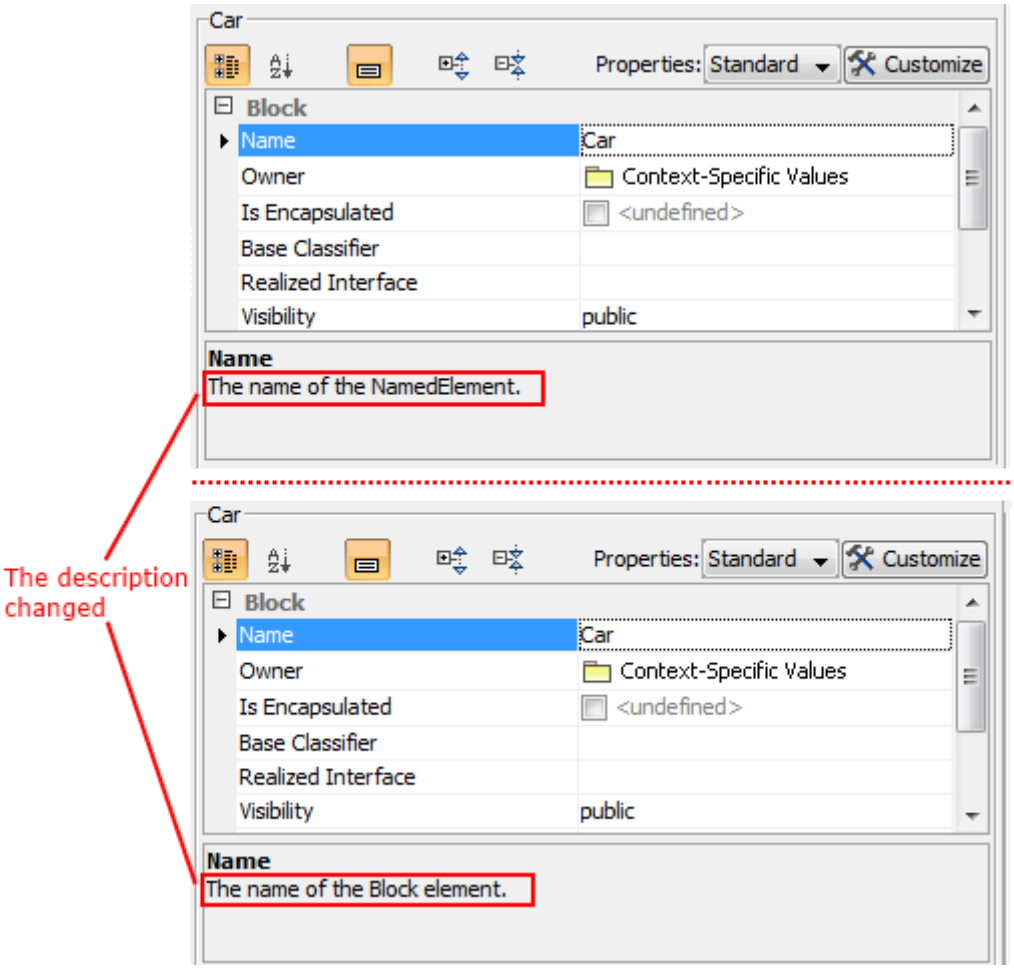
---

1. Create a new description attribute(see page 2090) in the customization element.
2. Apply the «*metaProperty*» stereotype(see page 2043) on the attribute.
3. Type a new description in the customization element Specification window, under the description attribute, **Documentation/Hyperlinks** property group.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



4. To apply the changes, save and reopen the project. The description of the *Name* property changes.



Example of property description substitution.

Type substitution (restriction)

To change the type

1. Create the *type* attribute(see page 2090) for the customization element.
2. Apply the «*metaProperty*» stereotype(see page 2043).
3. Click the **New Types** property in the type *attribute* Specification window.
4. Specify the «Block» stereotype.
5. To apply the changes, save and reopen the project.

Suggesting predefined values

To create properties to list predefined values suggested for the end user (such as "type modifier")

1. Create an attribute(see page 2090) for the customization element.
2. Name attribute as a property to customize.
3. Apply the «metaProperty» stereotype(see page 2043).
4. Enter suggested values into the Suggested Values property.

On this page

- Property name substitution(see page 2102)
- Property description substitution(see page 2102)
- Type substitution (restriction)(see page 2105)
- Suggesting predefined values(see page 2105)

Related Pages

- Creating Customization Data(see page 2068)
- Using Customization Data(see page 2093)
- Customization Element Properties(see page 2090)
- Customizing Specification Window(see page 2093)

Customizing Element Shortcut Menu

On this page

- Quick property editor(see page 0)
- Applying quick stereotypes(see page 0)

You can add new items to shortcut menus to use certain modeling tool commands more conveniently. To do so, you need to customize the model(see page 2070).

> **Important**
> Properties of all other types are ignored.

Customizing the model

There are two ways to customize the model:

Quick property editor

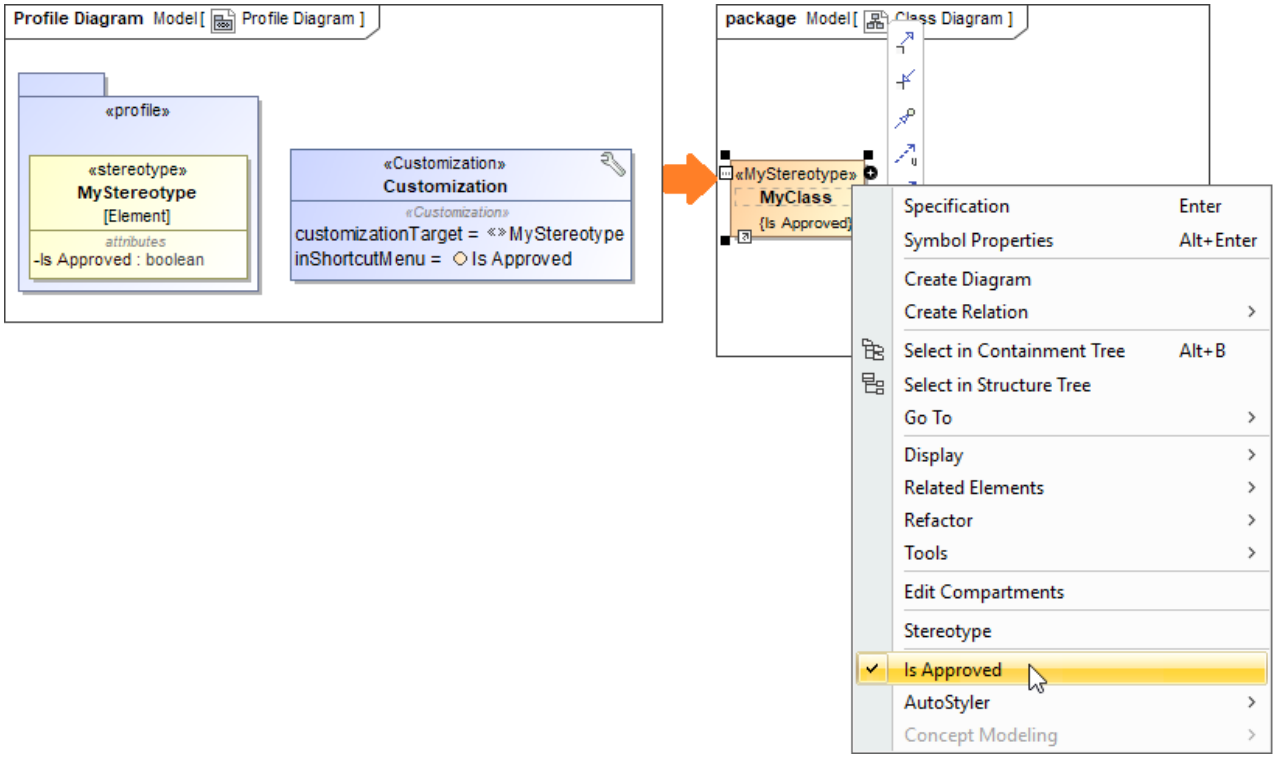
You can add those properties you frequently use to the customized element shortcut menu. To do so, open the project that includes your DSL customization(see page 2070), open the Specification window(see page 913) for your customization element, locate the **In Shortcut Menu** property, and specify the property you want to see in the element shortcut menu.

These are the types of properties whose values can be edited directly via the element shortcut menu:

- Boolean – allows selecting or clearing a checkbox.
- Enumeration – allows selecting one of the listed enumeration literals.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Reference to one element (a subclass of Classifier) – allows using an element list to select a type.



Example of the boolean property in the shortcut menu

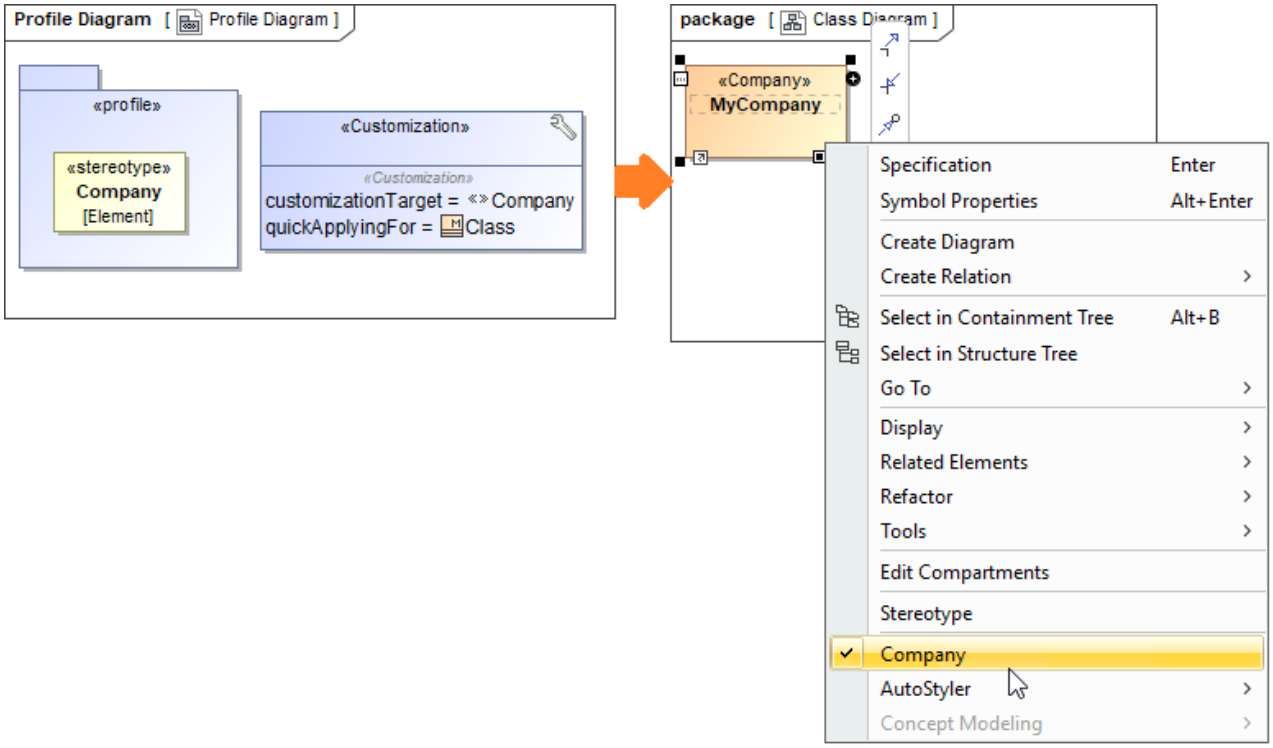
Applying quick stereotypes

Stereotypes, for example, «continuous», «buffer», etc., act as flags. You can set these stereotypes on selected elements with a single click from the element shortcut menu.

Use the **Quick Applying For** property in the customization element Specification window(see page 913) to add stereotypes to the customized element shortcut menu. Before you can do that, you need to create your DSL customization(see page 2070) for a stereotype.

The **Quick Applying for** property specifies a stereotype or a metaclass to apply customizations directly from the element shortcut menu. For example, if the **Quick Applying For** property value is the *Class* metaclass, the customization will be available for the class elements only.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Example of a customized element shortcut menu

> You can show/hide default modeling tool options in shortcut menus by using the Perspectives functionality(see page 70) (**Options** > **Perspectives** > **Customize**).

Related Pages

- Creating Customization Data(see page 2068)
- Using Customization Data(see page 2093)
- Customizing and Selecting Perspective(see page 67)

Customizing Drag-and-Drop

The drag-and-drop customization allows you to create relationships or assign property values by dragging and dropping one element onto another right in the diagram pane. By using the DSL engine, you may set up your own drag-and-drop specifications between elements:

- create relationships(see page 2110)
- assign property values(see page 2111)

There are a number of predefined rules to create relationships or assign property values automatically.

Drag and Drop Actions
You can customize the following drag-and-drop actions:

- drag-and-drop one element from the diagram pane onto another element on the diagram pane
- drag-and-drop one element from the Model Browser onto another element on the diagram pane
- drag-and-drop two or more elements from the Model Browser onto an element on the diagram pane

Below is a list of the properties to customize drag and drop rules.

Property	Description
Source Element	Specify the element type to drag to the target element. Specify a class(see page 666), metaclass, or stereotype(see page 856). This is a mandatory property: if the **Source Element** property is undefined, then the drag-and-drop rule is inactive.
Representation Text	Specify the name of the drag-and-drop rule.
Relation Action Result	Specify a relationship created between the source and target elements after drag-and-drop.
Property Action Result	Specify a property assigned to the target element after drag-and-drop.
Append Mode	Change the append mode for the property assignment. The possible **Append Mode** property value is *true* or *false*. If *true*, properties are assigned one by one. If *false*, the created property overwrites the former property.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related Pages

- Creating Customization Data(see page 2068)
- Using Customization Data(see page 2093)
- Dragging objects(see page 487)

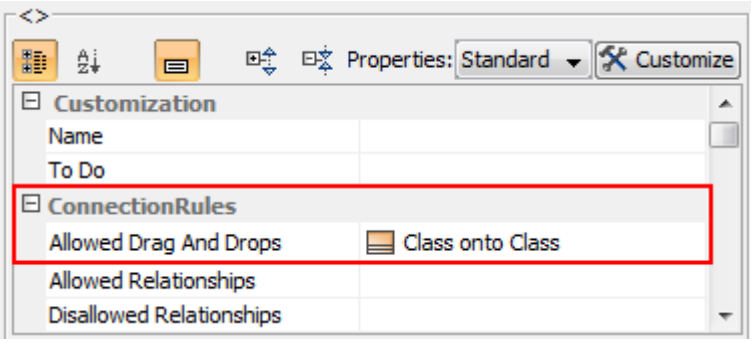
Case study #1: Adding a rule to create a Generalization relationship on drag-and-drop

This example presents step-by-step instructions for customizing a drag-and-drop rule, according to the Generalization relationship created between classes after the drag-and-drop.

1. Create a profile diagram(see page 483).
2. Create a class(see page 666), and name it *Class onto Class*.
3. For the *Class onto Class* class, apply the «*DragAndDropSpecification*» stereotype (see page 856).

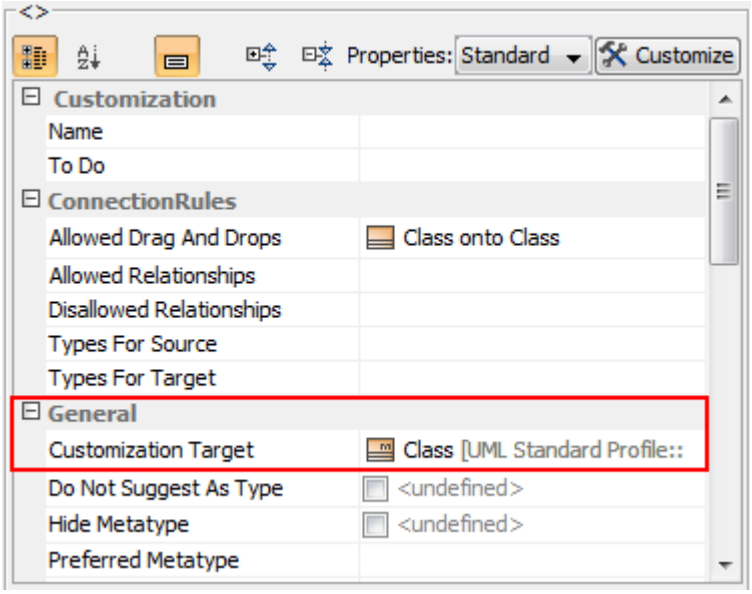
   New Button
   There is a new button in the Profile diagram palette - Drag and Drop Specification. Use this button to create a class element with the assigned «*DragAndDropSpecification*» stereotype.

4. Create a *Customization* element(see page 2073).
   In the **Customization** Specification window, assign the *Class onto Class* element to the **Allowed Drag and Drops** property under **Connection Rules**.



5. In the **Customization** Specification window, under the **General** category, for the **Customization Target** property, assign the *Class* metaclass.

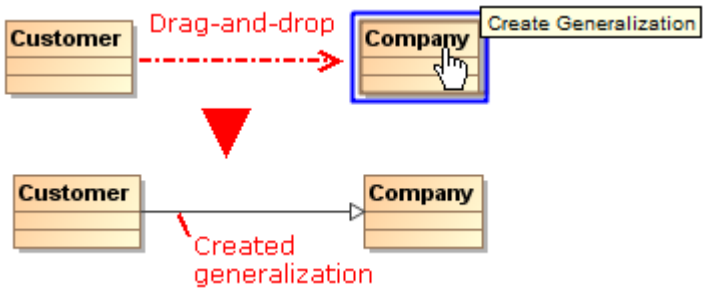
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



6. In the *Class onto Class* Specification window, specify the following property values:

   - For the **Source Element** property, specify the *Class* metaclass.
   - For the **Relation Action Result** property, specify the *Generalization* metaclass.
   - For the **Representation Text** property, specify the name of the rule, for example, *Create Generalization*.

7. Reopen the project. Drag the class to the other class, the Generalization relationship will be created between these classes.



Related Pages

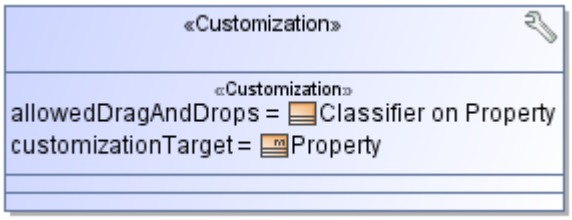
- Creating Customization Data(see page 2068)
- Using Customization Data(see page 2093)
- Creating Numbering Customizations(see page 2132)

Case study #2: Adding a rule to assign a property type on drag-and-drop

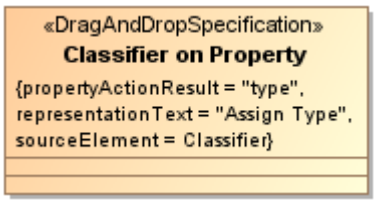
This example presents step-by-step instructions for customizing a drag-and-drop rule. You will assign the property type after dragging a class to the property.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

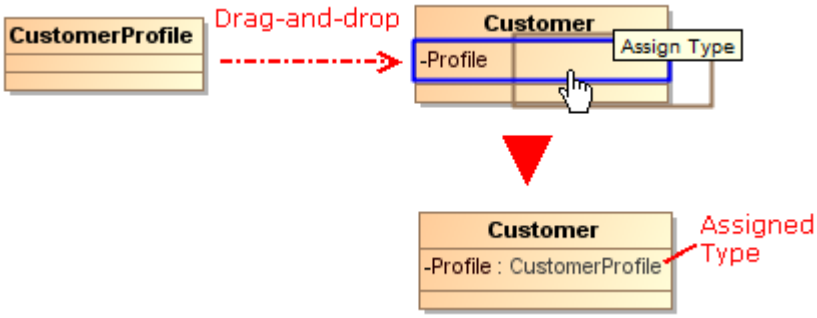
1. Create a profile diagram(see page 483).
2. Create a class(see page 666), and name it *Classifier on Property*.
3. For the *Classifier on Property* class, apply the «*DragAndDropSpecification*» stereotype(see page 856).
4. Create a customization element(see page 2073).
5. In the **Customization** Specification window(see page 913), assign the *Classifier on Property* element to the **Allowed Drag and Drops** property, .
6. In the **Customization** Specification window, assign the *Property* metaclass to the **Customization Target** property.



7. In the *Classifier on Property* Specification window, specify the following property values:
   - For the **Source Element** property, specify the *Classifier* metaclass.
   - For the **Property Action Result** property, specify *Type*.
   - For the **Representation Text** property, type the name of the rule, for example, *Assign Type*.



8. Reopen the project. The classifier is assigned as a property type after dragging it to the property.



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related Pages

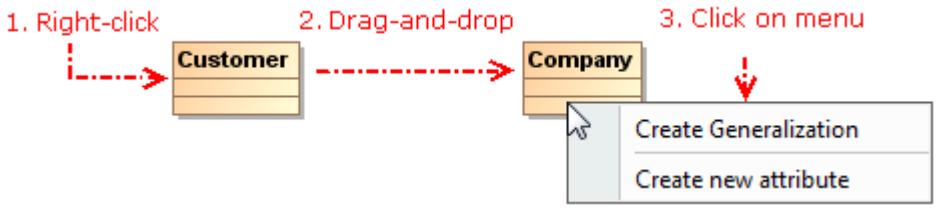
- Creating Customization Data(see page 2068)
- Using Customization Data(see page 2093)
- Creating Numbering Customizations(see page 2132)

Selecting more than one drag-and-drop rule

In some cases, more than one rule can be applied to a drag-and-drop operation. This enables you to select what to create.

To select a rule

1. Do one of the following:
   - Drag a selected element to another, hold it for a few seconds till the tool-tip opens, and then release. A shortcut menu is displayed.
   - Right-click an element, drag it onto another element, and then drop it. A shortcut menu is displayed. (**Note:** this procedure is valid on Windows OS only).
2. On the shortcut menu, click the rule to apply. The rule is applied.



Tip
In a modeling tool developed by NoMagic, Inc., there are a number of predefined drag-and-drop rules.

Related Pages

- Creating Customization Data(see page 2068)
- Using Customization Data(see page 2093)
- Creating Numbering Customizations(see page 2132)

Creating Custom Rules for Relationships

DSL relationships may have special rules to describe which elements can be connected. Use the following properties to define the relationships:

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Property	Description
Types for Source	Specifies metaclasses or stereotypes(see page 856) allowed to connect as the source of the relationship. Types cannot conflict with UML permitted types for this relationship. **Note** The **Types For Source** property is intended for relationships only. The **Customization Target** property can be a relationship or a metaclass of a relationship.
Types for Target	Specifies metaclasses or stereotypes (see page 856)allowed to connect as the target of the relationship. Types cannot conflict with UML permitted types for this relationship. **Note** The **Types For Source** property is intended for relationships only. The **Customization Target** property can be a relationship or a metaclass of a relationship.

Note and Example

Subtypes of a specified end type are allowed only if the type is abstract (for example, the Classifier metaclass), otherwise only concrete types must be used. For example:

- If the value of the **Types For Source** property is the Classifier metaclass, then the Class, Component and other classifiers are allowed to connect as a source of a relationship.
- If the value of the **Types For Source** property is the Class metaclass, then the Class elements are allowed to connect as a source of a relationship.
- If the value of the **Types For Source** property is the «Block» DSL element, then the «Block» elements are allowed to connect as a source of a relationship.

In the following table, see properties that are used for customizing non-relationships:

Property	Description
Allowed Relationships	Specifies types of relationships that are allowed to be connected to the DSL element.
Disallowed Relationships	Specifies types of relationships that are not allowed to be connected to the DSL element.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Note and Example
Subtypes of specified allowed or disallowed relationships are analyzed only when a specified relationship type is an abstract metaclass (for example, *DirectedRelationship*). In other cases, only concrete types are checked.

- If «Trace» is allowed, subtypes of trace «copy» and «verify» are allowed also.
- If *Association* is allowed, *Extension (a* subtype) is not allowed, but simple stereotyped Associations are allowed. All DSL Associations are not allowed.
- If *Dependency* is not allowed, simple stereotyped Dependencies like «delegate» are not allowed, but subtypes of *Dependency*, for example, *Abstraction* and DSL types based on *Dependency* are allowed.

Related Pages

- Creating Customization Data(see page 2068)
- Using Customization Data(see page 2093)
- Creating Numbering Customizations(see page 2132)
- Working with paths and relationships(see page 557)

Defining Preferred Metatype

The DSL customization engine allows specification of a preferred metatype if more than one metatype is defined for the DSL element.

To define a preferred metatype for the DSL type

1. Select the appropriate customization element(see page 2073) and open its Specification window(see page 913).
2. Click the value specification cell of the **Preferred Metatype** property and do one of the following:
   - Select the property value from the open list(see page 925) and close the Specification window.
   - Click the button to create a new one(see page 1035). In the element Selection dialog(see page 1031), select the created metatype and click OK. Close the Specification window.

   The selected metatype will be assigned as the **Preferred Metatype** property value.

   Multiple selection is not allowed. You must define a single value.

After updating the customization element specification, you must either reuse the DSL Customization profile(see page 242) in the model or reopen the model.

From now on, all DSL elements will be created with the preferred metatype specified in the customization.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related Pages

- Creating Customization Data(see page 2068)
- Using Customization Data(see page 2093)

Initializing a Custom Model

With DSL customization, you can modify a model element to act according to your needs depending on the stereotype you apply.

Default values

Tags(see page 864) have default values. Use the default values when you create the tag, making sure its definition has multiplicity greater than zero. In this case it is automatically created when applying a stereotype.

Stereotypes on relationship ends

At times, DSL requires applying stereotypes (see page 856)to some elements after DSL relationship connection to those elements.

Use the following properties for this purpose:

Property	Description
Apply to Source	Specifies stereotypes that must be applied to the source element of this relationship after connection.
Apply to Target	Specifies stereotypes that must be applied to the target element of this relationship after connection.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Source and Target in an example
"Source" and "Target" are respectively the "source" and "target" of a directed relationship. The first connected element acts as a"source" for a non-directed relationship.
Example:

1. Create a stereotype *«serve»* and select the Dependency metaclass.
2. Create two stereotypes *«master»* and *«servant»* and select the Element metaclass.
3. Create a customization element.
4. In the customization Specification window(see page 913), specify the following property values:
   a. for the **Customization Target** property, select the *«serve»* stereotype,
   b. for the **Apply To Source** property, select the *«master»* stereotype,
   c. for the **Apply To Target** property, select the *«servant»* stereotype.
5. Reopen the project. Draw the *«serve»* dependency from one element to another. The *«master»* and *«servant»* stereotypes are applied to these elements.

Required Generalization or Interface Realization

Sometimes DSL requires elements to be subtypes of a general abstract class (see page 666)or interface(see page 741). Use the **Super Types** property for this purpose. The **Super Types** property specifies types to be super types of the DSL element. The *Generalization* or *InterfaceRealization* (if using an interface) will be created in the model after applying the customized stereotype.

Example
The following is an example of customization. Every JAVA class will be a subclass of the *Object* cl ass. In other words, every Class marked with the «JAVAClass» stereotype is inherited from the Object.

1. Create a customization element.
2. In the customization Specification window, specify the **Customization Target** property as the «JAVAClass» stereotype.
3. In the customization Specification window, specify the **Super Types** property as the *Object* class.

On this page

- Default values(see page 2116)
- Stereotypes on relationship ends(see page 2116)
- Required Generalization or Interface Realization(see page 2117)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related Pages

- Creating Customization Data(see page 2068)
- Using Customization Data(see page 2093)
- Model(see page 780)
- Tag(see page 864)

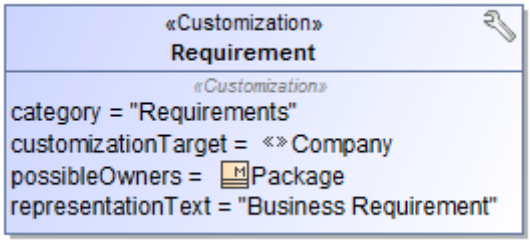
Creating a DSL Element from the Customized Category in the Shortcut Menu

The **Create Element** menu in the Model Browser allows you to group elements into categories. You can also use various menus in dialogs and specifications to group elements. Use the **Category** property of a customization element to create categories.

Case Study

The following example demonstrates how to create a category for a new requirement element within a Package element. This example refers to the project described in Case Study: Creating Your First Customization(see page 2070).

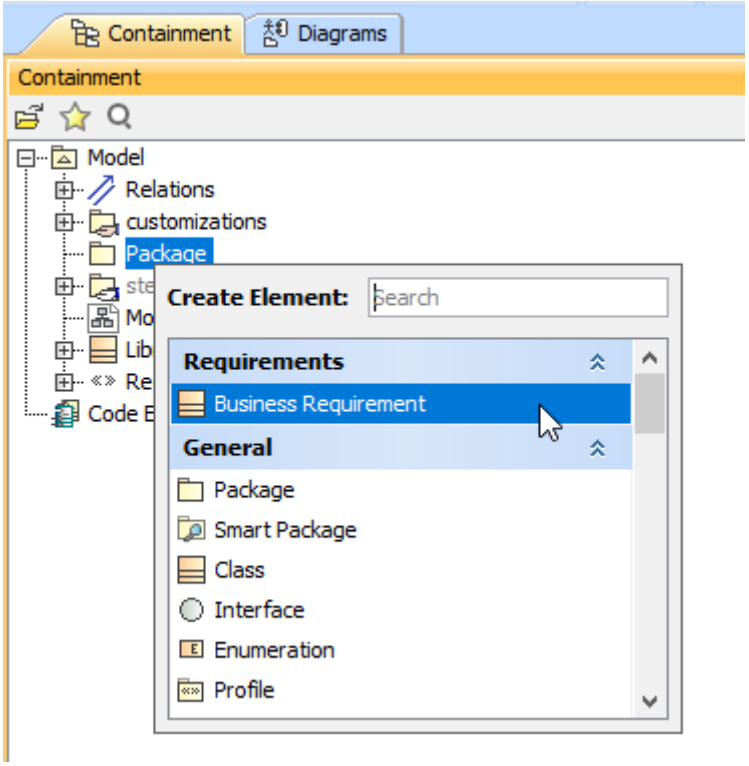
1. Assign the custom stereotype (in this case the *«Company»* stereotype) as a **Customization Target** property in the customization element Specification window. Specify the Package metaclass as a **Possible Owners** property value. Please see the Case Study: Creating Your First Customization(see page 2070) for more details.
2. In the customization element Specification window:
   - specify the name of a new category as the **Category** property value (for example, *Requirements)*.
   - specify the name of an element to create as the **Representation text** property value (for example, *Business Requirements)*.



In this example, we used the stereotype created in the example described in Case Study: Creating Your First Customization**.**

3. Save and reopen your project.
4. The *Business Requirement* element is added to the package shortcut menu, as shown below.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The custom Requirements category in the Create Element menu.

To create several items under the same category, you must create a customization element for each item. Specify the same name of the **Category** property.

Related Pages

- Creating Customization Data(see page 2068)
- Using Customization Data(see page 2093)

Customizing Possible Owned Elements

You can define situations when elements of particular predefined stereotypes or metaclasses are created inside other elements. For example:

- *Block* contains the *BlockProperty* property, instead of a standard property.
- *Block* owns *InternalBlockDiagram*, instead of a standard *CompositeStructureDiagram*.

Use properties listed in the customization element Specification window(see page 913), under the **Owned Elements** category, to customize these cases.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Customizing Owned Types

Customize the list of owned types in the DSL element shortcut menu (under the **Create Element** command) in the Model Browser(see page 29) and other lists. The properties used in customizing owned types are:

Property	Description
Suggested Owned Types	Specifies the list of stereotypes and metaclasses used when creating inner elements in the DSL element.
Hidden Owned Types	Specifies the list of stereotypes and metaclasses *not* used as inner elements.

You cannot customize a standard element when you use standard UML (rather than DSL) to create certain stereotyped elements (such as a package). In this case, you must specify possible owners using the **Possible Owners** property from the customization element Specification window. The **Possible Owners** property specifies the list of stereotypes and metaclasses used when creating owned elements.

Customizing Owned Diagrams

Customize the list of owned diagrams in the DSL element shortcut menu (under the **Create Diagram** command) in the Model Browser and in other lists. The following are the properties used in customizing owned diagrams:

Property	Description
Suggested Owned Diagrams	Specifies types of diagrams appearing in the DSL element shortcut menu when creating possible inner diagrams.
Hidden Owned Diagrams	Specifies types of diagrams owned by the DSL element. For example, if the property value is *Section*, then the *Section* elements are owned by the customized element.

Version
Until version 17.0.3, the **Suggested Owned Type** and **Hidden Owned Types** properties did not function if the customization target was a metaclass.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Customizing Suggested Relationships

You can customize the list of relationships created directly in the model from the Model Browser, starting from a source or target element.

Customize the relationships list according to the following rules:

- If a DSL relationship has *no* connection rules for restricted end types and the **Hide Metatype** property value specified is *true*, the DSL relationship will be added into all lists where the extended UML type is selected. For example, the «allocation» relationship in SysML model will be used everywhere Dependency is used.
- If a DSL relationship has connection rules, then it will be created only from or to these restricted types.

On this page

- Customizing Owned Types(see page 2120)
- Customizing Owned Diagrams(see page 2120)
- Customizing Suggested Relationships(see page 2121)

Related Pages

- Creating Customization Data(see page 2068)
- Using Customization Data(see page 2093)
- Customization Element Properties Description(see page 2075)

Creating Element - Specific Help Topics

Define a string value referring to an element-specific help topic for each DSL element:

- By clicking the **Help** button in the element's Specification window(see page 913).
- By pressing F1, when the element's symbol is selected.

To define a help topic ID for the DSL element

1. Select the appropriate customization element and open its Specification window.
2. In the general specification pane, select the **Help ID** property and type the help topic ID.
3. Click **Close** when you are done.

> **DSL Customization**
> After updating the customization element specification, you must either reuse the DSL Customization profile in the model or reopen the model.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

If an element represents two or more DSL elements, the first DSL element's help topic ID will be taken.

If the DSL type customization does not have the **Help ID** property defined, the help topic ID derives from the more general stereotype customization (ancestor) of the **Help ID** property. If there is no ancestor, the metaclass help opens.

Related Pages

- Using Customization Data(see page 2093)
- Creating Customization Data(see page 2068)

Creating a DSL Element from the Shortcut Menu and Other Locations

The Creating your First Customization(see page 2070) section introduced the step-by-step process of creating a custom element. Following are tips to create custom elements.

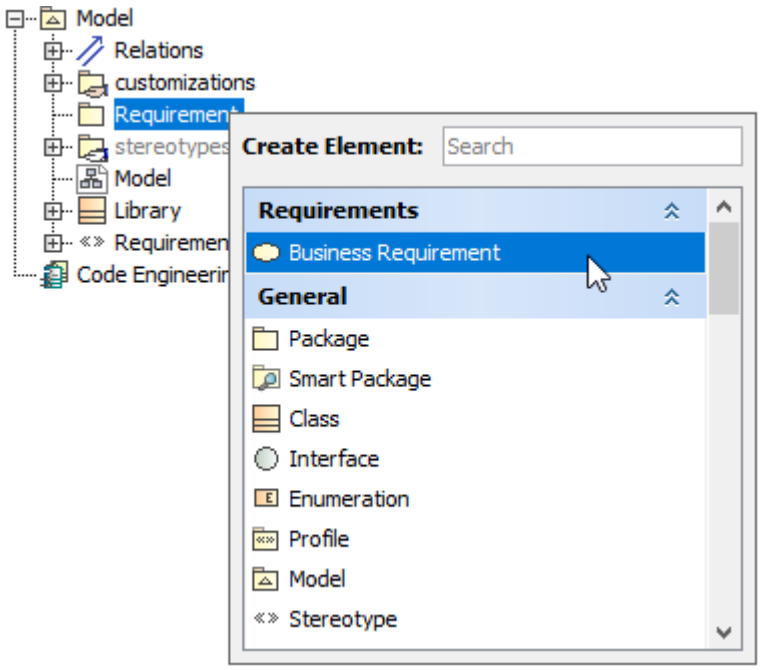
Use the **Possible Owners** property to define the possible owner for a customized element. You can also quickly create a customized element from the possible owner shortcut menu in the Model Browser(see page 29) and in other locations. For example, you can add a package as a possible owner for a customized use case element.

Use the **Possible Owners** property to quickly create a customized element from the following locations:

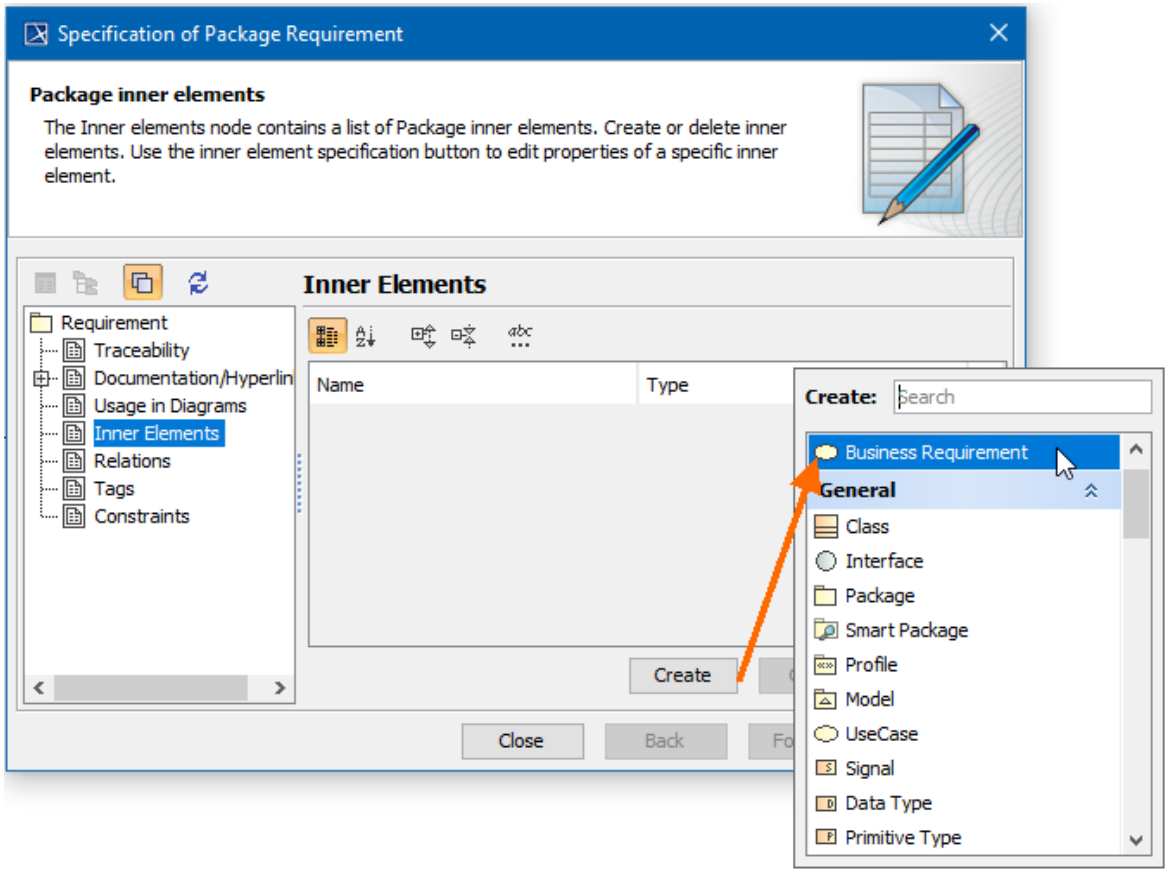
- The possible owner's shortcut menu, under the **Create Element** command, in the Model Browser(see page 29)
- The **Inner Elements** property group under the possible owner's Specification window.
- The **Select Element** dialog(see page 1031), when the element creation mode is turned on.

Examples of creating the *Business Requirement* element in Creating a DSL Element from the Customized Category in the Shortcut Menu(see page 2118) are shown below:

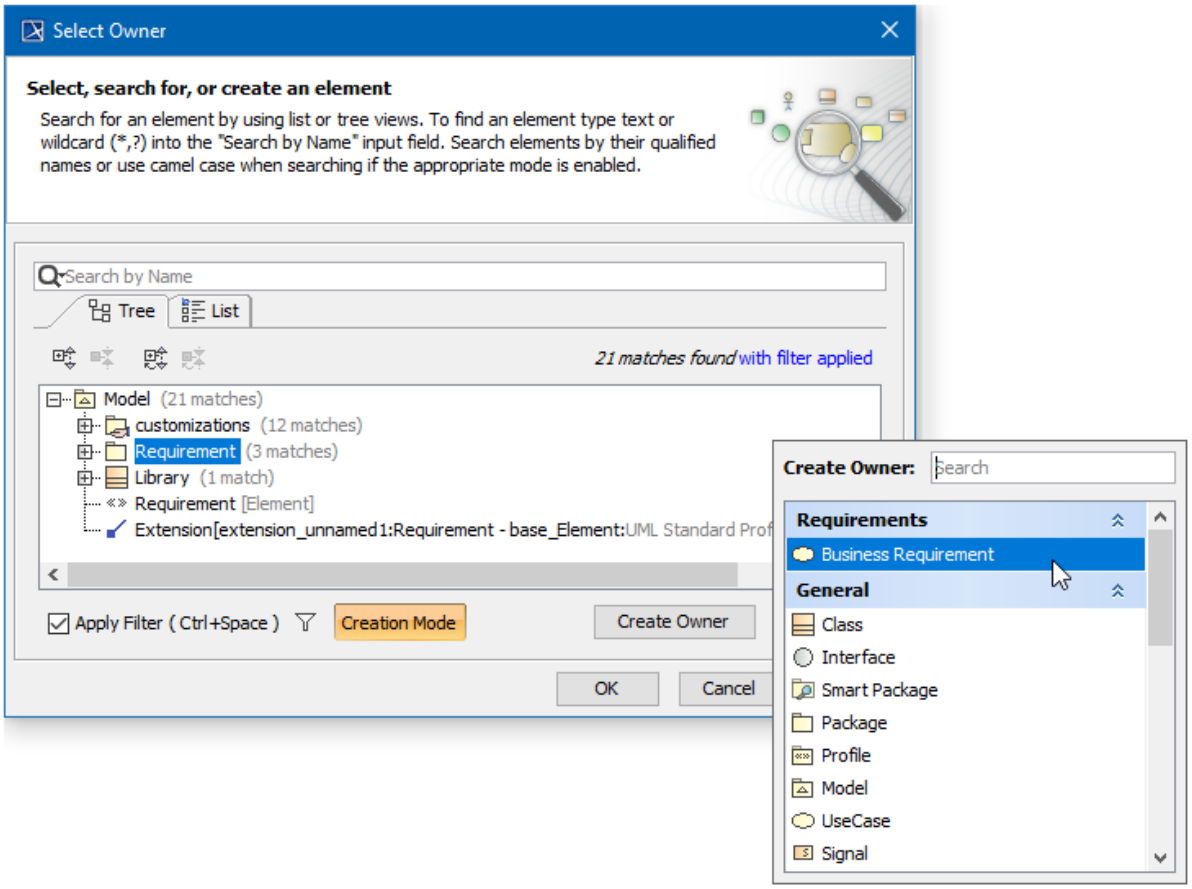
1. In the package shortcut menu (in the Model Browser) under the **Create Element** command.



2. In the possible owner's Specification window, the **Inner Elements** property group.



3. In the element Selection dialog, when creating a new element.



Related Pages

- Creating Customization Data(see page 2068)
- Using Customization Data(see page 2093)
- Customization Element Properties Description(see page 2075)

Customizing Symbols

Setting default symbol size

The **Default Shape Size** property defines the default size of a shape when you create a symbol in a diagram. Use the **Default Shape Size** property in the Customization element Specification window(see page 913) to define the default symbol size in DSL customization:

- The first value is the width (x) in pixels. Value= 0 is used if a default symbol width should stay, but the height should change.
- The second value is a height (y) in pixels.

A default shape size is used everywhere a new symbol is created - using drag-and-drop(see page 2108), diagram toolbar button, diagram wizards, etc.

Setting custom path style

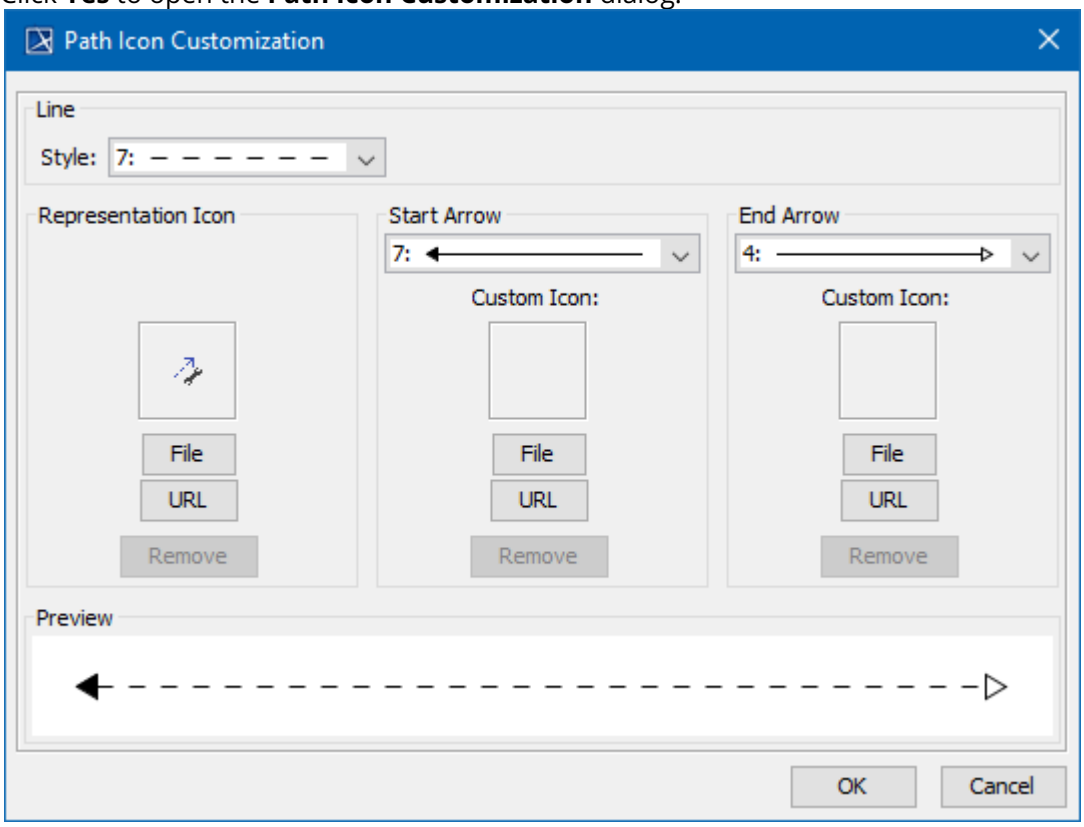
Customize the custom path style by changing:

- Line style (dashes, dots, solid line, etc).
- Arrow style at the first end and arrow style at the second end.
- Custom scalable icon for the first end and for the second style.

To create the custom path style

---

1. In a profile, create a stereotype.
2. Right-click either the stereotype or a created stereotype (usually created in the profile diagram). Select **Metaclass**. In the **Select Metaclass** dialog, clear all selections. Select the *Relationships* metaclass.
3. Customize the path style. In the stereotype Specification window, click the **Icon** property and then click the ... button.
4. Click **Yes** to open the **Path Icon Customization** dialog.



5. Specify the path style.
6. Click **OK** when you are done.
7. Apply the stereotype(see page 856) to a relationship.

> **Symbol Properties**
> Change the line width and the rest of the path properties in the **Symbol Properties** dialog(see page 975).

Path icon usage rules:

- Custom appearance is used only after applying the stereotype.
- End icons are rotated according to path position and direction.
- The end icon size is scaled according to font size (as with standard arrows).

On this page

- Setting default symbol size(see page 2124)
- Setting custom path style(see page 2125)

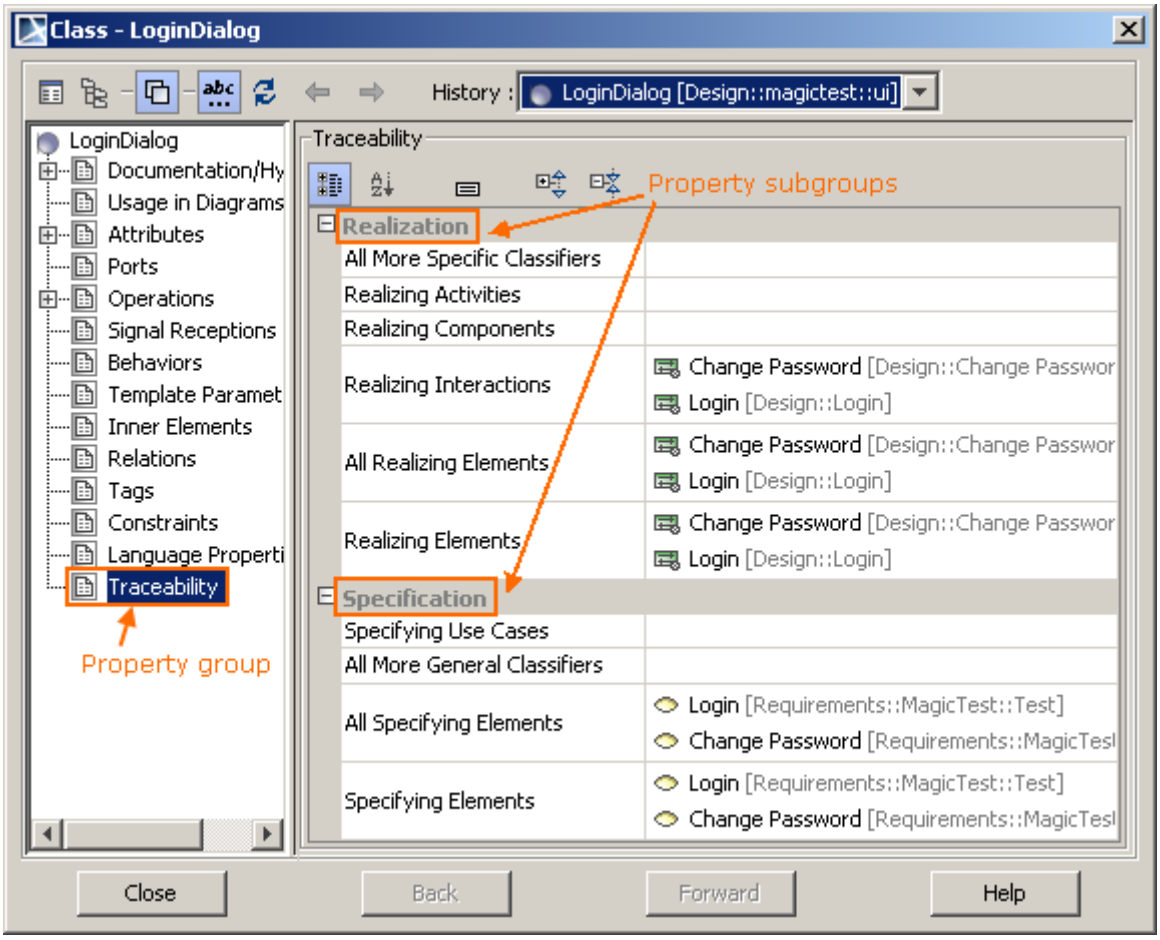
Related Pages

- Customization Element Properties Description(see page 2075)
- Formatting symbols(see page 975)

Creating Property Groups and Subgroups

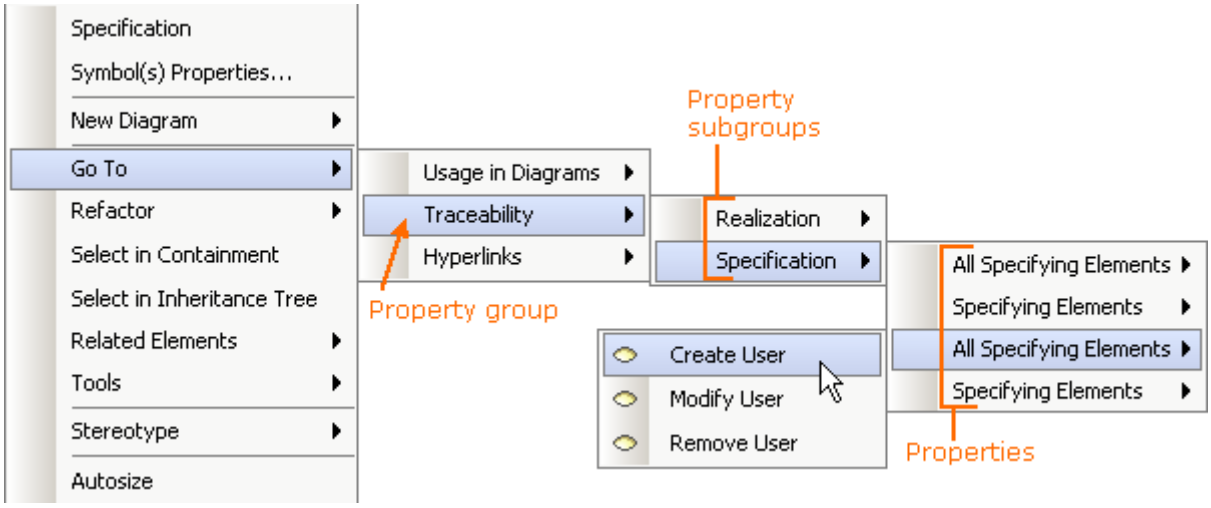
Create your own groups and subgroups to display the properties of both standard UML metaclasses or stereotypes(see page 856). You can also choose where you want to see these property groups and subgroups. Property groups and subgroups can be visible in the following places:

- Element's Specification window(see page 913).



Property groups and subgroups in the element's Specification Window

- Element's **Properties** panel (at the bottom of the Model Browser(see page 29)).
- Element's shortcut menu > **Go To** > submenu.



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Property groups and subgroups in the element's shortcut menu

- **Compartment Edit** dialog(see page 571).
- **Criterion Editor** dialog(see page 366) for editing the relation's criteria in the Relation Map diagram(see page 364).

To create a property group

1. In the Model Browser, right-click a customization element and choose **Create Element** > **Property Group**.
2. Type the attribute's name (used to name the property group).
3. Open the attribute's Specification window.
4. Edit the property values in the general specification pane. Each property is described in the description area in the Specification window.
5. Click **Close**.

A stereotype «propertyGroup»(see page 2129) has been automatically applied to the newly created attribute.

To create a property subgroup

1. Choose the customization element you want to create a property subgroup for. The customization element must have at least one attribute with the stereotype «propertyGroup» applied.
2. Right-click the attribute with a stereotype «propertyGroup» and choose **Create Element** > **Property Group**.
3. Type the attribute's name (used to name the property subgroup).
4. Open the attribute's Specification window.
5. In the general specification pane, edit the property values (properties are described in the table below). A subgroup does not inherit the properties of the group. You must specify them for each subgroup separately.
6. Click **Close**.

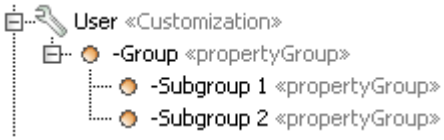
A stereotype «propertyGroup» has been automatically applied to the newly created attribute.

> **Note**
> The stereotype «propertyGroup» is applied when creating either a property group or a subgroup.

Depending on the attribute hierarchy, the attribute will either be a property group or subgroup:

1. If an attribute with a stereotype «propertyGroup» is owned by a customization element, it is a property group.

2. If an attribute with a stereotype «propertyGroup» is owned by another attribute with the same stereotype, then it is a property subgroup.



«propertyGroup» properties
All properties of a stereotype «propertyGroup» are described in «propertyGroup» Properties(see page 2129).

Related Pages

- «propertyGroup» Properties(see page 2129)
- Type Restriction for Custom Table in Specification Window(see page 2131)
- Merging property groups and subgroups(see page 2131)
- Default visibility of property groups and subgroups(see page 2132)

Demo

Property group creation[370]

«propertyGroup» Properties

Property	Description
Show Group In Compartment Edit	This property allows for displaying the element properties in the **Compartment Edit** dialog > the **Element Properties** tab and in a note on a diagram either within the property group/subgroup or without it. If the value is *true*, the property name is displayed as follows: *<property_group_name>::<property_name>*. If the value is **false**, the property name is displayed as follows: *<property_name>* .
Show Group In Element Specification	If the value is *true*, the property group/ subgroup is visible in the element's Specification window(see page 913).

370 https://www.nomagic.com/files/viewlets/md_viewlets_propertygroup_viewlet_swf.html?ml=1&iframe=1

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Property	Description
Show Group In Go To	If the value is *true*, the property group/ subgroup is visible in element's shortcut menu as a submenu under the **Go To** menu.
Show Group In Quick Properties	If the value is *true*, the property group/ subgroup is visible in the element's **Properties** panel.
Show Group In Relation Map	If the value is *true*, the property group/ subgroup is visible in the Criterion Editor dialog(see page 366), when you create or edit a Relation Map diagram(see page 364).
Use As Node	If the value is *true*, the property group is displayed in a separate tab in the element's Specification window and/ or **Properties** panel. If the value is *false*, the property group is displayed as a group in the general (default) pane of the element's Specification window and/ or **Properties** panel. Do not set this property value to *true* for the property subgroup.
Properties	Stores a list of properties that is visible in the property group or subgroup. Do not set this property value to *true* for the property subgroup.
Columns	Stores the specified table columns for showing the values of a multivalued property in the element's Specification window, for example, **ownedOperation** in Class. • Do not specify this property, when there is more than one property assigned to the properties group and/ or subgroup, i.e., when there is more than one value added to the **Properties** property. • The property group, to which the multivalued property is assigned, must be specified as a separate tab in the element's Specification window, for example, the **Use As Node** property must be set to true.
Filter	Stores the element types (UML metaclasses and custom stereotypes) that will be displayed as property values.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Type Restriction for Custom Table in Specification Window

DSL allows you to create a new node in the Specification window(see page 913) and show values of some UML properties in table form. Use the **Filter** property to filter elements by type. The **Filter** property stores the element types (UML metaclasses and custom stereotypes(see page 856)) displayed as property values. The DSL engine checks these values and displays only these types of elements in the table. The **Create** button allows you to create new elements of these restricted types.

Rules for subtypes

- If the type is a stereotype, all subtypes (sub stereotypes) are also accepted. For example, if the type is Requirement(see page 2131), BusinessRequirement(see page 2131) is also accepted.
- If the type is an abstract metaclass, all subclasses are accepted. For example, if the type is Classifier, Class(see page 666), Component(see page 685) and other are accepted. DSL elements are also accepted.

Related Pages

- Creating Customization Data(see page 2068)
- Using Customization Data(see page 2093)
- Creating Property Groups and Subgroups(see page 2126)

Merging property groups and subgroups

You can specify property groups and subgroups for different customization elements of the same element. That element will have groups and subgroups from all customization elements.

Example
Let's say we have two customizations of the same element type, the actor. The first customization has the property group *My Group* with the subgroup *Subgroup 1* containing the property *Prop 1* . The second customization has the same property group with the same subgroup specified, but the subgroup *Subgroup 1* contains the property *Prop 2* .
As a result, the actor element type will have *My Group,* with *Subgroup 1* containing both *Prop 1* and *Prop 2*.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related Pages

- Creating Customization Data(see page 2068)
- Using Customization Data(see page 2093)
- Creating Property Groups and Subgroups(see page 2126)

Default visibility of property groups and subgroups

You can create property groups and subgroups for both standard UML metaclasses and stereotypes(see page 856). The group will be added to either the metaclass or to the stereotype depending on the customization target of the customized property group.

If the customization target is a stereotype and the **Hide Metatype** property is set to true, the specified property group in this customization is visible only for the elements with this stereotype applied.

If the customization target is a metaclass, the specified group in this customization will be visible for the elements of this particular type. This group is also visible for the elements with applied stereotypes, except when the **Hide Metatype** property in this stereotype customization is set to true.

The table below lists the default property group visibility in the element's Specification window(see page 913).

Customization target	**Property group visibility in element's Specification window and Properties panel**		
	Element of metaclass type	**Element with stereotype with DSL customization Hide Metatype=*false***	**Element with stereotype with DSL customization Hide Metatype=*true***
Stereotype	Invisible	Visible in standard mode	Visible in standard mode
Metaclass	Visible in standard mode	Visible in standard mode	Visible in all modes

Related Pages

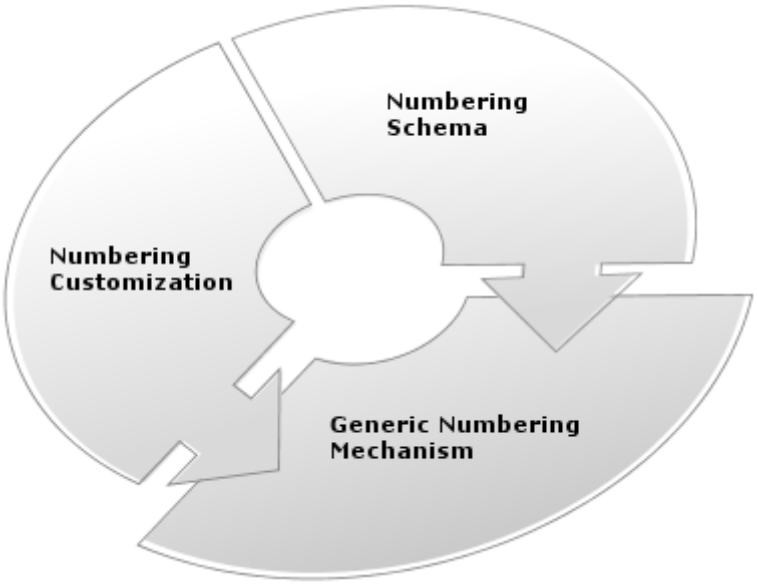
- Creating Customization Data(see page 2068)
- Using Customization Data(see page 2093)
- Creating Property Groups and Subgroups(see page 2126)

Creating Numbering Customizations

On this page

- Basic concepts(see page 2134)
- Basic steps for creating numbering customization(see page 2135)
- Specifying custom numbering properties(see page 2135)

The Generic Numbering Mechanism(see page 1040) consists of the Numbering Customization and Numbering Scheme, shown in the figure below.



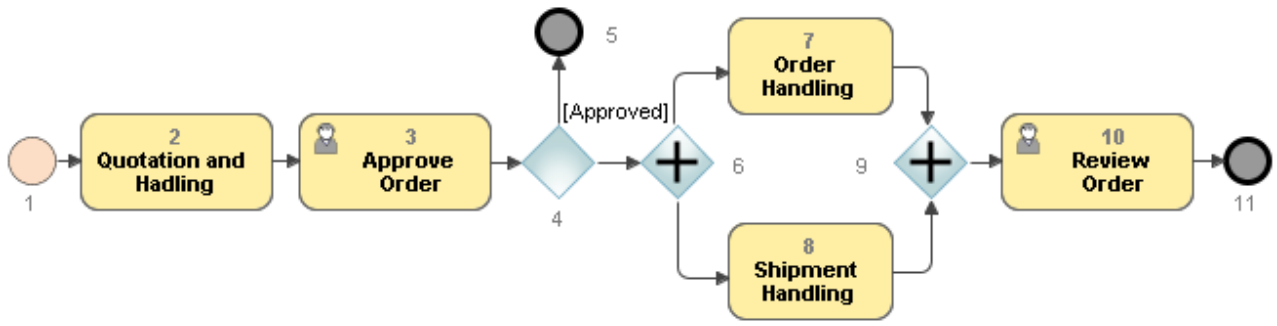
The structure of the Generic Numbering Mechanism.

Create numbering schemes (formats) and apply them to DSL elements in the modeling tool or plugin (SysML[371], Cameo Business Modeler[372], or UPDM plugin[373]).

You can create one or more numbering schemes for the same DSL element. You can also apply the same numbering scheme to several DSL elements. In this case, instances of different DSL elements will be numbered in sequence. For example, if you use the same numbering customization for both actors and use cases, all actors and use cases in a use case diagram will be numbered in sequence. Another example of numbering different DSL elements in sequence is shown in the figure below. The start events, end events, tasks, and gateways in the BPMN Process diagram are numbered using the same numbering scheme.

371 https://docs.nomagic.com/display/SYSMLP190SP4/SysML+Plugin+Documentation
372 https://docs.nomagic.com/display/CBMP190SP4/Cameo+Business+Modeler+Plugin+Documentation
373 https://docs.nomagic.com/display/UPDM3P

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Numbering instances of different BPMN elements in a sequence

Find the instructions to customize the generic numbering mechanism to create your own numbering scheme in Creating your first numbering customization(see page 2135). Create your own numbering customization defining specific numbering properties using expressions(see page 2152).

Basic concepts

Concept	Description
Numbering scheme	Defines a numbering style as well as number parts used to compose a DSL element number. A numbering schema can have one or more number parts. A numbering scheme is represented as a class with the «NumberingScheme» stereotype applied.
Number part	Represents a rule for calculating an individual part of the whole element number, as the element number is composed of one or more individual number parts. It can be a number, character, separator, or other. Number part is represented as a numbering scheme property with the «NumberPart» stereotype applied.
Numbering property	Indicates a DSL element property to store the element number and defines a numbering scheme used for the DSL element numbering. The numbering property is represented as a customization element property with the «AutoNumber» stereotype applied.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Basic steps for creating numbering customization

To create a numbering customization for an element:

1. Create a profile diagram(see page 2276).
2. Create a stereotype(see page 856) to customize a desired element.
3. Add a new attribute to the stereotype.

   This attribute will store the element number.

4. Create a numbering scheme to define a numbering style and number parts(see page 2138).
5. Create a customization element(see page 2073) for the previously created stereotype.
6. Add a numbering attribute to the customization element. Reopen the project to apply changes.

Specifying custom numbering properties

To define your own specific numbering customization, you need to define an expression(see page 2152). For general numbering it is a binary script. The following script example illustrates the multi-level numbering:

```
binary;Binary;com.nomagic.magicdraw.autoid.LinkedDUCBinary;
```

Where *com.nomagic.magicdraw.autoid.LinkedDUCBinary* is your binary class that implements *INumbering Action.*

For more information, see Custom elements numbering(see page 2347).

Related pages

- Creating your First Customization(see page 2070)
- Extending Metamodel with Derived Properties(see page 2146)

Creating your first numbering customization

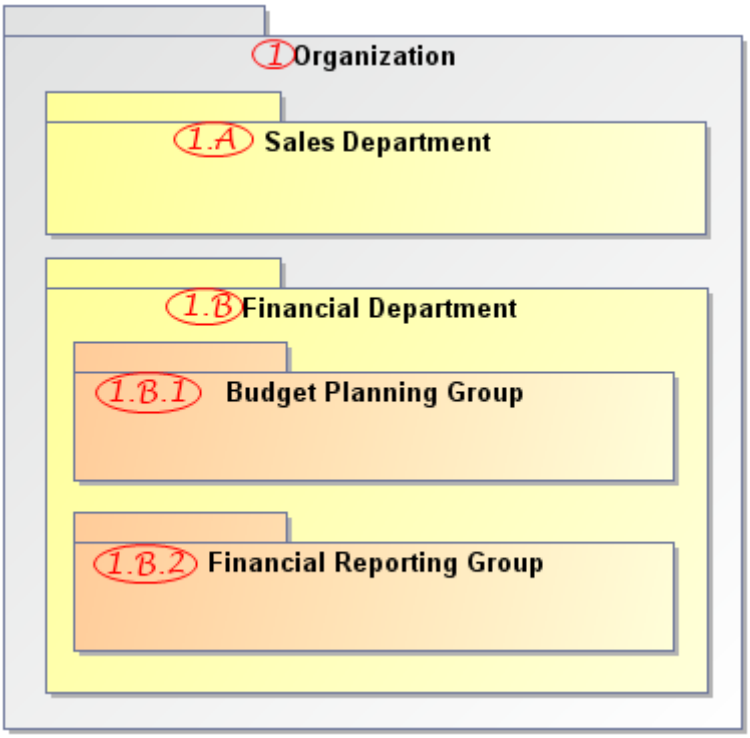
To create a numbering customization for an element

- 1. Creating a stereotype to customize a desired element(see page 2137)
- 2. Adding a new attribute for storing numbers(see page 2137)
- 3. Creating a numbering scheme to define a numbering style and number parts(see page 2138)
- 4. Creating a customization element for the stereotype that customizes the desired element(see page 2141)
- 5. Add a numbering attribute to the customization element(see page 2142)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- 6. Changing the number display mode(see page 2143)
- 7. Reopen the project to apply changes(see page 2143)

Assume you need to number packages as shown:



Task for numbering packages.

There should be three numbering levels. The following table describes the rules for calculating package numbers in each level:

Numbering level	Numbering format	Examples
1st	<number>	“1”, “2”, “3”
2nd	<number><separator><character>	“1.A”, “1.B”, “2.A”, “2.B”
3rd	<number><separator><character><separator><number>	“1.A.1”, “1.A.2”, “1.A.3”, “2.B.1”, “2.B.2”

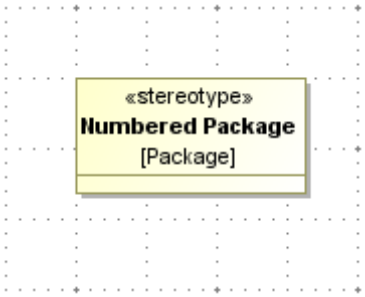
So we need to create a numbering scheme and use it in the numbering customization for Package(see page 800) elements.

Perform this task in the Quick Start perspective(see page 69).

1. Creating a stereotype to customize a desired element

To create a stereotype that customizes UML Packages

1. Create a Profile diagram(see page 483).
2. On the Profile diagram palette, click the **Stereotype** button.
3. Click a free space on the diagram pane. A new stereotype shape appears and the **Select Metaclass** dialog opens.
4. Select the *Package* metaclass and click **OK**.
5. Name the stereotype *Numbered Package*.



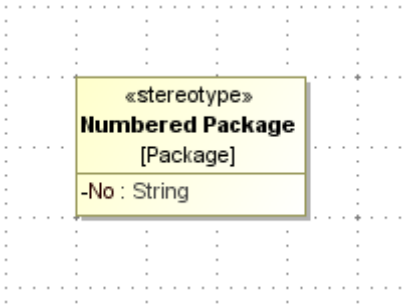
Stereotype for customizing UML packages.

> Make the «Numbered Package» stereotype invisible if you want the stereotype and its properties to be hidden in a diagram.

2. Adding a new attribute for storing numbers

To add a new attribute for storing package numbers to the «Numbered Package» stereotype

1. Select the shape of the «Numbered Package» stereotype on the Profile diagram pane.
2. Add a new stereotype attribute of the String type named *No*.



Property for storing package numbers.

3. Creating a numbering scheme to define a numbering style and number parts

This heading has incoming link from Creating Numbering Customizations[374]. Update the link if you change the heading.

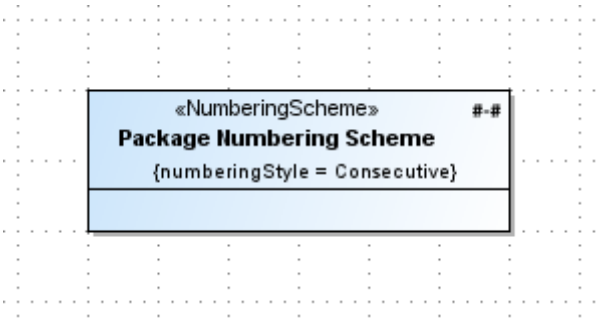
To create a numbering scheme

374 https://docs.nomagic.com/display/MD190/Creating+Numbering+Customizations

1. On the Profile diagram palette, click the **Class** button.
2. Click a free space on the diagram pane. A new Class shape appears.
3. Name the Class *Package Numbering Scheme*.

   The name of the numbering scheme will be displayed in the **Element Numbering dialog**(see page 1046). We recommend giving it a short and meaningful name.

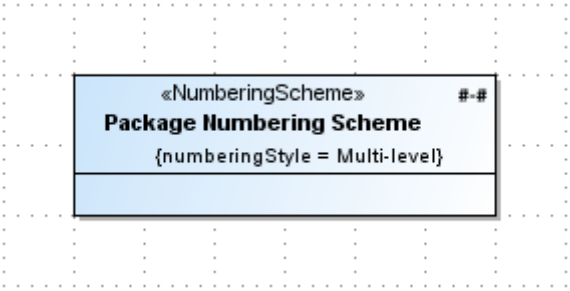
4. Apply the «NumberingScheme» stereotype to the Class.



Scheme for numbering UML packages.

To define a numbering style

1. Open the Specification window(see page 913) of the newly created numbering scheme.
2. Click the **Numbering Style** value cell to open the list of available numbering styles. Select the **Multi-level** numbering style in this list. Use the Multi-level numbering style (remember: you must have three numbering levels for numbering packages).
3. Click **Close** when you are done.



Numbering scheme with multilevel numbering style defined.

The following table shows number parts that should be used to create the numbering formats described in the preceding table.

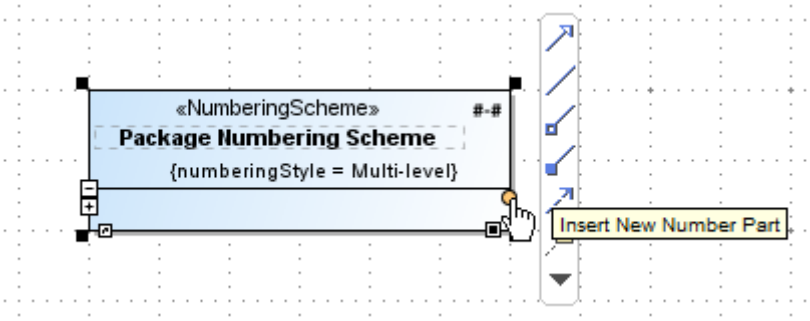
Number part	Number part name
<number>	Level 1
<separator>	Dot 1

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Number part	Number part name
<character>	Level 2
<separator>	Dot 2
<number>	Level 3

To create a number part

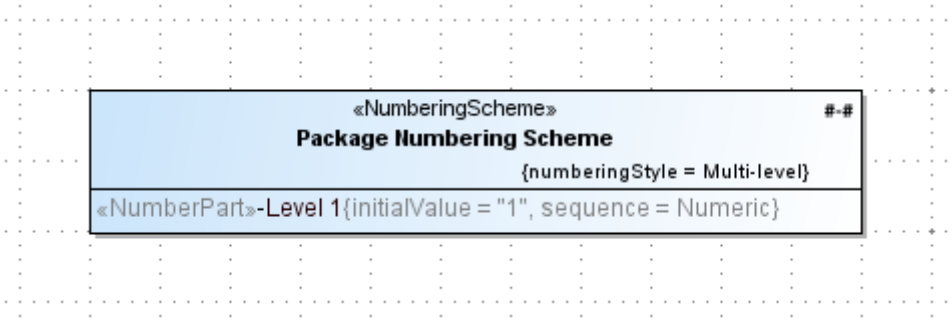
1. Select the shape of the numbering scheme on the profile diagram pane.
2. Add a new number part.



Creating a new number part.

To specify a number part for the first numbering level

1. Create a number part named Level 1(see page 2138).
2. Open the Specification window(see page 913) of the number part.
3. Click the **Sequence** property value specification cell and select **Numeric** in the values list. You must have numbers at the first numbering level.
4. Click the **Initial Value** property value cell and type the number **1** as its value. Numbering at the first level should start at **1**.
5. Click **Close** when you are done.



Numbering scheme with number part defined for first numbering level.

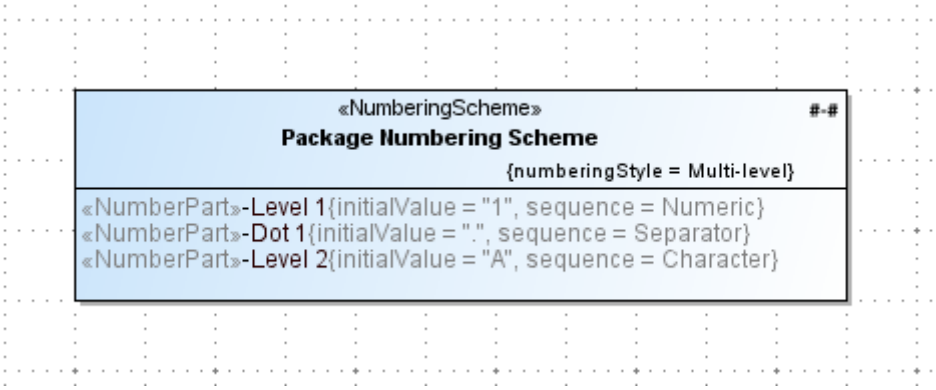
To specify a number part for the first separator

1. Create a number part named Dot 1(see page 2138).
2. Open the Specification window(see page 913) of the number part.
3. Click the **Sequence** property value specification cell. Select Separator in the values list.
4. Click the **Initial Value** property value cell and type the character (a dot) as its value. The separator between the first and the second numbering level should be a dot.
5. Click **Close** when you are done.

To specify a number part for the second numbering level

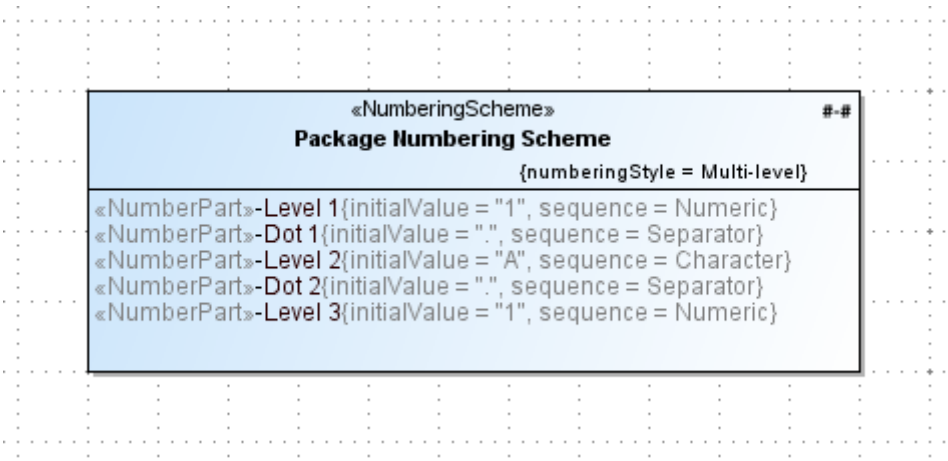
---

1. Create a number part named Level 2(see page 2138).
2. Open the Specification window(see page 913) of the number part.
3. Click the **Sequence** property value specification cell. Select **Character** in the values list. You must have letters at the second numbering level.
4. Click the **Initial Value** property value cell and type the letter A as its value. The numbering at the second level should start at A.
5. Click **Close** when you are done.



Numbering scheme with number parts defined for the second numbering level.

The rest of the number parts are created accordingly.



Numbering scheme with all number parts defined.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

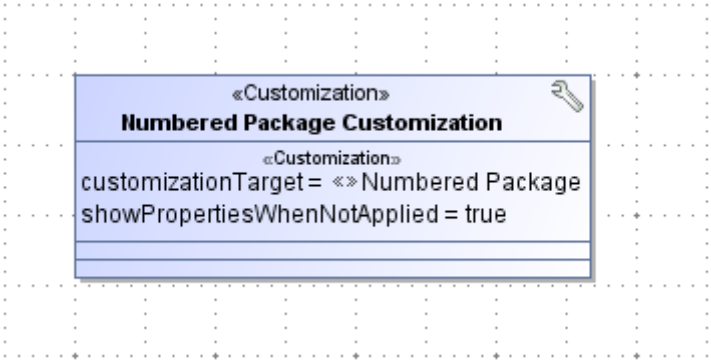
As there are no number parts specified for the fourth (and beyond) numbering levels, packages will be numbered using the number parts of the third numbering level. For example, “1.A.1.1”, “1.B.1.2”, and so forth.

4. Creating a customization element for the stereotype that customizes the desired element

In order to use the scheme for numbering packages, you must define it in the numbering customization for package elements. First, create a customization element.

To create a customization element for the “Numbered Package” stereotype

1. On the diagram palette, click the **Customization** button.
2. Click a free space on the diagram pane. The new customization shape appears.
3. Name the customization *Numbered Package Customization*.
4. Open the **Customization** Specification window(see page 913).
5. Click the **Customization Target** property. Select the *Numbered Package* stereotype in the element Selection dialog and click **OK**.
6. Set the **Show Properties When Not Applied** property to *true*. All properties of the *Numbered Package* stereotype will be visible as properties of the *Package* metaclass, even if the stereotype is not yet applied.
7. Click **Close** when you are done.



Customization element for the “Numbered Package” stereotype.

5. Add a numbering attribute to the customization element

After creating the customization for the *Numbered Package* stereotype, specify the stereotype attribute for storing element numbers as well as defining a numbering scheme to use. For this, you must create a numbering attribute.

To add a numbering attribute to the customization element of the *Numbered Package* stereotype

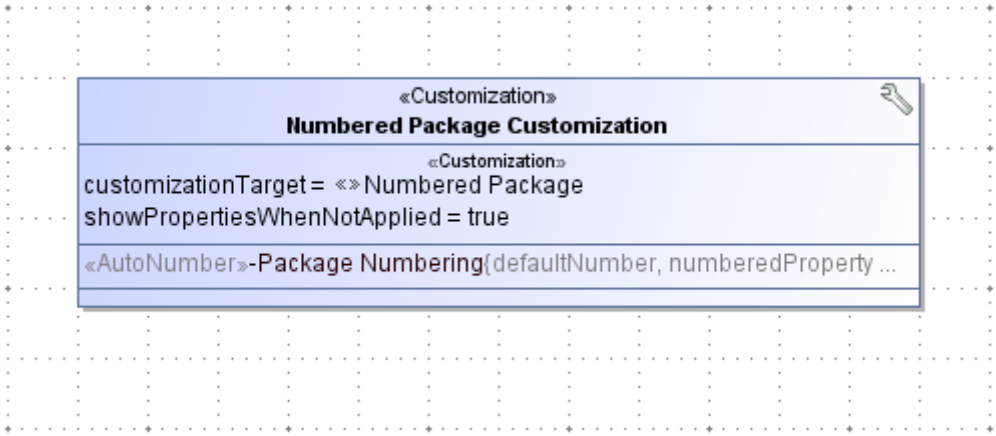
1. Select the shape of the customization.
2. Add a new **Auto Numbering Property** named *Package Numbering*.
3. Open the Specification window(see page 913) of the created attribute.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

4. Specify a value for the **Numbered Property** property. Click the property value cell and then click the **...** button. In the element Selection dialog, select the *No.* property of the *Numbered Package* stereotype and click **OK**.
5. Specify a value for the **Numbering Scheme** property. Click the value cell and either:

    - Select the *Package Numbering Scheme* class in the opened list.
    - Click the **...** button. In the element Selection dialog, select the *Package Numbering Scheme* class and click **OK**.

6. Change the **Default Number** property value to *true*. The *true* value indicates that this numbering attribute will be set as the default when numbering instances of the customized *Pack age* metaclass.
7. Specify the **Use Prefix On Owner** attribute value. The *true* value indicates that the element prefix will be used for its numbered owner as well.
8. Click **Close** when you are done.
9. Save and reopen the project to apply the created numbering customization.



Numbering attribute in the customization element of the “Numbered Package” stereotype.

6. Changing the number display mode

To see the number before the element name, you must change the number display mode.

To change the number display mode on packages

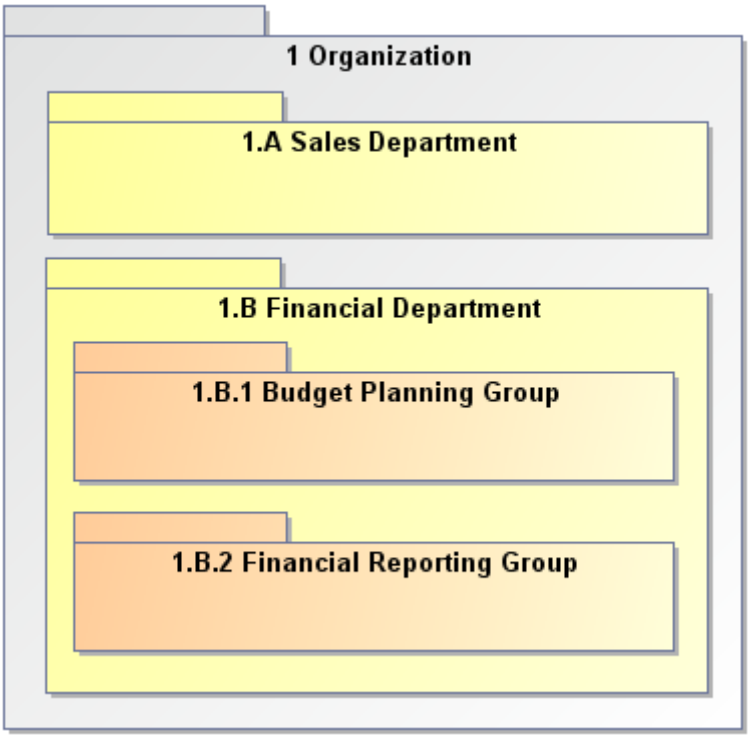
---

1. Open the appropriate package diagram.
2. Create a package to apply the numbering scheme.
3. Name the package *Organization*.
4. Apply the *Numbered Package* stereotype to the *Organization* package.
5. Right-click the shape of the package on the diagram pane. Click **Symbol Properties**.
6. Set the **Show Number Tag Name** property value to *false*. The *false* indicates that the element number tag name (the *No* property of the *Numbered Package* stereotype) will not be displayed on the package shape.
7. Click the **Element Number Display Mode** value cell to open the list of available modes. Select the **Before the element name** mode in this list.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

7. Reopen the project to apply changes

Open the appropriate package diagram and see the packages numbered using the newly created numbering customization.



Packages numbered using a numbering customization.

Related pages

- Creating Numbering Customizations(see page 2132)
- Numbering Scheme property values(see page 2144)
- Profile diagram(see page 324)

Numbering Scheme property values

On this page

- Numbering Style property values(see page 2145)
- Sequence property values(see page 2145)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Numbering Style property values

Property value	Description
Multi-level	Select this style to number elements in several numbering levels. Specify at least one number part for each numbering level. Top level elements will be numbered using the first number part. Elements in subsequent levels (directly or indirectly contained in the top level element) will be numbered using successive number parts. If you specified fewer number parts than there are element levels, both the last element number part of the separator type and the last element number part of the non-separator type (numeric or character) will be reused to number subsequent level elements. For instance, if number parts are specified only for two numbering levels, and elements in the second level are numbered as “1.A”, “1.B”, and “1.C”, elements in the third numbering level will be numbered as “1.A.A”, “1.A.B”, etc.
Consecutive	Select this style to number elements in one numbering level. Specify only one number part. If you specified more than one number part, elements will be numbered using only the last number part. All other number parts in this case will be treated as static. For instance, if an element number includes five number parts (i.e., “1.1.1”). and the consecutive numbering style is selected, all elements will be numbered by changing only the last number part (“1.1.1”, “1.1.2”, “1.1.3”, etc.). The same happens after changing a multilevel numbering style to consecutive.

Sequence property values

Property value	Description
Numeric	Select this type to use only positive numbers for calculating a value of the number part. Elements will be numbered starting with “1” in ascending order, if no initial value is specified.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Property value	Description
Character	Select this type to use only Latin letters for calculating a value of the number part. Elements will be numbered starting with “A” in ascending order, if no initial value is specified.
Expression	Select this type to use a Binary (Reference to a code class) expression for calculating a value of the number part. Use the **Expression Editor** dialog to define the expression.
Separator	Select this type to specify a separator between two number parts. Do not forget to specify the separator value for the **Initial Value** property. If the property value is not specified, the separator will not be used.
OwnerNumber	Select this type to use the number of an owner numbered using another numbering customization for calculating a value of the number part.

Related pages

- Creating Numbering Customizations(see page 2132)
- Creating your first numbering customization(see page 2135)

Extending Metamodel with Derived Properties

DSL customization allows extending the UML metamodel or its profile (such as SysML or UPDM) by creating new read-only derived properties.

A derived property's values are automatically calculated from the other properties’ values.

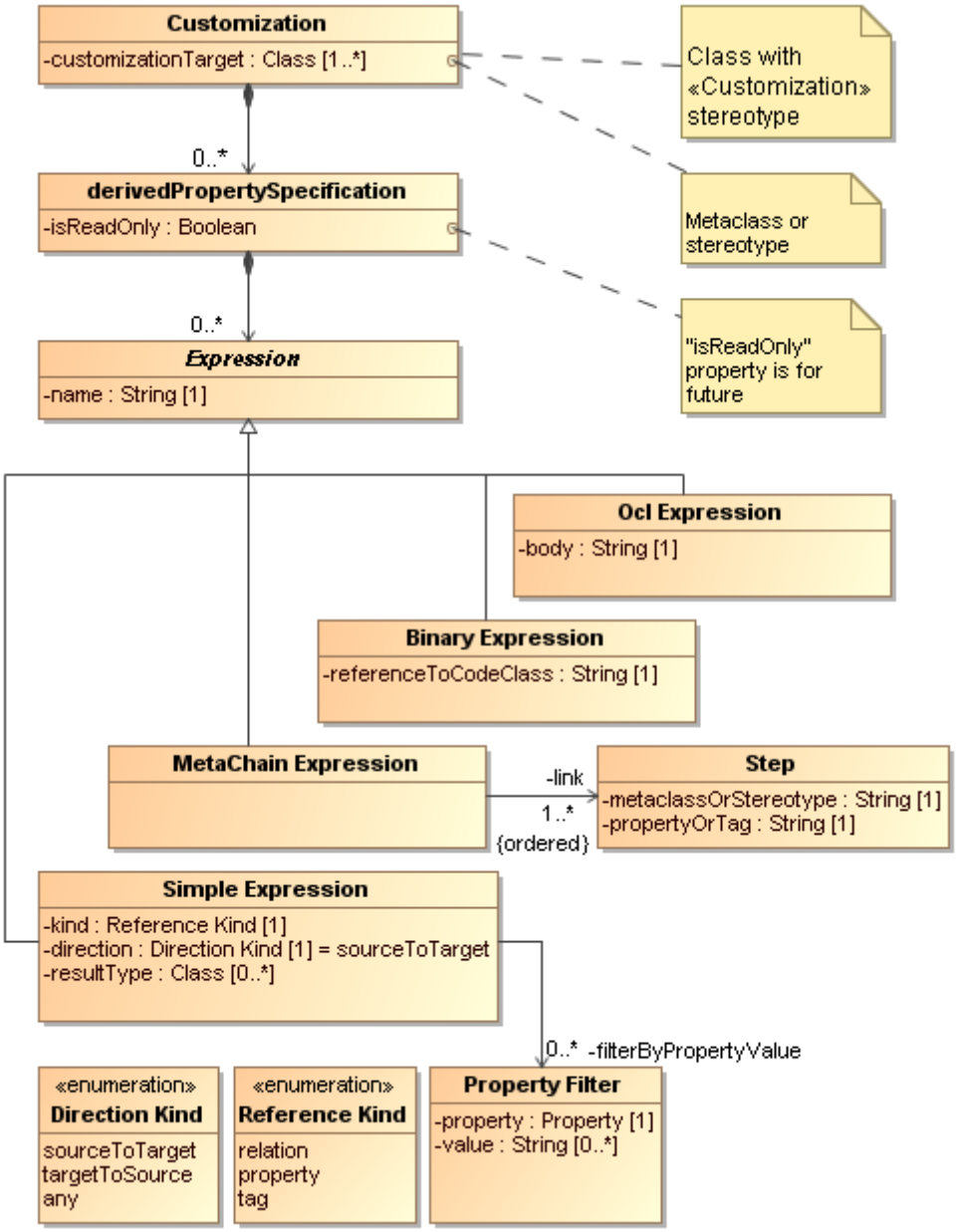
Derived properties are calculated on the fly. The modeling tool analyzes the derived properties and calculates them for existing model elements (when loading a model) and dynamically updates them according to model changes.

Derived properties can be added either to the standard UML elements, or to the ones extended with stereotypes. To extend the metamodel with the derived properties, there is no need to apply the stereotypes: the metamodel is extended in runtime.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Define derived properties using different expression types: Simple, Meta Chain, OCL, and Binary (reference to java code).

Easily access derived properties of a select model element, navigate to its specifications, or use the derived properties in model analysis tools (dependency matrices, report templates, and relation maps).



Derived properties specification metamodel.

Related pages

- Creating and specifying derived properties(see page 2146)
- Defining expressions(see page 2152)
- Recursive properties(see page 2153)
- Expressions merge(see page 2153)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Derived properties visibility(see page 2154)

Demo

Derived properties creation[375]

Creating and specifying derived properties

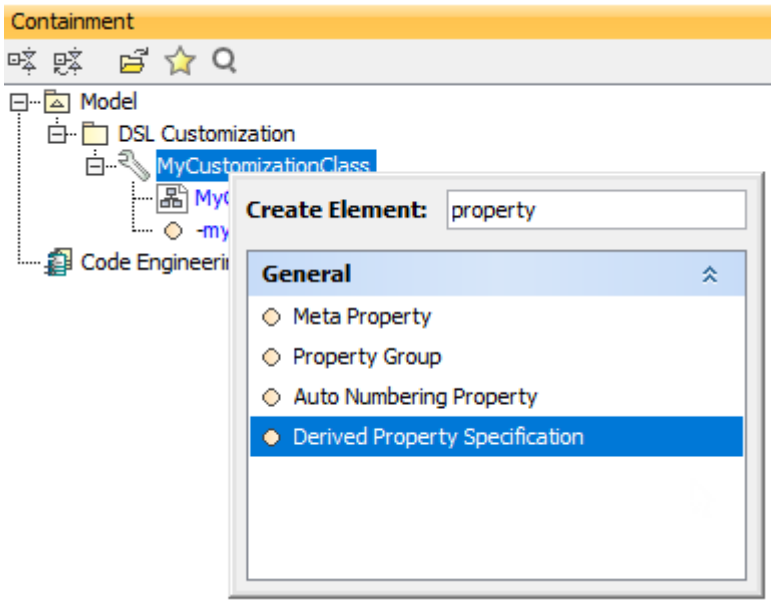
To be included in a MagicDraw project as a property extending a UML element, an attribute must be owned by a DSL customization element and stereotyped as «derivedPropertySpecification».

To create and specify a derived property

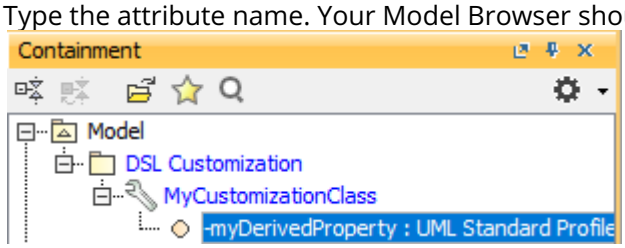
1. Select the customization element for which you want to create a derived property.

   The derived property will be created for the stereotype or metaclass, defined as a customization target in the selected customization element specification.

2. Select **Create Element** > **Derived Property Specification** from the shortcut menu. The attribute is created.



3. Type the attribute name. Your Model Browser should look like this at this stage:



375 https://www.nomagic.com/files/viewlets/md_viewlets_derivedproperties_viewlet_swf.html?ml=1&iframe=1

> The attribute's name corresponds to the name of the derived property.
> Type derived property names in camel case (i.e.,*realizingActivities*).
> A property name defined in camel case will be displayed in separate words with the first word capitalized. For example, realizingActivities will be changed to *Realizing Activities* (visible in the user interface).

4. Edit property values specifying the derived property in the attribute's Specification window.

> All properties are specified in the general specification pane. Some are also specified in the **Tags** specification pane. For detailed information, see the properties description table(see page 2149).

> If you do not see the newly created derived property in the element's Specification window(see page 913), please read the following:
> - In the element's Specification window(see page 913), the specified derived property is visible by default only in the **All** properties mode. First, try switching to this mode. Also try changing the mode in which you want the derived property to be visible (click the **Customize** button).
> - If you have specified the derived property for the element (with the **Used UML Properties** property of DSL customization specified) you must add the newly created derived property to the element's visible properties list. For detailed information, please refer to Derived properties visibility(see page 2154).

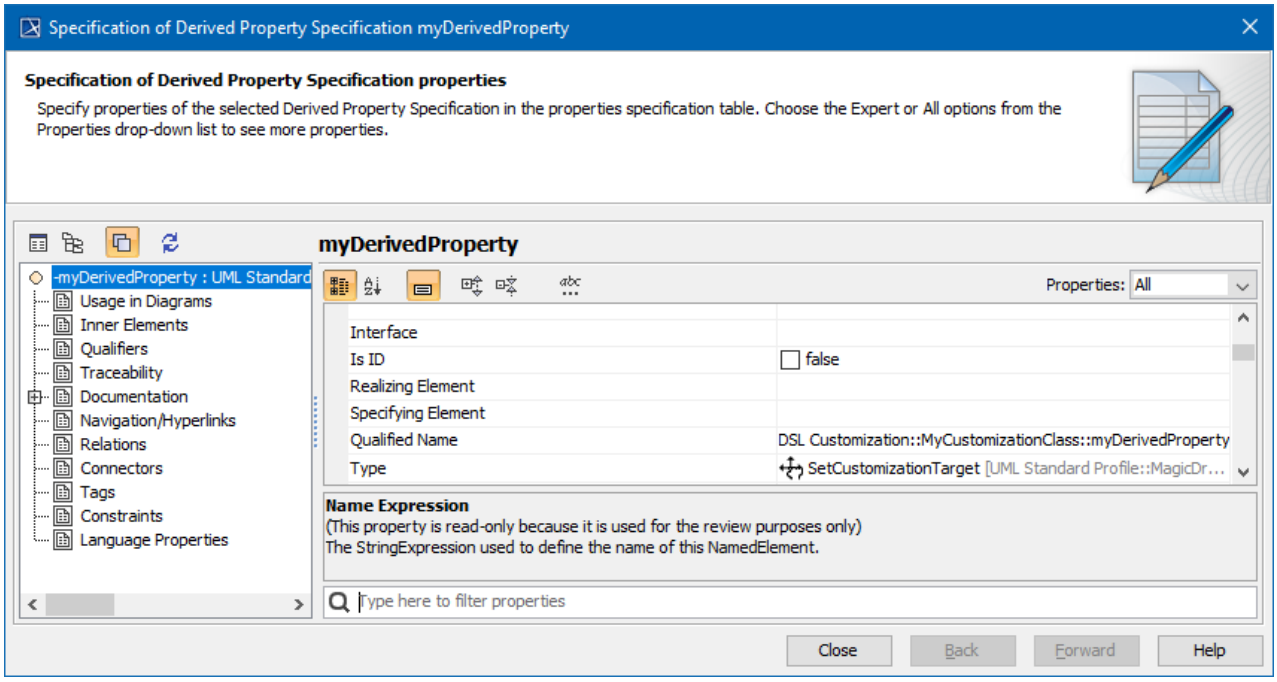
The following table describes the properties specified for the derived property.

Property	Description
Name	The name of the property. This name will be added and visible in the specification of the element whose type is defined as a customization target.
Type	The type of the derived property values. You may set a model element type or a data type (e.g., string, boolean, orintegerfrom the UML Standard Profile) as a property value. To display classes and components as property values, select the classifier as a type for the derived property value.

Property	Description
Multiplicity	The multiplicity of the derived property values. If the multiplicity is “0..1”, the derived property value will be shown as single. If the multiplicity is “0..*”, the derived property values will be shown as collection.
Is Read Only	If *true*, the derived property is read-only. You cannot edit the result elements for the derived property value. If *false*, the property is read-write. You can edit the result elements for the derived property according to the expressions defined.
Is Ordered	If *true*, the property values are always displayed in the same order. If *false*, the property values are displayed in nonpredictable order.
Is Unique	If *true*, the property values are unique. The same element is displayed only once. If *false*, the property values may be displayed more than once.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Property	Description
Expression	One or more expressions of the derived property. An expression defines the criterion for selecting result elements. There are ten possible expression types: • Metachain navigation (indirect relations through chains of properties) • Find • Union • Exclude • Filter • Various scripts • Implied Relation • Type Test • Property Test • Operation from Model The criterion for selecting the result elements can also be any combination of the expression types above. For the detailed information, see Defining expressions(see page 2152). Only one Simple Navigation operation criterion can be assigned to the expression of the derived property.
Documentation	The text displayed as the derived property description in the element's Specification window(see page 913) and Properties panel.
customizationTarget	The specified derived property will be added to one or more element types (stereotypes and/ or metaclasses). The expression calculations begin from the element type of the customization target.



The Specification window of the derived property

Related pages:

- Extending Metamodel with Derived Properties(see page 2145)
- Using Customization Data(see page 2093)
- Creating Customization Data(see page 2068)
- Defining Expressions(see page 2152)

Defining expressions

Each derived property must have one or more expressions defining the criteria for searching the result elements.

To open the dialog for a new expression definition

It is strongly recommended that you specify the customization target in the customization element before defining the expression for its derived property.

1. Select the attribute with the stereotype «derivedPropertySpecification»(see page 2147).
2. Open the attribute's Specification window(see page 913).
3. In the general specification pane (opened by default), click the **Expression** property and then click the **...** button. The **Expression** dialog opens, and you can then construct predefined expressions or define your custom operation for calculating derived property values.

   Select the **Expert** mode to see the full list of available operations.

Related pages

- Extending Metamodel with Derived Properties(see page 2145)

Recursive properties

A recursive property is defined as a binary class "*com.nomagic.magicdraw.derivedproperty.RecursiveExpression()*" with parameters *<property-name>* and *<recursive-property-name>*.

Specify the expression in the following form - *com.nomagic.magicdraw.derivedproperty.RecursiveExpression(<property-name>, <recursive-property-name>* (e.g. *com.nomagic.magicdraw.derivedproperty.RecursiveExpression(Classifier.specificClassifier, Classifier.allSpecificClassifiers)*

Recursive properties enable you to collect and represent:

- All elements directly or indirectly related to a specific element through the generalization relationships.

  Refer to the derived property customization expression, available at "UML Standard Profile::MagicDraw Profile::Traceability customization::Properties descriptors::Other::All Specific Classifiers::allSpecificClassifiers".

- All elements directly or indirectly related to a specific element through the abstraction and other types of relations.

  Refer to the derived property customization expression, available at "UML Standard Profile::MagicDraw Profile::Traceability customization::Properties descriptors::Realization::Element All Realizing Elements::allRealizingElements".

Related Pages

- Extending Metamodel with Derived Properties(see page 2145)
- Using Customization Data(see page 2093)
- Creating Customization Data(see page 2068)

Expressions merge

Suppose you have two customization elements extending the same UML element (for example, the same element specified as a customization target in both customization elements). Each customization element has a derived property. If the derived properties' names, types, and multiplicities are the same, then the expressions defined for these derived properties are merged. Consequently, the extended UML element will have only one derived property instead of two and the property values will be calculated according to the union of expressions defined in both derived properties.

This occurs when the derived property (already specified in, for example, the UML Standard Profile) is also specified in a project. Note that in contrast to the example above the real-life expression merge can be performed in more complex situations with more than two derived properties having more than one expression defined. The expression merge can combine different types of expressions.

Related Pages

- Extending Metamodel with Derived Properties(see page 2145)
- Using Customization Data(see page 2093)
- Creating Customization Data(see page 2068)

Derived properties visibility

A newly created derived property is visible by default in the following places:

- General specification pane of the DSL element's Specification window(see page 913).
- **Properties** panel of the DSL element.
- **Compartment Edit** dialog.

  To see derived properties of customized elements that have the **Used UML Properties** property specified, ensure that derived properties are assigned as values for the **Used UML Properties** property.

- **Criterion Editor** dialog for editing relation criteria in a Relation Map diagram.
- **Select Dependency Criteria** dialog for editing the dependency criteria in a Dependency Matrix diagram.

Additionally, DSL customization allows more precision when specifying a derived property's visibility, as this can be done through the specification of the DSL element's properties group or subgroup to which the derived property is assigned. As a result, the derived property will be visible in the group or subgroup according to this group's or subgroup's visibility settings.

To make derived properties of customized elements having the Used UML Properties property specified visible

1. Open the Specification window(see page 913) of the customization element.
2. In the general specification pane, select **Used UML Properties** and assign one or more derived property values.
3. Save and reopen the project.

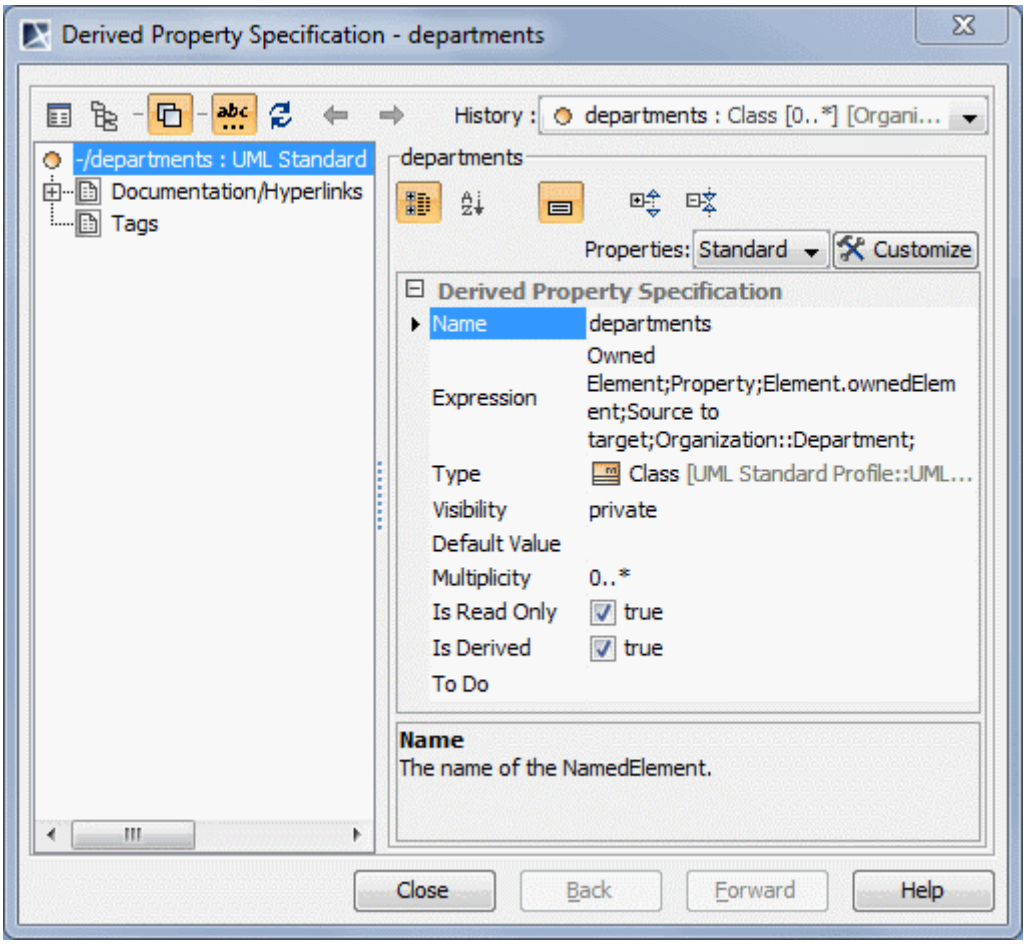
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To assign the derived property to a property group

1. Select the customization element with the derived property.
2. Create a property group for the customization element.
3. In the general specification pane, edit property values that specify the property group's visibility.
4. In the property group Specification window(see page 913), click the **Properties** property and then click the **...** button.
5. In the **Item Filter** dialog, select the check box near the appropriate derived property and click **OK**. As shown in the following figure, the selected derived property will be assigned to **Properties** as a new value.
6. Click **Close**.

Use a similar procedure to assign the derived property to a property subgroup. Create a property subgroup in addition and assign the derived property to the subgroup, rather than the property group.

> You can also assign a derived property to a property group not specified in the same customization element if both customization elements extend the same UML element.



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Derived property assigned to property group.

All derived properties, together with custom UML properties, are available for creating your own Report templates.

Related Pages

- Extending Metamodel with Derived Properties(see page 2145)
- Using Customization Data(see page 2093)
- Creating Customization Data(see page 2068)
- Creating Property Groups and Subgroups(see page 2126)

Distributing Profiles

You can distribute profiles with included customization data in the following two ways:

- Exchanging Profiles via XMI
- Distributing Resources by Using Resource Manager(see page 1150)

Exchanging Profiles via XMI

Using UML XMI interchange mechanisms it is possible to interchange profiles between tools, together with models to which they have been applied. A profile must therefore be defined as an interchangeable UML model. In addition to exchanging profiles together with models between tools, profile application should also be definable “by reference”; that is, a profile does not need to be interchanged if it is already present in the importing tool.

Related pages

- Resource Manager(see page 1150)
- Distributing Resources(see page 2185)

Developer Guide

This guide describes the Open Java API of MagicDraw or a Cameo Suite product, such as Cameo Systems Modeler, and provides instructions on how to implement custom plugins, add actions to the menus or toolbars, change UML model elements as well as create new patterns. Though MagicDraw is used in examples mainly, other modeling tools, since they are based on the MagicDraw modeling platform, performs the same.

The program core code and used libraries are packaged in jars files that are located in *<modeling tool installation directory>\lib* (and its subdirectories).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Some plugins also exposes Open API. Plugins libraries can be located in *<modeling tool installation directory>\plugins\<plugin_name>*

The code can be divided into the following three API scope groups:

- **OpenAPI** – a code for public usage, stable through builds and versions.
- **InternalAPI** – a code for the internal NoMagic usage only, may change through builds and versions without any restrictions.
- **NotAPI** – not an API code, it may change in each build.

Java annotations are used to distinguish which API scope in the provided code belongs to. Annotations of the type (class, interface, enum) declaration:

Annotation	API Scope	Scope applies to
@OpenApiAll	OpenAPI	The annotated type declaration and all declared members, constructors.
@OpenApi	OpenAPI	The annotated type declaration only. Each declared member, constructor is annotated explicitly with the @OpenApi or @InternalApi annotation (see the following table).
@InternalApi	InternalAPI	The annotated type declaration and all declared members, constructors.
@NotApi	NotAPI	The annotated type declaration and all declared members, constructors.

Annotations of the member, constructor declaration of the @OpenApi type:

Annotation	API Scope	Scope applies to
@OpenApi	OpenAPI	The annotated declared member or constructor.
@InternalApi	InternalAPI	The annotated declared member or constructor.

Only code from the OpenAPI scope should be used to extend the modeling tool. All the InternalAPI and NotAPI scope code are additionally annotated with the *@Deprecated* annotation. This allows seeing the illegal API usage in all modern IDEs without an additional configuration.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

We provide a set of plugin samples in *<modeling tool installation directory>\openapi\examples*. By using these examples, you can find out on how to use an OpenAPI.

In the generated javadoc[376], you will find the detailed description of classes, their attributes, and operations. JavaDoc is located in *<modeling tool installation directory>\openapi\docs*.

Plugins

Plugins are the only one way to change the functionality of a modeling tool. The main purpose of the plugin architecture is to add a new functionality to your modeling tool. Although there is a limited ability to remove an existing functionality using plugins.
A plugin must contain the following resources:

- A directory
- Compiled java files, packaged into a jar file
- A plugin descriptor file
- Optional files used by the plugin

Typically a plugin creates some GUI components allowing a user to use the plugin functionality. Generally, this is not necessary because the plugin can listen for some changes in a project and activate itself on the desired changes.

Related pages

How plugins work

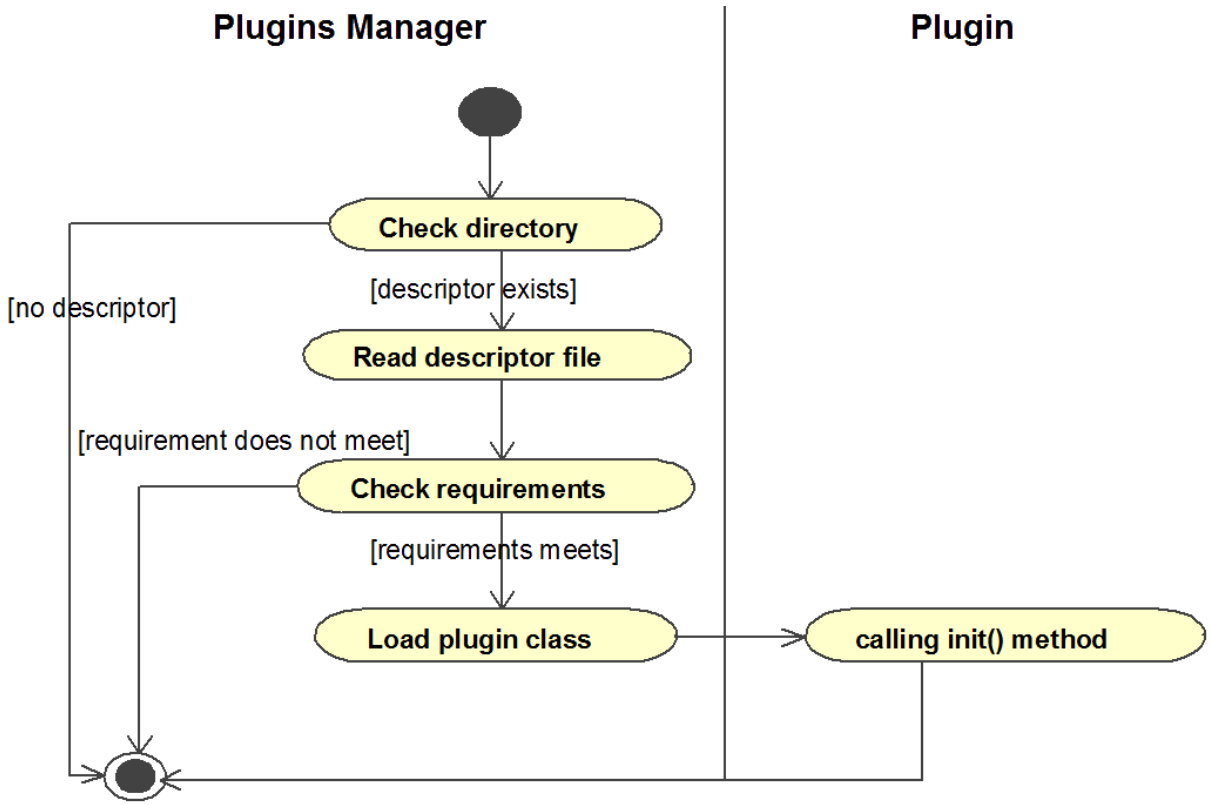
On every startup, a modeling tool scans the plugins directory(see page 2168), and searches for subdirectories there:

- If a subdirectory contains the plugin descriptor file, the plugin's manager reads the descriptor file.
- If requirements specified in a descriptor file are fulfilled, the plugin's manager loads a specified class (the specified plugin class must be derived from the com.nomagic.magicdraw.plugins.Plugin[377] class). Then a method *init()*[378] of the loaded class is called. The *init()*[379] method can add GUI components using the actions architecture or do other activities and return from the method. The *init()*[380] method is called only if *isSupported()*[381] returns *true*.

The following figure illustrates how do plugins work.

376 http://jdocs.nomagic.com/190beta/
377 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/plugins/Plugin.html
378 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/plugins/Plugin.html#init--
379 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/plugins/Plugin.html#init--
380 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/plugins/Plugin.html#init--
381 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/plugins/Plugin.html#isSupported--

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Related pages

- Plugins directories(see page 2168)

Writing plugins

With this example, we will create a plugin that displays a message on the program startup. To create the plugin, you need to write a plugin descriptor.

Step #1: Create your plugin folder in a *plugins* folder

Create a *myplugin* folder in the *plugins* folder in the installation directory of the modeling tool.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Step #2: Write a plugin code

The plugin must contain at least one class derived from the com.nomagic.magicdraw.plugins.Plugin[382] class.

```
package myplugin;
import com.nomagic.magicdraw.plugins.Plugin;
import javax.swing.*;
public class MyPlugin extends Plugin
{
    @Override
    public void init()
    {
        JOptionPane.showMessageDialog(null, "My Plugin init");
    }
    @Override
    public boolean close()
    {
        JOptionPane.showMessageDialog( null, "My Plugin close");
        return true;
    }
    @Override
    public boolean isSupported()
    {
        //plugin can check here for specific conditions
        //if false is returned plugin is not loaded.
        return true;
    }
}
```

This plugin shows the one message when it is initialized, and another message when it is closed.

Step #3: Compile and pack the plugin to a *.jar* file

> For more information on the classpath specification details, see Classpath(see page 2161).

The compiled code must be packed to a *.jar* file.

To create a .jar file, use a jar command in the plugins directory:

```
jar -cf myplugin.jar myplugin/*.class
```

382 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/plugins/Plugin.html

Step #4: Write a plugin descriptor

The plugin descriptor is a file named *plugin.xml*. This file should be placed in the *myplugin* folder.

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin id="my.first.plugin" name="My First Plugin" version="1.0" provider-name="Code
r" class="myplugin.MyPlugin">
    <requires>
        <api version="1.0"/>
    </requires>
    <runtime>
        <library name="myplugin.jar"/>
    </runtime>
</plugin>
```

For detailed information about the plugin descriptor, see Plugin descriptor(see page 2163).

Related pages

- Compilation classpath(see page 2161)
- Plugin descriptor(see page 2163)

Compilation classpath

Core libraries

The correct classpath must be specified when compiling your (plugin) code or running `*.jar` files from *`<modeling tool installation directory>\lib`*, as well as all its subdirectories must be included into the classpath.

You can open the properties file in *`<modeling tool installation directory>\bin`* and check the `CLASSPATH` property to be sure what files must be included.

Make sure that following *.jar files are added in such order:

1. `patch.jar`
2. `brand.jar` (if the file exists). This file is needed if the Cameo Enterprise Architecture program is used. MagicDraw does not requires this file.
3. brand_api.jar (if the file exists)*
4. Rest `*.jar` files. Order of them is not important

> Make sure that only the *.jar files coming with the current program version you are included. If you have upgraded your modeling tool from an earlier version, the *lib* directory may contain files from the earlier version. Do not include these obsolete *.jar files.

Plugins libraries

Keep in mind that some Open API classes may come from plugins (like the SysML plugin for example). In these case, make sure that jar files from a plugin directory are added into the compiled classpath.

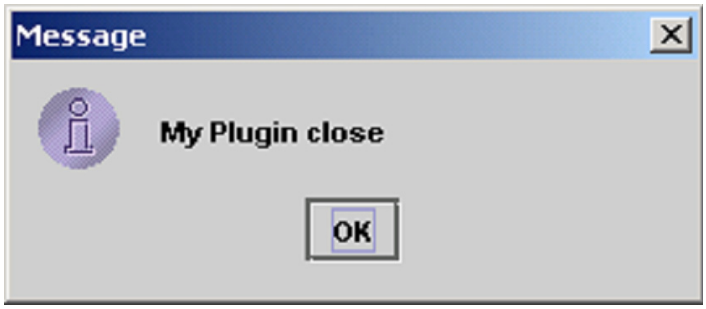
Testing plugin

You can test if your plugin sample works by checking described messages or by checking the *magicdraw .log* file.

To check messages

1. Restart MagicDraw (or other modeling tool you are using). On startup, a message should appear:

2. Then exit a program. Another message should appear:



To check the *magicdraw.log* file

- On the **Help** menu, click **About MagicDraw**. Then, in the **Environment** tab, click the **Log File** link.
- You can find the *magicdraw.log* file in a configuration file storage:
  - For Windows operating system, it is *C:\Users\<username>\AppData\Local\.magicdraw\<MagicDraw version number>*
  - For other operating systems, it is *<user.home>/.magicdraw/<MagicDraw version number>*

After successful startup, the following information should be provided at the end of the *magicdraw.log* file:

```
LOAD PLUGINS:
com.nomagic.magicdraw.plugins.PluginDescriptor@edf730[ id = my.first.plugin, name =
My First Plugin, provider = Coder, version = 1.0, class = myplugin.MyPlugin, requires
api = 1.0, runtime = \[Ljava.net.URL;@ff94b1\] (mailto:@ff94b1)

INIT PLUGINS:
[com.nomagic.magicdraw.plugins.PluginDescriptor@edf730[ id = my.first.plugin, name =|
http://Ljava.net.URL/] My First Plugin, provider = Coder, version = 1.0, class =
myplugin.MyPlugin, requires api = 1.0, runtime = [Ljava.net.URL;@ff94b1]
```

Also all plugins and their statuses are listed in the program **EnvironmentOptions** dialog, in the Plugins tab.

> Looking at the file is the best way to find problems when the plugin does not work.

Detail information

Plugin descriptor

A plugin descriptor is a file written in XML and named *plugin.xml*. Each descriptor contains properties of the one plugin. The descriptor file should contain the definitions of elements of the one plugin.

The *plugin.xml* file consists of the following elements:

- plugin(see page 2162)
- requires(see page 2162)
- api(see page 2162)
- required-plugin(see page 2162)
- runtime(see page 2162)
- library(see page 2162)

The following tables describe them in more details.

plugin

Attribute	Description
id	A plugin ID should be unique. It is used to identify a plugin by a plugin's manager internals and by requirements of other plugins. For example, *my.first.plugin.0*.
name	A plugin name. There is no strict rules applied to this attribute. For example, *Example plugin*.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Attribute	Description
version	A plugin version. The version can be used to check other plugins dependencies, if internalVersion is not defined.
internalVersion	A plugin internal version. It is used to check other plugins dependencies.
provider-name	A plugin provider name, which is a company or an author name. For example: *No Magic*.
class	A full qualified class name. The class must be derived from *com.nomagic.magicdraw.plugins*. A plugin and stored in plugin runtime library. This class will be loaded and initialized by the plugin's manager. For example, *myplugin.MyPlugin*.
ownClassloader	Optional; the default value is *false*. Indicates, if to use an own plugin's classloader (separate from other plugins). All program plugins are loaded by the **one** classloader. If there are plugins that cannot be loaded by the same classloader (for example, because of conflicts in plugin libraries versions or other), their descriptors must be defined to use own class-loaders.
class-lookup	Optional; possible values are *LocalFirst*, *GlobalFirst*, the default value is *GlobalFirst*. Specifies the priority of a "parent" class loader, if a plugin uses ownClassloader. LocalFirst forces to load classes from the plugin class loader even if such classes exist in the modeling tool core class path. This option should be used, if, in your plugin, you want to use different versions of same libraries that are used in the core.

Nested elements

Element name	Description
requires	A modeling tool API version required by the plugin. Plugins and their versions required by the plugin.
runtime	Runtime libraries needed by a plugin.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

requires
Nested elements

Element name	Description
api	A required modeling tool API version.
required-plugin	Plugin(s) that are required to run the plugin.

api

Attribute	Description
version	A required modeling tool API version. For example, *1.0*.

required-plugin

Attribute	Description
id	The ID of a required plugin. For example, *my.first.plugin.0*.
name	A name of a required plugin.
internalVersion	An internal version of a required plugin. If it is not defined, the "version" is used to check if the required plugin is suitable.
version	A version of a required plugin. If it is not defined, any required plugin version is suitable. For example: *1.1*.
optional	The default value of this attribute is *false*. If value is *false* (or attribute is skipped), the plugin will be loaded only if the required plugin is loaded. If value is *true,* the plugin will be loaded if the required plugin is loaded or required plugin is not installed.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

runtime

Nested elements

Element name	Description
library	A runtime library for a running plugin.

library

Attribute	Description
name	A name (path) of the required library. The path is relative to the plugin descriptor directory (slash should be used to separate files in path). For example, "pluginlib.jar", "lib/otherlib.jar".

Plugin classes

com.nomagic.magicdraw.plugins.Plugin[383] is the base abstract class for any modeling tool plugin. A plugin written by a user must be extended from this class. Every plugin has its own descriptor set by a plugin's manager. A plugin has three special methods:

- a *init()*[384] method is called on a program startup. The plugin must override this method and implement its own functionality there.
- a close()[385] method is called on a program exit. The plugin must override this method and return the value *true* if the plugin is ready to exit. In other case, it should return the value *false*. If the plugin returns *false*, the program exit is canceled.
- a *isSupported()*[386] method is called before the plugin initialization. If this method returns *false*, the plugin is not initialized. *isSupported()*[387]may be used to check if the plugin can be started, for example, on this OS.

Use *com.nomagic.magicdraw.plugins.PluginUtils*[388] to get references to loaded plugins. This is useful if you have dependencies among plugins and want to get a reference to other plugin object.

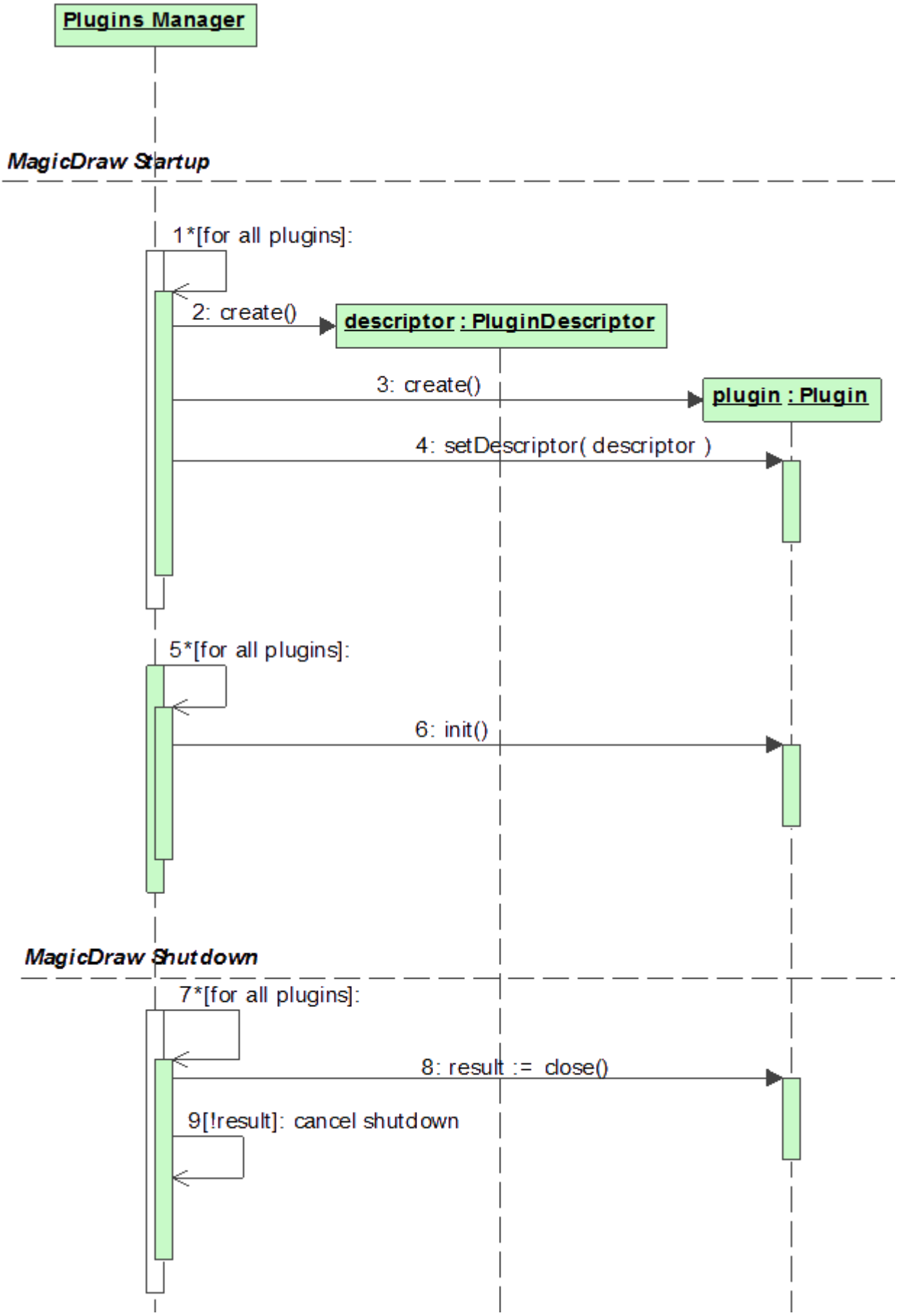
com.nomagic.magicdraw.plugins.PluginDescriptor[389] is the class that provides information loaded from the *plugin.xml* file (a plugin descriptor) to the plugin.
Use com.nomagic.magicdraw.plugins.Plugin.getDescriptor()[390] *to get the descriptor.*

More information is available in javadoc[391].

---

383 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/plugins/Plugin.html
384 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/plugins/Plugin.html#init--
385 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/plugins/Plugin.html#close--
386 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/plugins/Plugin.html#isSupported--
387 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/plugins/Plugin.html#isSupported--
388 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/plugins/PluginUtils.html
389 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/plugins/PluginDescriptor.html
390 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/plugins/Plugin.html#getDescriptor--
391 http://jdocs.nomagic.com/190beta/

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Plugin class loading

All modeling tool plugins (classes and runtime libraries) are loaded by the **one** classloader. If there are plugins that cannot be loaded by the same classloader (for example, conflicts appear in versions of plugin libraries or other), their descriptors should be defined to use own classloaders. In this case, the plugin classes are loaded by a separate classloader.

An optional property "class-lookup" controls how classes are loaded, if the plugin has its own classloader. If the value of this property is *LocalFirst*, the class is loaded from the plugin classpath even if such class is already loaded in the global core classloader of your modeling tool. This property is important if you want to use different versions of the same classes (libraries) used in the program core.

Plugins directories

Plugins are loaded from two locations on the program startup.

Installation directory

Plugins are loaded from the program installation directory (a global plugin directory) *<modeling tool installation directory>\plugins*.

This directory can be changed using a *md.plugins.dir* java system property and specifying paths separated with a semicolon (;) symbol.

To store modeling tool configuration files in a custom location

1. Open the *<modeling tool name>.properties* file, which is located in *<modeling tool installation directory>\bin*.
2. In the JAVA_ARGS line, add the following property:
   `-Dmd.plugins.dir="absolute path to plugins directory1;absolute path to plugins directory2"`

   Sample property value
   -Dmd.plugins.dir="C:\Program Files\MagicDraw\plugins;D:\MyPlugins"

Configuration files location

Additionally, plugins are loaded from a configuration files location (*<Configuration files location>\plugins*). This allows to have global and user plugins.

See “MagicDraw Configuration Files Location” in the MagicDraw UserManual.pdf.

Another issue on Unix systems is related to user permissions to write. If a modeling tool is installed in a root, user is not allowed to write in a global plugin directory if a user has not such permissions.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Resource dependent plugin

Starting with MagicDraw version 16.6, there is a new functionality to require to load plugins for a particular project. This feature was created to avoid an incorrect data load because of missing plugins. Every plugin can provide a name and version of a plugin to be required for the correct project load. To become a resource dependent plugin, your plugin class must implement the com.nomagic.magicdraw.plugins.ResourceDependentPlugin[392] interface.

ResourceDependentPlugin[393] has three special methods:

- a *isPluginRequired*[394]*(*[395]Project[396]*)* method is called on saving a project. The plugin must return *true,* if a given project uses resources from the plugin.
- a *getPluginName()*[397] method should return a plugin name.
- a *getPluginVersion()*[398] method should return a version of the plugin.

The following example illustrates an implementation of ResourceDependentPlugin[399]:

392 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/plugins/ResourceDependentPlugin.html
393 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/plugins/ResourceDependentPlugin.html
394 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/plugins/ResourceDependentPlugin.html#isPluginRequired-com.nomagic.magicdraw.core.Project-
395 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/plugins/ResourceDependentPlugin.html#isPluginRequired-com.nomagic.magicdraw.core.Project-
396 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Project.html
397 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/plugins/ResourceDependentPlugin.html#getPluginName--
398 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/plugins/ResourceDependentPlugin.html#getPluginVersion--
399 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/plugins/ResourceDependentPlugin.html

```
package myplugin;
import ...
public class MyPlugin extends Plugin implements ResourceDependentPlugin
{
   @Override
   public void init()
   {...}
   @Override
   public boolean close()
   {...}
   @Override
   public boolean isSupported()
   {...}

   @Override
   public boolean isPluginRequired(Project project)
   {
      return ProjectUtilities.findAttachedProjectByName(project,
"my_profile_filename") != null;
   }
   @Override
   public String getPluginName()
   {
      return this.getDescriptor().getName();
   }
   @Override
   public String getPluginVersion()
   {
      return this.getDescriptor().getVersion();
   }
}
```

This plugin is required for a project, if the project contains a used project "*my_profile_filename*". The plugin name and version will be saved into the project's XMI file.

Creating test cases

MagicDraw and other modeling tools developed by No Magic Inc. provide a JUnit (www.junit.org)[400]-based test frameworks. The main purpose of the test framework is to simplify the automatic unit and integration tests development for the program and its plugins. The test framework can be used by developers for testing their own plugins or standard essential program features.

On this page

- JUnit 3 test cases(see page 2169)
- JUnit 4 test cases(see page 2169)
- JUnit 5 test cases(see page 2169)

400 http://www.junit.org/

JUnit 3 test cases

The test framework consists of an abstract JUnit test case implementation (com.nomagic.magicdraw.tests.MagicDrawTestCase[401]) and a number of tools (com.nomagic.magicdraw.tests.common.TestEnvironment[402], com.nomagic.magicdraw.tests.common.comparators.ProjectsComparator[403]) that can be used for the following purposes:

- Starting the application
- Managing projects
- Checking program for memory leaks

JUnit 4 test cases

The test framework also has test class runner implementations (com.nomagic.magicdraw.tests.MagicDrawTestRunner[404], com.nomagic.magicdraw.tests.ParametrizedMagicDrawTestRunner[405]) that:

- Start the application
- Check program for memory leaks.

JUnit 5 test cases

Extend your test class with *com.nomagic.magicdraw.tests.MagicDrawApplication* [406]to start the application before running the test methods.

> You can find the examples of the code in *<modeling tool installation directory>\openapi\examples\testframework*

Related pages

Creating JUnit test case

Let's use MagicDraw as an example to describe the creation of JUnit test cases, as other modeling tools developed by No Magic Inc. name them.

401 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/MagicDrawTestCase.html
402 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/common/TestEnvironment.html
403 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/common/comparators/ProjectsComparator.html
404 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/MagicDrawTestRunner.html
405 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/ParametrizedMagicDrawTestRunner.html
406 http://jdocs.nomagic.com/190_sp3/com/nomagic/magicdraw/tests/MagicDrawApplication.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

JUnit 3 test case

The JUnit test case should be created by extending the abstract *com.nomagic.magicdraw.tests.MagicDrawTestCase*[407] class from Open API (see the following *MyTest* sample). The *MagicDrawTestCase*[408] class starts MagicDraw automatically for each test case method defined in MagicDrawTestCase[409] and performs the memory leak test after each completed test case.

MagicDrawTestCase[410] provides default and specific constructors for creating a test case instance with a specific custom name for the specific test case method. Test cases with specific names might be helpful when several instances of the same test case are created and analyzed.

The *MagicDrawTestCase*[411] initialization and tear down should be implemented in overridden setUpTest()[412] and *tearDownTest()*[413] methods. The standard JUnit method *suite* might be used for preparing the tests suite with several instances of test cases which may use different test data.

407 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/MagicDrawTestCase.html
408 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/MagicDrawTestCase.html
409 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/MagicDrawTestCase.html
410 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/MagicDrawTestCase.html
411 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/MagicDrawTestCase.html
412 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/MagicDrawTestCase.html#setUpTest--
413 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/MagicDrawTestCase.html#tearDownTest--

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

```
    import com.nomagic.magicdraw.tests.MagicDrawTestCase;
    public class MyTest extends MagicDrawTestCase
    {
        public MyTest(String testMethodToRun, String testName)
        {
            super(testMethodToRun, testName);
        }

        @Override
        protected void setUpTest() throws Exception
        {
            super.setUpTest();
            //do setup here
        }

        @Override
        protected void tearDownTest() throws Exception
        {
            super.tearDownTest();
            //do tear down here
        }

        public void testSomething()
        {
              //implement the unit test here
        }

        public static Test suite() throws Exception
        {
            //you may create a test suite with several instances of the test.
            TestSuite suite = new TestSuite();
            suite.addTest(new MyTest("testSomething", "MagicDraw Test Sample"));
            suite.addTest(new MyTest("testSomething", "Another MagicDraw Test
Sample"));
            return suite;
        }
    }
```

MagicDrawTestCase[414]also provides methods for opening, saving, and closing MagicDraw projects. It also performs the memory leak test after the test case has been completed and the project has been closed. The memory leak test for the whole test case can be disabled using the setMemoryTestReady(boolean)[415] method, while the *setSkipMemoryTest(boolean)*[416] method can be used to disable the memory leak test on closing the MagicDraw project.

The list of required MagicDraw plugins for the test case can be configured by overriding the *getRequiredPlugins()*[417] method of *MagicDrawTestCase*[418]. The overridden method implementation should return the list of required plugins IDs. Textual information about required plugins and their

414 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/MagicDrawTestCase.html
415 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/MagicDrawTestCase.html#setMemoryTestReady-boolean-
416 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/MagicDrawTestCase.html#setSkipMemoryTest-boolean-
417 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/MagicDrawTestCase.html#getRequiredPlugins--
418 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/MagicDrawTestCase.html

loading status can be obtained using getRequiredPluginsDescription()[419] and *isRequiredPluginsLoaded()*[420] methods.

Textual information about the test process can be logged using the Log4j logger. The test case logger for the TEST category can be accessed using the *getLogger()*[421] method of MagicDrawTestCase[422]. More information about the Log4j logger configuration and usage can be found at http://logging.apache.org/log4j/1.2/manual.html.

JUnit 4 test case

The JUnit4 test class should be annotated to use *com.nomagic.magicdraw.tests.MagicDrawTestRunner*[423] from Open API (see a code sample bellow). The *MagicDrawTestRunner*[424]class starts MagicDraw automatically and performs the memory leak test after each completed test case.

```
import com.nomagic.magicdraw.tests.MagicDrawTestRunner;
import org.junit.*;
import org.junit.runner.RunWith;

@RunWith(MagicDrawTestRunner.class)
public class MyTest
{
   @Before
   public void beforeTestBegins()
   {
      //do test case setup here, e.g. load project
   }

   @Test
   public void testSomething()
   {
      //implement the unit test case here
   }

   @After
   public void afterTestDone()
   {
      //do tear down here, e.g. close project
   }
}
```

419 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/MagicDrawTestCase.html#getRequiredPluginsDescription--
420 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/MagicDrawTestCase.html#isRequiredPluginsLoaded--
421 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/MagicDrawTestCase.html#getLogger--
422 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/MagicDrawTestCase.html
423 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/MagicDrawTestRunner.html
424 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/MagicDrawTestRunner.html

JUnit 5 test case

The JUnit5 test class should be annotated to extend *com.nomagic.magicdraw.tests.MagicDrawApplication*[425]f rom Open API (see a code sample bellow). The MagicDrawApplication[426] extension starts MagicDraw automatically before running the test cases.

```
import com.nomagic.magicdraw.tests.MagicDrawApplication;
import org.junit.jupiter.api.*;
import org.junit.jupiter.api.extension.ExtendWith;

@ExtendWith(MagicDrawApplication.class)
public class MyTest
{
   @BeforeEach
   void beforeTestBegins()
   {
      //do test case setup here, e.g. load project
   }

   @Test
   void testSomething()
   {
      //implement the unit test case here
   }

   @AfterEach
   void afterTestDone()
   {
      //do tear down here, e.g. close project
   }
}
```

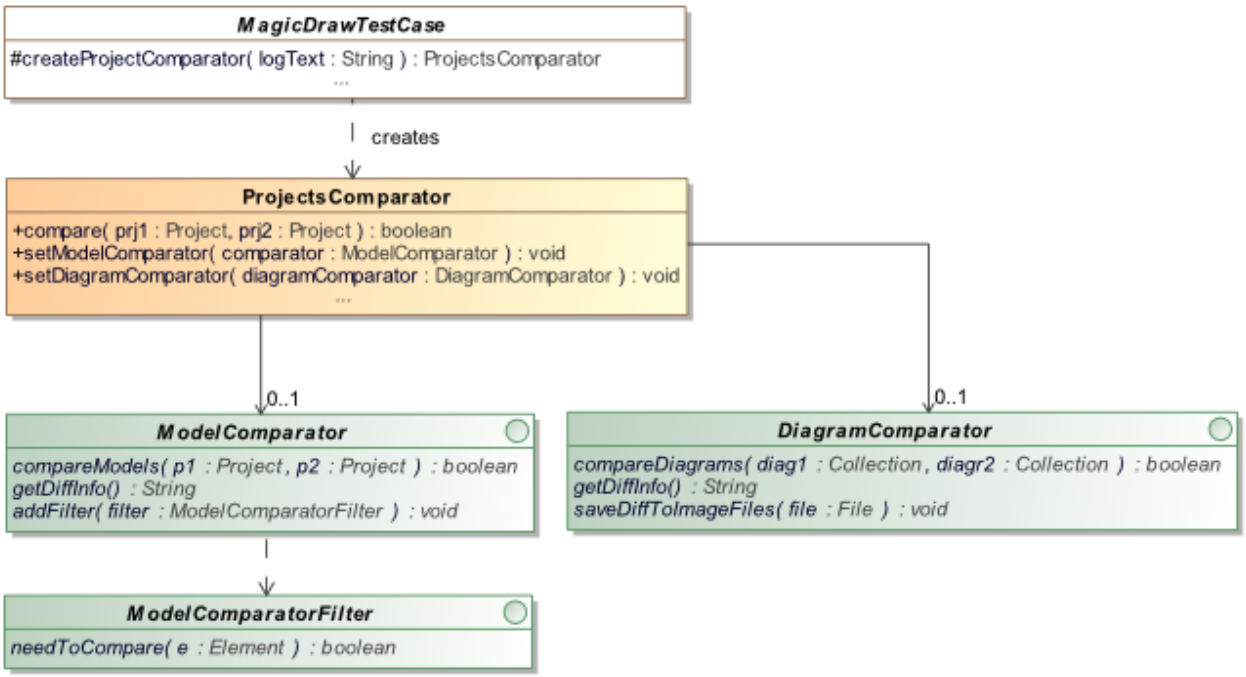
On this page

- JUnit 3 test case(see page 2171)
- JUnit 4 test case(see page 2173)
- JUnit 5 test case(see page 2174)

425 http://jdocs.nomagic.com/190_sp3/com/nomagic/magicdraw/tests/MagicDrawApplication.html
426 http://jdocs.nomagic.com/190_sp3/com/nomagic/magicdraw/tests/MagicDrawApplication.html

Comparing projects for testing purposes

To compare two projects for testing purpose, the test framework provides the *com.nomagic.magicdraw.tests.common.comparators.ProjectsComparator*[427] class with the compare([428]Project[429],[430]Project[431])[432] method for comparing two projects (see the following figure). The default implementation of *ProjectComparator*[433]can be created using the *createProjectComparator(java.lang.String)*[434] method of the *MagicDrawTestCase* class. Projects can be compared by checking their model elements and element symbols in diagrams for equality.



The model elements comparison can be configured using the *com.nomagic.magicdraw.tests.common.comparators.ModelComparator*[435] interface. Developers may exclude some model elements from the comparison by providing the custom *com.nomagic.magicdraw.tests.common.comparators.ModelComparatorFilter*[436] interface implementation.

---

427 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/common/comparators/ProjectsComparator.html

428 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/common/comparators/ProjectsComparator.html#compare-com.nomagic.magicdraw.core.Project-com.nomagic.magicdraw.core.Project-

429 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Project.html

430 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/common/comparators/ProjectsComparator.html#compare-com.nomagic.magicdraw.core.Project-com.nomagic.magicdraw.core.Project-

431 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Project.html

432 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/common/comparators/ProjectsComparator.html#compare-com.nomagic.magicdraw.core.Project-com.nomagic.magicdraw.core.Project-

433 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/common/comparators/ProjectsComparator.html

434 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/MagicDrawTestCase.html#createProjectComparator-java.lang.String-

435 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/common/comparators/ModelComparator.html

Please note that the default implementation of *ProjectComparator*[437]does not compare the diagram information table, root model comment, and elements from used projects.

The default implementation of *com.nomagic.magicdraw.tests.common.comparators.DiagramComparator*[438] compares diagrams for changes by analyzing a location and a size of the symbols presented in diagrams. The graphical differences of compared diagrams can be saved in the graphical PNG file using the saveDiffToImageFiles(java.io.File)[439] method of the DiagramComparator[440]interface.

Developers may create their own comparators for the custom diagrams and model elements comparison by implementing *ModelComparator*[441]and *DiagramComparator*[442]interfaces. The custom *Model* and *Diagram* comparators implementation can be set to *ProjectComparator*[443]using appropriate setter methods.

Working with test resources

Test cases may use external resources such as configuration files and projects for testing purposes. It is recommended to keep external test resources apart from test case implementation class files. The test framework provides methods for retrieving test resources from the one specific resource directory.

Resource directory can be accessed by calling the *com.nomagic.magicdraw.tests.common.TestEnvironment.getResourceDir()*[444]method. This method obtains the directory specified by the *tests.resources* system property. For information about specifying the *tests.resources* system property see Configure test environment(see page 2178). The test framework also provides methods for collecting projects from the resource directory. Developers may use *TestEnvironment.getProjects(java.io.File)*[445] for collecting projects recursively from a specific directory. Unnecessary projects can be filtered out by specifying their names in *skip.txt* files that are placed in the same directory as project files.

Test cases may produce some resources as a test output. The output directory for a specific test case can be created using TestEnvironment.getOutputDir(java.lang.Class)[446]. The class here specifies a class of the test case which output directory should be accessed.

---

437 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/common/comparators/ProjectsComparator.html
438 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/common/comparators/DiagramComparator.html
439 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/common/comparators/DiagramComparator.html#saveDiffToImageFiles-java.io.File-
440 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/common/comparators/DiagramComparator.html
441 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/common/comparators/ModelComparator.html
442 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/common/comparators/DiagramComparator.html
443 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/common/comparators/ProjectsComparator.html
444 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/common/TestEnvironment.html#getResourceDir--
445 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/common/TestEnvironment.html#getProjects-java.io.File-
446 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/common/TestEnvironment.html#getOutputDir-java.lang.Class-

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

There are also methods for working with output files. An output file for a specific project can be simply created by changing the project file extension (TestEnvironment.getFileWithExtension(java.io.File, java.lang.String)[447]) or adding a special prefix to its name (*TestEnvironment.createFileNameWithPrefix(java.io.File, java.lang.String)*[448]).

Configure test environment

Test cases can be run as a regular JUnit test case using the JUnit *TestRunner* class or configuration tools such as Ant or Maven. However, some test case specific system properties should be set.

The program installation directory should be also specified using the *install.root* system property which can be passed to JVM as a runtime argument *-Dinstall.root=path_to_modeling_tool_installation_directory*.

It is possible to setup the program installation to force using the floating license server. The following properties should be passed to JVM as runtime arguments:

```
-DFL_FORCE_USAGE=true
-DFL_SERVER_ADDRESS=flexnet_license_server_address
-DFL_SERVER_PORT=license_server_port
-DFL_EDITION=magicdraw_edition
```

Test cases may use external resources such as configuration files and projects during tests. It is recommended to keep external test resources apart from test case implementation class files. The test framework provides methods for retrieving test resources from the one specific resource directory. The resource directory for test cases can be configured by specifying the *tests.resources* system property of JVM. The tests resource property can be passed to JVM as a runtime argument

```
-Dtests.resources=path_to_resources_directory
```

The Log4j logger used in test cases can be configured by specifying the *test.properties* file which should be placed in *<modeling tool installation directory>\data*.

The program is started in a "silent" mode when running tests - the user interference requiring dialogs are not shown, instead the warning "*WARN GENERAL - TRYING TO DISPLAY "<dialog title>" DIALOG IN SILENT MODE*" is logged. The stack trace is logged too if a special system property *-DprintSilentDialogStackTrace=true* is set.

Related Pages

- Developing plugins using IDE(see page 2177)

---

[447] http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/common/TestEnvironment.html#getFileWithExtension-java.io.File-java.lang.String-

[448] http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/tests/common/TestEnvironment.html#createFileNameWithPrefix-java.io.File-java.lang.String-

Developing plugins using IDE

Plugins extending your modeling tool can be effectively developed using popular IDEs (Integrated Development Environment) such as Eclipse, IntelliJ, and others. Developers may use theirs favorite IDE for an agile modeling tool plugin coding, building, testing, and debugging. This chapter describes the plugin development in Eclipse (www.eclipse.org[449]) and IntelliJ IDEA (www.jetbrains.com[450]) IDEs.

Related pages

Development in Eclipse

The preconfigured Eclipse projects for two sample plugins and a batch mode(see page 2375) (command-line) tool are provided with a program installation. These projects can be found in *<modeling tool installation directory>\openapi\ide\eclipse.zip*.

Let's use MagicDraw as an example to describe the following procedures.

> Make sure MagicDraw installation directory does not contain spaces.

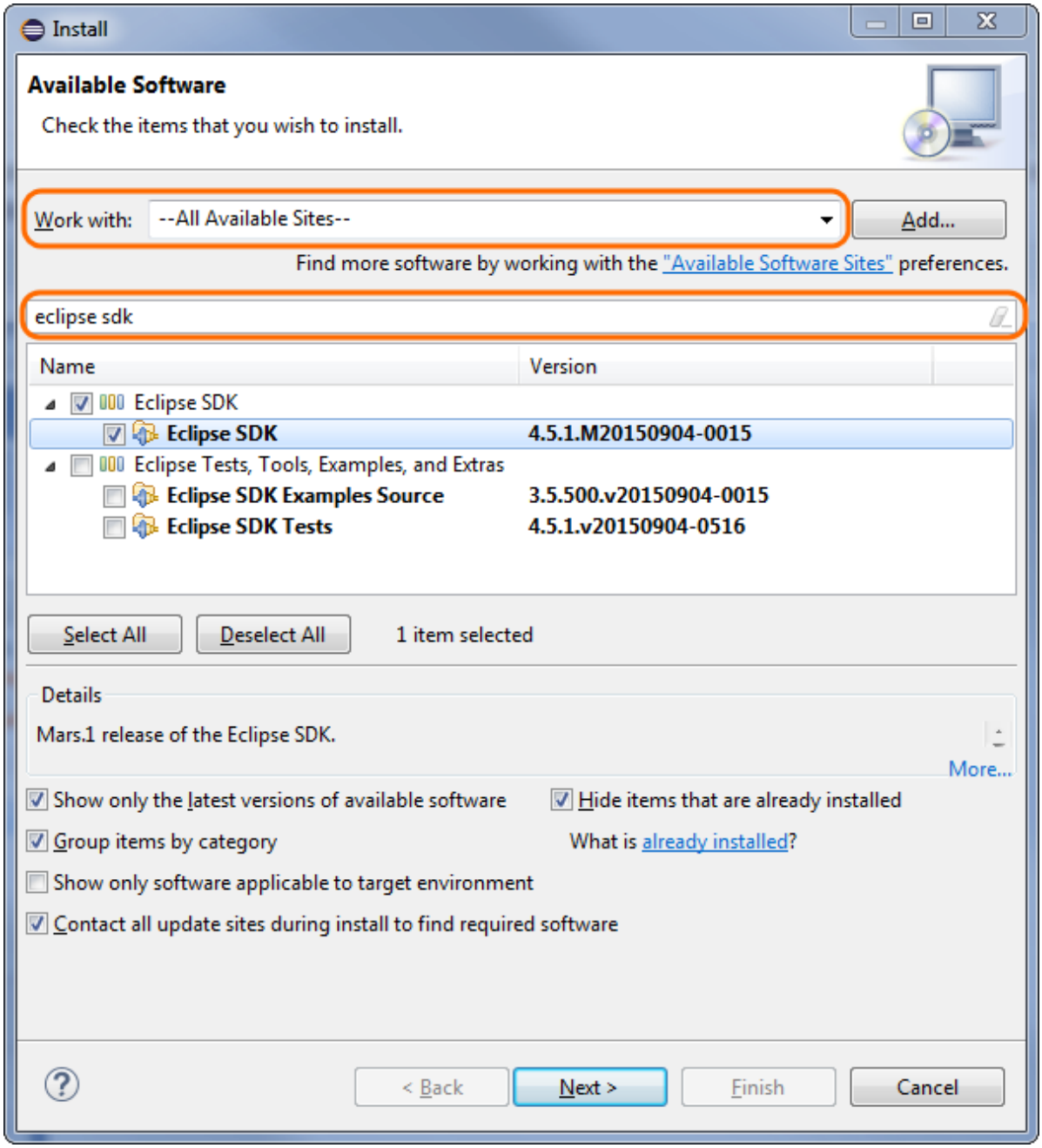
To setup the Eclipse environment for the modeling tool development

1. Start Eclipse IDE. To install Eclipse SDK, from the main menu, select **Help** > **Install New Software**. The **Install** dialog opens. In the **Work with** list, select **All Available Sites**, and, in the **filter** box, type *eclipse sdk* and press Enter. All available Eclipse versions are listed. Click to select **Eclipse SDK** and go **Next**. Follow the installation wizard steps to proceed software installation process.

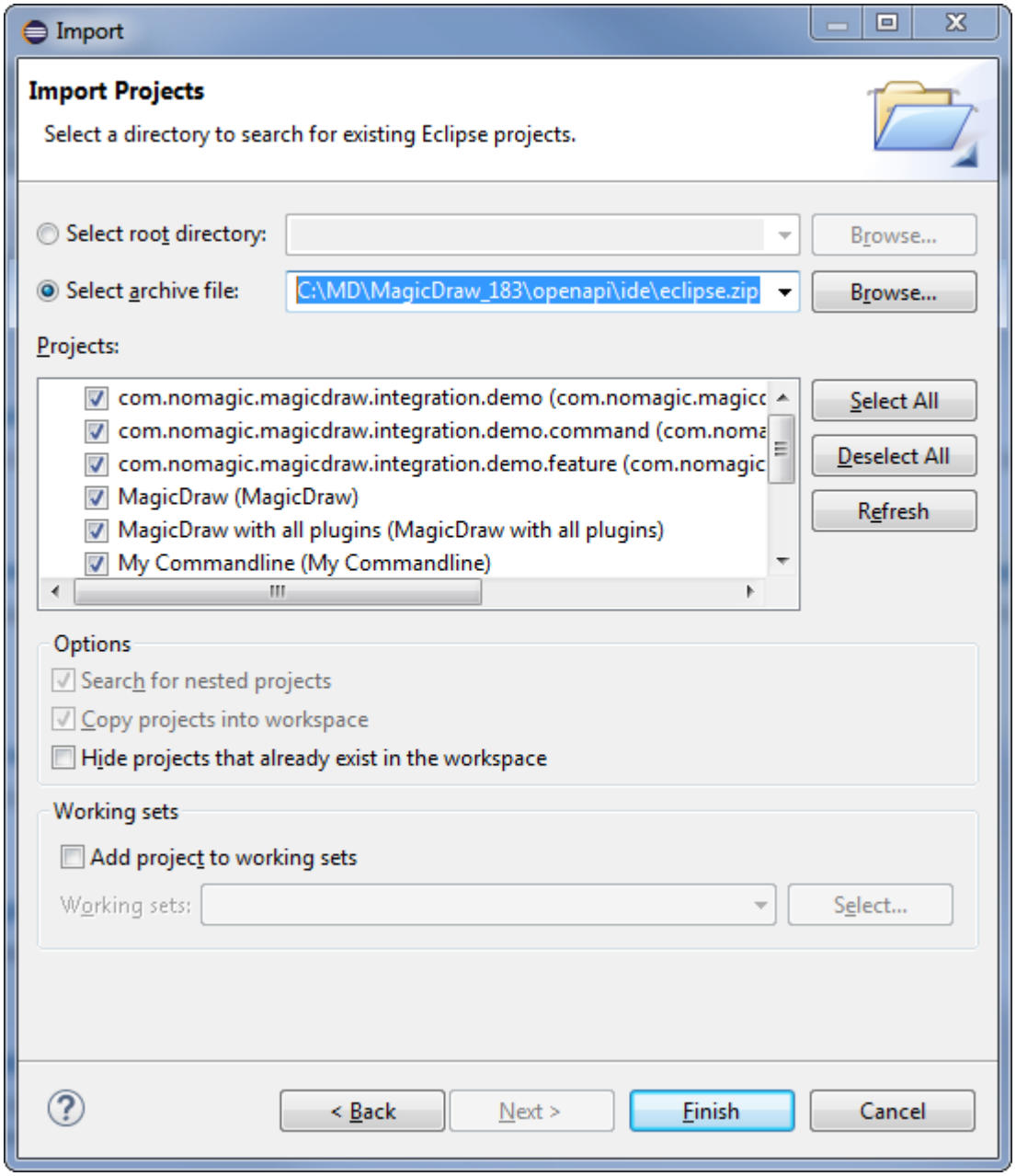
   > In case you have Eclipse IDE for Eclipse Committers, Eclipse SDK is already available in your IDE and you can skip this step.

449 http://www.eclipse.org/
450 http://www.jetbrains.com/

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



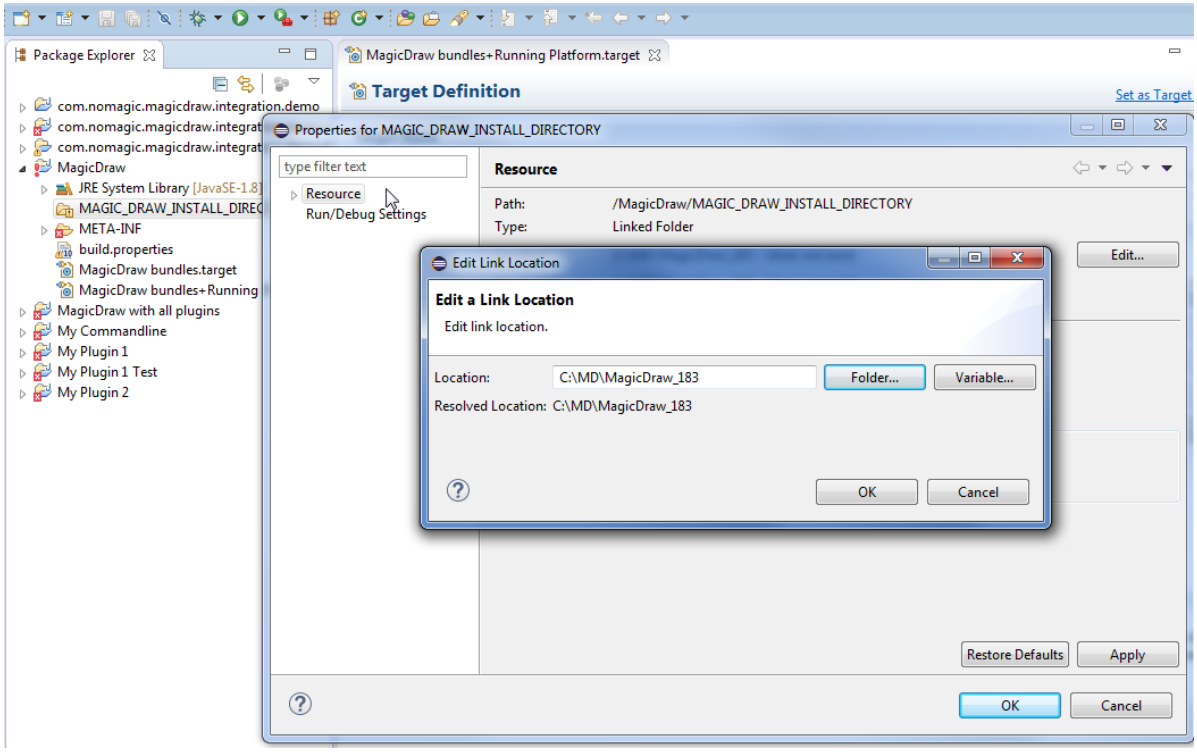
2. After the installation completes, import java projects from *<modeling tool installation directory>\openapi\ide\eclipse.zip* into your Eclipse Workspace. On the main menu, click **File** > **Import**. The **Import** dialog opens. In the **Select an import source** list, select **General** > **Existing Projects into Workspace** and click **Next**. Copy the java project location to the **Select archive file** box or browse to the java projects and click **Browse**. The project list appears in the **Projects** box. Select projects you need and click **Finish.**



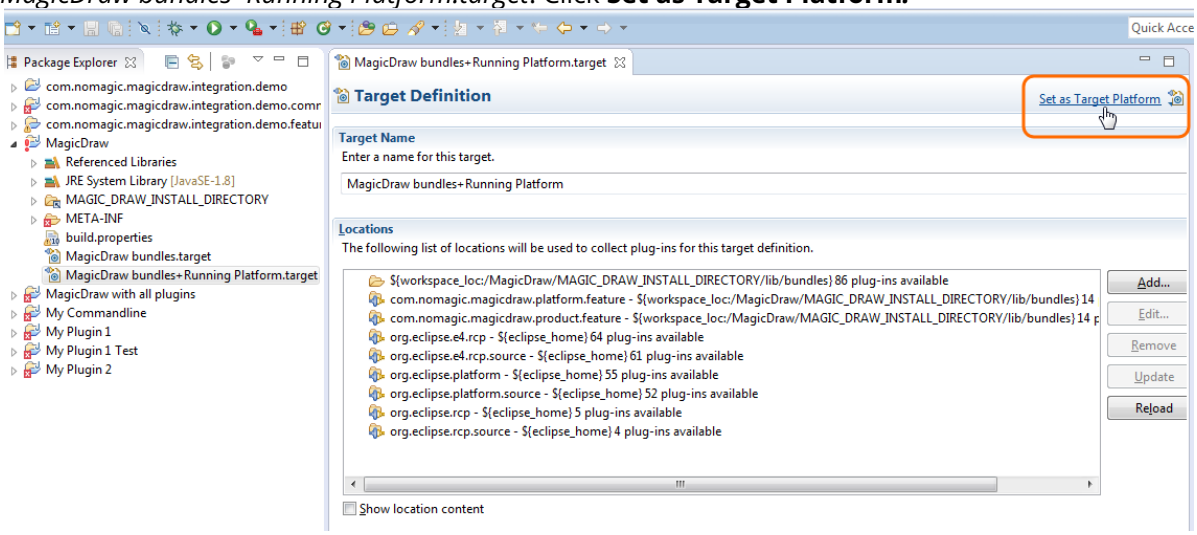
3. Open the **Package Explorer** view, expand the MagicDraw project and edit the MAGIC_DRAW_INSTALL_DIRECTORY link:
   3.1. Right-click the link and select **Properties**.
   3.2. On the left side of the opened dialog, select **Resource** and click the **Edit** button on the right side of the dialog. The **Edit Link Location** dialog opens.
   3.3. Select the location and click **OK**.

> Be sure the link points to your MagicDraw (or other according to a modeling tool you are using) installation directory (see the following figure). **Installation directory must not contain spaces.**

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



4. In the **Package Explorer**, expand the MagicDraw project and open target definition file *MagicDraw bundles+Running Platform.target*. Click **Set as Target Platform.**



5. Locate two exemplary launch configurations in the **Package Explorer**.
   5.1. *MagicDraw with all plugins.launch* is in the project *MagicDraw with all plugins*. It launches MagicDraw (or other modeling tool) with plugins available in the IDE workspace namely *My Plugin 1* and *My Plugin 2*.

> ⚠ The launch configuration is designed to load plugins from the MagicDraw (or other according to a modeling tool you are using) installation directory (see step #3(see page 2180)) and two plugins from the Eclipse Workspace. Thus, if the *md.plugins.dir* java system property is not defined (see Plugins directories(see page 2167)), developing plugins are not loaded.

5.2. *My Commandline.launch* is in the project *My Commandline*. It launches batch mode tool *My Commandline*.

Eclipse Workspace is ready for the source code development and running/debugging.

> ⚠ The libraries (jar files) of the plugin must be added to the development class path throughout the plugin dependency hierarchy if the developing code depends on that plugin.
> For example, if the code depends on plugin A; plugin A depends on plugins B and C; plugin B depends on plugin D, the libraries of all plugins (A, B, C, and D) must be added to the class path.

> ⚠ When you launch your own plugin, you need to add all jar files that are required by your plugin from appropriate plugins. The MagicDraw jar files can be found in MAGIC_DRAW_INSTALL_DIRECTORY/lib and its sub directories, whereas plugins' jar files can be found in MAGIC_DRAW_INSTALL_DIRECTORY/plugins and its sub directories.

> ⓘ Even if the plugin descriptor file contains information about the runtime plugin *.jar* file, it is not necessary to build and deploy this *.jar* file to a plugin directory while the plugin is developed under Eclipse.

Development in IntelliJ IDEA

The preconfigured IntelliJ IDEA project with modules for two sample plugins and a batch mode(see page 2375) (command-line) tool is provided with a program installation directory. The project can be found in *<modeling tool installation directory>\openapi\ide\intellij.zip.*

Let's use MagicDraw as an example to describe the following procedures.

To setup the IntelliJ IDEA environment for the MagicDraw (or other program according to a modeling tool you are using) development

---

1. In IntelliJ IDEA, add a path variable named MAGIC_DRAW_INSTALL_DIRECTORY and pointing to your version of MagicDraw (CSM, CEA that you have the licence for, as it starts like a normal version and will require an active licence) installation directory.
2. Extract *<MagicDraw installation directory>\openapi\ide\intellij.zip* to a chosen folder.
3. Open the extracted IntelliJ IDEA project *MagicDraw development.ipr* from that folder.

> ⚠ The launch configuration is designed to load plugins from the program installation directory (see step #1) and two plugins from the IDEA project. Thus, if the *md.plugins.dir* java system property is not defined (see Plugins directories(see page 2167)), developing plugins are not loaded.

After projects are imported and the launch configurations are prepared, the IDEA project are ready for the source code development and running/debugging.

The libraries (jar files) of the plugin must be added to the development class path throughout the plugin dependency hierarchy if the developing code depends on that plugin.
For example, if the code depends on plugin A; plugin A depends on plugins B and C; plugin B depends on plugin D, the libraries of all plugins (A, B, C, and D) must be added to the class path.

When you launch your own plugin, you need to add all jar files that are required by your plugin from appropriate plugins. The MagicDraw jar files can be found in MAGIC_DRAW_INSTALL_DIRECTORY/lib and its sub directories, whereas plugins' jar files can be found in MAGIC_DRAW_INSTALL_DIRECTORY/plugins and its sub directories.

Even if the plugin descriptor file contains information about a runtime plugin *.jar* file, it is not necessary to build and deploy this *.jar* file to a plugin directory while a plugin is developed under IntelliJ IDEA.

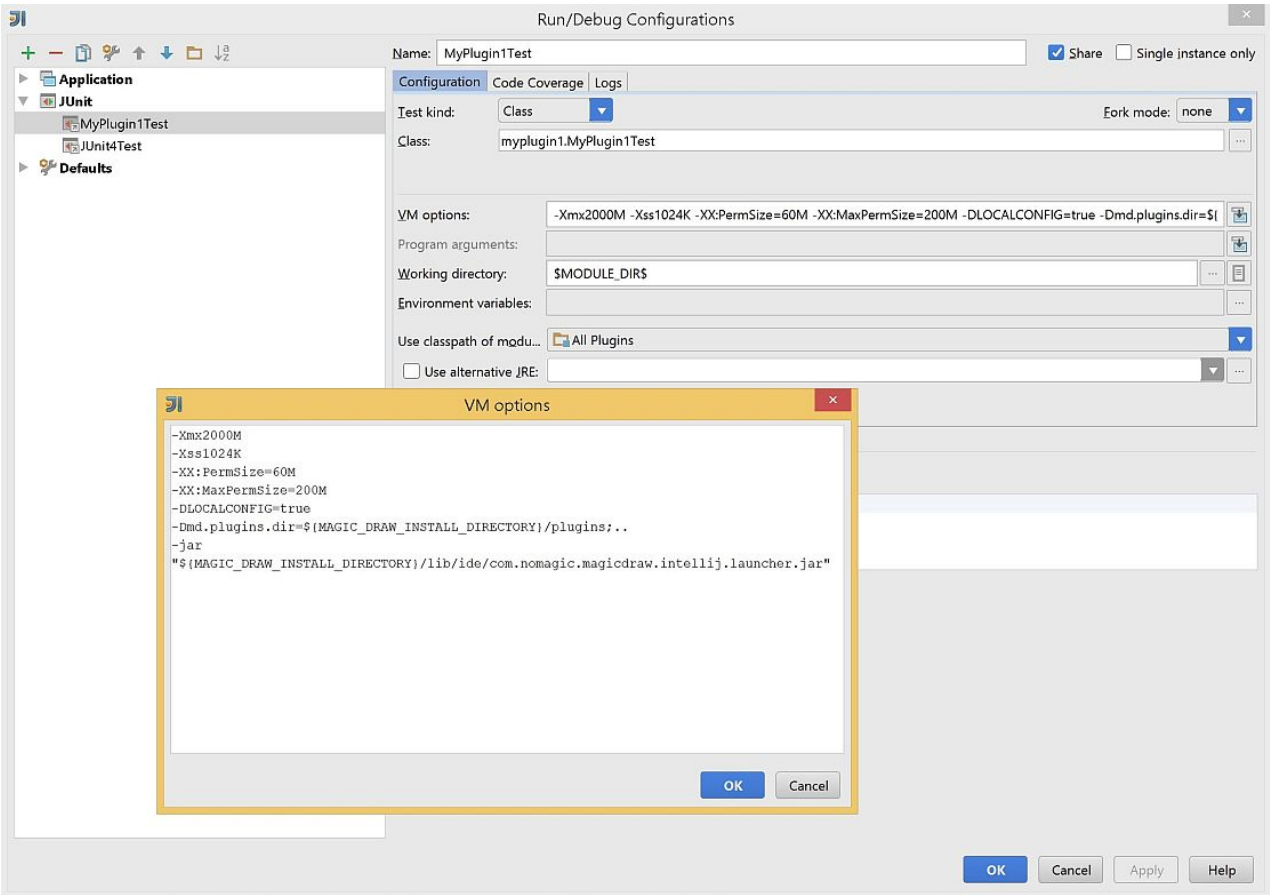
Launching tests

A custom test launching mechanism is used to launch MagicDraw tests, because of an Eclipse OSGi environment (which is not officially supported by the recent version of IntelliJ IDEA).

The test launcher classes are defined at the end of *VM options* (without prefix *-D*) and the classpath to those classes is additionally defined as CLASSPATH variable in *Environment variables* section of the run configuration. These settings should be used for any test configuration for MagicDraw on IntelliJ IDEA by default (see the picture below and the provided sample configuration).

An IntelliJ IDEA module that is used for classpath in the test configuration should include all contents of the *<modeling tool installation directory>\lib* directory recursively as it's dependencies.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The test launching mechanism is designed to use IntelliJ IDEA classpath file (autmatically generated) to support long classpaths. It is mandatory to setup Intellij IDEA to create that classpath file for provided test launchers. That can be done by adding or modifying a property *dynamic.classpath* to be *true* in IntelliJ IDEA workspace properties file *<project_name>.iws* (it usually resides beside project file *<project_name>.ipr)* or *.idea/workspace.xml* (if the directory based project structure is used).

.iws or .idea/workspace.xml

```
...
<component name="PropertiesComponent">
    <property name="recentsLimit" value="5" />
    ...
    <property name="dynamic.classpath" value="true" />
```

Distributing Resources

In this section, you will find information about how to share your created resources with other users. The easiest way to distribute data is to store resources in the one zip archive and import files to your modeling tool with the MagicDraw Resource Manager(see page 1150).

You can distribute the following types of resources:

- Custom Diagrams
- Profiles
- Templates
- Samples and Documentation
- Plugins

You can distribute the one resource type or you may distribute the whole set of resources.

Related pages

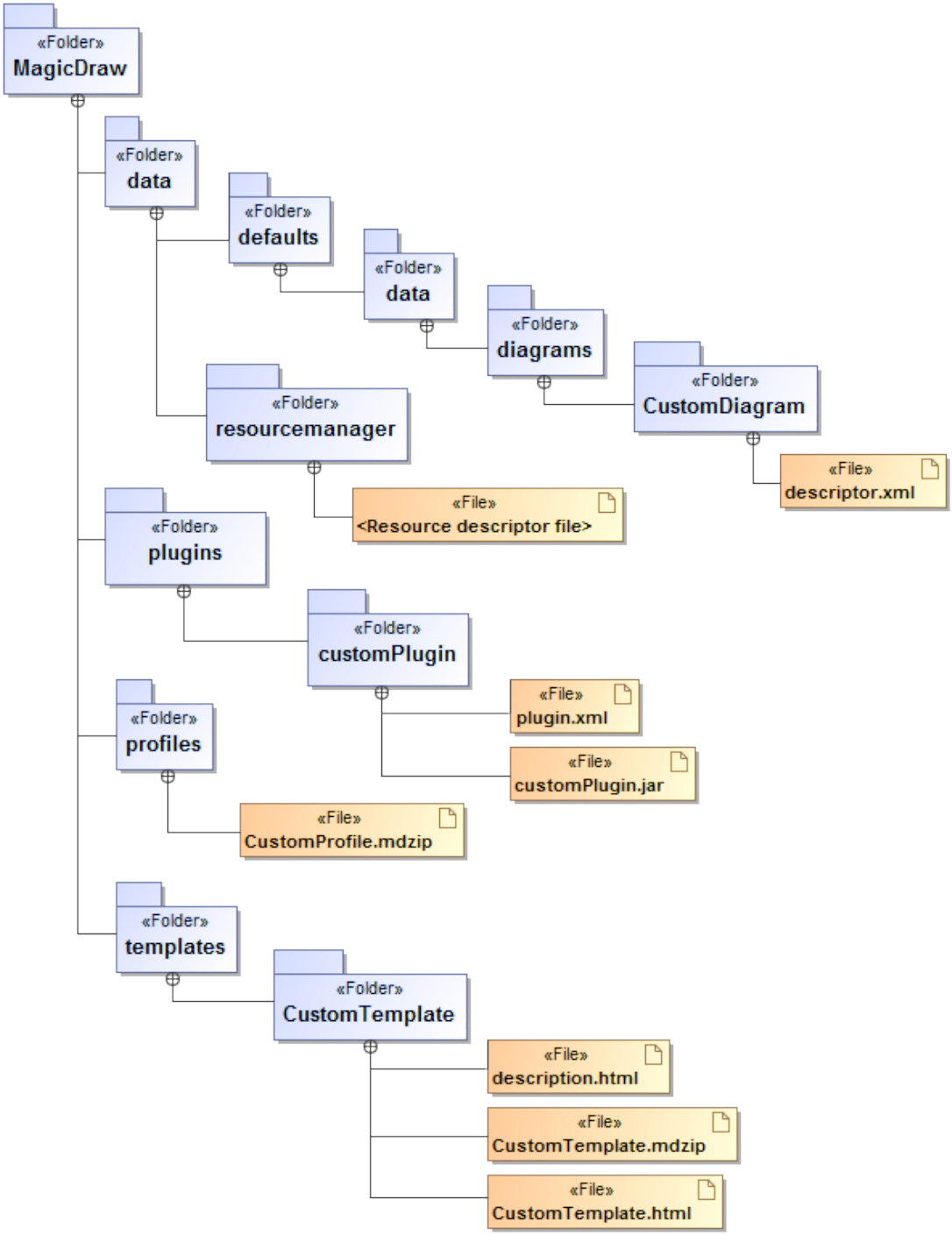
How to distribute resources

You can distribute custom resources in the following two ways:

- Using **Custom Resource Building Wizard**:
  1. Install the **Development Tools** plugin. Click **Help** > **Resource/Plugin Manager**, download the plugin and restart the program.
  2. Create required files. For information about what files are required for each type of the resource, see Creating required files and folders structure(see page 2188).

     ⚠ The structure must not contain the "data/resourcemanager" folders yet.

  3. Start Resource Builder (**Tools** > **Development Tools** > **Build Custom Resource...**), select a *MagicDraw* folder structure you have constructed in the step #2 step.
  4. Fill the properties in the provided form.
  5. Select the desired resource (a ZIP archive) output file, and click **Finish**. Resource Builder will create the "data/resourcemanager" folder containing an XML file with the properties you provided and assemble everything inside the *MagicDraw* folder into the ZIP file you specified as the output file. The resource (ZIP) file you have created can be distributed to your team members and installed using the Resource/Plugin Manager.
- Distribute manually. The manual distribution consists of the following steps:
  1. Create required files. For information about what files are required for each type of resource, see Creating required files and folders structure(see page 2188).
  2. Create a Resource descriptor file. For more information about the Resource descriptor file, see Resource manager descriptor file(see page 2192).
  3. Archive created files to a zip file. The zip file should include the required files, as well as folders that have a structure matching the structure of a modeling tool. Information about the folders structure for each type of resource and the general view of the file structure is depicted in the following figure:



4. Import your prepared data to your modeling tool through the Resource Manager.

   For more information about the MagicDraw Resource Manager, see *MagicDraw UserManual.pdf*.

Resource Manager supports zip archives only!

Related pages

- Creating required files and folders structure(see page 2188)
- Resource manager descriptor file(see page 2192)

Creating required files and folders structure

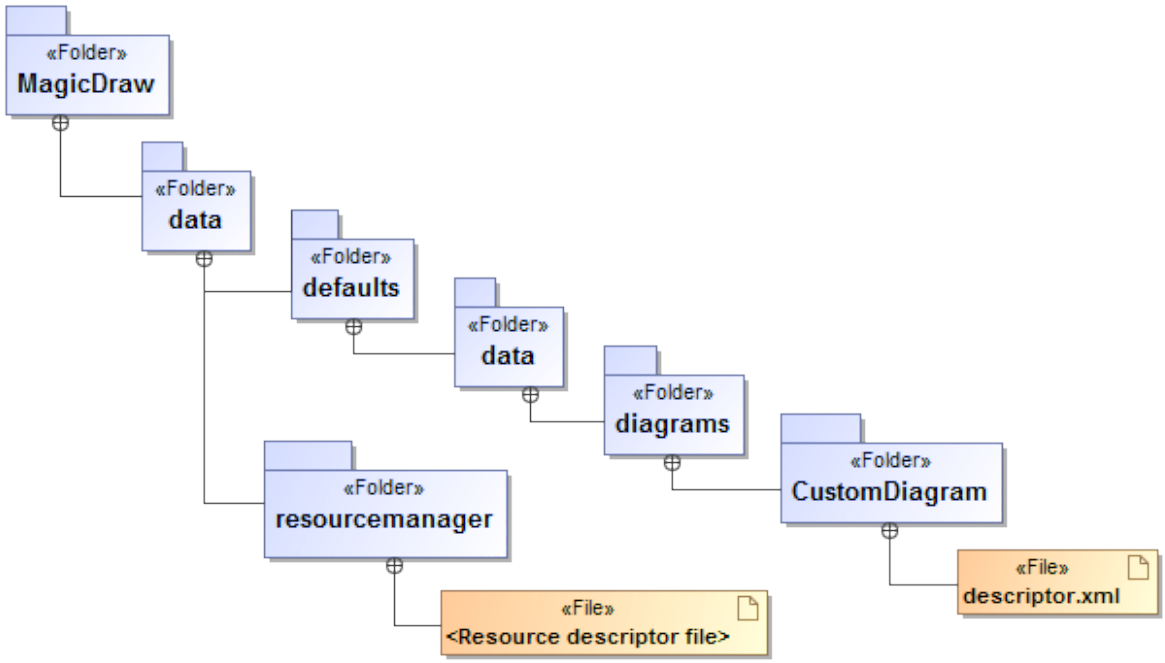
To distribute resources, you must create the required files and folders for a particular resource type. Some of the resource file names should match the standard names.

For each resource file, there should be a created folders structure, which should match the folders structure of the program installation folder.

To distribute resources, you must create a resource manager descriptor file(see page 2187).

Distributing Custom Diagrams

A required file for the custom diagram distribution is *descriptor.xml* - a Custom Diagram descriptor provided by MagicDraw Resource Manager. The file and folder structure is depicted in the following figure:



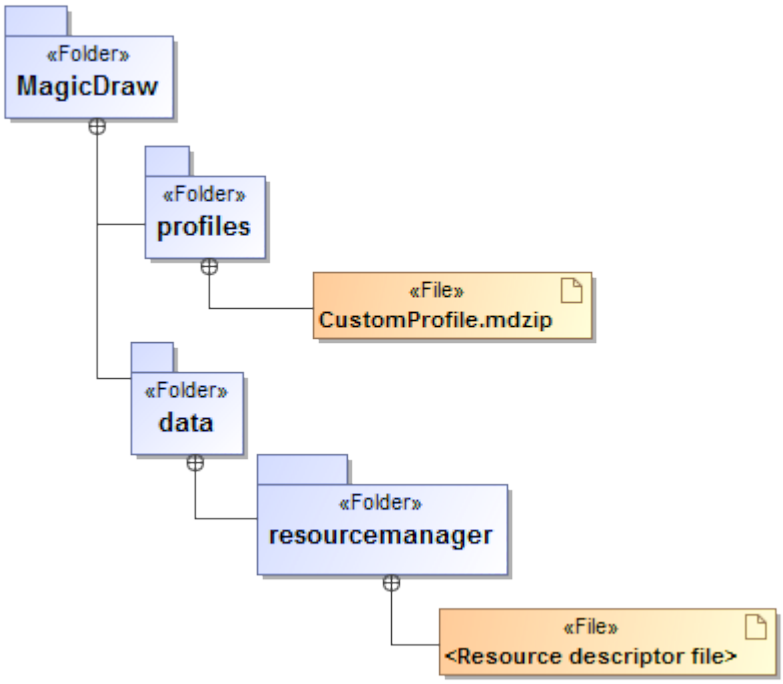
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

The custom diagram type (in diagram descriptor.xml file) should match the folder name it is placed in.
For example, the custom diagram with type *My Custom Structure Diagram* should be placed in a folder named **My Custom Structure Diagram**.

For more information about creating new diagram types, see the section “Diagram Types” in MagicDraw UserGuide.pdf or see the *UML Profiling and DSL UserGuide.pdf* for the custom diagrams creation.

Distributing Profiles

You can distribute your profile as the following structure:



For more information about working with Profiles see *MagicDraw UserManual.pdf* and *UML Profiling and DSL UserGuide.pdf*.

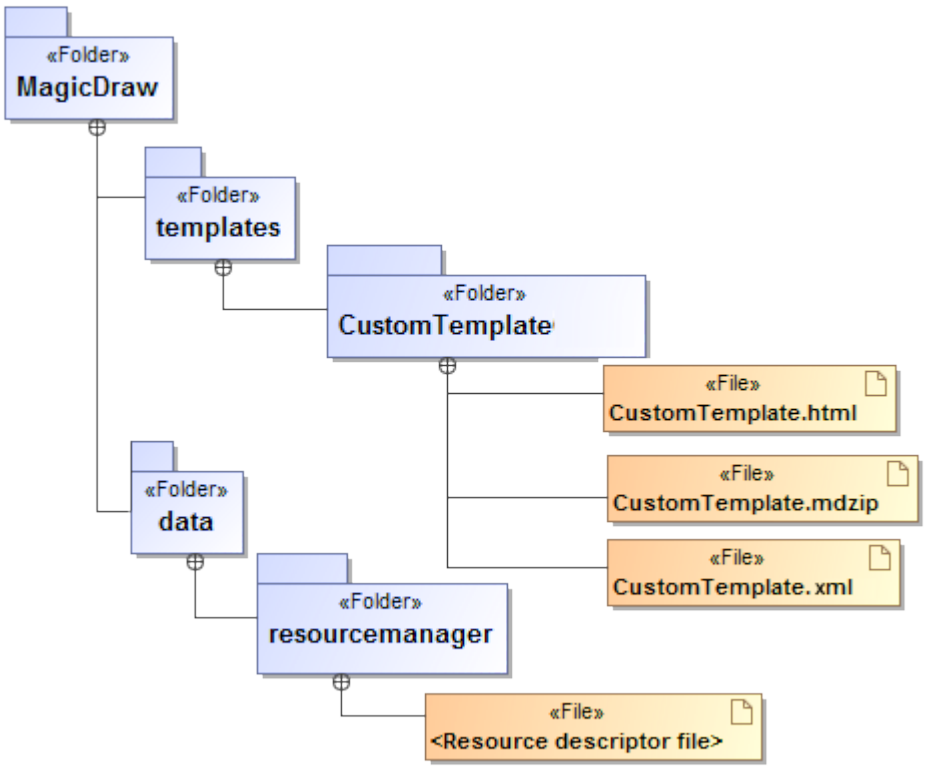
Distributing Templates

Files for the template distribution are as follows:

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

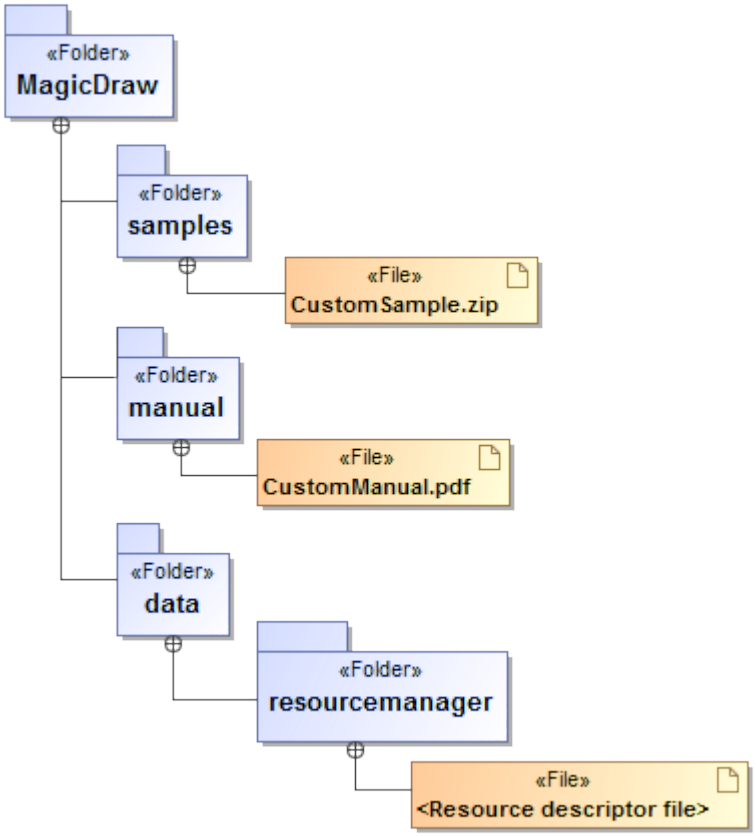
- *CustomTemplate.mdzip* (required) - the template project file.
- *CustomTemplate.html* - the description of the template project.
- *CustomTemplate.html* - the description of the template group.

The file and folder structure is depicted in the following figure:



Distributing Samples and Documentation

You can distribute your created samples and documentation and import them into your modeling tool with the MagicDraw ResourceManager. The file and folder structure is depicted in the following figure:

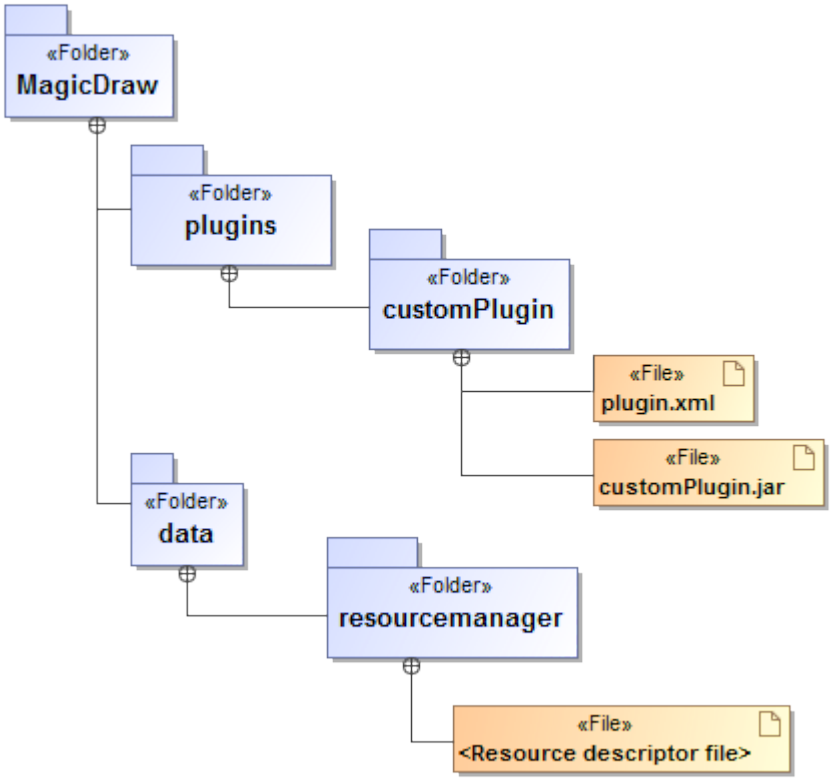


Distributing Plugins

Required files for a plugin distribution are as follows:

- *plugins.xml* - A plugin description(see page 2162).
- *customPlugin.jar* - jarred plugin class files(see page 2165). You may select any title for this file.

The file and folder structure is depicted in the following figure:



The plugin term may include all resources that could be distributed, such as custom diagrams, profiles, templates, samples, and others.

Related pages

- Resource manager descriptor file(see page 2192)
- Plugin descriptor(see page 2162)
- Plugin classes(see page 2165)

Resource manager descriptor file

If you are planning to import and export your resources with the Resource Manager, you should read this section. For importing resources to your modeling tool, the resource manager descriptor file should be created. In this section you will find information about the descriptor file naming, location, and a sample of this file.

> ⓘ For more information about importing resources with Resource Manager, see *MagicDraw UserManual.pdf*.

Resource Manager descriptor file naming

The Resource Manager descriptor file has a specific naming. The structure of the name is *MDR_<type>_<plugin name>_<plugin id>_descriptor.xml.* For example, *MDR_Plugin_CustomPlugin_3001_descriptor.xml*.

All spaces are replaced with the underscore symbol "_".

Resource Manager descriptor file location

The file location is *<modeling tool installation directory>\data\resourcemanager*.

Resource Manager descriptor file content and sample

This section presents a basic sample of the Resource Manager descriptor file structure. This sample represents a plugin distribution, which also includes a custom diagram, profile, template, sample, and documentation.

You can also distribute custom diagrams, profiles, templates, or samples separately, you only need to change the "*type*" value.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

```
<resourceDescriptor
    critical="false"
    date="2007-12-24+02:00"
    description="My Plug-in"
    homePage="http://www.nomagic.com"
    id="3001"
    mdVersionMax="higher"
    mdVersionMin="15.0"
    name="CustomPlugin"
    product="CustomPlugin"
    type="Plugin">

    <version human="1.0 beta" internal="1" resource="10"/>
    <provider name="No Magic" />

    <edition>Enterprise</edition>
    <edition>Architect</edition>
    <edition>Standard</edition>
    <edition>Professional Java</edition>
    <edition>Professional C++</edition>
    <edition>Professional C#</edition>
    <edition>Professional ArcStyler</edition>
    <edition>Reader</edition>
    <edition>OptimalJ</edition>

    <installation>
        <file from="data/defaults/data/diagrams/CustomDiagram/descriptor.xml" to="dat
a/defaults/data/diagrams/CustomDiagram/descriptor.xml" />
        <file from="profiles/CustomProfile.xml.zip" to="profiles/
CustomProfile.xml.zip" />
        <file from="templates/CustomTemplate.xml.zip" to="templates/
CustomTemplate.xml.zip" />
        <file from="samples/CustomPluginSample.mdzip" to="samples/
CustomPluginSample.mdzip" />
        <file from="manual/CustomPluginManual.pdf" to="manual/
CustomPluginManual.pdf" />
        <file from="plugins/customPlugin/*.*" to="plugins/customPlugin/*.*" />
        <file from="data/resourcemanager/MDR_Plugin_CustomPlugin_3001_descriptor.xml"
 to="data/resourcemanager/MDR_Plugin_CustomPlugin_3001_descriptor.xml" />

    </installation>
</resourceDescriptor>
```

The following table describes the terms used in the sample:

Element	Description
id	A unique plugin id. *id* is used to form a descriptor file name. This id must be unique. To prevent duplicates, use a number starting from 3000 or contact the No Magic support at http://www.nomagic.com/support.html.

Element	Description
name	A plugin name. A name is used to form a descriptor file name. The plugin name must be unique between all program resources.
product	A product name. Used only for the “Plugin (commercial)” resource type.
type	A type may be one of the following types: Custom Diagram, Plugin, Plugin (commercial), Profile, Sample, Template.
version resource	This number is used to identify the resource build/release. For example, a new resource build must be released, while the resource human and internal versions should not change. A suggested value is a *version internal* value multiplied by 10. *version resource* should be a number.
version internal	This version number is not visible to the user and may be used for an internal count. *version internal* should be a number.
version human	A human readable version number of the resource. This version number is visible to users. The *version human* number may consist of numbers and/or words.
edition	Supported program editions.
installation	*installation* includes files, which will be copied from the custom plugin archive to the program folder. Do not use "*.*" ! If a file name includes “*.*“, all defined files will be removed when uninstalling the plugin. For example, if "samples/*.*" is defined, all files from the “samples” folder will be removed after uninstalling the resource.
critical	If the resource is selected as critical, it cannot be removed because MagicDraw does not allow this.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Building a resource distribution file

Users can create their own resource distribution files and install resources from them using the Resource Manager. These files can contain plugins/profiles/model libraries etc. The resources must be packed to separate **.zip** files.

This can be useful for distributing a set of resources accompanied with custom resources or in a locked down environment where the resource bundle is hosted in a local network.

There are multiple ways to create a resource bundle:

1) Using the **Development Tools** plugin GUI

1. Install the **Development Tools** plugin using your tool using Resource Manager.
2. Restart the modeling tool.
3. In the modeling tool main menu, click **Tools** > **Development Tools** > **Build Resource Distribution File**.
4. Browse and select the resources folder, choose the output location and click **OK**. A resource distribution file will be built in the output folder.

> The **Development Tools** plugin can either be installed from the NoMagic Resource Server or can be found in the no-cost resource distribution file.

2) Using command line tool.

In case the process of building a resource distribution files should be automated, it can be done using command line tool.

Three arguments are required:

- Resource directory path
- Output directory path
- Result distribution file name

Example:

```
java -cp "lib/*;lib/bundles/*"

com.nomagic.magicdraw.resourcemanager.distribution.GenerateResourceDistributionFileTa
sk "E:\myResources" "E:\myBundles" "myCustomResourceDistributionFile"
```

> Resource distribution file creator checks for validity of each resource. Only valid resources are included in the resource distribution file. The resources must meet the required folder/file structure and must contain appropriate descriptor file. For more information about the folder/file structure visit Creating required files and folders structure(see page 2187)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

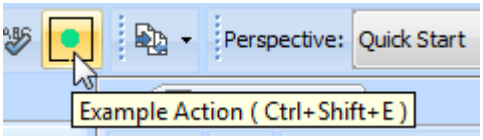
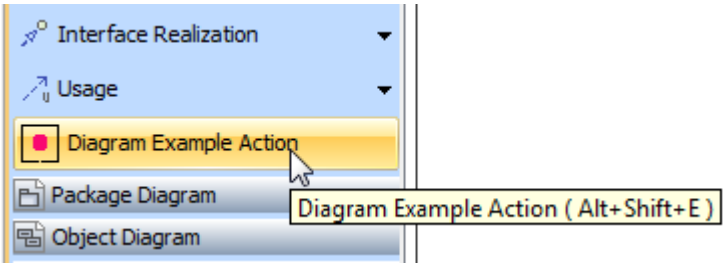
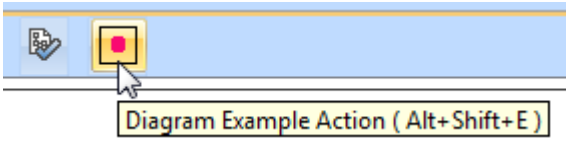
Adding new functionality

A program actions mechanism enables to add new functionality to the modeling tool and the way to invoke it through GUI. In this case, all menu commands and buttons are considered as actions.

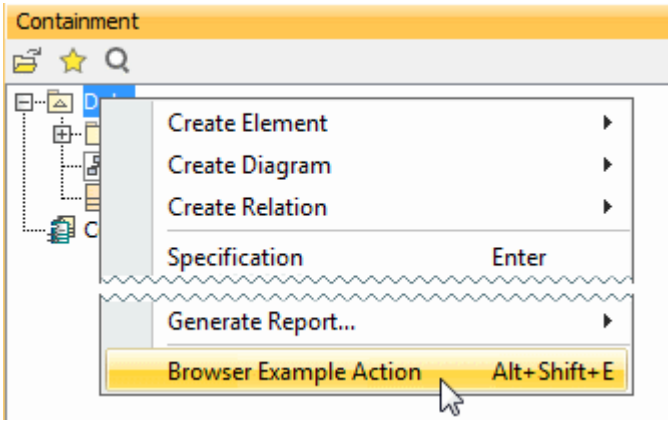
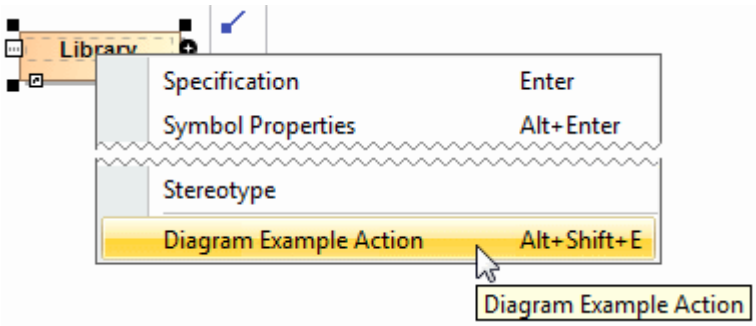
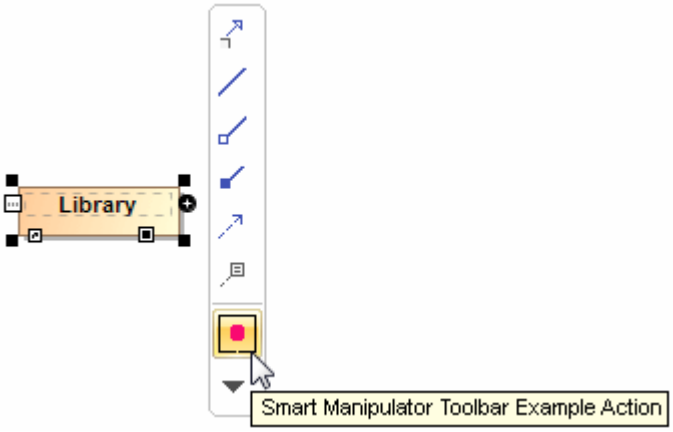
Related pages
Using and extending other UI component(see page 2313)

Invoking actions

Actions can be invoked from:

Main menu	
Main toolbar	
Diagram pallet	
Diagram toolbar	

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Browser shortcut menu	
Diagram shortcut menu	
Smart manipulator toolbar	
Keyboard shortcuts	This action cannot be represented in GUI. Create a new action and assign a keyboard shortcut to invoke that action.

Creating new actions

> You can find the code examples in
> - *<modeling tool installation directory>\openapi\examples\actiontypes*
> - *<modeling tool installation directory>\openapi\examples\simpleconfigurators*
> - *<modeling tool installation directory>\openapi\examples\selectionactions*

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Step #1: Create a new action class

All actions used in a modeling tool must be subclasses of the *com.nomagic.magicdraw.actions.MDAction*[451] class (see javadoc[452] for more details). The following *MDAction*[453]subclasses that are used for basic purposes are already created in a modeling tool:

- *com.nomagic.magicdraw.ui.browser.actions.DefaultBrowserAction*[454] – an action class, used for a browser action. It enables to access some browser tree and nodes and is recommended to use for performing some actions with the selected browser nodes.
- *com.nomagic.magicdraw.ui.actions.DefaultDiagramAction*[455]– an action class for a diagram action. It enables to access some diagram elements and is recommended to use for performing some actions with the selected diagram elements.
- *com.nomagic.magicdraw.actions.PropertyAction*[456] – an action for changing some element or application property. It can be used for changing properties defined by a user.

You must override at least the *actionPerformed(java.awt.event.ActionEvent)*[457] method and implement in it what this actions is going to do.

Example #1: A simple action

```
    class SimpleAction extends MDAction
    {
        public SimpleAction(String id, String name)
        {
             super(id, name, null, null);
        }
        /**
        * Shows message.
        */
        public void actionPerformed(ActionEvent e)
        {

JOptionPane.showMessageDialog(MDDialogParentProvider.getProvider().getDialogParent(),
"This is:" + getName());
        }
    }
```

451 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/actions/MDAction.html
452 http://jdocs.nomagic.com/190beta/
453 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/actions/MDAction.html
454 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ui/browser/actions/DefaultBrowserAction.html
455 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ui/actions/DefaultDiagramAction.html
456 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/actions/PropertyAction.html
457 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/actions/MDAction.html#actionPerformed-java.awt.event.ActionEvent-

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example #2: An action for a browser

```
    public class BrowserAction extends DefaultBrowserAction
    {
        /**
        * Creates action with name "ExampleAction"
        */
        public BrowserAction()
        {
          super("", "ExampleAction", null, null);
        }
        public void actionPerformed(ActionEvent e)
        {
            Tree tree = getTree();
            String text="Selected elements:";
            for (int i = 0; i < tree.getSelectedNodes().length; i++)
            {
                Node node = tree.getSelectedNodes()[i];
                Object userObject = node.getUserObject();
                if (userObject instanceof BaseElement)
                {
                    BaseElement element = (BaseElement) userObject;
                    text += "\n"+element.getHumanName();
                }
            }

JOptionPane.showMessageDialog(MDDialogParentProvider.getProvider().getDialogParent(),
text);
        }
    }
```

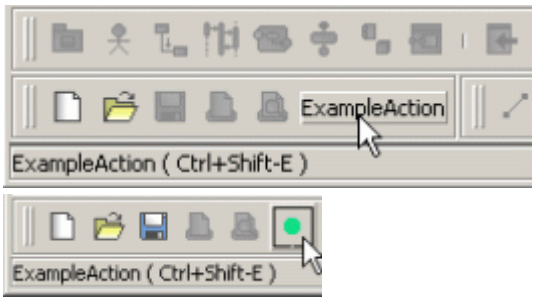
Step #2: Specify action properties

Property	Function
ID	ID is a unique String for identifying an action. If the action ID is set to *null, a* new ID is generated. An action name can be used as ID. All MagicDraw (or other modeling tool developed by No Magic Inc.) default actions IDs are defined in the *com.nomagic.magicdraw.actions.ActionsID*[458] class (for more details, see JavaDoc[459]). You can use these constants for accessing actions. Do not use it as new actions ID.
Name	The action name is visible in all GUI elements.

458 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/actions/ActionsID.html
459 http://jdocs.nomagic.com/190beta/

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Property	Function
Shortcut and mnemonic	Every action can have an assigned keyboard shortcut. Shortcuts can be customized from the **Environment Options** dialog box. (see MagicDraw User Manual, Section “Environment Options”.) The action must have an ID assigned (not null). In other case, shortcuts cannot be restored after restarting an application.

Icon	Every action can have a small and large icon. A small icon is described as *javax.swing.Action.SMALL_ICON* and can be used in menu items. A large icon is used in toolbar buttons. The action for a toolbar must have a large icon, otherwise it is displayed as a button with an action name.
	
	 
	
	`// setting an icon. On the toolbar, the button with an icon looks better than with the text.`
	`action.setLargeIcon( new ImageIcon(getClass().getResource("main_toolbar_icon.gif") ) );`

Description	The action’s description that is shown as a tool tip text.

Step #3: Describe Enabling/Disabling logic

There are two ways for controlling the update of the actions state:

1. **Add an action to the predefined actions group.**

   Actions can be added into one of predefined actions groups. All actions of the one group are disabled/enabled together. Conditions for groups enabling/disabling and status updating are predefined and cannot be changed.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example:

```
MDAction action = new MDAction("Example","Example",KeyEvent.VK_E,
ActionsGroups.PROJECT_OPENED_RELATED);
```

2. **Implement the *updateState()*[460] method for the action.**

   Here you may describe all conditions when an action must be enabled and disabled. Example of the *updateState()*[461] method for the browser shortcut menu action:

```
        public void updateState()
        {
           //action will be enabled only if there are some selected nodes.
            Tree tree = getTree();
            if(tree != null)
            {
                setEnabled(tree.getSelectedNode() != null);
            }
        }
```

If the action is not added to any group, the *updateState()*[462] method for all such actions will be invoked after executing any command and after closing/opening a project or window.

When some actions need to refresh their state, all actions without a group can be updated manually:

```
ActionsStateUpdater.updateActionsState();
```

Step #4: Configure actions

Every action must be added into an *com.nomagic.actions.ActionsCategory*[463] class. *ActionsCategory*[464]is a small group for actions. It can be represented as a separator or submenu (a nested category).

Categories are added into the com.nomagic.actions.ActionsManager[465] class, which is some kind of an actions container. One *ActionsManager* [466]represents one GUI element – a menu bar, a shortcut menu, or a toolbar.

The following table explains how classes of a modeling tool maps into GUI elements:

	ActionsManager	**Category**	**Action**
Menu	Menu bar	Menu	Menu item

460 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/actions/MDAction.html#updateState--
461 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/actions/MDAction.html#updateState--
462 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/actions/MDAction.html#updateState--
463 http://jdocs.nomagic.com/190beta/com/nomagic/actions/ActionsCategory.html
464 http://jdocs.nomagic.com/190beta/com/nomagic/actions/ActionsCategory.html
465 http://jdocs.nomagic.com/190beta/com/nomagic/actions/ActionsManager.html
466 http://jdocs.nomagic.com/190beta/com/nomagic/actions/ActionsManager.html

	ActionsManager	Category	Action
Toolbar	All toolbars	One toolbar	Button
Shortcut Menu	Shortcut menu	Submenu	Menu item

Actions in *ActionsManagers* are configured by many *Configurators*. A *Configurator* is responsible for adding or removing the action into a strictly defined place and position between other actions.

There are three types of configurators:

- *com.nomagic.actions.AMConfigurator*[467] - a configurator for a general purpose. It is used for menus, toolbars, browser, and diagrams shortcuts.
- *com.nomagic.magicdraw.actions.BrowserContextAMConfigurator*[468] - a configurator for configuring managers for a browser shortcut (pop-up) menu. It can access a browser tree and nodes.
- *com.nomagic.magicdraw.actions.DiagramContextAMConfigurator*[469] - a configurator for configuring shortcut menus in a diagram. It can access a diagram, selected diagram elements, and an element that requests a shortcut menu.

Once *ActionsManagers* for the main menu and all toolbars are created and configured, the actions can be only disabled but not removed later.

Shortcut menus are created on every invoking, so *ActionsManagers* are created and configured every time and actions can be added or removed every time.

Example #1: Add some action into a browser's shortcut menu

```
final DefaultBrowserAction browserAction = ...
BrowserContextAMConfigurator brCfg = new BrowserContextAMConfigurator()
{
    // Implement configuration.
    // Add or remove some actions in ActionsManager.
    // A tree is passed as an argument, provides ability to access nodes.
   public void configure(ActionsManager mngr, Tree browser)
   {
        // Actions must be added into some category.
        // So create the new one, or add an action into the existing category.
        MDActionsCategory category = new MDActionsCategory("", "");
        category.addAction(browserAction);
        // Add a category into the manager.
        // A category isn't displayed in a shortcut menu.
        mngr.addCategory(category);
   }
   public int getPriority()
   {
      return AMConfigurator.MEDIUM_PRIORITY;
   }
};
```

467 http://jdocs.nomagic.com/190beta/com/nomagic/actions/AMConfigurator.html
468 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/actions/BrowserContextAMConfigurator.html
469 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/actions/DiagramContextAMConfigurator.html

Example #2: add some action into main menu, after creating a new project

```
    // Create an action.
    final MDAction someAction = ...
    AMConfigurator conf = new AMConfigurator()
    {
        public void configure(ActionsManager mngr)
        {
           // Searching for an action after which an insertion should be done.
           NMAction found= mngr.getActionFor(ActionsID.NEW_PROJECT);

           // The action found, inserting
           if( found != null )
           {
                // find the category of the "New Project" action.
                ActionsCategory category =
(ActionsCategory)mngr.getActionParent(found);


                // Get all actions from this category (menu).
                List actionsInCategory = category.getActions();


                //Add the action after the "New Project" action.
                int indexOfFound = actionsInCategory.indexOf(found);
                actionsInCategory.add(indexOfFound+1, someAction);

                // Set all actions.
                category.setActions(actionsInCategory);
            }
        }

        public int getPriority()
        {
            return AMConfigurator.MEDIUM_PRIORITY;
        }
    };
```

Step #5: Register Configurator

All configurators are registered in the *com.nomagic.magicdraw.actions.ActionsConfiguratorsManager*[470] class. *ActionsConfiguratorsManager*[471]enables to add or remove many configurators to every configuration predefined by a modeling tool.

All available configurations are accessible in the following way:

470 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/actions/ActionsConfiguratorsManager.html
471 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/actions/ActionsConfiguratorsManager.html

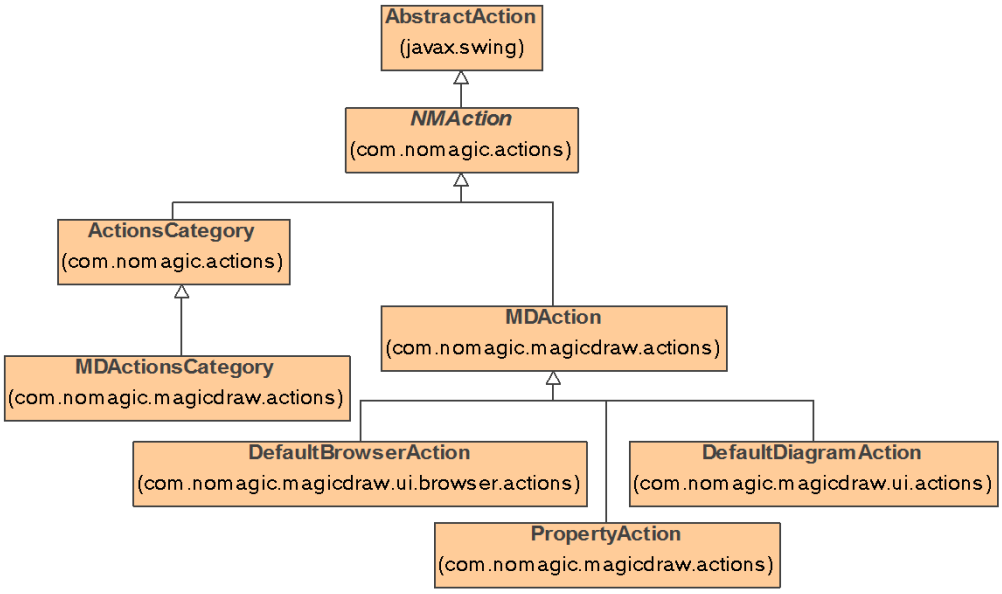
```
ActionsConfiguratorsManager.getInstance().add<configuration_name>Configurator(configu
rator);
```

Example: Add new configurator for CONTAINMENT_BROWSER_CONTEXT configuration

```
    //See previous examples, how to create a configurator for browser actions.
    //Add the configurator into ActionsConfiguratorsManager.

ActionsConfiguratorsManager.getInstance().addContainmentBrowserContextConfigurator(br
Cfg);
```

Actions' hierarchy

The following figure depicts the hierarchy of actions:



Predefined Actions Configurations

MAIN_MENU

MAIN_TOOLBAR

MAIN_SHORTCUTS

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

CUSTOMIZABLE_SHORTCUTS

CONTAINMENT_BROWSER_CONTEXT

CONTAINMENT_BROWSER_SHORTCUTS

CONTAINMENT_BROWSER_TOOLBAR

INHERITANCE_BROWSER_CONTEXT

INHERITANCE_BROWSER_SHORTCUTS

INHERITANCE_BROWSER_TOOLBAR

DIAGRAMS_BROWSER_SHORTCUTS

DIAGRAMS_BROWSER_TOOLBAR

EXTENSIONS_BROWSER_CONTEXT

EXTENSIONS_BROWSER_SHORTCUTS

EXTENSIONS_BROWSER_TOOLBAR

SEARCH_BROWSER_CONTEXT

SEARCH_BROWSER_SHORTCUTS

SEARCH_BROWSER_TOOLBAR

Working with icons

Use com.nomagic.magicdraw.icons.IconsFactory[472] to load the icons.

As of version 18.2, modeling tools support HiDPI/Retina monitors and chooses dynamically which icon to load - a lower or higher resolution. All this logic is encapsulated in com.nomagic.ui.ScalableImageIcon[473]. IconsFactory[474]adds additional caching and ensures that icon from the same URL will be loaded just once.

The HiDPI/Retina compatibility requires two images - for a normal resolution and for a high resolution. Use the SVG format for a high resolution image. Put both icons with the same name (but a different extension) at the same location. Use a normal icon URL to load the icon, but the SVG icon will be loaded on the high resolution monitor.

472 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/icons/IconsFactory.html
473 http://jdocs.nomagic.com/190beta/com/nomagic/ui/ScalableImageIcon.html
474 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/icons/IconsFactory.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Icon classes

Class	Description
com.nomagic.ui.ResizableIcon[475]	The extended version of *javax.swing.Icon.* It can be scaled (resized) during the painting.
com.nomagic.ui.ResizableIconAdapter[476]	An adapter of a simple *javax.swing.Icon* to a *com.nomagic.ui.ResizableIcon*[477].
com.nomagic.ui.SquareIcon[478]	An icon which fits another icon into a square of a given size.
com.nomagic.ui.ScalableImageIcon[479]	The HiDPI (Retina) friendly *javax.swing.ImageIcon* implementation. This class dynamically chooses a right image by the current screen dpi and/or the scaling factor defined in the system (Retina and etc). If scaling is needed and the SVG icon with the same name is available at the given location, the SVG icon will be used instead of a bitmap icon. Dynamic choosing will work only if two icons are provided at the same location. For example, */com/product/icons/a.png* and */com/product/icons/a.svg. ScalableImageIcon(Product.class, "/com/product/icons/a.png")* will load *a.png* on a regular dpi monitor, but *a.svg* will be loaded on the HiDPI monitor. Keep in mind that the SVG icon size can be any. The SVG icon will be resized according to the size of the *a.png* icon and a scaling factor. For example, if a size of *a.png* is 16x16 and a scaling factor is 2, the loaded SVG icon will be resized to 32x32 if needed. The original icon will be loaded and scaled if the SVG icon is not provided, but HiDPI with scaling is used.

Utility classes

Class	Description
com.nomagic.magicdraw.icons.IconsFactory[480]	A utility class to load icons from the MagicDraw icons package or other locations.
com.nomagic.ui.IconUtilities[481]	A utility class to work with Icons.

475 http://jdocs.nomagic.com/190beta/com/nomagic/ui/ResizableIcon.html
476 http://jdocs.nomagic.com/190beta/com/nomagic/ui/ResizableIconAdapter.html
477 http://jdocs.nomagic.com/190beta/com/nomagic/ui/ResizableIcon.html
478 http://jdocs.nomagic.com/190beta/com/nomagic/ui/SquareIcon.html
479 http://jdocs.nomagic.com/190beta/com/nomagic/ui/ScalableImageIcon.html
480 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/icons/IconsFactory.html
481 http://jdocs.nomagic.com/190beta/com/nomagic/ui/IconUtilities.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Advanced icons classes

Class	Description
com.nomagic.ui.AlphaCompositeIcon[482]	Paints a wrapped icon with an alpha composite.
com.nomagic.ui.BorderIcon[483]	An icon for adding a border around a wrapped icon.
com.nomagic.ui.DoubleIcon[484]	An Icon for painting two wrapped icons on a top of each other.
com.nomagic.ui.RetinaImageIcon[485]	A retina friendly image icon implementation. The icon (and the image returned by this icon) is twice smaller then the wrapped icon itself. A wrapped icon is used for painting. On painting, Graphics is scaled down by a retina scale factor and the wrapped icon is painted.
com.nomagic.ui.DoubleSizeImageIcon[486]	An icon combines two other icons and chooses which one to paint depending on the graphics scaling.
com.nomagic.ui.ResizableIconImageIcon[487]	An image icon which wraps other icon and provides an image for a wrapped icon. The provided image is HiDPI/Retina friendly. On Mac, a special instance of *java.awt.Image* (*sun.awt.image.MultiResolutionToolkitImage*) is returned which supports multi-resolution.

On this page

- Icon classes(see page 2206)
- Utility classes(see page 2206)
- Advanced icons classes(see page 2207)

482 http://jdocs.nomagic.com/190beta/com/nomagic/ui/AlphaCompositeIcon.html
483 http://jdocs.nomagic.com/190beta/com/nomagic/ui/BorderIcon.html
484 http://jdocs.nomagic.com/190beta/com/nomagic/ui/DoubleIcon.html
485 http://jdocs.nomagic.com/190beta/com/nomagic/ui/RetinaImageIcon.html
486 http://jdocs.nomagic.com/190beta/com/nomagic/ui/DoubleSizeImageIcon.html
487 http://jdocs.nomagic.com/190beta/com/nomagic/ui/ResizableIconImageIcon.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related pages

MDP protocol

MagicDraw registers a custom URL protocol handler **mdp:// ** for loading resources (icons) from MagicDraw or plugins jars/classpath. Resources can also be loaded from attached files in an opened project.

Usage samples:

- Use the *mdp* protocol in the HTML based model element documentation to point to some image from a plugin jar.
- Use the *mdp* protocol in the HTML based model element documentation to point to some attached image from a project.

URL syntax

mdp://resource/	Use the "*resource*" host to load resources from jars/classpaths. For example, to load the *activitydiagram.png* icon from MagicDraw jar, use such URL: `mdp://resource/com/nomagic/magicdraw/icons/diagrams/activitydiagram.png`
mdp://attachedFile/	Use the "*attachedFile*" host to load resources from the attached file into the active project. For example, to load the icon from the attached file element with ID "*_18_2_8ca0285_1436339188295_24687_13288*", use such URL: `mdp://attachedFile#_18_2_8ca0285_1436339188295_24687_13288`

Running action with progress

For long lasting actions, there is a need to show a progress and run the asynchronous code.

There are two options to execute these actions:

- a foreground task - use com.nomagic.task.RunnableWithProgress[488] or com.nomagic.magicdraw.ui.MagicDrawProgressStatusRunner[489]
- a background task - use com.nomagic.magicdraw.task.BackgroundTaskRunner[490]

> You can find the code examples in *<modeling tool installation directory>\openapi\examples\progressstatus*

488 http://jdocs.nomagic.com/190beta/com/nomagic/task/RunnableWithProgress.html
489 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ui/MagicDrawProgressStatusRunner.html
490 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/task/BackgroundTaskRunner.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related pages

- Multi-threading(see page 2395)

Working with project

Project concept

The *com.nomagic.magicdraw.core.Project*[491] represents the storage of all project specific data like the UML model, all diagrams, project specific options and etc.

Multiple projects can be opened at the same time, but just one Project can be active in UI. Data from one project can not reference data in other opened project. The active (current) project can be accessible in the following way:

```
Project project = Application.getInstance().getProject();
```

Also, the *Project*[492]is accessible for any other com.nomagic.magicdraw.uml.BaseElement[493] *(UML Element or Presentation Element)*

```
Project project = Project.getProject(element);
```

The project keeps references to a root *Model(s)*, also has references to all diagrams.

Project always have one primary root Model and may have zero or more attached Models from used projects. Attached models are available only in projects loaded from the Cameo Enterprise Dataware server(see page 1367). Locally stored projects or projects loaded from Teamwork Server(see page 1479) has only a primary root model.

The primary root model can be accessed in the following way:

```
Package model = project.getPrimaryModel();
```

All root models can be accessed in the following way:

```
List<Package> models = project.getModels();
```

491 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Project.html
492 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Project.html
493 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/BaseElement.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related pages

- Project structure decomposition(see page 2221)
- UML Model(see page 2225)
- Working with diagrams(see page 2269)
- Project options(see page 2221)

Project descriptor

com.nomagic.magicdraw.core.project.ProjectDescriptor[494] represents a project (or a used project) as a resource for storing and loading. The same project can have multiple *ProjectDescriptors*.

There are two types of *ProjectDescriptors*:

- **A local project descriptor.** It represents a local ordinary project. A descriptor can be created for a project or file object.
- **A remote project descriptor.** It represents a project stored in the server.

A server project has both descriptors (local and remote) because it can be saved locally.

Every *ProjectDescriptor*[495]provides the following properties:

- **URI.** The location is a specific *java.lang.String* value that holds all information that is needed to access a project.
- **Representation String.** A string used for a project identification in the user interface (GUI).
- **Remote flag.** The remote flag indicates if the project descriptor represents a server project.

com.nomagic.magicdraw.core.project.ProjectDescriptorsFactory[496] can create an appropriate *ProjectDescriptor*[497]object,

com.nomagic.magicdraw.teamwork.application.TeamworkUtils[498] helps to find an existing Teamwork Server(see page 1479) (remote) project descriptors.

Projects management

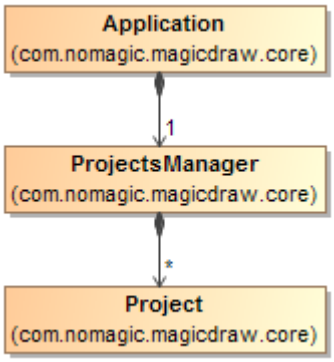
The *com.nomagic.magicdraw.core.project.ProjectsManager*[499] class is responsible for containment and management of projects.

Use the code below to access ProjectsManager[500]

494 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectDescriptor.html
495 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectDescriptor.html
496 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectDescriptorsFactory.html
497 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectDescriptor.html
498 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/teamwork/application/TeamworkUtils.html
499 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectsManager.html
500 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectsManager.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

```
ProjectsManager projectsManager = Application.getInstance().getProjectsManager();
```



ProjectsManager[501]provides the API for *com.nomagic.magicdraw.core.Project*[502] creating, closing, saving, loading, and activating. A program can have multiple opened projects, but only one project can be active.

```
//Gets all the projects
List<Project> projects = projectsManager.getProjects();

//Gets an active project
Project activeProject = projectsManager.getActiveProject();
```

An active project can also be accessed directly from *com.nomagic.magicdraw.core.Application*[503]:

```
//Gets an active project
Project project = Application.getInstance().getProject();
```

You can find the code examples in *<programinstallation directory>\openapi\examples\projects*

Related pages

Project loading and saving

Projects are saved and loaded by using two methods in the *com.nomagic.magicdraw.core.project.ProjectsManager*[504] class.

501 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectsManager.html
502 http://com.nomagic.magicdraw.core.Project
503 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Application.html
504 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectsManager.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- saveProject([505]ProjectDescriptor[506], boolean)[507]
- loadProject([508]ProjectDescriptor[509], boolean)[510]

> ⚠ "Save" and "Load" means "Commit" and "Update" for the server project.

A project cannot be saved using a descriptor, if the project isn't specified, and a project cannot be loaded, if the file isn't specified. In such cases *java.lang.IllegalStateException* is thrown.

The silent mode means that during the saving or load process no GUI interruptions for a user input is used, for example, there is no the **Commit Project** dialog box while committing a server project or there is no the **Save** dialog box while saving a new project (a project is saved into the last used directory).

Save participant

Use *com.nomagic.magicdraw.core.SaveParticipant* [511]to plugin into a save/commit operation.

Register it using *com.nomagic.magicdraw.core.Application.addSaveParticipant(*[512]SaveParticipant[513]*)*.

Implementing *SaveParticipant.isReadyForSave(*[514]Project[515], ProjectDescriptor[516]*)* you can "disable" the save/commit operation until some conditions are met.

---

50 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectsManager.html#saveProject-com.nomagic.magicdraw.core.project.ProjectDescriptor-boolean-
506 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectDescriptor.html
70 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectsManager.html#saveProject-com.nomagic.magicdraw.core.project.ProjectDescriptor-boolean-
80 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectsManager.html#loadProject-com.nomagic.magicdraw.core.project.ProjectDescriptor-boolean-
509 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectDescriptor.html
91 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectsManager.html#loadProject-com.nomagic.magicdraw.core.project.ProjectDescriptor-boolean-
511 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/SaveParticipant.html
21 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Application.html#addSaveParticipant-com.nomagic.magicdraw.core.SaveParticipant-
513 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/SaveParticipant.html
41 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/SaveParticipant.html#isReadyForSave-com.nomagic.magicdraw.core.Project-com.nomagic.magicdraw.core.project.ProjectDescriptor-
515 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Project.html
516 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectDescriptor.html

Example #1. Saving an active project

```
    ProjectsManager projectsManager = Application.getInstance().getProjectsManager();
    // An active project
    Project project = projectsManager.getActiveProject();

    // Get a project descriptor
    ProjectDescriptor projectDescriptor =
ProjectDescriptorsFactory.getDescriptorForProject(project);

    // Save a project
    projectsManager.saveProject(projectDescriptor, true);
```

Example #2. Loading a project from the file

The project can be loaded, if the project's file name is known:

```
    ProjectsManager projectsManager = Application.getInstance().getProjectsManager();
    File file = new File(projectFilePath);
    // Create a project descriptor
    ProjectDescriptor projectDescriptor =
ProjectDescriptorsFactory.createProjectDescriptor(file.toURI());
    projectsManager.loadProject(projectDescriptor, true);
```

Example #3. Importing another project file

The project can be imported, if the project's file name is known:

```
    ProjectsManager projectsManager = Application.getInstance().getProjectsManager();
    File file = new File(projectFilePath);
    // Create a project descriptor
    ProjectDescriptor projectDescriptor =
ProjectDescriptorsFactory.createProjectDescriptor(file.toURI());
    projectsManager.importProject(projectDescriptor);
```

Example #4. Loading a server project

The project can be loaded, if the project's qualified name is known:

```
    ProjectsManager projectsManager = Application.getInstance().getProjectsManager();
    // Create a project descriptor
    ProjectDescriptor projectDescriptor =
TeamworkUtils.getRemoteProjectDescriptorByQualifiedName(remoteProjectQualifiedName);
    if (projectDescriptor != null)
    {
        projectsManager.loadProject(projectDescriptor, true);
    }
```

Related pages

- Project descriptor(see page 2210)

Used project management

com.nomagic.magicdraw.core.project.ProjectsManager[517] provides the used project management (project usage, export, import, reload, and package sharing) methods.

Example #1. Exporting a used local project

The collection of project packages can be exported as a used project.

```
    ProjectsManager projectsManager = Application.getInstance().getProjectsManager();
    File file = new File(moduleFilePath);
    ProjectDescriptor projectDescriptor =
ProjectDescriptorsFactory.createProjectDescriptor(file.toURI());
    // Export a collection of packages as a used project
    projectsManager.exportModule(project, packages, "My used local project",
projectDescriptor);
```

517 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectsManager.html

Example #2. Exporting a used server project

The com.nomagic.magicdraw.teamwork.application.TeamworkUtils[518] class is used to export a used server project.

```
    TeamworkUtils.exportTeamworkModule(project, packages, "My used remote project",
remoteProjectQualifiedName);
```

Example #3. Using a project

The local and server project usage is similar - an appropriate project descriptor must be used, but module directories project option should be updated before using local project:

```
    ProjectsManager projectsManager = Application.getInstance().getProjectsManager();
    File file = new File(moduleFilePath);
    // update project options only when using local project
    String moduleDir = file.getParent();
    ProjectOptions projectOptions = project.getOptions();
    List<String> directories = projectOptions.getModulesDirectories(true);
    if (!directories.contains(moduleDir))
    {
        projectOptions.addModuleDirectory(moduleDir);
    }
    ProjectDescriptor projectDescriptor =
ProjectDescriptorsFactory.createProjectDescriptor(file.toURI());
    // Use a project
    projectsManager.useModule(project, projectDescriptor);
```

Example #4. Importing a used project

The local and server project import does not differ. Just an appropriate project descriptor must be used:

```
    ProjectsManager projectsManager = Application.getInstance().getProjectsManager();
    File file = new File(moduleFilePath);
    ProjectDescriptor projectDescriptor =
ProjectDescriptorsFactory.createProjectDescriptor(file.toURI());
    projectsManager.importModule(project, projectDescriptor);
```

Example #5. Reloading a used project

The local and server project reload does not differ. Just an appropriate project descriptor must be used:

518 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/teamwork/application/TeamworkUtils.html

```
    ProjectsManager projectsManager = Application.getInstance().getProjectsManager();
    File file = new File(moduleFilePath);
    ProjectDescriptor projectDescriptor =
ProjectDescriptorsFactory.createProjectDescriptor(file.toURI());
    projectsManager.reloadModule(project, projectDescriptor);
```

Example #6. The package sharing

```
    ProjectsManager projectsManager = Application.getInstance().getProjectsManager();
    SessionManager.getInstance().createSession("Create shared package");
    // Create a package to share
    Package aPackage = project.getElementsFactory().createPackageInstance();
    aPackage.setOwner(project.getModel());
    aPackage.setName("myShare");
    SessionManager.getInstance().closeSession();

    // Share a package
    projectsManager.sharePackage(project, Arrays.asList(aPackage), "my used
project");


    // Save a project
    ProjectDescriptor projectDescriptor =
ProjectDescriptorsFactory.getDescriptorForProject(project);
    projectsManager.saveProject(projectDescriptor, true);
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example #7. Editing a used project within the project

```
    final String moduleFileName = new File(moduleFilePath).getName();
    // Find a used project directly used by the project (the primary project)
    final IAttachedProject attachedProject =
ProjectUtilities.findAttachedProjectByName(project, moduleFileName, false);
    if (attachedProject != null)
    {
        final ModuleUsage moduleUsage = new ModuleUsage(project.getPrimaryProject(),
attachedProject);
        final Set<ModuleUsage> moduleUsages = Collections.singleton(moduleUsage);

        final boolean readOnly = attachedProject.isReadOnly();
        if (readOnly)
        {
            // Make the used project editable (the read-write accessibility mode)
            ModulesService.setReadOnlyOnTask(moduleUsages, false);
        }
        // Get the first shared package of the used project
        final Collection<Package> sharedPackages =
ProjectUtilities.getSharedPackages(attachedProject);
        final Package aPackage = sharedPackages.iterator().next();
        // Create a class in the used project
        SessionManager.getInstance().createSession("Create use case in used
project");
        final Class aCase = project.getElementsFactory().createClassInstance();
        aCase.setOwner(aPackage);
        aCase.setName("myClass");
        SessionManager.getInstance().closeSession();
        // save the used project
        Application.getInstance().getProjectsManager().saveModule(project,
attachedProject, true, false);
        if (readOnly)
        {
             // Make a used project not editable (the read-only accessibility mode)
             ModulesService.setReadOnlyOnTask(moduleUsages, true);
        }
    }
```

Advanced management

Use the com.nomagic.magicdraw.core.ProjectUtilities[519] utility class for retrieving more project decomposition related information.

Use the *com.nomagic.magicdraw.core.modules.ModulesService*[520] or *ProjectsManager* [521]classes for managing attached projects.

519 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/ProjectUtilities.html
520 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/modules/ModulesService.html
521 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectsManager.html

Related pages

- Project descriptor(see page 2210)
- Project structure decomposition(see page 2221)

Working with projects from Teamwork Server

Use the com.nomagic.magicdraw.teamwork.application.TeamworkUtils[522] utility class to work with projects from Teamwork Server(see page 1479).

The Open API provides Teamwork Server(see page 1479) accessing methods. The example of the code:

522 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/teamwork/application/TeamworkUtils.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

```
    // check the logged user
    if (!user.equals(TeamworkUtils.getLoggedUserName()))
    {
        // login to the teamwork server
        if (!TeamworkUtils.login(server, -1, user, password))
        {
            // login failed return;
        }
    }

    // load a teamwork project
    ProjectDescriptor projectDescriptor =
TeamworkUtils.getRemoteProjectDescriptorByQualifiedName(projectName);
    ProjectsManager projectsManager = Application.getInstance().getProjectsManager();
    projectsManager.loadProject(projectDescriptor, true); Project project =
Application.getInstance().getProject();
    Model model = project.getModel();

    // get locked by a user
    Collection userLockedElements = TeamworkUtils.getLockedElement(project, user);
    if (!userLockedElements.contains(model))
    {
        // a model is not locked by a user, get all locked
        Collection allLockedElements = TeamworkUtils.getLockedElement(project, null);
        if (!allLockedElements.contains(model))
        {
            // a model is not locked, lock it
            TeamworkUtils.lockElement(project, model, false);
        }
    }

    SessionManager.getInstance().createSession("Rename Model");
    // change a name
    model.setName("MyModel");
    SessionManager.getInstance().closeSession();
    // unlcok and commit (because do not discard)
    TeamworkUtils.unlockElement(project, model, false, false);
    projectsManager.closeProject();
    // logout
    TeamworkUtils.logout();
```

You can find the code examples in *<modeling tool installation directory>\openapi\examples\teamwork*

Working with projects from Teamwork Cloud server

Use the com.nomagic.magicdraw.esi.EsiUtils[523] utility class to work with projects from Teamwork Cloud(see page 1367).

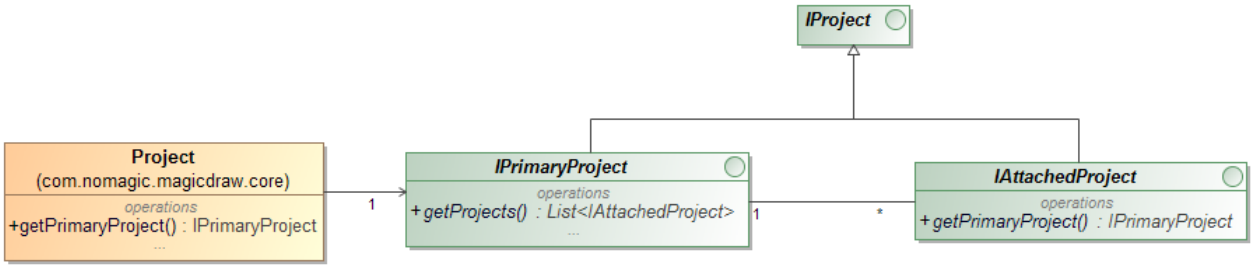
523 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/esi/EsiUtils.html

Check javadoc[524] for more details.

> You can find the code examples in *<modeling tool installation directory>\openapi\examples\teamworkcloud*

Project structure decomposition

Project structure is decomposed into several parts - a primary project and attached projects. The primary project part belong to a project itself. Attached project parts comes from "used projects" (modules). Used projects can use other projects, so there is a deep structure of attached projects.



Use the *com.nomagic.magicdraw.core.ProjectUtilities*[525] utility class for retrieving more project decomposition related information.

Use com.nomagic.magicdraw.core.modules.ModulesService[526] or *com.nomagic.magicdraw.core.project.ProjectsManager*[527] classes for managing attached projects.

Related pages

- Projects management(see page 2210)
- Used project management(see page 2214)

Project options

Every project has project related options. These options are stored together with the project data. Options are organized using *com.nomagic.magicdraw.properties.Property*[528] and *com.nomagic.magicdraw.properties.PropertyManager*[529] classes.

Project options belong to three categories:

524 http://jdocs.nomagic.com/190beta/
525 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/ProjectUtilities.html
526 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/modules/ModulesService.html
527 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectsManager.html
528 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/properties/Property.html
529 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/properties/PropertyManager.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

1. Visible in the Project Options UI window
2. Invisible in the Project Options UI window
3. Personal invisible in the Project Options UI window

Personal options are specific to every user in a server project. It means, every user may have different values of the same option if it is marked as "personal". The personal option is treated as a regular option in a local project.

Retrieving Project Options

Use *com.nomagic.magicdraw.core.Project.getOptions()*[530].

Retrieving Project Option value

The following example shows how to access project option's value:

```
    //retrieving a visible in UI option
    Property property =
project.getOptions().getProperty(ProjectOptions.PROJECT_GENERAL_PROPERTIES,
"TEST_PROPERTY_ID");
    //retrieving an invisible in UI option
    Property property =
project.getOptions().getProperty(ProjectOptions.PROJECT_INVISIBLE_PROPERTIES,
"TEST_PROPERTY_ID");
    //retrieving a personal invisible in UI option
    Property property =
project.getOptions().getProperty(ProjectOptions.PERSONAL_INVISIBLE_PROPERTIES,
"TEST_PROPERTY_ID");

    //retrieving an option value
    if (property != null)
    {
        Object value = property.getValue();
    }
```

Adding custom project options

Configurators are used for defining a new project options.

The following example shows, how to add the project option's configurator at the plugin's init()[531] method in order to have the additional project option "Test Property" for every project.

530 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Project.html#getOptions--
531 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/plugins/Plugin.html#init--

```
    ProjectOptions.addConfigurator(new ProjectOptionsConfigurator()
    {
        public void configure(ProjectOptions projectOptions)
        {
            com.nomagic.magicdraw.properties.Property property =
projectOptions.getProperty(ProjectOptions.PROJECT_GENERAL_PROPERTIES,
"TEST_PROPERTY_ID");
            if (property == null)
            {
                // Create a property, if it does not exist
                property = new StringProperty("TEST_PROPERTY_ID", "description");
                // Group
                property.setGroup("MY_GROUP");
                // The custom resource provider
                property.setResourceProvider(new PropertyResourceProvider()
                {
                    public String getString(String string, Property property)
                    {
                        if ("TEST_PROPERTY_ID".equals(string))
                        {
                            // Translate ID
                            return "Test Property";
                         }
                         if ("TEST_PROPERTY_ID_DESCRIPTION".equals(string))
                        {
                            // Translate a description
                            return "Test Property in My Group";
                        }
                        if ("MY_GROUP".equals(string))
                        {
                            // Translate a group
                            return "My Group";
                        }
                        return string;
                    }
                });

                // Add a property
                projectOptions.addProperty(ProjectOptions.PROJECT_GENERAL_PROPERTIES,
property);
            }
        }
    }
```

Related pages

- Properties(see page 2308)

Project life-cycle events

Use com.nomagic.magicdraw.core.project.ProjectsManager.addProjectListener(ProjectEventListener)[532] to register *com.nomagic.magicdraw.core.project.ProjectEventListener*[533]. The listener will be invoked on various project life-cycle related actions.

A recommendation is to register a listener in the *com.nomagic.magicdraw.plugins.Plugin.init()*[534] method.

Save participant

Use *com.nomagic.magicdraw.core.SaveParticipant*[535] to plugin into a save/commit operation.

Register it using com.nomagic.magicdraw.core.Application.addSaveParticipant([536]SaveParticipant[537])[538].

Related pages

- Projects management(see page 2210)

UML model

In projects, created by the modeling tools developed by No Magic Inc., the UML model is an implementation of the OMG UML 2.5.1 metamodel. We do not provide a very detailed description of all UML metamodel elements and their properties in this documentation or javadoc[539]. Our UML implementation covers whole UML specification as its and also extends it by introducing some additional Elements (like Diagram). You can find all this information in the UML 2.5.1 superstructure specification on the Object Management Group official Web site at http://www.omg.org/spec/UML/.

You should use UML model interfaces from the *com.nomagic.uml2.ext.magicdraw* package.

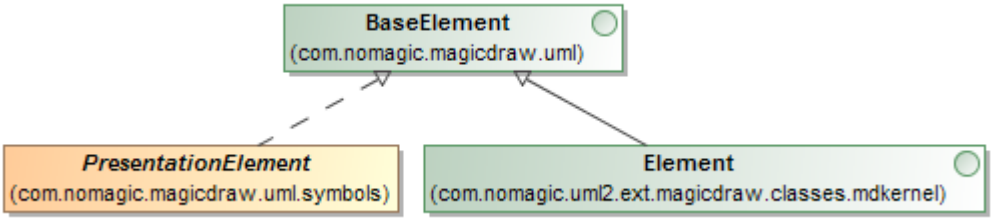
> Never use implementation classes like com.nomagic.uml2.ext.magicdraw.classes.mdkernel.impl.ElementImpl[540] in your code!

---

[532] http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectsManager.html#addProjectListener-com.nomagic.magicdraw.core.project.ProjectEventListener-
[533] http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectEventListener.html
[534] http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/plugins/Plugin.html#init--
[535] http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/SaveParticipant.html
[536] http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Application.html#addSaveParticipant-com.nomagic.magicdraw.core.SaveParticipant-
[537] http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/SaveParticipant.html
[538] http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Application.html#addSaveParticipant-com.nomagic.magicdraw.core.SaveParticipant-
[539] http://jdocs.nomagic.com/190beta/
[540] http://jdocs.nomagic.com/190/com/nomagic/magicdraw/uml/ElementImpl.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

The base interface of all model classes is *com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element*[541], it implements the *com.nomagic.magicdraw.uml.BaseElement*[542] inteface.



All structure derived from the *Element*[543]you can see in the OMG-UML 2.5.1 Superstructure Specification.

All attributes described in the UML specification are implemented and accessible through setters and getters.

> The UML model is thread safe for reading, but still the **UML model writing (modification)** is not thread safe(see page 2395).

Related pages

Working with UML model

Session management

com.nomagic.magicdraw.openapi.uml.SessionManager[544] is the singleton manager used for editing model Elements. All modifications to model elements should be performed between the createSession([545]Project[546], java.lang.String)[547] and closeSession([548]Project[549])[550] method calls.

To edit some *com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element*[551], a session with this manager must be created. After editing a model element, a session must be closed. After that, all changes will be

541 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
542 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/BaseElement.html
543 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
544 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/openapi/uml/SessionManager.html
54 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/openapi/uml/SessionManager.html#createSession-com.nomagic.magicdraw.core.Project-java.lang.String-
546 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Project.html
74 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/openapi/uml/SessionManager.html#createSession-com.nomagic.magicdraw.core.Project-java.lang.String-
84 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/openapi/uml/SessionManager.html#closeSession-com.nomagic.magicdraw.core.Project-
549 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Project.html
85 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/openapi/uml/SessionManager.html#closeSession-com.nomagic.magicdraw.core.Project-
551 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

applied to the model and registered in the command history (for undo/redo) as one command with a session name. Only one session can be active.

The following code can be used for accessing, checking, and creating the session:

```
// access a singleton instance by using getInstance()
// only one session can be active, so check this.
if (!SessionManager.getInstance().isSessionCreated(project))
{
    // create a new session.
    SessionManager.getInstance().createSession(project, "Edit");
}
```

If other session is already created and not closed, the createSession([552]Project[553], java.lang.String)[554] method throws the *java.lang.IllegalStateException* runtime exception.

Checking element editing permissions

It is a programmer responsibility to check model modification permissions before doing model changes. Some Elements in model can be read-only, but API **does not prevent** modifications of these elements.

The Element is read-only if:

- The Element belongs to the read-only used project (module)
- The Element is not locked for editing in a teamwork environment
- Some other restrictions are added in some specific modeling case

Checking editing permission

Use *com.nomagic.magicdraw.uml.BaseElement.isEditable()*[555] to check if Element's properties can be changed.

Editing permissions should be checked if primitive Element properties are modified (like a name, for example) or references are modified (like a Property type).

Check the adding permission if you are going to modify some containment property (like *Element.ownedElement*).

Checking adding permission

Use *BaseElement.canAddChild()*[556] to check if children can be added into the Element.

---

[552] http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/openapi/uml/SessionManager.html#createSession-com.nomagic.magicdraw.core.Project-java.lang.String-
[553] http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Project.html
[554] http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/openapi/uml/SessionManager.html#createSession-com.nomagic.magicdraw.core.Project-java.lang.String-
[555] http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/BaseElement.html#isEditable--
[556] http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/BaseElement.html#canAddChild--

Use *BaseElement.canAdd(com.nomagic.magicdraw.uml.BaseElement)*[557] to check if a parent can own a given element and a parent has permissions to own. This method will fail, for example, if you will try to add an Operation into a Package.

Checking removing permission

Use BaseElement.isEditable()[558] to check if an Element can be removed (disposed) from the model.

Checking moving permission

The moving permission is similar to the adding permission, but checks more conditions.

Plugging custom permissions handler

Use *com.nomagic.magicdraw.uml.permissions.ElementPermissionsManager.addPermissionsHandler(*[559] ElementPermissions[560]*)* to register a custom element permissions handler. Using this API you can handle custom rules for putting a permission on the top of some specific elements.

Accessing and modifying model element properties

Model element properties can be accessed with simple setters, getters:

```
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.NamedElement el = ...;
String name = el.getName();
el.setName("new name");
```

Container Properties

Modeling tools developed by No Magic Inc. uses a composite structure of the model.

Every model element is a container and contains its own children and knows about its own parent.

A model element parent can be accessed with the *com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element.getOwner()*[561] method. Owned children can be received with the Element.getOwnedElement()[562] method. Different types of children are stored in separate container properties.

---

557 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/BaseElement.html#canAdd-com.nomagic.magicdraw.uml.BaseElement-
558 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/BaseElement.html#isEditable--
559 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/permissions/ElementPermissionsManager.html#addPermissionsHandler-com.nomagic.magicdraw.uml.permissions.ElementPermissions-
560 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/permissions/ElementPermissions.html
561 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html#getOwner--
562 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html#getOwnedElement--

You can access these container properties by names that are described in the UML specification. The *getOwnedElement()*[563]method collects all children from all inner container properties.

The container properties modification and iteration is straightforward using the *java.util.Collection* interface. Property change events are fired automatically when container properties are modified.

Containers implement subsets and unions constraints from the UML metamodel specification. This explains how the modification of one container can affect other containers. Make sure you understand subsets and unions in the UML metamodel. If you want to add some inner Element to the union collection, you need to add it into a specific subset of *union.*

Some containers are read-only. This is true for the most of DERIVED UML metamodel properties.

Some derived properties are editable. For example, *Element.ownedElement* is editable.

It is enough to set one UML meta-association property value and an opposite property will be set too. For example, adding a *Class* into a *Package* can be done in two ways:

```
Class myClass= ...;
Package myPackage ...;
myClass.setOwner(myPackage);
```

or

```
myPackage.getOwnedElement().add(myClass);
```

Accessing elements in container properties

The following example illustrates retrieving children of model elements:

```
Element el = ...;
for(Element element : el.getOwnedElement())
{
    // work with element
}
```

Modifying elements in container

Use standard *java.util.Collection or java.util.List* methods:

563 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html#getOwnedElement--

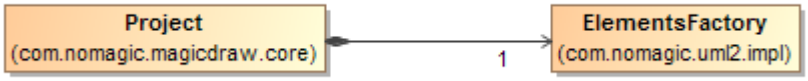
```
modelElement.get<SomeContainer>().add(child);
modelElement.get<SomeContainer>().remove(child);
```

Related pages

- Session management(see page 2224)
- Checking element editing permissions(see page 2225)

Creating new model elements

Use the com.nomagic.uml2.impl.ElementsFactory[564] class for creating model elements. To create a model element, a session with the com.nomagic.magicdraw.openapi.uml.SessionManager[565] must be created.



The *create<model element type>Instance()* method creates a new model element instance. The figure above shows only a subset of all available *create<...>Instance()* methods.

All changes in the UML model will be registered and, on the session closing, will be added into the command history.

```
Project project =...;
ElementsFactory f = project.getElementsFactory();
SessionManager.getInstance().createSession(project, "Create a package");
Package packageA = f.createPackageInstance();
//add created package into a root of the project
packageA.setOwner(project.getPrimaryModel());
...
// apply changes and add a command into the command history.
SessionManager.getInstance().closeSession(project);
```

You can find the code examples in *<modeling tool installation directory>\openapi\examples\accessors*

Related pages

- Session management(see page 2224)
- Creating a diagram(see page 2276)

564 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/impl/ElementsFactory.html
565 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/openapi/uml/SessionManager.html

Creating new relationship objects

A model element is a relationship, if it implements one of the following interfaces:

- com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Relationship[566],
- *com.nomagic.uml2.ext.magicdraw.activities.mdbasicactivities.ActivityEdge*[567],
- *com.nomagic.uml2.ext.magicdraw.statemachines.mdbehaviorstatemachines.Transition*[568],
- *com.nomagic.uml2.ext.magicdraw.activities.mdextrastructuredactivities.ExceptionHandler*[569],
- *com.nomagic.uml2.ext.magicdraw.compositestructures.mdinternalstructures.Connector*[570].

For checking if the model element is a relationship, call the com.nomagic.uml2.ext.jmi.helpers.CoreHelper.isRelationship([571]Element[572])[573] method.

For getting supplier and client elements of a relationship, use *CoreHelper*[574] *class* getSupplierElement([575]Element[576])[577], getClientElement([578]Element[579])[580] methods.

---

566 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Relationship.html
567 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/activities/mdbasicactivities/ActivityEdge.html
568 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/statemachines/mdbehaviorstatemachines/Transition.html
569 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/activities/mdextrastructuredactivities/ExceptionHandler.html
570 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/compositestructures/mdinternalstructures/Connector.html
571 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/CoreHelper.html#isRelationship-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-
572 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
573 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/CoreHelper.html#isRelationship-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-
574 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/CoreHelper.html
575 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/CoreHelper.html#getSupplierElement-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-
576 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
577 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/CoreHelper.html#getSupplierElement-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-
578 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/CoreHelper.html#getClientElement-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-
579 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
580 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/CoreHelper.html#getClientElement-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-

Use *CoreHelper*[581] *class setSupplierElement(*[582]Element[583]*,* Element[584]*)* , setClientElement([585]Element[586], [587]Element[588])[589] methods to set supplier and client elements of a relationship.

To create a new Relationship

---

1. Create a new relationship model element.
2. Set client and supplier model elements by using *CoreHelper*[590] class *setSupplierElement(*[591]Element[592]*,* Element[593]*)*, setClientElement([594]Element[595], [596]Element[597])[598] methods.
3. Add a new relationship into some parent by using the *com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element.setOwner(*[599]Element[600]*)* method.

---

581 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/CoreHelper.html
582 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/CoreHelper.html#setSupplierElement-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-
583 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
584 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
585 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/CoreHelper.html#setClientElement-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-
586 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
587 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/CoreHelper.html#setClientElement-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-
588 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
589 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/CoreHelper.html#setClientElement-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-
590 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/CoreHelper.html
591 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/CoreHelper.html#setSupplierElement-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-
592 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
593 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
594 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/CoreHelper.html#setClientElement-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-
595 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
596 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/CoreHelper.html#setClientElement-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-
597 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
598 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/CoreHelper.html#setClientElement-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-
599 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html#setOwner-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-
600 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html

Related pages

- Session management(see page 2224)
- Checking element editing permissions(see page 2225)

Editing model elements

To edit some *com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element*[601], a session with the *com.nomagic.magicdraw.openapi.uml.SessionManager*[602] must be created.

All changes in the UML model will be registered and, on the session closing, will be added into the command history.

```
Project project = ...;
SessionManager.getInstance().createSession(project, "Edit class A");
if (classA.isEditable())
{
    classA.setName(newName);
}
SessionManager.getInstance().closeSession(project);
```

It is the programmer responsibility to ensure that the modified element is not read-only by checking element editing permissions.

Related pages

- Session management(see page 2224)
- Checking element editing permissions(see page 2225)

Adding, moving, deleting model elements

There are two ways to add/move/delete model elements:

- Use com.nomagic.magicdraw.openapi.uml.ModelElementsManager[603]
- Call direct functions on *Elements*

The *ModelElementsManager*[604] is the singleton utility class which checks *com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element*[605] editing permissions and the session existence before executing the function. Also, it performs a check, if an element can be added to a parent. If the *ModelElementsManager* [606]is not used, a programmer must perform these checks in the code explicitly.

601 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
602 http://com.nomagic.magicdraw.openapi.uml.SessionManager
603 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/openapi/uml/ModelElementsManager.html
604 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/openapi/uml/ModelElementsManager.html
605 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
606 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/openapi/uml/ModelElementsManager.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Adding and moving functions are similar with mostly one - move functions does more checking such as checking for a name conflict if *com.nomagic.uml2.ext.magicdraw.classes.mdkernel.NamedElement*[607] is moved. The general rule would be to use an adding function for newly created *Element*[608]*(s)* and use a moving function for already existing elements in the model.
More advanced model refactoring functions are described in the Refactoring model elements(see page 2232) page.

Adding element to owner with *ModelElementsManager*

For adding a new model *Element*[609]into a model, use the *addElement(*[610]Element[611], Element[612]*)* method provided by the *ModelElementsManager*[613].

```
Project project = ...;
Class classA = ...;
Package package = ...;
// create a new session
SessionManager.getInstance().createSession(project, "Add class into package");
try
{
    // add a class into a package
    ModelElementsManager.getInstance().addElement(classA, package);
}
catch (ReadOnlyElementException e)
{
}
// apply changes and add a command into the command history.
SessionManager.getInstance().closeSession(project);
```

If a given model element cannot be added into a given parent, *java.lang.IllegalArgumentException* is thrown. For example, an *com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Operation*[614] cannot be added into a *com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Package*[615] or an *Operation*[616] cannot be added into a not locked for editing *com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class*[617] (in the server project). If an element or parent is null, *java.lang.IllegalArgumentException* also is thrown. If a given element is not editable (read-only), *com.nomagic.magicdraw.openapi.uml.ReadOnlyElementException*[618] is thrown.

607 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/NamedElement.html
608 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
609 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
610 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/openapi/uml/ModelElementsManager.html#addElement-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-
611 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
612 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
613 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/openapi/uml/ModelElementsManager.html
614 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Operation.html
615 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Package.html
616 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Operation.html
617 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Class.html
618 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/openapi/uml/ReadOnlyElementException.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Adding element to owner using a direct function

```
Element parent = ...;
Class classA = ...;
Project project = ...;
// create a new session
SessionManager.getInstance().createSession(project, "Add class into parent");
if (parent.canAdd(classA))
{
    classA.setOwner(parent);
}
// apply changes and add a command into the command history.
SessionManager.getInstance().closeSession(project);
```

Moving element with *ModelElementsManager*

For moving an existing model *Element*[619], use the *moveElement(*[620]Element[621], Element[622]*)* method provided by the *ModelElementsManager*[623].

```
Project project = ...;
Class classA = ...;
Package package = ...;
// create a new session
SessionManager.getInstance().createSession(project, "Add class into package");
try
{
    // add a class into a package
    ModelElementsManager.getInstance().moveElement(classA, package);
}
catch (ReadOnlyElementException e)
{
}
// apply changes and add a command into the command history.
SessionManager.getInstance().closeSession(project);
```

If a given model element cannot be moved into a given parent, *java.lang.IllegalArgumentException*is thrown. For example, an *Operation*[624] cannot be moved into a *Package*[625] or an *Operation*[626] cannot be moved into a not locked for editing *Class*[627] (in the server project). If an element or parent is null,

619 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
620 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/openapi/uml/ModelElementsManager.html#moveElement-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-
621 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
622 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
623 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/openapi/uml/ModelElementsManager.html
624 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Operation.html
625 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Package.html
626 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Operation.html
627 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Class.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

java.lang.IllegalArgumentException also is thrown. If a given element is not editable (read-only), *com.nomagic.magicdraw.openapi.uml.ReadOnlyElementException*[628] is thrown.

Moving element using a direct function

```
    Element parent = ...;
    Class classA = ...;
    Project project = ...;
    // create a new session
    SessionManager.getInstance().createSession(project, "Add class into parent");
    if (com.nomagic.uml2.ext.jmi.helpers.ModelHelper.canMoveChildInto(parent,
classA))
    {
        classA.setOwner(parent);
    }
    // apply changes and add a command into the command history.
    SessionManager.getInstance().closeSession(project);
```

Removing element with *ModelElementsManager*

```
    Class classA = ...;
    Project project = ...;
    // create a new session
    SessionManager.getInstance().createSession(project, "Remove class");
    try
    {
         // remove a class
        ModelElementsManager.getInstance().removeElement(classA);
    }
    catch (ReadOnlyElementException e)
    {
    }
    // apply changes and add a command into the command history.
    SessionManager.getInstance().closeSession(project);
```

628 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/openapi/uml/ReadOnlyElementException.html

Removing element using direct call

```
Class classA = ...;
Project project = ...;
// create a new session
SessionManager.getInstance().createSession(project, "Remove class");
if (classA.isEditable()))
{
     classA.dispose();
}
// apply changes and add a command into the command history.
SessionManager.getInstance().closeSession(project);
```

You can find the code examples in *<modeling tool installation directory>\openapi\examples\hierarchyremover*

On this page

- Adding element to owner with ModelElementsManager(see page 2232)
- Adding element to owner using a direct function(see page 2233)
- Moving element with ModelElementsManager(see page 2233)
- Moving element using a direct function(see page 2234)
- Removing element with ModelElementsManager(see page 2234)
- Removing element using direct call(see page 2235)

Related pages

- Session management(see page 2224)
- Checking element editing permissions(see page 2225)
- Refactoring model elements(see page 2237)

Refactoring model elements

To refactor an element in a model, use the *com.nomagic.magicdraw.uml.Refactoring*[629] class.

629 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/Refactoring.html

Example #1: Converting an element to an interface

```
    Element elementToConvert = ...;
    Project project = ...;
    SessionManager sessionManager = SessionManager.getInstance();
    sessionManager.createSession(project, "Convert");

    // Converts the element to an interface.
    ConvertElementInfo info = new ConvertElementInfo(Interface.class);

    // Preserves the old element ID for the new element.
    info.setPreserveElementID(true);
    Element conversionTarget = Refactoring.Converting.convert(elementToConvert,
info);
    sessionManager.closeSession(project);
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example #2: Replacing an element with another element

```
    Element elementToReplace = ...;
    Project project = ...;
    SessionManager sessionManager = SessionManager.getInstance();
    sessionManager.createSession(project, "Replace");

    ConvertElementInfo info = new
 ConvertElementInfo(elementToReplace.getClassType());
    info.setConvertOnlyIncomingReferences(true);

    Refactoring.Replacing.replace(element, elementToReplace, info);

    sessionManager.closeSession(project);
```

Example #3: Extract refactoring

Use the *com.nomagic.magicdraw.uml.Refactoring.Extracting*[630] class to create the extract manager for a symbol which you want to extract. Configure the extract refactoring by changing *com.nomagic.magicdraw.uml.refactor.extract.ExtractSource*[631] and *com.nomagic.magicdraw.uml.refactor.extract.ExtractTarget*[632]. Invoke the refactoring with com.nomagic.magicdraw.uml.refactor.extract.ExtractManager.extract()[633]. Review refactoring results by inspecting *ExtractSource*[634] and *ExtractTarget*[635].

630 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/Refactoring.Extracting.html
631 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/refactor/extract/ExtractSource.html
632 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/refactor/extract/ExtractTarget.html
633 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/refactor/extract/ExtractManager.html#extract--
634 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/refactor/extract/ExtractSource.html
635 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/refactor/extract/ExtractTarget.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

```
    // Creates an extract refactor manager.
    ExtractManager extractManager =
Refactoring.Extracting.createExtractManager(symbols);
    if (extractManager != null)
    {
        Project project = ...;
        // A session has to be started before refactoring.
        SessionManager sessionManager = SessionManager.getInstance();
        sessionManager.createSession(project, "Extract Refactor Symbols");

        // We may control the extract refactor result by modifying extract target.
        ExtractTarget extractTarget = extractManager.getExtractTarget();

        // Create a namespace to which we are going to refactor.
        Project project = Project.getProject(symbols[0]);
        Package package1 = project.getElementsFactory().createPackageInstance();
        package1.setOwner(project.getPrimaryModel());


        // Set the namespace to which the extract result should go.
        extractTarget.setTargetNamespace(package1);

        // Choose target diagram type from allowed diagram types if the default type
does not suite.
        List<String>  allowedTargetDiagramTypes =
extractTarget.getAllowedTargetDiagramTypes();
        extractTarget.setTargetDiagramType(allowedTargetDiagramTypes.get(0));

        // Modify reference names which link the extract refactor source to the
target.
        List<? extends ExtractReference> references = extractTarget.getReferences();

        for (int i = 0; i < references.size(); i++)
        {
            ExtractReference reference = references.get(i);
            reference.setName(Integer.toString(i));
        }

        // We may control the extract refactor source by modifying the extract
source.
        ExtractSource extractSource = extractManager.getExtractSource();
        extractSource.setElementName("sourceElementName");


        // Perform actual refactoring.
        extractManager.extract();
        sessionManager.closeSession(project);
```

```
        // The element which was created in the source during refactoring.
        Element sourceElement = extractSource.getElement();


        // The element which was created in the target during refactoring.
        Element targetElement = extractTarget.getElement();


        // The diagram which was created in the target during refactoring.
        DiagramPresentationElement targetDiagram = extractTarget.getDiagram();
    }
```

Example #4: Reverse relationship refactoring

The relation reverse refactoring can be done using the *com.nomagic.magicdraw.uml.Refactoring.RelationReversing.reverseRelationDirection(*[636]Element[637]*)* method.

```
     // We have an arbitrary element or symbol which represents a relationship.
    BaseElement baseElement = ...;
    Project project = ...;
    // Relationship reversing should be wrapped with session create/close calls.
    SessionManager sessionManager = SessionManager.getInstance();
    sessionManager.createSession(project, "Reverse relation");

    // Reverse the relationship.
    Refactoring.RelationReversing.reverseRelationDirection(baseElement);

    // Close the session.
    sessionManager.closeSession(project);
```

Example #5: Moving element with connected relationships to other owner

Use *com.nomagic.magicdraw.uml.Refactoring.Moving.moveElementsWithRelation(java.util.Collection<*[638]Element[639]*>,* Element[640]*)* to move *com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element*[641] and all connected com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Relationship[642]*(s)* to a new owner.

> You can find the code examples in *<modeling tool installation directory>\openapi\examples\refactor*

---

63 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/Refactoring.RelationReversing.html#reverseRelationDirection-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-
637 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
63 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/Refactoring.Moving.html#moveElementsWithRelations-java.util.Collection-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-
639 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
640 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
641 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
642 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Relationship.html

Related pages

- Session management(see page 2224)
- Checking element editing permissions(see page 2225)

Copying elements and symbols

You can copy model elements and symbols either to another location in the same project or to another project. This must be done in the same session.

There are two modes of making copies:

- Deep copying (with new data)
- Shallow copying (with reused data)

Example #1: Copying an element

```
    Element element = ...; // An element to copy/paste.
    Element parent = ...; // A parent to which the element has to be pasted: either
the same project or another project.
    Project project = ...;
    SessionManager sessionManager = SessionManager.getInstance();
    sessionManager.createSession(project, "Clone");

    // A 3rd parameter indicates whether an element name uniqueness should be
preserved in the parent.
    CopyPasting.copyPasteElement(element, parent, true);

    sessionManager.closeSession(project);
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example #2: Copying multiple elements and symbols

```
    List elements = ...; // Elements to copy/paste.
    List views = ...; // Symbols to copy/paste.
    Element parent = ...; // A parent to which elements should be pasted: either the
same project or another project.
    BaseElement symbolParent = ...; // A parent to which symbols should be pasted.
    Project project = ...;
    SessionManager sessionManager = SessionManager.getInstance();
    sessionManager.createSession(project, "Clone");


    // A 4th parameter indicates whether deep or shallow copy is applied.
    // A 5th parameter indicates whether an element name uniqueness should be
preserved in the parent.
    List baseElements = CopyPasting.copyPasteElements(views, parent, symbolParent,
true, true);

    sessionManager.closeSession(project);
```

You can find the code examples in *<programinstallation directory>\openapi\examples\copypaste*

Related pages

- Session management(see page 2224)
- Checking element editing permissions(see page 2225)

Identifying elements

Two methods provide an identifier of an *com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element*. Either method can be used depending on the situation. This section explains the difference between the two:

1. com.nomagic.uml2.ext.magicdraw.base.ModelObject.getLocalID()[643]
2. com.nomagic.magicdraw.foundation.MDObject.getID()[644]

The meaning of the returned identifiers depends on the project type.

Local (.mdzip) and Teamwork Server projects

Both methods return the same identifier, which is permanent and unique in the scope of the project. In most cases, this ID is unique among different projects, unless you create a copy of the project by copying the file or using the **Save As** command.

643 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/base/ModelObject.html#getLocalID--
644 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/foundation/MDObject.html#getID--

Examples

1. If a project *A* is used in projects *B* and *C*, then the elements of the project *A* have the same identifiers in both projects *B* and *C*.
2. If a project *A* is copied as a project *A1*, and a project *B* uses the project *A*, using the project *A1* additionally will not be allowed since IDs must remain unique in the scope of the project *B*.

Teamwork Cloud (TWC) projects

If a local project is added to the Teamwork Cloud (or vice versa), getLocalID()[645] consistently returns the same identifier value. Note that several elements with the same local ID may exist in the scope of a single TWC project.

getID()[646] returns UUID, which is unique both in the project and in the scope of the whole TWC server repository.

The Teamwork Cloud imports (clones) the contents of used projects. This means copies of the elements are created inside the project. While getLocalID()[647] returns the same value for different copies, getID()[648] will return a unique value for each copy (project usage).

Example

If a project *A* is copied as a project *A1*, and a TWC project *B* uses both *A* and *A1*, getLocalID()[649] returns the same value for corresponding elements of *A* and *A1*, but getID()[650] returns different identifiers for the same elements.

Tip

Both identifiers can be used to retrieve the element using the method com.nomagic.magicdraw.core.Project.getElementByID(java.lang.String)[651]. Be aware that if several elements exist in the project by the provided local ID, only a single one of them is returned.

Related pages

- Working with UML model(see page 2224)

645 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/base/ModelObject.html#getLocalID--
646 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/foundation/MDObject.html#getID--
647 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/base/ModelObject.html#getLocalID--
648 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/foundation/MDObject.html#getID--
649 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/base/ModelObject.html#getLocalID--
650 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/foundation/MDObject.html#getID--
651 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Project.html#getElementByID-java.lang.String-

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Finding elements by name or by meta-type

Use *com.nomagic.magicdraw.uml.Finder*[652] for finding elements by the name or type (meta-type) in a project. This utility class provides several internal finders for executing different types of search operations.

For example:

```
   Project project = ...;
   //find a Model element under the project primary model with a name "MyModel". It
is not a recursive search
   Model model = Finder.byName().find(project.getPrimaryModel(), Model.class,
"MyModel");
   //find a first Model element under project primary model with a name "MyModel". It
is a recursive search.
   Model model = Finder.byNameRecursively().find(project.getPrimaryModel(), Model.cla
ss, "MyModel");
   //find all Model elements under project primary model with a name "MyModel". It is
a recursive search.
   Collection<Model> models =
Finder.byNameAllRecursively().find(project.getPrimaryModel(), Model.class, "MyModel")
;
```

Finding or creating Classifiers

A special utility class is available for search or creating Classifiers by metatype, simple or qualified name - com.nomagic.magicdraw.uml.ClassifierFinder[653]

Related pages

- Retrieving meta information about element(see page 2243)

Model traversing, Visitor pattern

Here is an example of how to collect all children from the *com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element*[654] and avoid recursion using a simple *for* cycle:

652 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/Finder.html
653 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/ClassifierFinder.html
654 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

```
List<Element> children = new ArrayList<Element>();
Element current = ...
children.add(current);
// if the current element has children, list will be increased.
for (int i = 0; i < children.size(); i++)
{
    current = children.get(i);
    // add all children into end of this list, so it emulates a recursion.
    children.addAll(current.getOwnedElement());
}
```

It is highly recommended to use more advanced collecting methods from *com.nomagic.magicdraw.uml.Finder*[655] utility:

- byScope()[656]
- byType()[657]
- byTypeRecursively()[658]

Related pages

Visitors

Visitor design pattern is implemented in UML model structure. Every *com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element*[659] has the *accept(*[660]AbstractVisitor[661]*)* method for visiting it.

The *com.nomagic.uml2.impl.ModelVisitor*[662] has the *visit<element_metatype>* method for all types of model elements. This is very useful when you are working with a large collection of elements and need to perform actions, specific for every type of *Element*[663](for example, save/load, copy/paste, or a specific properties setting).

Just derive your class from *com.nomagic.magicdraw.uml.InheritanceVisitor*[664] and override some *visit...* methods and call *accept(*[665]AbstractVisitor[666]*)* for Elements[667].

655 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/Finder.html
656 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/Finder.html#byScope--
657 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/Finder.html#byType--
658 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/Finder.html#byTypeRecursively--
659 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
660 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/BaseElement.html#accept-com.nomagic.magicdraw.uml.AbstractVisitor-
661 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/AbstractVisitor.html
662 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/impl/ModelVisitor.html
663 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
664 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/InheritanceVisitor.html
665 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/BaseElement.html#accept-com.nomagic.magicdraw.uml.AbstractVisitor-
666 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/AbstractVisitor.html
667 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html

```
ModelVisitor myVisitor = new ModelVisitor()
{
    // override some visit methods ...
    public void visitClass(Class element, VisitorContext context)
    {
        //this is my UML Class
    }
};
Project project = ...;
Package root = project.getPrimaryModel();
Iterator<Element> it = root.getOwnedElement().iterator();
while (it.hasNext())
{
    it.next().accept(myVisitor);
}
```

You can find the code examples in *<programinstallation directory>\openapi\examples\statistics*

ModelHierarchyVisitor

The com.nomagic.uml2.impl.ModelHierarchyVisitor[668] is an enhanced implementation of the *Visitor* pattern and a subclass of com.nomagic.uml2.impl.ModelVisitor[669], used for visiting model elements.

Every *visit<element_metatype>* method calls a *visit<element_metatype>* method for a super type.

Model elements *visit...* methods have an additional context parameter of the *com.nomagic.uml2.ext.jmi.reflect.VisitorContext*[670] type. The visitor context is used to track what super type *visit<element_metatype>* methods were already visited (to avoid multiple visits because some model elements have a multiple inheritance). Open API users should not work with the visitor context. All tracking is done in internal implementation.

For example, you can put some code into *com.nomagic.uml2.impl.ModelHierarchyVisitor.visitClassifier(*[671]Classifier[672]*)* and it will be executed for all subclasses of a *com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Classifier*[673].

668 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/impl/ModelHierarchyVisitor.html
669 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/impl/ModelVisitor.html
670 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/reflect/VisitorContext.html
671 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/impl/ModelHierarchyVisitor.html#visitClassifier-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Classifier-com.nomagic.uml2.ext.jmi.reflect.VisitorContext-
672 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Classifier.html
673 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Classifier.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

```
ModelVisitor myVisitor = new ModelHierarchyVisitor()
{
    public void visitClassifier(Classifier element, VisitorContext context)
    {
        //this is my classifier
    }
};
Project project = ...;
Package root = project.getPrimaryModel();
Iterator<Element> it = root.getOwnedElement().iterator();
while (it.hasNext())
{
    it.next().accept(myVisitor);
}
```

You can find the code examples in *<programinstallation directory>\openapi\examples\statistics*

Retrieving meta information about element

Every Element is identified by it's "metaclass". Element metaclass is mapped into java interface (for example *com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class*[674]).

Use *com.nomagic.magicdraw.uml.BaseElement.getClassType()*[675] to retrieve the element metaclass.

Use *com.nomagic.magicdraw.uml.ClassTypes*[676] for managing the metaclasses - collecting all available, collecting derived and etc.

Creating textual element representation

Use *com.nomagic.magicdraw.uml.RepresentationTextCreator*[677] to create a textual representation of Element. This utility class is used for creating a textual element representation in various UI - trees, list and etc.

```
   Element element = ...
   //create a preformatted text for any kind of element
   String text =RepresentationTextCreator.getRepresentedText(element);

   //create a text with special options for
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Property
   Property property = ...;
   String propertyString = RepresentationTextCreator.createPropertyText(property,
true, true, true, true, true, false, false, false, true, false);
```

674 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Class.html
675 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/BaseElement.html#getClassType--
676 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/ClassTypes.html
677 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/RepresentationTextCreator.html

Providing a custom representation text

You can override default custom text creation for the specific element.

Use *RepresentationTextCreator*[678]class method *addProvider(*[679]RepresentationTextCreator.RepresentationTextProvider[680]*)* to register a custom representation text provider.

Retrieving element icon

Use com.nomagic.magicdraw.uml.ElementIcon[681] utility class for retrieving element icon. It also provides methods for retrieving icons by element metatype, stereotype and etc.

```
   Element element = ...;
   //get element icon
   ResizableIcon icon = ElementIcon.getIcon(element);

   //get UML Package icon
   ResizableIcon packageIcon =
ElementIcon.getIconByClassType(com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Packag
e.class);
```

Related pages

- Retrieving meta information about element(see page 2246)
- Working with icons(see page 2205)

Advanced utility functions

Use *com.nomagic.uml2.ext.jmi.helpers.ModelHelper*[682] for various advanced UML model related functions.

It provides utility methods for

- Some core functions
  - working with element's documentation (Comments)
  - getting/setting Relationships client and supplier ends
  - working with Associations - getting member ends, changing navigability

678 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/RepresentationTextCreator.html
679 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/RepresentationTextCreator.html#addProvider-com.nomagic.magicdraw.uml.RepresentationTextCreator.RepresentationTextProvider-
680 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/RepresentationTextCreator.RepresentationTextProvider.html
681 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/ElementIcon.html
682 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/ModelHelper.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

    - working with multiplicity of MultiplicityElements
    - ownership related functions - looking for valid owners, checking ownership and etc.
- Working with Value Specifications
    - Creating value specifications
    - Setting value to Value specifications
- Working with Classifiers
    - collecting derived classifiers, checking legal inheritance
    - collecting inherited members
    - working with redefined elements
    - checking Classifiers compatibility
    - working with Operation parameters, comparing Operations signatures
- Working with InstanceSpecifications
    - creating Slots, initializing slots values
    - working with Links

Use com.nomagic.uml2.ext.jmi.helpers.InformationFlowHelper[683] for advanced Information Flows related functions

Use com.nomagic.uml2.ext.jmi.helpers.InteractionHelper[684] for Interaction domain related advanced functions

Use *com.nomagic.uml2.ext.jmi.helpers.StateMachineHelper*[685] for State Machine related advanced functions

Use *com.nomagic.uml2.ext.jmi.helpers.UseCaseHelper*[686] for Use Case related advanced functions

Use *com.nomagic.magicdraw.uml.ConnectorsCollector*[687] to collect connected Connector elements

Event support

Modeling tools developed by No magic Inc. provides the API for listening to the events, while changing a model. There is a possibility either to get an event immediately after the property has been changed, or get the event about all the changes in the session, after this session has been closed.

There are four different listener registration types:

- **The whole repository listener.** You will get events about the changes in all the elements.
- **The specific element listener.** You will get events about the changes in any property of this element.
- **The element's specific property listener.** You will get events about the changes of this property.
- **The specific element type listener.** You will get events about the changes in all the elements of the specific metatype.

The transaction listener is notified, when all the changes within the session are done and the session is closed.

683 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/InformationFlowHelper.html
684 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/InteractionHelper.html
685 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StateMachineHelper.html
686 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/UseCaseHelper.html
687 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/ConnectorsCollector.html

You can find the code examples in *<program installation directory>\openapi\examples\events*.

Related pages

Property change event concept

UML model uses standard *java.beans.PropertyChangeEvent* events for notifying listeners about changes.

The main properties of the *PropertyChangeEvent* are as follows:

- **Property name.** The changed property name.
- **New value.** The new (current) changed property value.
- **Old value.** The old property value, which was before the property change.

Properties names are described in UML model properties names(see page 2249) page

Example of the element's name change event

```
propertyChangeEvent.getPropertyName().equals(PropertyNames.NAME);
propertyChangeEvent.getNewValue() // will be new name of element;
propertyChangeEvent.getOldValue() // will be name of element before change.
```

Example of the new element creation event

```
propertyChangeEvent.getPropertyName().equals(UML2MetamodelConstants.INSTANCE_CREATED)
;
    propertyChangeEvent.getNewValue() // will be new created element;
    propertyChangeEvent.getOldValue() // will be null.
```

Related pages

- UML model properties names(see page 2249)

UML model properties names

Constants for UML model properties names are defined in several classes.

Use *com.nomagic.uml2.impl.PropertyNames*[688] for standart UML metamodel property names.

Use com.nomagic.uml2.impl.PrivatePropertyNames[689] for auto-generated opposite property names.

The events' names related with model creation/ deletion are listed in the *com.nomagic.uml2.ext.jmi.UML2MetamodelConstants*[690] class.

Listening to property change events

The *PropertyChangeEvents* support provides the ability to listen to the event by different scopes. The scope depends on how the listener is registered:

- If the listener is registered at the whole repository, it receives events about the changes in all the elements.
- If the listener is registered at the specific element, it receives events about the changes in any property of this element.
- If the listener registered with the specific property at the element, it receives events about the changes in this property.
- If the listener is registered with the specific property for the element type, it receives events about the changes in all the elements of the specific type.

The *java.beans.PropertyChangeListener* should be registered to receive these events. There are several different ways to register a listener:

- To listen for the whole repository, use:

```
        project.getRepositoryListenerRegistry().addPropertyChangeListener(listener,
(RefObject)null);
```

This listener will get all the property change events from any element.

- To listen for any delete operation in the repository and get notified before the delete action is performed, use:

```
        project.getRepositoryListenerRegistry().addPropertyChangeListener(listener,
UML2MetamodelConstants.BEFORE_DELETE);
```

This listener will be notified, when any model element is set to be deleted.

688 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/impl/PropertyNames.html
689 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/impl/PrivatePropertyNames.html
690 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/UML2MetamodelConstants.html

- To listen for any property changes of the specific element, use:

```
element.addPropertyChangeListener(listener);
```

This listener will be notified when any element property is changed.

- To listen for any property changes of the specific element type, use:

```
project.getSmartEventSupport().registerConfig(aClass, configs, listener);
```

This listener will be notified, when any property of any element in the project with this type is changed. If "configs" is NULL, the listener will get all property change events.

EventSupport could be disabled from the event firing:

- To check if it is enabled, use:

```
project.getRepository().getEventSupport().isEnableEventFiring()
```

- To stop/ start event firing, use:

```
project.getRepository().getEventSupport().setEnableEventFiring(...)
```

These examples show, how to create the property change listeners to listen to the different kind of properties.

Listener for listening to the specific element's any property changes:

```
element.addPropertyChangeListener(new PropertyChangeListener()
{
    public void propertyChange(PropertyChangeEvent evt)
    {
        // evt.getPropertyName() is changed
    }

});
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Listener for listening to the specific property (NAME) of the specific element:

```
    element.addPropertyChangeListener(new PropertyChangeListener()
    {
        public void propertyChange(PropertyChangeEvent evt)
        {
            if (PropertyNames.NAME.equals(evt.getPropertyName()))
            {
                // name is Changed
            }
        }
    });
```

Listening to related elements in hierarchy events

There is a possibility to listen for changes of the elements, which are somehow related with the given element. For example, we want to be notified, when the element's owner name is changed. The *com.nomagic.uml2.ext.jmi.smartlistener.SmartPropertyChangeListener*[691] class is dedicated for the situations like this. The main idea of the solution is to provide the *com.nomagic.uml2.ext.jmi.smartlistener.SmartListenerConfig*[692], which will provide the chain of the property names to listen to.

The *SmartListenerConfig*[693]class provides static methods for the default configuration of property chains. This is useful, when listening for common property change events.

Use the SmartPropertyChangeListener[694] class *createSmartPropertyListener(...)* method to create and register such listener for the particular element, which will be notified with events provided by the *SmartListenerConfig*[695].

> ⚠ If the listener is not needed anymore, unregister it using the *removePropertyChangeListener(...)* m ethod.

Example #1. Listen to the element owner's name

691 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/smartlistener/SmartPropertyChangeListener.html
692 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/smartlistener/SmartListenerConfig.html
693 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/smartlistener/SmartListenerConfig.html
694 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/smartlistener/SmartPropertyChangeListener.html
695 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/smartlistener/SmartListenerConfig.html

```
        Element element = ...; // some element

        SmartListenerConfig config = new SmartListenerConfig();
        config.listenTo(PropertyNames.NAME); // listen to the element's name
        config.listenToNested(PropertyNames.OWNER).listenTo(PropertyNames.NAME); //
listen to the element owner's name

        // create the listener
        SmartPropertyChangeListener.createSmartPropertyListener(element, new
 PropertyChangeListener()
        {
            public void propertyChange(PropertyChangeEvent evt)
            {
               // the change event comes here
            }
        }, config);
```

Example #2. Listen recursively to the owner's name and the "*Is Abstract*" property

```
        Element element = ...; // some element

        SmartListenerConfig config = new SmartListenerConfig();
        config.listenTo(PropertyNames.NAME);
        config.listenTo(PropertyNames.IS_ABSTRACT);
        config.listenTo(PropertyNames.OWNER, config);

        SmartPropertyChangeListener.createSmartPropertyListener(element, new
 PropertyChangeListener()
        {
            public void propertyChange(PropertyChangeEvent evt)
            {
                // the change event comes here
            }
        }, config);
```

Listening to transaction commit events on session closing

Every session creating and closing starts and commits the model editing transaction. The *com.nomagic.uml2.transaction.TransactionCommitListener*[696] is a special listener, which is notified when the all changes inside a transaction are done and the transaction is closed.

The *TransactionCommitListener*[697] contains *transactionCommited(java.util.Collection<java.beans.PropertyChangeEvent>)*[698] method, which provides a collection of all *java.beans.PropertyChangeEvent(s)* that were executed in a transaction.

696 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/transaction/TransactionCommitListener.html
697 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/transaction/TransactionCommitListener.html
698 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/transaction/TransactionCommitListener.html#transactionCommited-java.util.Collection-

Create a custom transaction commit listener:

```
    public class MyTransactionListener implements TransactionCommitListener
    {
        public Runnable transactionCommited(final Collection<PropertyChangeEvent>
events)
        {
            return new Runnable()
            {
                public void run()
                {
                    for (PropertyChangeEvent event : events)
                    {
                        if
 (UML2MetamodelConstants.INSTANCE_CREATED.equals(event.getPropertyName()))
                        {
                            Object source = event.getSource();
                            if (source instanceof Property)
                            {
                                Property property = (Property) source; Element owner
= property.getOwner();
                                if (owner instanceof Classifier)
                                {
                                    Classifier propertyOwner = (Classifier) owner;
                                    propertyOwner.setName("Contains (" +
propertyOwner.getAttribute().size() + ") attributes");
                                }


                                // additionally for this Property we register
listener to listen for any property changes in this Element properties.
                                property.addPropertyChangeListener(new
 DerivedValuePropertyChangeListener());
                            }
                        }
                    }
                }
            };
        }
    }
```

This “transaction commit listener” checks, if a new property is created in a classifier and updates the classifier’s (a property owner) name. All changes are done in the same session.

Register the custom “transaction commit listener” into the project

```
    MyTransactionListener transactionListener = new MyTransactionListener();
    TransactionManager transactionManager =
project.getRepository().getTransactionManager();
    transactionManager.addTransactionCommitListener(transactionListener);
```

Related pages

- Session management(see page 2224)

Working with stereotypes and tagged values

The UML metamodel implementation itself does not provide a way to assign stereotypes and tags directly to Elements. The UML metamodel implementation in a modeling tool provides a pretty complex mapping — every UML Element can have an Instance of assigned Stereotypes. It is stored as the *Element.appliedStereotypeInstance* property. Slots of this instance are *TaggedValues*. If several stereotypes are applied, same InstanceSpecification is used with many classifiers set.

There are two ways to work with profiling information

- We provide a *com.nomagic.uml2.ext.jmi.helpers.StereotypesHelper*[699] helper class for hiding this mapping complexity. It has set of useful methods for assigning or unassigning stereotypes and creating *TaggedValues*. Keep in mind that *TaggedValues* in this helper class are called Slots.
- Use profile implementation classes

Related pages

Using StereotypesHelper

Use utility class *com.nomagic.uml2.ext.jmi.helpers.StereotypesHelper*[700] to work with profiles, stereotypes, tagged values. Keep in mind that *Tagged Values* in this helper class are called Slots or stereotype properties.

699 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html
700 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

StereotypeHelper[701]provides API to check for stereotypes by a stereotype name. Try to avoid this API and pass Stereotype as object if possible. In many models several profiles with stereotypes with same names can be used and this "name based" API can cause unpredictable results.
So general rules are:

- Put URI for your profiles and find Profile in model by URI, not by profile name
- Find Stereotype element inside a specific Profile, not in a whole model
- Pass Stereotype element to *StereotypesHelper*[702]methods instead of passing a stereotype name

Find a stereotype by name

Find profile by URI (or by name) first using

StereotypesHelper.getProfileByURI([703]Project[704], java.lang.String)[705]

StereotypesHelper.getProfile([706]Project[707], java.lang.String)[708]

Find stereotype by name inside a profile using

StereotypesHelper.getStereotype([709]Project[710], java.lang.String,[711] Profile[712])[713]

Check if stereotype is applied

Check if any stereotype is applied

701 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html
702 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html
703 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#getProfileByURI-com.nomagic.magicdraw.core.Project-java.lang.String-
704 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Project.html
705 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#getProfileByURI-com.nomagic.magicdraw.core.Project-java.lang.String-
706 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#getProfile-com.nomagic.magicdraw.core.Project-java.lang.String-
707 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Project.html
708 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#getProfile-com.nomagic.magicdraw.core.Project-java.lang.String-
709 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#getStereotype-com.nomagic.magicdraw.core.Project-java.lang.String-com.nomagic.uml2.ext.magicdraw.mdprofiles.Profile-
710 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Project.html
711 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#getStereotype-com.nomagic.magicdraw.core.Project-java.lang.String-com.nomagic.uml2.ext.magicdraw.mdprofiles.Profile-
712 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/mdprofiles/Profile.html
713 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#getStereotype-com.nomagic.magicdraw.core.Project-java.lang.String-com.nomagic.uml2.ext.magicdraw.mdprofiles.Profile-

StereotypesHelper.hasStereotype([714]Element[715])[716]

Check if concrete stereotype is assigned
StereotypesHelper.hasStereotype([717]Element[718],[719]Stereotype[720])[721]

Check if concreate stereotype or derived stereotype is applied

StereotypesHelper.hasStereotypeOrDerived([722]Element[723],[724]Stereotype[725])[726]

Get value of tagged value

> ⚠ Values of tagged values are returned as *java.util.List* even if upper multiplicity of tag 1. In this case List will have zero or one object.

---

[714] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#hasStereotype-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-
[715] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
[716] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#hasStereotype-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-
[717] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#hasStereotype-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype-
[718] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
[719] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#hasStereotype-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype-
[720] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#hasStereotype-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype-
[721] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#hasStereotype-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype-
[722] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#hasStereotypeOrDerived-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype-
[723] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
[724] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#hasStereotypeOrDerived-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype-
[725] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/mdprofiles/Stereotype.html
[726] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#hasStereotypeOrDerived-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype-

StereotypesHelper.getStereotypePropertyValue([727]Element[728],[729]Stereotype[730], java.lang.String)[731]

StereotypesHelper.getStereotypePropertyFirst([732]Element[733],[734]Stereotype[735], java.lang.String)[736]

Set value of tagged value

StereotypesHelper.setStereotypePropertyValue([737]Element[738],[739]Stereotype[740], java.lang.String, java.lang.Object)[741]

Collect all elements with applied stereotype

For a concrete stereotype use

---

[72] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#getStereotypePropertyValue-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype-java.lang.String-

[728] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html

[729] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#getStereotypePropertyValue-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype-java.lang.String-

[730] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/mdprofiles/Stereotype.html

[73] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#getStereotypePropertyValue-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype-java.lang.String-

[73] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#getStereotypePropertyFirst-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype-java.lang.String-

[733] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html

[73] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#getStereotypePropertyFirst-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype-java.lang.String-

[735] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/mdprofiles/Stereotype.html

[73] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#getStereotypePropertyFirst-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype-java.lang.String-

[7] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#setStereotypePropertyValue-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype-java.lang.String-java.lang.Object-

[738] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html

[9] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#setStereotypePropertyValue-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype-java.lang.String-java.lang.Object-

[740] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/mdprofiles/Stereotype.html

[4] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#setStereotypePropertyValue-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype-java.lang.String-java.lang.Object-

StereotypesHelper.getExtendedElements([742]Stereotype[743])[744]
For a concrete stereotype and derived stereotypes from it use

StereotypesHelper.getExtendedElementsIncludingDerived([745]Stereotype[746])[747]

Check if stereotype can be applied to Element

StereotypesHelper.canApplyStereotype([748]Element[749],[750]Stereotype[751])[752]

Remove stereotype from element

StereotypesHelper.removeStereotype([753]Element[754],[755]Stereotype[756])[757]

---

[742] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#getExtendedElements-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype-
[743] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/mdprofiles/Stereotype.html
[744] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#getExtendedElements-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype-
[745] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#getExtendedElementsIncludingDerived-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype-
[746] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/mdprofiles/Stereotype.html
[747] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#getExtendedElementsIncludingDerived-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype-
[748] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#canApplyStereotype-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype-
[749] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
[750] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#canApplyStereotype-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype-
[751] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/mdprofiles/Stereotype.html
[752] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#canApplyStereotype-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype-
[753] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#removeStereotype-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype-
[754] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
[755] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#removeStereotype-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype-
[756] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/mdprofiles/Stereotype.html
[757] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#removeStereotype-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype-

Apply stereotype to element

StereotypesHelper.addStereotype([758]Element[759],[760]Stereotype[761])[762]

Sample

```
Element element = ....; //Element for which we add stereotype, set tag value and then
remove the stereotype.
Project project = ...; //Project
String tagValue = "test value";
Profile profile = StereotypesHelper.getProfile(project, "ProfileNameExample");
Stereotype stereotype = StereotypesHelper.getStereotype(project,
"StereotypeNameExample", profile);
StereotypesHelper.addStereotype(element, stereotype); //We add stereptype to element
StereotypesHelper.setStereotypePropertyValue(element, stereotype, "TagName",
tagValue); //we set stereotype tag named "TagName"
StereotypesHelper.removeStereotype(element, stereotype); //we remove stereotype from
element
```

Custom profile implementation

Instead of using low level *com.nomagic.uml2.ext.jmi.helpers.StereotypesHelper*[763] API to access information about profile elements, it is possible to generate a profile implementation class out of the UML Profile model. This implementation class wraps all mapping, naming details with a nice Java API. MagicDraw comes with already generated profiles implementation for standard profiles. See Standard profiles implementation(see page 2260) page for details.

To create your own custom profile implementation you need to follow these steps

1. Install Development Tools plugin from Help->Resource Manager
2. Open UML Profile model with MagicDraw.
3. Generate profile implementation code using Tools->Development Tools->Generate Profile Class Implementation

Profile implementation class has these features

- There is a constant defined for every DataType, Enumeration or EnumerationLiteral name
- There is a getter for every DataType element defined in the profile
- There is a getter for every Stereotype element defined in the profile
- There is a static check method to check if Element has specific stereotype applied

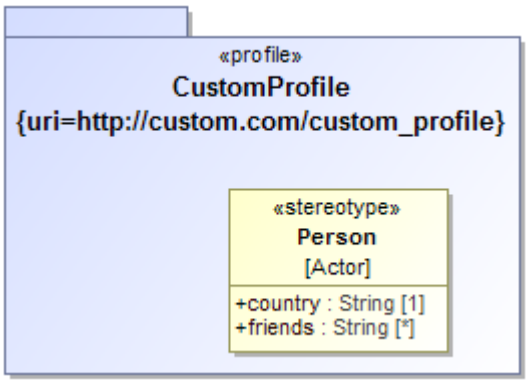
---

[758] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#addStereotype-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype-
[759] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
[760] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#addStereotype-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype-
[761] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/mdprofiles/Stereotype.html
[762] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#addStereotype-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype-
[763] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html

- Java enumeration is generated for every Enumeration element defined in the profile
- Inner java class is generate for every Stereotype defined in the profile. This class has following features
    - There is a constant defined for Stereotype name
    - There are constants defined for all stereotype tags names
    - There is a setter/getter method available for every stereotype tag. Signature of these methods corresponds to tag multiplicity type and etc. No more need to work with strings and casting values to primitives as in *StereotypesHelper* case.

> It is highly recommended to assign URI to Profile element in the UML model before generation. URI is an unique Profile identifier and helps to avoid conflicts if model has several profiles with same name applied.

Custom sample profile



Usages example

```
Element element = ...;
//check if element has Person stereotype applied
CustomProfile.isPerson(element)
//get Person.country tag value
String country = CustomProfile.Person.getCountry(element);
//set Person.country tag value
CustomProfile.Person.setCountry(element, "US");
```

Generated implementation class for this custom profile

```java
import com.nomagic.magicdraw.core.Project;
import com.nomagic.magicdraw.uml.BaseElement;
import com.nomagic.uml2.ext.jmi.helpers.StereotypeByProfileCache;
import com.nomagic.uml2.ext.jmi.helpers.StereotypesHelper;
import com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element;
import com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype;
import javax.annotation.CheckForNull;
import javax.annotation.Nonnull;
import java.util.Collection;
import java.util.HashSet;

public class CustomProfile extends StereotypeByProfileCache
{
   public static final String PROFILE_URI = "http://custom.com/custom_profile";
```

```
    public static final String PROFILE_NAME = "Custom Profile";

    public static CustomProfile getInstance(@Nonnull BaseElement element)
    {
       return getInstance(Project.getProject(element));
    }

    public CustomProfile(@Nonnull Project project)
    {
       super(project, PROFILE_NAME, PROFILE_URI);
    }

    public static CustomProfile getInstance(@Nonnull Project project)
    {
       CustomProfile instance = _getInstance(CustomProfile.class, project);
       if (instance == null)
       {
          instance = new CustomProfile(project);
```

```
        }
        return instance;
    }

    public static class Person extends AbstractStereotypeWrapper
    {
        //stereotype Person and its tags
        @SuppressWarnings("UnusedDeclaration")
        public static final String STEREOTYPE_NAME = "Person";
        @SuppressWarnings("UnusedDeclaration")
        public static final String COUNTRY = "country";
        @SuppressWarnings("UnusedDeclaration")
        public static final String FRIENDS = "friends";

        @SuppressWarnings("UnusedDeclaration")
        public static void setCountry(Element element, String value)
        {
            StereotypesHelper.setStereotypePropertyValue(element,
getInstance(element).getPerson(), COUNTRY, value);
        }

        @SuppressWarnings("UnusedDeclaration, unchecked")
        @CheckForNull
        public static String getCountry(Element element)
        {
            return toString(StereotypesHelper.getStereotypePropertyFirst(element,
getInstance(element).getPerson(), COUNTRY));
        }

        @SuppressWarnings("UnusedDeclaration")
        public static void setFriends(Element element, java.util.List<String> value)
        {
            StereotypesHelper.setStereotypePropertyValue(element,
getInstance(element).getPerson(), FRIENDS, value);
        }

        @SuppressWarnings("UnusedDeclaration")
        public static void addFriends(Element element, String value)
        {
            StereotypesHelper.setStereotypePropertyValue(element,
getInstance(element).getPerson(), FRIENDS, value, true);
        }

        @SuppressWarnings("UnusedDeclaration")
        public static void removeFriends(Element element, String value)
        {
            java.util.List<String> values = new
 java.util.ArrayList<>(getFriends(element));
            values.remove(value);
            setFriends(element, values);
        }
```

```
      @SuppressWarnings("UnusedDeclaration, unchecked")
      @Nonnull
      public static java.util.List<String> getFriends(Element element)
      {
         return (java.util.List<String>)
(StereotypesHelper.getStereotypePropertyValue(element,
getInstance(element).getPerson(), FRIENDS));
      }

   }

   @SuppressWarnings({"UnusedDeclaration", "ConstantConditions"})
   public Stereotype getPerson()
   {
      return getStereotype(Person.STEREOTYPE_NAME);
   }

   @SuppressWarnings("UnusedDeclaration")
   public static boolean isPerson(@CheckForNull Element element)
   {
      if (element instanceof com.nomagic.uml2.ext.magicdraw.mdusecases.Actor)
      {
         CustomProfile instance = getInstance(element);
         return instance.isTypeOf(element, instance.getPerson());
      }
      return false;
   }

   @Override
   protected Collection<Stereotype> generatedGetAllStereotypes()
   {
      if (getProfile() != null)
      {
         final Collection<Stereotype> stereotypes = new HashSet<>();
         stereotypes.add(getPerson());
         return stereotypes;
      }
      return super.generatedGetAllStereotypes();
   }

}
```

Related pages

- Standard profiles implementation(see page 2267)

Standard profiles implementation

Standard profiles implementation are available as part of API.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Use com.nomagic.uml2.MagicDrawProfile[764] to access "MagicDraw Profile" elements

Use *com.nomagic.uml2.StandardProfile*[765] to access "StandardProfile" elements

Use *com.nomagic.uml2.ValidationProfile*[766] to access "Validation Profile" elements

MagicDraw plugins (like SysML) can also expose implementations of profiles. Check their APIs.

UML Model Implementation Using EMF

Starting from version 17.0, a UML model is implemented using Eclipse Modeling Framework (EMF), that is the UML model is an EMF model. All UML model classes implement the *org.eclipse.emf.ecore.EObject* interface. The UML model can be accessed and changed using the EMF API. For example:

```
// get project model
Package model = project.getPrimaryModel();
// create session
SessionManager.getInstance().createSession(project, "create class");


// get name attribute
EAttribute element_name = UMLPackage.eINSTANCE.getNamedElement_Name();
// get name value (same as model.getName())
Object name = model.eGet(element_name);
System.out.println("name = " + name);
// change name value (same as model.setName(name + "_1");)
model.eSet(element_name, name + "_1");


Class aClass = UMLFactory.eINSTANCE.createClass();


// get packaged element collection
Collection collection = (Collection)
model.eGet(UMLPackage.eINSTANCE.getPackage_PackagedElement());
// add new class (same result as model.getPackagedElement().add(aClass))
collection.add(aClass);


// close session
SessionManager.getInstance().closeSession(project);
```

Related pages

764 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/MagicDrawProfile.html
765 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/StandardProfile.html
766 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ValidationProfile.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

UML model implementation in version 17.0 details

The MagicDraw (or other modeling tool developed by No Magic Inc.) 17.0 UML metamodel has a nested package structure. UML is a root package that has nested subpackages, for example, classes/mdkernel, components/mdbasiccomponents, mdusecases. Metadata about a specific model element can be found in the package of the element. For example, a UML model element *Property* is from the package *uml/classes/mdkernel*, so metadata about *Property* can be found in the package *com.nomagic.uml2.ext.magicdraw.classes.mdkernel.metadata.MdkernelPackage*.

```
EAttribute nameAttribute = UMLPackage.Literals.NAMED_ELEMENT__NAME;
```

Model elements can be created using the EMF UML factory *com.nomagic.uml2.ext.magicdraw.metadata.UMLFactory*: For example:

```
Component component = UMLFactory.eINSTANCE.createComponent();
```

The element will be created in the repository of the active project. UML model elements are not contained by any EMF resource.

UML model implementation in 17.0.1 and up details

In MagicDraw (or other modeling tool developed by No Magic Inc.) version 17.0.1, the UML metamodel has only one package - UML, and all UML metadata exist in this package. All UML metadata can be found in com.nomagic.uml2.ext.magicdraw.metadata.UMLPackage[767]. For example:

```
EAttribute nameAttribute = UMLPackage.Literals.NAMED_ELEMENT__NAME;
```

UML model elements can be created using the EMF factory com.nomagic.uml2.ext.magicdraw.metadata.UMLFactory[768]. For example:

```
Component component = UMLFactory.eINSTANCE.createComponent();
```

In version 17.0.1, an element created using the EMF factory will be placed in the repository that does not belong to any project. The created object will be automatically added into the project's repository after it will be added into the container that is in the project's repository.

All UML model elements are directly or indirectly contained by EMF resources. Resource of an element can be retrieved in the standard way:

```
Element element;
org.eclipse.emf.ecore.resource.Resource resource = element.eResource();
```

767 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/metadata/UMLPackage.html
768 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/metadata/UMLFactory.html

There is no strict definition how EMF resources are organized in MagicDraw project. It depends on the project type (local, from Teamwork Server or from the Teamwork Cloud server).

Working with diagrams

Diagrams are not a part of UML metamodel, but we extended it and introduced a new Diagram element. Diagram element extends a standard UML NamedElement.

Related pages

Diagram types

There are two groups of the diagrams in modeling tools developed by No magic Inc. – creatable and not creatable. Only the diagrams of a creatable type can be created (instantiated). A not creatable diagram serves as the base for other types of diagrams.

There are 17 predefined types of diagrams (13 creatable and 4 not). Creatable diagrams are:

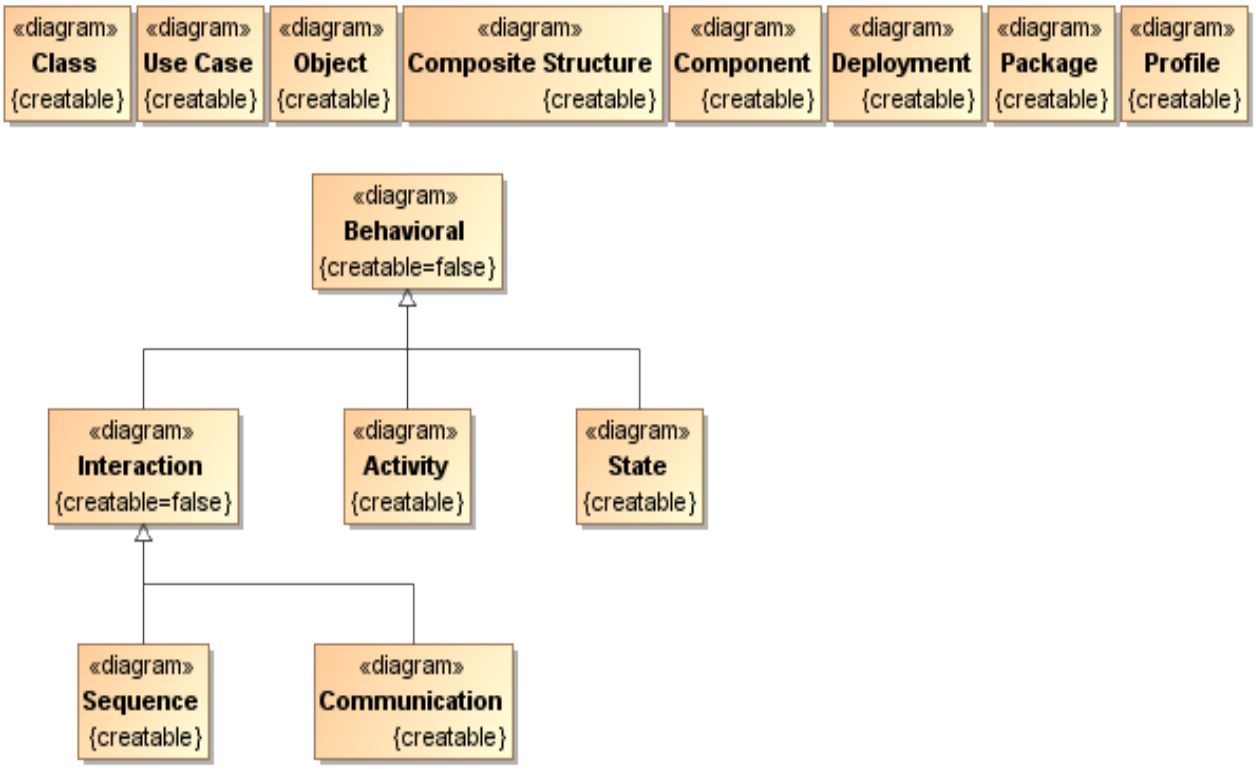
- Class
- Use Case
- Object
- Communication
- Sequence
- State Machine
- Protocol State Machine
- Activity
- Composite Structure
- Component
- Deployment
- Package
- Profile

Not creatable diagrams:

- Interaction
- Behavior
- Any
- Static

Communication and Sequence diagrams are subdiagrams of the Interaction diagram. Activity, Interaction, State Machine, and Protocol State Machine diagrams are subdiagrams of the Behavior diagram.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The only way to add a new diagram type to the modeling tool is to extend one of the already existing diagram types and register it. This mechanism is described in Adding new diagram types(see page 2270).

Related pages

Adding new diagram types

Adding new diagram types to the modeling tool consists of two steps:

1. Overriding the abstract *com.nomagic.magicdraw.uml.DiagramDescriptor* class(see page 2270)
2. Registering a new diagram type(see page 2270)

Step #1. Override the abstract com.nomagic.magicdraw.uml.DiagramDescriptor[769] class

Override the abstract class *DiagramDescriptor*[770]and implement the following abstract methods:

- *getDiagramTypeId()*[771]– this method must return a diagram type id, which is used to identify the diagram

769 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/DiagramDescriptor.html
770 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/DiagramDescriptor.html
771 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/DiagramDescriptor.html#getDiagramTypeId--

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- getSingularDiagramTypeHumanName()[772] – the method returns the diagram type human name
- *getPluralDiagramTypeHumanName()*[773] – the method returns the diagram type human name in a plural form
- getSuperType()[774] – this method must return a super type (super diagram) of this diagram type
- *isCreatable()*[775] – returns the flag indicating if this diagram type is creatable
- getSVGIcon()[776] – returns a large icon for this type of the diagram (see Adding New Functionality(see page 2196))
- getSmallIconURL()[777] – returns a small icon URL for this type of the diagram (see Adding New Functionality(see page 2196))
- *getDiagramActions()*[778] – returns the manager of actions used in the diagram
- getDiagramToolbarConfigurator()[779] – returns the action manager configurator which configures the described diagram toolbar (see Adding New Functionality(see page 2196))
- *getDiagramShortcutsConfigurator()*[780] – returns the action manager configurator which configures described diagram shortcuts (see Adding New Functionality(see page 2196))
- *getDiagramContextConfigurator()*[781] – returns the action manager configurator which configures described diagram shortcut menu actions (see Adding New Functionality(see page 2196)).

The diagram descriptor example

> For the full source code, see the Open API examples in *<program installation directory>\openapi\examples.*

---

772 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/DiagramDescriptor.html#getSingularDiagramTypeHumanName--
773 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/DiagramDescriptor.html#getPluralDiagramTypeHumanName--
774 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/DiagramDescriptor.html#getSuperType--
775 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/DiagramDescriptor.html#isCreatable--
776 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/DiagramDescriptor.html#getSVGIcon--
777 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/DiagramDescriptor.html#getSmallIconURL--
778 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/DiagramDescriptor.html#getDiagramActions--
779 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/DiagramDescriptor.html#getDiagramToolbarConfigurator--
780 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/DiagramDescriptor.html#getDiagramShortcutsConfigurator--
781 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/DiagramDescriptor.html#getDiagramContextConfigurator--

```
/**
* Descriptor of a specific diagram.
*/
public class SpecificDiagramDescriptor extends DiagramDescriptor
{
     public static final String SPECIFIC_DIAGRAM = "Specific Diagram";

    /**
    * Let this diagram type be a sub type of a class diagram type.
    */
    public String getSuperType()
    {
        return DiagramType.UML_CLASS_DIAGRAM;
    }

    /**
    * This is a creatable diagram.
    */
    public boolean isCreatable()
    {
        return true;
    }

    /**
    * Actions used in this diagram.
    */
    public MDActionsManager getDiagramActions()
    {
        return SpecificDiagramActions.ACTIONS;
    }

    /**
    * A configurator for a diagram toolbar.
    */
    public AMConfigurator getDiagramToolbarConfigurator()
    {
        return new SpecificDiagramToolbarConfigurator();
    }

    /**
```

```
        * A configurator for diagram shortcuts.
        */
        public AMConfigurator getDiagramShortcutsConfigurator()
        {
            return new ClassDiagramShortcutsConfigurator();
        }

        /**
        * A configurator for a diagram shortcut menu.
        */
        public DiagramContextAMConfigurator getDiagramContextConfigurator()
        {
            return new BaseDiagramContextAMConfigurator();
        }

        /**
        * Id of the diagram type.
        */
        public String getDiagramTypeId()
        {
            return SPECIFIC_DIAGRAM;
        }

        /**
        * A diagram type human name.
        */
        public String getSingularDiagramTypeHumanName()
        {
            return "Specific Diagram";
        }

        /**
        * A diagram type human name in a plural form.
        */
        public String getPluralDiagramTypeHumanName()
        {
            return "Specific Diagrams";
        }

        /**
        * Resizable svg icon for diagram.
        */
        public ResizableIcon getSVGIcon()
        {
            ResizableIcon icon = null;
            try
```

```
            {
                icon = IconsFactory.getSvgIcon(new File("icons/
specificdiagram.svg").toURI().toURL());
            }
            catch(Exception e){e.printStackTrace();}
            return icon;
        }

        /**
        * URL to a small icon for a diagram.
        */
        public URL getSmallIconURL()
        {
             return getClass().getResource("icons/specificdiagram.svg");
        }
    }
```

Step #2. Register a new diagram type

The new diagram descriptor should be registered in a modeling tool using the *com.nomagic.magicdraw.core.Application.addNewDiagramType(*[782]DiagramDescriptor[783]*)* method of the modeling tool you are using. This method can be invoked when a plugin(see page 2157) is initialized.

The diagram descriptor registration example

> For the full source code, see the Open API examples in *<program installation directory>\openapi\examples\specificdiagram.*

782 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Application.html#addNewDiagramType-com.nomagic.magicdraw.uml.DiagramDescriptor-

783 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/DiagramDescriptor.html

```
    class SpecificDiagramPlugin extends Plugin{
        /**
        * Initializing the plugin.
        */
        public void init()
        {
            // Registering a new diagram type
            Application.getInstance().addNewDiagramType(new
 SpecificDiagramDescriptor());
        }
        /**
        * Always returns true, because this plugin does not have any close specific
actions.
        */
        public boolean close()
        {
            return true;
        }
        /**
        * Always returns true, because this plugin does not have any specific
suportability conditions.
        */
        public boolean isSupported()
        {
            return true;
        }
    }
```

Creating a diagram

Every diagram has a diagram type and this type should be provided during diagram creation.
An example shows how to create and add a diagram to a parent element:

```
    Project project = ...;
    Package parent = project.getPrimaryModel();
    // create a new session
    SessionManager.getInstance().createSession(project, "Create a diagram");
    try
    {
        //a class diagram is created and added to a parent model element
        Diagram diagram =
ModelElementsManager.getInstance().createDiagram(DiagramTypeConstants.UML_CLASS_DIAGR
AM, parent);
        //open a diagram
        project.getDiagram(diagram).open();
    }
    catch (ReadOnlyElementException e)
    {
    }
    SessionManager.getInstance().closeSession(project);
```

Related pages

- Session management(see page 2224)

Presentation elements (symbols)

The UML semantic defines only the UML metamodel. A modeling tool developed by No Magic Inc. has its own structure of classes for UML elements representation in the diagram. The base class of this structure is *com.nomagic.magicdraw.uml.symbols.PresentationElement*[784]. A presentation element is a textual or graphical presentation of one or more model elements.

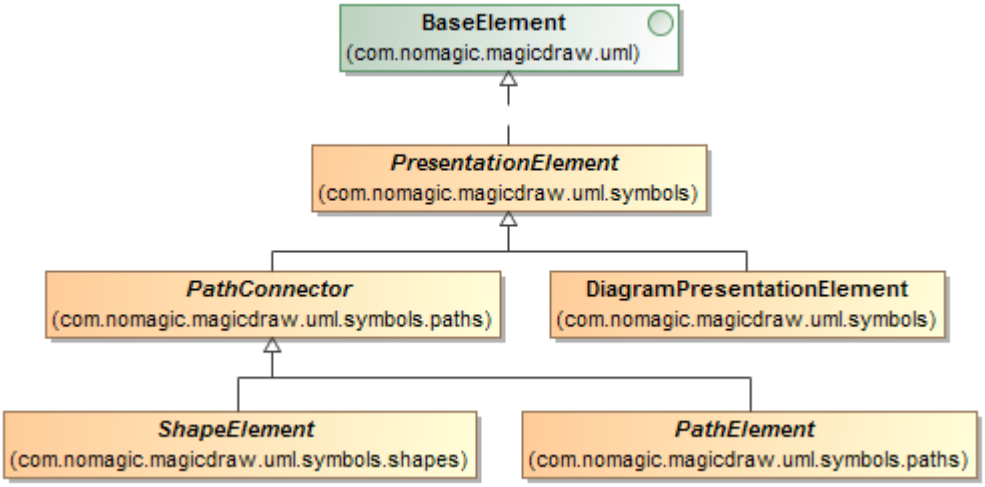
Presentation elements are sometimes called symbols also.

In the metamodel, a *PresentationElement*[785]is the *com.nomagic.magicdraw.uml.BaseElement*[786] that presents a set of model elements to a user. It is the base for all metaclasses used for the presentation. All other metaclasses with this purpose are indirect subclasses of *PresentationElement*[787].

A current version of Open API provides just a basic structure of presentation elements:

784 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/PresentationElement.html
785 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/PresentationElement.html
786 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/BaseElement.html
787 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/PresentationElement.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Every presentation element can have children. For example, *com.nomagic.magicdraw.uml.symbols.DiagramPresentationElement*[788] has a collection of inner presentation elements. *PresentationElement*[789]of some *Package* can have a collection of presentation elements for inner *Package* elements.

Current version *PresentationElement* API allows you to:

- Access element bounds *PresentationElement.getBounds()*[790]
- Access a model *com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element*[791] of a presentation element *PresentationElement.getElement()*[792]. A presentation element can have no *Element*[793], for example, a *com.nomagic.magicdraw.uml.symbols.shapes.TextBoxView*[794].
- Access children *PresentationElement.getPresentationElements()*[795]
- Access properties of the element *PresentationElement.getProperty(java.lang.String)*[796], PresentationElement.getPropertyManager()[797]. The sample of properties would be the *Suppress Operations* property for a class presentation element, the *Autosize* property for any *com.nomagic.magicdraw.uml.symbols.shapes.ShapeElement*[798].
- Select/unselect or access a selection state of the presentation element (*PresentationElement.isSelected()*[799], *PresentationElement.setSelected(boolean)*[800]).

A subclass of presentation elements *com.nomagic.magicdraw.uml.symbols.paths.PathConnector*[801] provides information about connected paths to the presentation element. To get a collection of connected paths to the presentation element, use the method *PathConnector.getConnectedPathElements()*[802]

788 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/DiagramPresentationElement.html
789 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/PresentationElement.html
790 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/PresentationElement.html#getBounds--
791 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
792 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/PresentationElement.html#getElement--
793 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
794 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/shapes/TextBoxView.html
795 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/PresentationElement.html#getPresentationElements--
796 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/PresentationElement.html#getProperty-java.lang.String-
797 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/PresentationElement.html#getPropertyManager--
798 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/shapes/ShapeElement.html
799 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/PresentationElement.html#isSelected--
800 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/PresentationElement.html#setSelected-boolean-
801 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/paths/PathConnector.html
802 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/paths/PathConnector.html#getConnectedPathElements--

Related pages

Diagram presentation element

com.nomagic.magicdraw.uml.symbols.DiagramPresentationElement[803] is a one to one mapping to UML *com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram*[804]. It means for every *Diagram*[805]in a model exists one and only one *DiagramPresentationElement*[806].
Use com.nomagic.magicdraw.core.Project.getDiagram([807]Diagram[808])[809] to get it for a Diagram.

The purpose of this presentation element is to manage all other presentation elements used in a Diagram:

- To get a collection of inner presentation elements, use the *com.nomagic.magicdraw.uml.symbols.PresentationElement.getPresentationElements()*[810] method. This method returns only direct children of the diagram.
- To open a diagram in the program UI, call DiagramPresentationElement.open()[811].
- To create/modify/delete inner presentation elements of the diagram, use com.nomagic.magicdraw.openapi.uml.PresentationElementsManager[812].

Shapes

Shapes are presentation elements created for such model elements as classes, packages, models, subsystems and others.

Paths

Paths are presentation elements created for such model elements as Relationships. Path has the following attributes:

---

803 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/DiagramPresentationElement.html
804 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Diagram.html
805 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Diagram.html
806 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/DiagramPresentationElement.html
807 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Project.html#getDiagram-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram-
808 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Diagram.html
809 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Project.html#getDiagram-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram-
810 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/PresentationElement.html#getPresentationElements--
811 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/DiagramPresentationElement.html#open--
812 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/openapi/uml/PresentationElementsManager.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- **Supplier** – the presentation element of a Relationship supplier. Use method *com.nomagic.magicdraw.uml.symbols.paths.PathElement.getSupplier()*[813] to access this element.
- **Client** – the presentation element of a Relationship client. Use method *PathElement.getClient()*[814] to access this element.
- **Supplier point** – the connection point of a path and supplier element. Use method *PathElement.getSupplierPoint()*[815] to access this point.
- **Client point** – the connection point of a path and client element. Use method *PathElement.getClientPoint()*[816]to access this point.
- **Break points** – a list of path breaking points between supplier and client points. This list can be empty if path is straight. Use method *PathElement.getBreakPoints()*[817] to access this list.

Related pages

- Working with paths and relationships(see page 557)
- Displaying paths(see page 513)

Working with presentation elements

Use only getter in Presentation elements' classes even if setters are available. Setters are not part of OpenAPI and in most cases should not be called directly.

Majority of functions for manipulating/creating/deleting the symbols is available in utility class *com.nomagic.magicdraw.openapi.uml.PresentationElementsManager*[818].

PresentationElementManager[819]can be used only inside the created session with *com.nomagic.magicdraw.openapi.uml.SessionManager*[820]. If a session is not created, *java.lang.IllegalStateException* is thrown.

PresentationElementManager[821]can be used only in already loaded and active project. In other cases, results can be unpredictable.

PresentationElementManager[822]is implemented as a singleton. Use method PresentationElementManager.getInstance()[823] to get a shared instance of this manager.

Related pages

---

813 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/paths/PathElement.html#getSupplier--
814 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/paths/PathElement.html#getClient--
815 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/paths/PathElement.html#getSupplierPoint--
816 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/paths/PathElement.html#getClientPoint--
817 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/paths/PathElement.html#getBreakPoints--
818 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/openapi/uml/PresentationElementsManager.html
819 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/openapi/uml/PresentationElementsManager.html
820 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/openapi/uml/SessionManager.html
821 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/openapi/uml/PresentationElementsManager.html
822 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/openapi/uml/PresentationElementsManager.html
823 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/openapi/uml/PresentationElementsManager.html#getInstance--

- Session management(see page 2224)

Creating shape elements

To create a *com.nomagic.magicdraw.uml.symbols.shapes.ShapeElement*[824] for given *ModelElement* in the given *com.nomagic.magicdraw.uml.symbols.DiagramPresentationElement*[825], use method *com.nomagic.magicdraw.openapi.uml.PresentationElementsManager#createShapeElement(...)*. The location of the created shape will be (0,0).

The following code example shows how to do this:

```
    com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class clazz = ...;
    DiagramPresentationElement diagram = ...;
    SessionManager.getInstance().createSession("Test");
    ShapeElement shape =
PresentationElementsManager.getInstance().createShapeElement(clazz, diagram);
    SessionManager.getInstance().closeSession();
```

You can find the code examples in
- *<programinstallation directory>\openapi\examples\symbolcreation*
- *<programinstallation directory>\openapi\examples\sequencecreation*

Related pages

- Session management(see page 2224)

Creating path elements

To create a *com.nomagic.magicdraw.uml.symbols.paths.PathElement*[826] for given *ModelLink* between given client and supplier elements, use method *com.nomagic.magicdraw.openapi.uml.PresentationElementsManager.createPathElement(*[827]Element[828], PresentationElement[829], PresentationElement[830]*)*:

824 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/shapes/ShapeElement.html
825 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/DiagramPresentationElement.html
826 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/paths/PathElement.html
827 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/openapi/uml/PresentationElementsManager.html#createPathElement-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.magicdraw.uml.symbols.PresentationElement-com.nomagic.magicdraw.uml.symbols.PresentationElement-
828 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
829 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/PresentationElement.html
830 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/PresentationElement.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

```
    com.nomagic.uml2.ext.magicdraw.classes.mddependencies.Dependency link = ...;
    PresentationElement clientPE = ...;
    PresentationElement supplierPE = ...;
    SessionManager.getInstance().createSession("Test");
    PathElement path =
PresentationElementsManager.getInstance().createPathElement(link, clientPE,
supplierPE);
    SessionManager.getInstance().closeSession();
```

The diagram is not passed into *createPathElement(*[831]Element[832], PresentationElement[833], PresentationElement[834]*)* method. A new path element is created in the same diagram as client or supplier presentation elements.

> You can find the code examples in
> - *<programinstallation directory>\openapi\examples\symbolcreation*
> - *<programinstallation directory>\openapi\examples\sequencecreation*

Related pages

- Session management(see page 2224)

Reshaping shape elements

To change bounds for the existing *com.nomagic.magicdraw.uml.symbols.shapes.ShapeElement*[835], use the method *com.nomagic.magicdraw.openapi.uml.PresentationElementsManager.reshapeShapeElement(*[836] ShapeElement[837], java.awt.Rectangle)[838]:

---

831 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/openapi/uml/PresentationElementsManager.html#createPathElement-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.magicdraw.uml.symbols.PresentationElement-com.nomagic.magicdraw.uml.symbols.PresentationElement-
832 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
833 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/PresentationElement.html
834 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/PresentationElement.html
835 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/shapes/ShapeElement.html
836 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/openapi/uml/PresentationElementsManager.html#reshapeShapeElement-com.nomagic.magicdraw.uml.symbols.shapes.ShapeElement-java.awt.Rectangle-
837 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/shapes/ShapeElement.html
838 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/openapi/uml/PresentationElementsManager.html#reshapeShapeElement-com.nomagic.magicdraw.uml.symbols.shapes.ShapeElement-java.awt.Rectangle-

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

```
    ShapeElement element = ...;
    Rectangle newBounds = new Rectangle(100,100,80,50);
    SessionManager.getInstance().createSession("Test");
    PresentationElementsManager.getInstance().reshapeShapeElement(element,
newBounds);
    SessionManager.getInstance().closeSession();
```

Every shape element has a preferred size. The shape size cannot be smaller than the preferred size. If you will try to set smaller bounds, these bounds will be increased to the preferred size.
If shape has the *Autosize* property set to TRUE, bounds will be reduced to the preferred size.

Related pages

- Session management(see page 2224)

Moving symbols on diagram to another location

To move a symbol on the diagram to another location, you can use the following method:

```
PresentationElementManager.getInstance().movePresentationElement(PresentationElement
view, Point location)
```

```
    PresentationElement view = ...;
    SessionsManager.getInstance().createSession("Move symbol");
    PresentationsElementsManager.getInstance().movePresentationElement(view, new
 Point(50, 50));
    SessionsManager.getInstance().closeSession();
```

Similarly, a group of symbols can be moved by using the following method:

```
    movePresentationElements(List<PresentationElement> views, Point location)
```

You can find the code examples in
- *<programinstallation directory>\openapi\examples\symbolmove*

Related pages

- Session management(see page 2224)

Moving symbols on diagram to another location and to different parent

To move a symbol on the diagram to another location and to a different parent, you can use the following method:

```
    movePresentationElement(PresentationElement view, PresentationElement parent,
Point location)
```

```
    PresentationElement view = ...;
    SessionManager.getInstance().createSession("Move symbol");
    PresentationElementsManager.getInstance().movePresentationElement(view,
newParent, new Point(50, 50));
    SessionManager.getInstance().closeSession();
```

Similarly, a group of symbols can be moved by using the following method:

```
    movePresentationElements(List<PresentationElement> views, PresentationElement
parent, Point location)
```

You can find the code examples in

- *<programinstallation directory>\openapi\examples\symbolmove*

Related pages

- Session management(see page 2224)

Changing path break points

To change a break points list for the given path element, use the method com.nomagic.magicdraw.openapi.uml.PresentationElementsManager.changePathBreakPoints([839]PathElement[840], java.util.List)[841]. The same method must be used if you want to change a client or supplier connection point.

The following code snippet shows how to do this:

```
PathElement element = ...;
ArrayList points = new ArrayList();
points.add(new Point(100, 100));
points.add(new Point(100, 150));
SessionManager.getInstance().createSession("Test");
PresentationElementsManager.getInstance().changePathBreakPoints(element, points);
SessionManager.getInstance().closeSession();
```

The order of points in the list must be from the supplier to client connection point. The list may or may not include the client and supplier connection points. At first the given points list will be adopted for the current path style (Rectilinear, Bezier. or Oblique) and only then applied for the path.

Related pages

- Session management(see page 2224)

Deleting presentation elements

To remove the given *com.nomagic.magicdraw.uml.symbols.PresentationElement*[842] from the diagram, use the method *com.nomagic.magicdraw.openapi.uml.PresentationElementsManager.deletePresentationElement(*[843] PresentationElement[844]*)*:

---

839 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/openapi/uml/PresentationElementsManager.html#changePathBreakPoints-com.nomagic.magicdraw.uml.symbols.paths.PathElement-java.util.List-

840 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/paths/PathElement.html

841 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/openapi/uml/PresentationElementsManager.html#changePathBreakPoints-com.nomagic.magicdraw.uml.symbols.paths.PathElement-java.util.List-

842 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/PresentationElement.html

843 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/openapi/uml/PresentationElementsManager.html#deletePresentationElement-com.nomagic.magicdraw.uml.symbols.PresentationElement-

844 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/PresentationElement.html

```
PresentationElement element = ...;
SessionManager.getInstance().createSession("Test");
PresentationElementsManager.getInstance().removePresentationElement(element);
SessionManager.getInstance().closeSession();
```

All related presentation elements with all children and connected paths will be removed too.

Related pages

- Session management(see page 2224)

Changing properties of presentation elements

Most of presentation elements can have properties (for example *Autosize* for shapes or *Path Style* for paths):

- To check if an element has some properties (or uses properties from its parent), use the method *com.nomagic.magicdraw.uml.symbols.PresentationElement.useParentProperties()*[845].
- To get all properties of this element, use the method *PresentationElement.getPropertyManager()*[846].
- To get a property of this element with a given ID, use the method *com.nomagic.magicdraw.uml.symbols.PresentationElement.getProperty(java.lang.String)*[847].
- To change element properties, use the method *com.nomagic.magicdraw.openapi.uml.PresentationElementsManager.setPresentationElementProperties(*[848]PresentationElement[849], PropertyManager[850]*)*. The given *com.nomagic.magicdraw.properties.PropertyManager*[851] can have only few element's properties (for example, just properties you want to change).

The Ids of all used properties are defined in the class *com.nomagic.magicdraw.properties.PropertyID*[852].

Look at Properties(see page 2308) page for more details about properties structure.

845 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/PresentationElement.html#useParentProperties--
846 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/PresentationElement.html#getPropertyManager--
847 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/PresentationElement.html#getProperty-java.lang.String-
848 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/openapi/uml/PresentationElementsManager.html#setPresentationElementProperties-com.nomagic.magicdraw.uml.symbols.PresentationElement-com.nomagic.magicdraw.properties.PropertyManager-
849 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/PresentationElement.html
850 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/properties/PropertyManager.html
851 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/properties/PropertyManager.html
852 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/properties/PropertyID.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

There is no information about what exactly properties every presentation element has. Use java debugger and analyze *PresentationElement.getPropertyManager()*[853] object. You can also guess property ID from the Symbol Properties name.

If you want to change a value of existing property, recommendation would be clone that property, change value for a clone and set it. Don't use constructor if possible.

The following code example shows how to change element properties:

```
    ShapeElement element = ...;
    Property property = element.getProperty(PropertyID.AUTOSIZE);
    if (property instanceof BooleanProperty)
    {
        BooleanProperty autoSize = (BooleanProperty) property.clone();
        autoSize.setValue(true);

        PropertyManager properties = new PropertyManager();
        properties.addProperty(autoSize);

        SessionManager.getInstance().createSession(project, "Test");

PresentationElementsManager.getInstance().setPresentationElementProperties(element,
properties);
        SessionManager.getInstance().closeSession(project);
}
```

The properties must be new instances. You cannot do something like this:

```
    ShapeElement element = ...;
    PropertyManager properties = element.getPropertyManager();
    properties.getProperty(PropertyID.AUTOSIZE).setValue(new Boolean(true));
    SessionManager.getInstance().createSession(project, "Test");

PresentationElementsManager.getInstance().setPresentationElementProperties(ele
ment, properties);
    SessionManager.getInstance().closeSession(project);
```

Related pages

- Properties(see page 2308)
- Session management(see page 2224)

853 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/PresentationElement.html#getPropertyManager--

Managing compartments

Some presentation elements displays information in so called compartments. For example UML Class shape has attributes, operations, ports compartments.

Compartments can show only a subset of information. The rest information can be hidden. For example show only a couple of specific Operations on a Class shape in the diagram.

Use com.nomagic.magicdraw.uml.symbols.CompartmentManager[854] for managing the visibility of compartment entries.

More information is available in javadoc[855].

> You can find the code examples in
> - *<programinstallation directory>\openapi\examples\compartmentedit*

Related pages

- Session management(see page 2224)

Collecting presentation elements, finding usages in diagrams

Very often you need to find out if some Element has a presentation element in a diagram. You also might need to collect all existing presentation elements for Element. There are different approaches for doing it. Choose the one which suites your needs.

Keep in mind that diagrams in MagicDraw uses lazy content loading approach. It means that content (presentation elements) are loaded only on demand. For example when diagram is opened.

Use SymbolElementMap

You can take this map using com.nomagic.magicdraw.core.Project.getSymbolElementMap()[856]. Map has API for collecting all symbols in all diagrams for some specific Element.

> Presentation elements can be collected only from loaded diagrams. If diagram is not loaded, presentation elements of this diagram are not registered in a map.
> Load diagram using com.nomagic.magicdraw.uml.symbols.DiagramPresentationElement.ensureLoaded()[857]

Find element usages in loaded and not loaded diagrams (do not load diagrams)

854 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/CompartmentManager.html
855 http://jdocs.nomagic.com/190beta/
856 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Project.html#getSymbolElementMap--
857 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/DiagramPresentationElement.html#ensureLoaded--

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Use this approach if you are not sure if presentation element exists in a diagram. This approach checks existing of presentation elements without loading a diagram.

DiagramPresentationElement[858] API provides very useful methods for that. This API works both in loaded and not loaded diagrams.

DiagramPresentationElement.getUsedModelElements(boolean)[859] to get all displayed elements in the diagram

Use *DiagramPresentationElement.isElementInDiagram(*[860]Element[861]*)* to check if Element has a presentation element in a diagram

If you have not loaded diagram and you need to get presentation element of some symbol, you can do the following

- Use *DiagramPresentationElement.isElementInDiagram(*[862]Element[863]*)* to check if element is in diagram
- Use DiagramPresentationElement.[864]ensureLoaded()[865] to load a diagram
- Use Project.getSymbolElementMap()[866] API to get a reference to presentation element in a diagram

Find presentation elements in diagrams and load them

Use this approach only if you are sure that presentation element of Element exists in diagram, because diagram will be loaded always.

Use DiagramPresentationElement.findPresentationElement([867]Element[868], java.lang.Class)[869] method

858 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/DiagramPresentationElement.html
859 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/DiagramPresentationElement.html#getUsedModelElements-boolean-
860 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/DiagramPresentationElement.html#isElementInDiagram-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-
861 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
862 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/DiagramPresentationElement.html#isElementInDiagram-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-
863 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
864 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/DiagramPresentationElement.html#ensureLoaded--
865 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/DiagramPresentationElement.html#ensureLoaded--
866 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Project.html#getSymbolElementMap--
867 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/DiagramPresentationElement.html#findPresentationElement-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-java.lang.Class-
868 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
869 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/DiagramPresentationElement.html#findPresentationElement-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-java.lang.Class-

More information is available in javadoc[870]

Displaying related elements in the diagram

A modeling tool developed by No Magic Inc. allows displaying symbols that are related to a given symbol via relationships. The *com.nomagic.magicdraw.uml.symbols.DisplayRelatedSymbols*[871] class provides API methods for this. Using this class you can control any of the following behaviors for displaying the related symbols logic:

- What relationship types should be included.
- The depth of a relationship tree.
- Whether or not existing symbols should be reused.

Example: Displaying related generalizations and interface realizations

```
    SessionManager sessionManager = SessionManager.getInstance();
    sessionManager.createSession(project, "Display related");

    Set linkTypes = new HashSet();
    linkTypes.add(new LinkType(Generalization.class));
    linkTypes.add(new LinkType(InterfaceRealization.class));

    DisplayRelatedSymbolsInfo info = new DisplayRelatedSymbolsInfo(linkTypes);
    info.setDepthLimited(true);
    info.setDepthLimit(3);

    PresentationElement view = ...; // A symbol for which you need to invoke the
displaying related symbols action.
    DisplayRelatedSymbols.displayRelatedSymbols(view, info);
    sessionManager.closeSession(project);
```

You can find the code examples in
- *<programinstallation directory>\openapi\examples\displayrelated*

Related pages

- Session management(see page 2224)

Layouting symbols in the diagram

We provide an API to layout presentation elements in a diagram using different strategies (layouter).

- com.nomagic.magicdraw.uml.symbols.DiagramPresentationElement.layout(boolean)[872]

870 http://jdocs.nomagic.com/190beta/
871 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/DisplayRelatedSymbols.html
872 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/DiagramPresentationElement.html#layout-boolean-

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- DiagramPresentationElement.layout(boolean,[873]DiagramLayouter[874])[875]
- DiagramPresentationElement.layout(boolean,[876]DiagramLayouter[877],[878]AbstractDiagramLayouterOptionsGroup[879])[880]

Make sure the diagram is opened and the session is closed before doing a diagram layout. These are the current limitation of layout API

You can find the code examples in
- *<programinstallation directory>\openapi\examples\testframework, com.nomagic.magicdraw.examples.testframework.DiagramLayoutTest class*

Related pages

- Session management(see page 2224)

Custom presentation elements rendering

The custom symbol rendering API allows change a default default symbol view. This API includes:

1. Renderer provider API, which allows to provide a custom renderer for the specific presentation element.
2. Renderers, which can modify the default presentation element appearance.

In this section, we will review, how to register the specific presentation element views provider and give the example covering the custom renderers for the package, slots, and dependency link symbols.

---

87 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/DiagramPresentationElement.html#layout-boolean-com.nomagic.magicdraw.uml.symbols.layout.DiagramLayouter-

874 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/layout/DiagramLayouter.html

87 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/DiagramPresentationElement.html#layout-boolean-com.nomagic.magicdraw.uml.symbols.layout.DiagramLayouter-

87http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/DiagramPresentationElement.html#layout-boolean-com.nomagic.magicdraw.uml.symbols.layout.DiagramLayouter-com.nomagic.magicdraw.core.options.AbstractDiagramLayouterOptionsGroup-

877 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/layout/DiagramLayouter.html

87http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/DiagramPresentationElement.html#layout-boolean-com.nomagic.magicdraw.uml.symbols.layout.DiagramLayouter-com.nomagic.magicdraw.core.options.AbstractDiagramLayouterOptionsGroup-

879 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/options/AbstractDiagramLayouterOptionsGroup.html

88http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/DiagramPresentationElement.html#layout-boolean-com.nomagic.magicdraw.uml.symbols.layout.DiagramLayouter-com.nomagic.magicdraw.core.options.AbstractDiagramLayouterOptionsGroup-

You can find the code examples in *<programinstallation directory>\openapi\examples\symbolrendering.*

Related pages

Custom renderer provider

A *com.nomagic.magicdraw.uml.symbols.PresentationElementRendererProvider*[881] is an object, which provides a custom symbol renderer for a given presentation element.

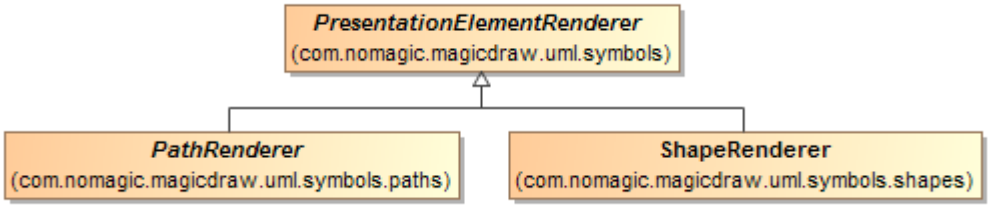
PresentationElementRendererProvider.getRenderer([882]PresentationElement[883]*)* method provides a specific renderer for the given *com.nomagic.magicdraw.uml.symbols.PresentationElement*[884] (Symbol) or null.

To make your provider start working, you must register it into the com.nomagic.magicdraw.uml.symbols.PresentationElementRendererManager[885] using:

```
PresentationElementRendererManager.getInstance().addProvider(new RendererProvider());
```

Custom symbol renderer

To have a custom element renderer, extend abstract *com.nomagic.magicdraw.uml.symbols.PresentationElementRenderer*[886] class.



Custom renderers sample

This example covers:

881 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/PresentationElementRendererProvider.html
882 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/PresentationElementRendererProvider.html#getRenderer-com.nomagic.magicdraw.uml.symbols.PresentationElement-
883 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/PresentationElement.html
884 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/PresentationElement.html
885 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/PresentationElementRendererManager.html
886 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/PresentationElementRenderer.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- A custom renderer for the package.(see page 2291) The package without inner elements is filled with a green color.
- A custom renderer for the slot.(see page 2292) The slot values are rounded.
- A custom renderer for the dependency link.(see page 2292) The dependency link is a blue thicker line with custom line ends.

This example shows, how to create custom renderers for the preceding pointed symbols.

The custom package renderer. The empty package (that is without inner elements) is filled with green color:

```
    class PackageRenderer extends ShapeRenderer
    {
        public Color getColor(PresentationElement presentationElement,
PresentationElementColorEnum colorEnum)
        {
            if (PresentationElementColorEnum.FILL.equals(colorEnum))
            {
                // the color to fill
                Element element = presentationElement.getElement();
                if (element instanceof
 com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Package &&
                    !((com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Package)
element).hasOwnedElement())
                {
                    // the package has no elements, use the green color to fill
                    return Color.GREEN;
                }
            }
            return super.getColor(presentationElement, colorEnum);
        }
    }
```

The custom slot renderer (used in the instance specification symbol). The slot values are rounded:

```
    class SlotRenderer extends ShapeRenderer
    {
        public String getText(PresentationElement presentationElement,
PresentationElementTextEnum textEnum)
        {
            if (PresentationElementTextEnum.NAME.equals(textEnum))
            {
                // the slot text is shown as a name
                Element element = presentationElement.getElement();
                if (element instanceof Slot)
                {
                    Slot slot = (Slot) element;
                    if (slot.hasValue())
                    {
                        String string = "";
                        List<ValueSpecification> values = slot.getValue();
                        for (ValueSpecification value : values)
                        {
                            if (value instanceof LiteralString)
                            {
                                LiteralString literalString = (LiteralString) value;
                                if (string.length() > 0)
                                {
                                   string += "; ";

                                }
                                String literalValue = literalString.getValue();
```

```
                                try
                                {
                                    // round a value
                                    double doubleValue =
Double.parseDouble(literalValue);
                                    double rounded = Math.round(doubleValue * 100) /
100;
                                    literalValue = Double.toString(rounded);
                                }
                                catch (NumberFormatException e)
                                {
                                }
                                string += literalValue;
                            }
                        }
                        return slot.getDefiningFeature().getName() + "=" + string;
                    }
                }
            }
            return super.getText(presentationElement, textEnum);
        }
    }
```

The custom dependency link renderer. The dependency link is a blue thicker line with custom line ends:

```
    class DependencyRenderer extends PathRenderer
    {
        private PathEndRenderer mClientEndRenderer;
        DependencyRenderer()
        {
            // a custom client end renderer, use a filled circle at the end
            mClientEndRenderer = new PathEndRenderer(PathEndAdornment.CIRCLE,
PathEndAdornmentModifier.FILLED);
        }

        public Color getColor(PresentationElement presentationElement,
PresentationElementColorEnum colorEnum)
        {
            if (PresentationElementColorEnum.LINE.equals(colorEnum))
            {
                // use a blue color for a line
                return Color.BLUE;
            }
            return super.getColor(presentationElement, colorEnum);
        }


        protected PathEndRenderer getClientEndRenderer(PathElement pathElement)
        {
            // use a custom end renderer
            return mClientEndRenderer;
        }

        public int getLineWidth(PresentationElement presentationElement)
        {
            // a line width is 2
            return 2;
        }

        protected void drawPathAdornment(Graphics g, PathElement pathElement)
        {
            super.drawPathAdornment(g, pathElement);

            // draw a circle at the middle of the dependency line
            Color background = Color.WHITE;
            Property property =
pathElement.getDiagramPresentationElement().getProperty(PropertyID.DIAGRAM_BACKGROUND
_COLOR);
            if (property != null)
            {
                Object value = property.getValue();
                if (value instanceof Color)
```

```
                {
                    background = (Color) value;
                }
            }
            Point middlePoint = pathElement.getMiddlePoint();
            int diameter = 10;
            int radius = diameter / 2;
            int x = middlePoint.x - radius;
            int y = middlePoint.y - radius;
            Color color = g.getColor();
            g.setColor(background);
            g.fillOval(x, y, diameter, diameter);
            g.setColor(color);
            g.drawOval(x, y, diameter, diameter);
        }
    }
```

Registering custom symbol renderer provider

The following example shows the custom symbol renderer provider, which provides the *SlotRenderer* for slot symbols, the *PackageRenderer* for package symbols, and the *DependencyRenderer* for a dependency link symbols. The created renderer provider must be registered into the *com.nomagic.magicdraw.uml.symbols.PresentationElementRendererManager*[887].

Step 1. Creating the RendererProvider class

887 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/PresentationElementRendererManager.html

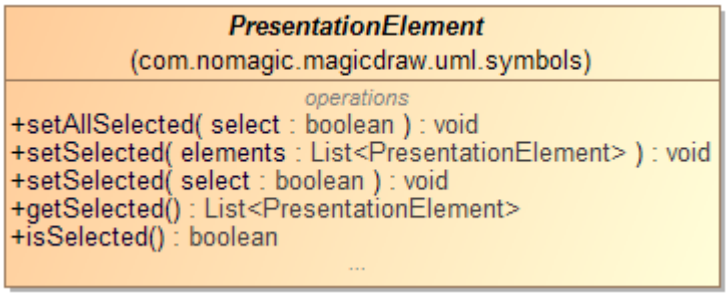
```
     /**
    * A custom renderers provider.
    */
    class RendererProvider implements PresentationElementRendererProvider
    {
        private SlotRenderer slotRenderer;
        private DependencyRenderer dependencyRenderer;
        private PackageRenderer packageRenderer;

        RendererProvider()
        {
            slotRenderer = new SlotRenderer();
            dependencyRenderer= new DependencyRenderer();
            packageRenderer = new PackageRenderer();
        }

        public PresentationElementRenderer getRenderer(PresentationElement
presentationElement)
        {
            if (presentationElement instanceof SlotView || presentationElement
instanceof SlotListElementView)
            {
                // A slot renderer
                return slotRenderer;
            }

            if (presentationElement instanceof DependencyView)
            {
                // dependency renderer
                return dependencyRenderer;
            }

            if (presentationElement instanceof PackageView)
            {
                // package renderer
                return packageRenderer;
            }

            return null;
        }
    };
```

Step 2. Registering RendererProvider

```
    PresentationElementRendererManager.getInstance().addProvider(new
 RendererProvider());
```

Selection in diagrams

Every *com.nomagic.magicdraw.uml.symbols.PresentationElement*[888] can access selected elements or change their own selection status.



Collecting selected presentation elements in a diagram

```
Project project = ;
DiagramPresentationElement diagram = project.getActiveDiagram();
List<PresentationElement> selected = diagram.getSelected();
```

Selecting element in a diagram

```
presentationElement.setSelected(true);
```

Presentation element selection events(see page 2301) page describes selection change events

Related pages

- Presentation element selection events(see page 2301)

Diagram events

Presentation elements modification events

Modeling tools developed by No Magic Inc. provides an API to listen to all diagram changes in a single adapter that works in the following order:

1. Receives events from all *opened* diagrams.
2. Delegates these events to your own listener.

888 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/PresentationElement.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

The adapter works in a project scope.

To listen to diagram change events, you need to create the *com.nomagic.magicdraw.uml.symbols.DiagramListenerAdapter*[889] object, that is, to pass the property change listener as a delegator, which receives all events.

To create the *DiagramListenerAdapter* [890]object, call:

```
    new DiagramListenerAdapter(propertyChangeListener)
```

The *DiagramListenerAdapter* [891]object is registered for a project on its creation.

To start using the adapter, install it. Uninstall it when it is no longer necessary, that is, you do not need to receive any events about diagram changes anymore.

Property names related to presentation elements modification are defined in *com.nomagic.magicdraw.uml.ExtendedPropertyNames*[892] class.

Example: Listening to symbol creation / removal

```
    DiagramListenerAdapter adapter = new DiagramListenerAdapter(new
 PropertyChangeListener()
    {
        public void propertyChange(PropertyChangeEvent evt)
        {
            String propertyName = evt.getPropertyName();
            if (ExtendedPropertyNames.VIEW_ADDED.equals(propertyName) ||
ExtendedPropertyNames.VIEW_REMOVED.equals(propertyName))
            {
                // an added / removed symbol
                PresentationElement presentationElement = (PresentationElement)
evt.getNewValue();
            }
        }
    }

    adapter.install(project);

    // When the adapter is no longer needed, uninstall it.
    adapter.uninstall(project);
```

889 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/DiagramListenerAdapter.html
890 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/DiagramListenerAdapter.html
891 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/DiagramListenerAdapter.html
892 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/ExtendedPropertyNames.html

Presentation element selection events

You need to register a *java.beans.PropertyChangeListener* on a project. Selection changes fire *java.beans.PropertyChangeEvent*.

```
Project project = ;

Listener listener = new Listener();
SelectionProvider selectionProvider = SelectionProvider.getInstance(project);
selectionProvider.addSelectionChangedListener(listener);

private static class Listener implements SelectionProvider.SelectionChangedListener
{
    @Override
    public void selectionChanged(SelectionProvider.SelectionChangedEvent event)
    {
        // was selected
        SelectionProvider.Selection oldSelection = event.getOldSelection();

        // now selected
        SelectionProvider.Selection newSelection = event.getNewSelection();

        // do something
    }
}
```

Notification of presentation element draw

com.nomagic.magicdraw.uml.symbols.SymbolDrawNotification[893] notifies when the presentation element is drawn (added to a diagram):

893 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/SymbolDrawNotification.html

```
    // An element removal listener
    final PropertyChangeListener removeListener = new PropertyChangeListener()
    {
        public void propertyChange(PropertyChangeEvent evt)
        {
            if (ExtendedPropertyNames.REMOVE.equals(evt.getPropertyName()))
            {
                // symbol removed
            }
        }
    };

    // An element drawing listener
    SymbolDrawListener symbolDrawListener = new SymbolDrawListener()
    {
        public void symbolAdded(DiagramPresentationElement diagram,
PresentationElement symbol, String actionID)
        {
            // A symbol added
            // A register listener
            symbol.addPropertyChangeListener(removeListener);
        }
    };

    // A register draw listener
    SymbolDrawNotification symbolDrawNotification
=SymbolDrawNotification.getSymbolDrawNotification(project);
    symbolDrawNotification.addSymbolDrawListener(symbolDrawListener);
```

Custom diagram painters

The Open API provides a way to add your own custom diagram painters for painting some additional stuff on the diagram canvas. A good sample would be some highlighting in the diagram.

> ⚠ The painter can be added only into the opened diagram's *com.nomagic.magicdraw.uml.symbols.DiagramSurface*[894]. Only the opened diagram has *DiagramSurface*[895]. A closed diagram returns *null*.

894 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/DiagramSurface.html
895 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/DiagramSurface.html

Example of the code:

```
Application.getInstance().addProjectEventListener(new ProjectEventListenerAdapter()
{
   public void projectOpened(Project project)
   {
      project.addPropertyChangeListener(new PropertyChangeListener()
      {
         public void propertyChange(PropertyChangeEvent evt)
         {
            if(evt.getPropertyName().equals(Project.DIAGRAM_OPENED))
            {
               DiagramPresentationElement diagram =
Application.getInstance().getProject().getActiveDiagram();
               diagram.getDiagramSurface().addPainter(new DiagramSurfacePainter()
               {
                  public void paint(Graphics g, DiagramPresentationElement diagram)
                  {
                     g.setColor(Color.BLUE);
                     List symbols = diagram.getPresentationElements();
                     for (int i = 0; i < symbols.size(); i++)
                     {
                        PresentationElement o = (PresentationElement)symbols.get(i);
                        if( o instanceof ShapeElement)
                        {
                           Rectangle bounds = o.getBounds();
                           bounds.grow(5,5);
                           ((Graphics2D)g).draw(bounds);
                        }

                     }

                  };

               });

            }

         }

      });

   }

});
```

> You can find the code examples in *<programinstallation directory>\openapi\examples\customdiagrampainter*

Creating diagram images

The Open API of a modeling tool provides the com.nomagic.magicdraw.export.image.ImageExporter[896] class for creating images from the whole diagram or just a part of it.

Image formats are predefined as constants in *ImageExporter*[897].

To create an image from the whole diagram, use the export([898]DiagramPresentationElement[899], int, java.io.File)[900]method.

To create an image from selected symbols in the diagram, use the export([901]DiagramPresentationElement[902], int, java.io.File, boolean)[903] method.

> You can find the code examples in *<programinstallation directory>\openapi\examples\imagegenerator*

Creating custom Drag&Drop handlers on diagram

MagicDraw itself has many default DnD handlers for doing various things while dropping Elements from Browser(trees) component on the diagram or on other presentation elements in the diagram.

These default handlers creates new presentation elements, changes the properties of elements and etc. Same is true if presentation elements are dropped on top of other presentation elements.

API allows to register a custom DnD handler for doing some specific things using DnD mechanism.

There are different registrations

- Use *com.nomagic.magicdraw.ui.dnd.CustomDropDiagramHandlerFactory.register(*[904] CustomDragAndDropHandler[905]*)* to register a handler for dropping elements from browser(trees).

---

896 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/export/image/ImageExporter.html
897 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/export/image/ImageExporter.html
89 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/export/image/ImageExporter.html#export-com.nomagic.magicdraw.uml.symbols.DiagramPresentationElement-int-java.io.File-
899 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/DiagramPresentationElement.html
90 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/export/image/ImageExporter.html#export-com.nomagic.magicdraw.uml.symbols.DiagramPresentationElement-int-java.io.File-
90 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/export/image/ImageExporter.html#export-com.nomagic.magicdraw.uml.symbols.DiagramPresentationElement-int-java.io.File-boolean-
902 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/DiagramPresentationElement.html
90 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/export/image/ImageExporter.html#export-com.nomagic.magicdraw.uml.symbols.DiagramPresentationElement-int-java.io.File-boolean-
90 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ui/dnd/CustomDropDiagramHandlerFactory.html#register-com.nomagic.magicdraw.ui.dnd.CustomDragAndDropHandler-
905 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ui/dnd/CustomDragAndDropHandler.html

- Use com.nomagic.magicdraw.ui.dnd.CustomShapeMoveHandlerFactory.register([906] CustomDragAndDropHandler[907])[908] to register a handler for dropping one presentation element on other presentation element in the same diagram

> You can find the code examples in
> - *<programinstallation directory>\openapi\examples\customdraganddrop*

Managing navigation in model

Open API allows to change how navigation in a model is handled. Using that API you can create and manage hyperlinks, change the mouse double-click behavior on elements.

See these pages for more details:

Customizing double-click behavior

The default application behavior on a double-click on the element is to open the Specification window(see page 913) or open a diagram. This behavior can be changed by registering custom element activators.

Use com.nomagic.magicdraw.ui.ElementActivationManager.addActivator([909]ElementActivator[910])[911] for that purpose.

Managing hyperlinks

UML models may have a different kind of hyperlinks assigned to model elements. There is API for managing these hyperlinks - you can assign new hyperlinks, remove existing.

It is also possible to register your own hyperlink handler for some custom hyperlink type.

See the following pages for more information:

---

[906] http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ui/dnd/CustomShapeMoveHandlerFactory.html#register-com.nomagic.magicdraw.ui.dnd.CustomDragAndDropHandler-
[907] http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ui/dnd/CustomDragAndDropHandler.html
[908] http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ui/dnd/CustomShapeMoveHandlerFactory.html#register-com.nomagic.magicdraw.ui.dnd.CustomDragAndDropHandler-
[909] http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ui/ElementActivationManager.html#addActivator-com.nomagic.magicdraw.ui.ElementActivator-
[910] http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ui/ElementActivator.html
[911] http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ui/ElementActivationManager.html#addActivator-com.nomagic.magicdraw.ui.ElementActivator-

Editing hyperlinks

Hyperlinks are implemented using stereotypes and tag values. The *HyperlinkOwner* stereotype has the following its properties (tags):

- *hyperlinkText* - all simple hyperlinks
- *hyperlinkTextActive* - an active simple hyperlink
- *hyperlinkModel* - all hyperlinks to a model element
- *hyperlinkModelActive* - an active hyperlink to a model element

You can use generic profiling API to edit/add/remove hyperlinks to model elements. See Working with stereotypes and tagged values(see page 2255) for more details.

Recommended way is to use utility class *com.nomagic.magicdraw.hyperlinks.HyperlinkUtils*[912]. This class provides utility methods for adding/removing/retrieving *com.nomagic.magicdraw.hyperlinks.Hyperlink*[913]f rom model Element,

Related pages

- Working with stereotypes and tagged values(see page 2255)

MDEL hyperlink

MagicDraw has a functionality to expose every element as URL. Later this URL can be added into external documents. Also that URL can be used to open/activate an element in a project.

For that purpose the **mdel://** protocol is introduced.

Use *com.nomagic.magicdraw.hyperlinks.HyperlinkUtils*[914]to retrieve mdel URL of elements.

Registering custom hyperlink handler

It is possible to register a new custom hyperlink handler for a custom protocol. The custom hyperlink may have a specific UI editor for editing it's properties.

Use com.nomagic.magicdraw.hyperlinks.HyperlinksHandlersRegistry.addHandler([915] HyperlinkHandler[916])[917] to add a custom handler.

---

912 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/hyperlinks/HyperlinkUtils.html
913 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/hyperlinks/Hyperlink.html
914 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/hyperlinks/HyperlinkUtils.html
915 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/hyperlinks/HyperlinksHandlersRegistry.html#addHandler-com.nomagic.magicdraw.hyperlinks.HyperlinkHandler-
916 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/hyperlinks/HyperlinkHandler.html
917 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/hyperlinks/HyperlinksHandlersRegistry.html#addHandler-com.nomagic.magicdraw.hyperlinks.HyperlinkHandler-

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

You can find the code examples in
- *<programinstallation directory>\openapi\examples\customhyperlink*

Properties

In modeling tools, No Magic Inc. Open API provides a set of classes used as properties for diagrams' symbols, project or environment options and etc.

Every property has two major attributes – the ID and value. A property has a value of the type *java.lang.Object*. Every specific property has a value of the specific type. For example, the value of *com.nomagic.magicdraw.properties.BooleanProperty*[918] is *java.lang.Boolean*, the value of a *com.nomagic.magicdraw.properties.StringProperty*[919] is *java.lang.String*.

The property ID identifies the specific property in the properties set.

You must set some *com.nomagic.magicdraw.properties.PropertyResourceProvider*[920] to your property instance in order to display a normal name, not the ID of the property in the UI of a modeling tool. *PropertyResourceProvider*[921]has just one method getString(java.lang.String ,[922]Property[923])[924]*, where a key is an ID of your property.*

The collection of properties is grouped by *com.nomagic.magicdraw.properties.PropertyManager*[925]. Every *PropertyManager*[926]has a name and a list of properties. It can return the property by the ID, properties with the same value, properties whose values are different.

For more details about every specific kind of a property, see javadoc[927].

Related pages

- Changing properties of presentation elements(see page 2285)
- Project options(see page 2220)

Information logging

The Log4j logger is used in MagicDraw and other modeling tools developed by No Magic Inc. The configuration file is *<program installation directory>/data/debug.properties*.

---

918 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/properties/BooleanProperty.html
919 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/properties/StringProperty.html
920 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/properties/PropertyResourceProvider.html
921 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/properties/PropertyResourceProvider.html
922 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/properties/PropertyResourceProvider.html#getString-java.lang.String-com.nomagic.magicdraw.properties.Property-
923 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/properties/Property.html
924 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/properties/PropertyResourceProvider.html#getString-java.lang.String-com.nomagic.magicdraw.properties.Property-
925 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/properties/PropertyManager.html
926 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/properties/PropertyManager.html
927 http://jdocs.nomagic.com/190beta/

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

The *<program installation directory>/data/test.properties* file is used as a configuration file for program test cases.

The following sample presents the content of the configuration file. It configures the logger to print INFO category messages to a console with a specific pattern.

```
log4j.appender.SO=org.apache.log4j.ConsoleAppender
log4j.appender.SO.layout=org.apache.log4j.PatternLayout
log4j.appender.SO.layout.ConversionPattern=%d [%t] %-5p %c - %m%n
log4j.rootCategory=INFO,SO
```

The example below displays how information can be logged to the configured log file:

```
//used to get logger with category name
org.apache.log4j.Logger logger = org.apache.log4j.Logger.getLogger("GENERAL");

logger.info("Logging info");
logger.error("Logging error");

//by default the text will be displayed in the form of :
//0000-00-00 00:00:00,000 [THREAD] INFO  GENERAL - Logging info
//0000-00-00 00:00:00,000 [THREAD] ERROR GENERAL - Logging error
```

More information about Log4j can be found at http://logging.apache.org/log4j/1.2/.

Environment Options

Application-related options are referred as environment options. If you are using MagicDraw, they are saved in the *global.opt* file that is located in *<user home directory>\AppData\Local\.magicdraw\<version number>\data.* The location for other modeling tools developed by No Magic Inc. is adequate.

You can add custom environment options for your modeling tool.

To add your own environment options

1. Extend the *com.nomagic.magicdraw.core.options.AbstractPropertyOptionsGroup*[928] class.
2. Add the extending class to application environment options.

Example of adding custom environment options

```
class MyOptionsGroup extends AbstractPropertyOptionsGroup
{
    ...
}
Application application = Application.getInstance();
EnvironmentOptions options = application.getEnvironmentOptions();
options.addGroup(new MyOptionsGroup());
```

Example of accessing environment options

```
Application application = Application.getInstance();
EnvironmentOptions options = application.getEnvironmentOptions();
int imageDpi = options.getGeneralOptions().getImageResolutionDpi();
```

You can find the code examples in
- *<programinstallation directory>\openapi\examples\environmentoptions*

Browser

The Model Browser (Containment tree, Diagrams tree, etc.) is a based GUI part for displaying various model trees and panels

```
Browser browser = Application.getInstance().getMainFrame().getBrowser();
```

The browser has five trees by default (some of them are be default hidden):

- containment
- diagrams
- inheritance
- extensions
- search results

The browser has 3 panel by default

- documentation
- properties

928 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/options/AbstractPropertyOptionsGroup.html

- zoom

Last activated tree is marked as active:

```
BrowserTabTree activeTree = browser.getActiveTree();
```

Every tree is based on Swing JTree and all manipulations can be done by using the API provided by Swing:

```
JTree tree = activeTree.getTree();
```

Selected nodes are accessible in the following way:

```
Node[] nodes = activeTree.getSelectedNodes();
```

Node is derived from *javax.swing.tree.DefaultMutableTreeNode*.

Selecting/opening element in tree

In order to select some element in tree, you need

- Open(activate) a tree using *com.nomagic.magicdraw.ui.browser.Browser#getContainmentTree*[929];
- Call com.nomagic.magicdraw.ui.browser.Tree#openNode(com.nomagic.magicdraw.uml.BaseElement)[930].

Related pages

Adding custom panel or tree into browser

Use com.nomagic.magicdraw.ui.browser.Browser.addBrowserInitializer([931] Browser.BrowserInitializer[932])[933] to register a custom browser initializer and add your custom panel or tree into a browser.

929 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ui/browser/Browser.html#getContainmentTree--
930 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ui/browser/Tree.html#openNode-com.nomagic.magicdraw.uml.BaseElement-
931 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ui/browser/Browser.html#addBrowserInitializer-com.nomagic.magicdraw.ui.browser.Browser.BrowserInitializer-
932 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ui/browser/Browser.BrowserInitializer.html
933 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ui/browser/Browser.html#addBrowserInitializer-com.nomagic.magicdraw.ui.browser.Browser.BrowserInitializer-

> You can find the code examples in
> - *<programinstallation directory>\openapi\examples\browserpanel*

Configuring node text and icon

You might need to change a text or icon for some Elements(nodes) in trees. For example, the adorn standard Element icon for showing that Element is annotated like the Validation framework does.

Option #1 for changing just a text

The text can be changed using a custom representation provider as described in Creating textual element representation(see page 2246). Using this approach, the text will be changed everywhere in UI, not just in trees.

Option #2 for changing a node text or icon

Use *com.nomagic.magicdraw.ui.browser.TreeNodeAdornmentManager.addIconAdornment(*[934] IconAdornment[935]*)* to register a node icon customizer.

Use TreeNodeAdornmentManager.addTextAdornment([936]TextAdornment[937])[938] to register a node text customizer.

Related pages

- Creating textual element representation(see page 2246)

Using and extending other UI components

Open API exposes some reusable and customizable UI components. They can be used for example from custom actions.

Configuring element Specification window

The Open API provides a way to configure the elements' Specification windows(see page 913). With your own configurator you can create new nodes or remove already existing nodes. Nodes are items of the tree visible on the left side in every Specification window(see page 913).

---

934 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ui/browser/TreeNodeAdornmentManager.html#addIconAdornment-com.nomagic.magicdraw.ui.IconAdornment-
935 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ui/IconAdornment.html
936 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ui/browser/TreeNodeAdornmentManager.html#addTextAdornment-com.nomagic.magicdraw.ui.browser.TextAdornment-
937 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ui/browser/TextAdornment.html
938 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ui/browser/TreeNodeAdornmentManager.html#addTextAdornment-com.nomagic.magicdraw.ui.browser.TextAdornment-

The com.nomagic.magicdraw.ui.dialogs.specifications.SpecificationDialogManager[939] class should be used for registering a specification dialog configurator *com.nomagic.magicdraw.ui.dialogs.specifications.configurator.ISpecificationNodeConfigurator*[940].

More information is available in javadoc[941].

> You can find the code examples in *<programinstallation directory>/openapi/examples/elementspecification*

Showing Element seletion dialog

Selecting Elements via Element Selection Dialog

```
    // Use ElementSelectionDlgFactory.create(...) methods to create the element
selection dialog.
    Frame dialogParent = MDDialogParentProvider.getProvider().getDialogParent();
    ElementSelectionDlg dlg = ElementSelectionDlgFactory.create(dialogParent);

    // Use ElementSelectionDlgFactory.initSingle(...) methods to initialize the
dialog with a single element selection mode.
    ElementSelectionDlgFactory.initSingle(....);

    // Use ElementSelectionDlgFactory.initMultiple(...) methods to initialize the
dialog with a multiple element selection mode.
    ElementSelectionDlgFactory.initMultiple(....);

    // Display the dialog for the user to select elements.
    dlg.setVisible(true);

    // Check if the user has clicked "Ok".
    if (dlg.isOkClicked())
    {
         // Get the selected element in a single selection mode.
        BaseElement selected = dlg.getSelectedElement();

        // Get selected elements in a multiple selection mode.
        BaseElement selected = dlg.getSelectedElements();
    }
```

> You can find the code examples in *<programinstallation directory>/openapi/examples/elementselection*

939 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ui/dialogs/specifications/SpecificationDialogManager.html
940 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ui/dialogs/specifications/configurator/ISpecificationNodeConfigurator.html
941 http://jdocs.nomagic.com/190beta/

Showing question, error, warning dialogs

Use com.nomagic.magicdraw.core.GUILog[942] class to show various simple UI dialogs.

Error dialog

Use *GUILog.showError(java.lang.String)*[943]

Use GUILog.showError(java.awt.Frame, java.lang.String, java.lang.Throwable)[944]to show error dialog with exception stack trace

Warning dialog

Use GUILog.showWarning(java.lang.String)[945]

Question dialog

Use GUILog.showQuestion(java.lang.String)[946]

Message dialog

Use GUILog.showMessage(java.lang.String)[947]

Use *GUILog.showHTMLMessage(java.lang.String,*[948]GUILog.URLActionHandler[949]*)* to show html message with some active links

Log messages into Message Window

Use GUILog.log(java.lang.String)[950] *to* add some information into a Message Window

See Showing notifications, adding text into Message Window(see page 2312) for more information about showing Notifications and appending text into a Message Window

For more information about other API choices see javadoc[951].

Related pages

- Showing notifications, adding text into Message Window(see page 2312)

942 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/GUILog.html
943 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/GUILog.html#showError-java.lang.String-
944 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/GUILog.html#showError-java.awt.Frame-java.lang.String-java.lang.Throwable-
945 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/GUILog.html#showWarning-java.lang.String-
946 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/GUILog.html#showQuestion-java.lang.String-
947 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/GUILog.html#showMessage-java.lang.String-
948 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/GUILog.html#showHTMLMessage-java.lang.String-com.nomagic.magicdraw.core.GUILog.URLActionHandler-
949 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/GUILog.URLActionHandler.html
950 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/GUILog.html#log-java.lang.String-
951 http://jdocs.nomagic.com/190beta/

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Showing notifications, adding text into Message Window

Use API described bellow to show Notifications (a balloon windows on the bottom right corner of application window) and appending text into a Message Window.

NotificationManager

This is an entry point to display notifications in MagicDraw. Two types of notifications can be displayed: application level and container level. Container lever notifications are displayed in special container that implements *com.nomagic.ui.notification.NotificationsContainer*[952] interface.

Use com.nomagic.magicdraw.ui.notification.NotificationManage.[953]

Use com.nomagic.magicdraw.core.GUILog.log(java.lang.String)[954] *to* add some information into a Message Window,

com.nomagic.magicdraw.ui.notification.Notification.getContext()[955]controls where notification is logged in the Message Window - into Environment or Project tab.

GUILog

Use GUILog.log(java.lang.String)[956] to put a text into Project Message Window message.

Use GUILog.log([957]Notification[958], boolean)[959] to log information from notification into a Message Window.

Use *GUILog.openLog()*[960] to open Message Window.

For more details about every specific kind of a property, see javadoc[961].

> You can find the code examples in *<programinstallation directory>/openapi/examples/notifications*

952 http://jdocs.nomagic.com/190beta/com/nomagic/ui/notification/NotificationsContainer.html
953 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ui/notification/NotificationManager.html
954 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/GUILog.html#log-java.lang.String-
955 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ui/notification/Notification.html#getContext--
956 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/GUILog.html#log-java.lang.String-
957 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/GUILog.html#log-com.nomagic.magicdraw.ui.notification.Notification-boolean-
958 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ui/notification/Notification.html
959 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/GUILog.html#log-com.nomagic.magicdraw.ui.notification.Notification-boolean-
960 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/GUILog.html#openLog--
961 http://jdocs.nomagic.com/190beta/

Related pages

- Showing question, error, warning dialogs(see page 2312)

Creating PopupMenu or other menu

API provides a way to create a popup menu from actions. Of course, you can use the standard java api for that, but we provide our own creators if you use our Actions.

Use com.nomagic.magicdraw.actions.MenuCreatorFactory.getMenuCreator()[962] to get an instance of a creator.

Use *com.nomagic.actions.ActionsMenuCreator.createPopupMenu(*[963]ActionsManager[964]*)* to create a popup menu.

Steps to create a popup menu would be

1. Create an instance of com.nomagic.actions.ActionsManager[965].
2. Add *com.nomagic.actions.NMAction*[966] actions and com.nomagic.actions.ActionsCategory[967] categories into *ActionsManager*[968].
3. Call *ActionsMenuCreator.createPopupMenu(*[969]ActionsManager[970]*)* to create a popup for your actions.

Related pages

- Creating new actions(see page 2197)

Adding custom project window

MagicDraw allows to open internal windows for the project and display some project related information in these windows. For example, Validation Result, Used By, Depend on, and etc. It is possible to add a custom internal window using API.

---

962 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/actions/MenuCreatorFactory.html#getMenuCreator--
96 http://jdocs.nomagic.com/190beta/com/nomagic/actions/ActionsMenuCreator.html#createPopupMenu-com.nomagic.actions.ActionsManager-
964 http://jdocs.nomagic.com/190beta/com/nomagic/actions/ActionsManager.html
965 http://jdocs.nomagic.com/190beta/com/nomagic/actions/ActionsManager.html
966 http://jdocs.nomagic.com/190beta/com/nomagic/actions/NMAction.html
967 http://jdocs.nomagic.com/190beta/com/nomagic/actions/ActionsCategory.html
968 http://jdocs.nomagic.com/190beta/com/nomagic/actions/ActionsManager.html
96 http://jdocs.nomagic.com/190beta/com/nomagic/actions/ActionsMenuCreator.html#createPopupMenu-com.nomagic.actions.ActionsManager-
970 http://jdocs.nomagic.com/190beta/com/nomagic/actions/ActionsManager.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Use *com.nomagic.magicdraw.ui.ProjectWindowsManager.ConfiguratorRegistry.addConfigurator(*[971] ProjectWindowsConfigurator[972]*)* to register project window configurator.
- Implement *com.nomagic.magicdraw.ui.ProjectWindowsConfigurator*[973] and add window using ProjectWindowsManager.addWindow([974]ProjectWindow[975])[976].

> You can find the code examples in *<programinstallation directory>/openapi/examples/projectwindow*

Generic tables

A generic table is a diagram with a specific stereotype. It is used to represent the element properties in a table form. It can represent different element types in one table and show all their properties. Table rows represent elements, and columns represent element properties. For more information about generic tables and their managing, see “Generic Table” in MagicDraw UserManual.pdf.

The Open API of modeling tools developed by No Magic Inc. provides the *com.nomagic.generictable.GenericTableManager*[977] class for creating and manipulating generic tables.

This API is used for creating a generic table, adding, editing, and removing table columns and rows, as well as getting cell values.

> Find the Generic Table Manager Example Plugin in *<MagicDraw_installation_directory>\openapi\examples\generictablemanager*.

Creating generic table

To create a generic table, use the *GenericTableManager.createGenericTable(*[978]Project[979]*, java.lang.String)*[980] method.

```
Diagram createdDiagram = GenericTableManager.createGenericTable(project, "Generic
table name");
```

---

971 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ui/ProjectWindowsManager.ConfiguratorRegistry.html#addConfigurator-com.nomagic.magicdraw.ui.ProjectWindowsConfigurator-
972 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ui/ProjectWindowsConfigurator.html
973 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ui/ProjectWindowsConfigurator.html
974 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ui/ProjectWindowsManager.html#addWindow-com.nomagic.magicdraw.ui.ProjectWindow-
975 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ui/ProjectWindow.html
976 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ui/ProjectWindowsManager.html#addWindow-com.nomagic.magicdraw.ui.ProjectWindow-
977 http://jdocs.nomagic.com/190beta/com/nomagic/generictable/GenericTableManager.html
978 http://jdocs.nomagic.com/190beta/com/nomagic/generictable/GenericTableManager.html#createGenericTable-com.nomagic.magicdraw.core.Project-java.lang.String-
979 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Project.html
980 http://jdocs.nomagic.com/190beta/com/nomagic/generictable/GenericTableManager.html#createGenericTable-com.nomagic.magicdraw.core.Project-java.lang.String-

Setting generic table filter

To set a generic table element types, use the *GenericTableManager.setTableElementTypes(*[981]Diagram[982]*, java.util.List<java.lang.Object>)*[983] method. The type of the element types list could be the Class or the Stereotype. See an example of the table element types:

```
//The table element types list
Set<Class> set = new HashSet<Class>();
List<Object> tableElementTypes = new ArrayList<Object>();

//Get all row elements
rowElements = GenericTableManager.getRowElements(createdDiagram);
for (int i = 0; i < rowElements.size(); i++)
{
    set.add(rowElements.get(i).getClassType());
}
tableElementTypes.addAll(set);
```

Getting generic table element types

To get the generic table element types, use the *GenericTableManager.getTableElementTypes(*[984]Diagram[985]*)* method:

```
List<Object> setElementTypes =
GenericTableManager.getTableElementTypes(createdDiagram);
```

Adding/getting columns

To add columns to the generic table, use the *GenericTableManager.addColumnsById(*[986]Diagram[987]*, java.util.List<java.lang.String>)*[988] method.

To get all possible columns from the generic table, use the *GenericTableManager.getPossibleColumnIds(*[989]Element[990]*)* method. To get set columns for the generic table, use the *GenericTableManager.getColumnIds(*[991]Diagram[992]*)* method.

[981] http://jdocs.nomagic.com/190beta/com/nomagic/generictable/GenericTableManager.html#setTableElementTypes-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram-java.util.List-
[982] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Diagram.html
[983] http://jdocs.nomagic.com/190beta/com/nomagic/generictable/GenericTableManager.html#setTableElementTypes-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram-java.util.List-
[984] http://jdocs.nomagic.com/190beta/com/nomagic/generictable/GenericTableManager.html#getTableElementTypes-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram-
[985] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Diagram.html
[986] http://jdocs.nomagic.com/190beta/com/nomagic/generictable/GenericTableManager.html#addColumnsById-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram-java.util.List-
[987] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Diagram.html
[988] http://jdocs.nomagic.com/190beta/com/nomagic/generictable/GenericTableManager.html#addColumnsById-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram-java.util.List-
[989] http://jdocs.nomagic.com/190beta/com/nomagic/generictable/GenericTableManager.html#getPossibleColumnIDs-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-
[990] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
[991] http://jdocs.nomagic.com/190beta/com/nomagic/generictable/GenericTableManager.html#getColumnIds-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram-
[992] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Diagram.html

```
List<String> columnIDs = GenericTableManager.getPossibleColumnIDs(elementType);
```

getColumnIds([993]Diagram[994]*)* returns columns from the generic table:

```
List<String> columnIDs = GenericTableManager.getColumnIds(createdDiagram);
GenericTableManager.addColumnsById(createdDiagram, columnIDs);
```

How to get *columnIDs*, please look at the *com.nomagic.magicdraw.examples.generictablemanager.GenericTableManagerExamplePlugin.GenericTableManagerAction* class in the *GenericTableManagerExamplePlugin.java* file. The file can be found in *<program_installation_directory>\openapi\examples\generictablemanager*.

Getting cell value

To get a value of the generic table call, use the GenericTableManager.getCellValue([995]Diagram[996], [997]Element[998], java.lang.String)[999] method:

```
Property mdProperty = GenericTableManager.getCellValue(createdDiagram, element,
columnId);
```

Getting column names

To get column names from the generic table, use the GenericTableManager.getColumnNames([1000]Diagram[1001])[1002] method.

---

993 http://jdocs.nomagic.com/190beta/com/nomagic/generictable/GenericTableManager.html#getColumnIds-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram-

994 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Diagram.html

995 http://jdocs.nomagic.com/190beta/com/nomagic/generictable/GenericTableManager.html#getCellValue-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-java.lang.String-

996 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Diagram.html

997 http://jdocs.nomagic.com/190beta/com/nomagic/generictable/GenericTableManager.html#getCellValue-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-java.lang.String-

998 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html

999 http://jdocs.nomagic.com/190beta/com/nomagic/generictable/GenericTableManager.html#getCellValue-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-java.lang.String-

1000 http://jdocs.nomagic.com/190beta/com/nomagic/generictable/GenericTableManager.html#getColumnNames-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram-

1001 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Diagram.html

1002 http://jdocs.nomagic.com/190beta/com/nomagic/generictable/GenericTableManager.html#getColumnNames-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram-

To get a single column name from the generic table, use the *GenericTableManager.getColumnNameById(*[1003]Diagram[1004]*, java.lang.String)*[1005] method.

```
List<String> columnNames = GenericTableManager.getColumnNames(createdDiagram);
String columnName = GenericTableManager.getColumnNameById(createdDiagram, columnID);
```

Getting row elements

To get row elements from the generic table, use the *GenericTableManager.getRowElements(*[1006]Diagram[1007]*)* method:

```
List<Element> rows = GenericTableManager.getRowElements(createdDiagram);
```

Getting visible row elements

To get visible row elements from the generic table, use the *GenericTableManager.getVisibleRowElements(*[1008]Diagram[1009]*)* method:

```
List<Element> rows = GenericTableManager.getVisibleRowElements(createdDiagram);
```

[1003] http://jdocs.nomagic.com/190beta/com/nomagic/generictable/GenericTableManager.html#getColumnNameById-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram-java.lang.String-

[1004] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Diagram.html

[1005] http://jdocs.nomagic.com/190beta/com/nomagic/generictable/GenericTableManager.html#getColumnNameById-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram-java.lang.String-

[1006] http://jdocs.nomagic.com/190beta/com/nomagic/generictable/GenericTableManager.html#getRowElements-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram-

[1007] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Diagram.html

[1008] http://jdocs.nomagic.com/190beta/com/nomagic/generictable/GenericTableManager.html#getVisibleRowElements-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram-

[1009] http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Diagram.html

Adding row element

To add a new row for the model element to the generic table, use the GenericTableManager.addRowElement([1010]Diagram[1011],[1012]Element[1013])[1014] method:

```
GenericTableManager.addRowElement(createdDiagram, element);
```

The element you want to add must be of the same type as one of generic table filter types.

Removing row element

To remove a row from the generic table, use the *GenericTableManager.removeRowElement(*[1015]Diagram[1016], Element[1017]*)* method:

```
GenericTableManager.removeRowElement(createdDiagram, element);
```

On this page

- Creating generic table(see page 2315)
- Setting generic table filter(see page 2316)
- Getting generic table element types(see page 2316)
- Adding/getting columns(see page 2316)
- Getting cell value(see page 2317)
- Getting column names(see page 2317)
- Getting row elements(see page 2318)
- Adding row element(see page 2319)
- Removing row element(see page 2319)

1010 http://jdocs.nomagic.com/190beta/com/nomagic/generictable/GenericTableManager.html#addRowElement-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-
1011 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Diagram.html
1012 http://jdocs.nomagic.com/190beta/com/nomagic/generictable/GenericTableManager.html#addRowElement-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-
1013 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
1014 http://jdocs.nomagic.com/190beta/com/nomagic/generictable/GenericTableManager.html#addRowElement-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-
1015 http://jdocs.nomagic.com/190beta/com/nomagic/generictable/GenericTableManager.html#removeRowElement-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-
1016 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Diagram.html
1017 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related pages

- Session management(see page 2224)

Working with Dependency Matrix

Dependency Matrix(see page 334) is a special type of diagram, that is, a diagram with a specific stereotype applied. Modeling tools developed by No Magic Inc. provide an API to

- Create custom types of the Dependency Matrix.
- Customize a Dependency Matrix as follows:
  - Configure the command bar, shortcut menus of the row or column header element, cell, and filter panel.
  - Assign shortcut keys for commands.
  - Define custom dependencies that can be created in the matrix.
  - Specify the appearance of the matrix, that is, change the default colors of the cell and both row and column headers, assign custom icons to represent dependencies, and so forth.
  - Define cases when specific dependencies should be updated without rebuilding the whole matrix.
  - Export the data of a Dependency Matrix to any format.

You can find the code examples in
- *<programinstallation directory>\openapi\examples\dependencymatrix*

Related pages

Creating a custom dependency matrix type

To create a custom dependency matrix type:

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

1. Create a new plugin to register components of the new dependency matrix type.(see page 2320)
2. Register the new dependency matrix type that it was added to modeling tool toolbars and menus(see page 2321).

Creating a plugin for custom dependency matrix type

Use the *com.nomagic.magicdraw.dependencymatrix.configuration.DependencyMatrixConfigurator*[1018] class to register components of the new dependency matrix type as shown in the following example:

```
    public class DependencyMatrixExample extends Plugin
    {
        @Override
        public void init()
        {
            DependencyMatrixConfigurator.registerConfiguration(new
DependencyMatrixSampleConfigurator("Sample Extended Matrix"));
        }

        @Override
        public boolean close()
        {
            return true;
        }

        @Override
        public boolean isSupported()
        {
            return true;
        }
    }
```

Remember the name of the new dependency matrix type that you pass as an argument while creating a new instance of the *DependencyMatrixConfigurator*[1019]class (in the preceding example, it is “Sample Extended Matrix”). You will have to specify the exact name when registering a new diagram type.

The sample of more detailed configuration code is provided in *<program installation folder>\openapi\examples\dependencymatrix*.

---

801 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/dependencymatrix/configuration/DependencyMatrixConfigurator.html

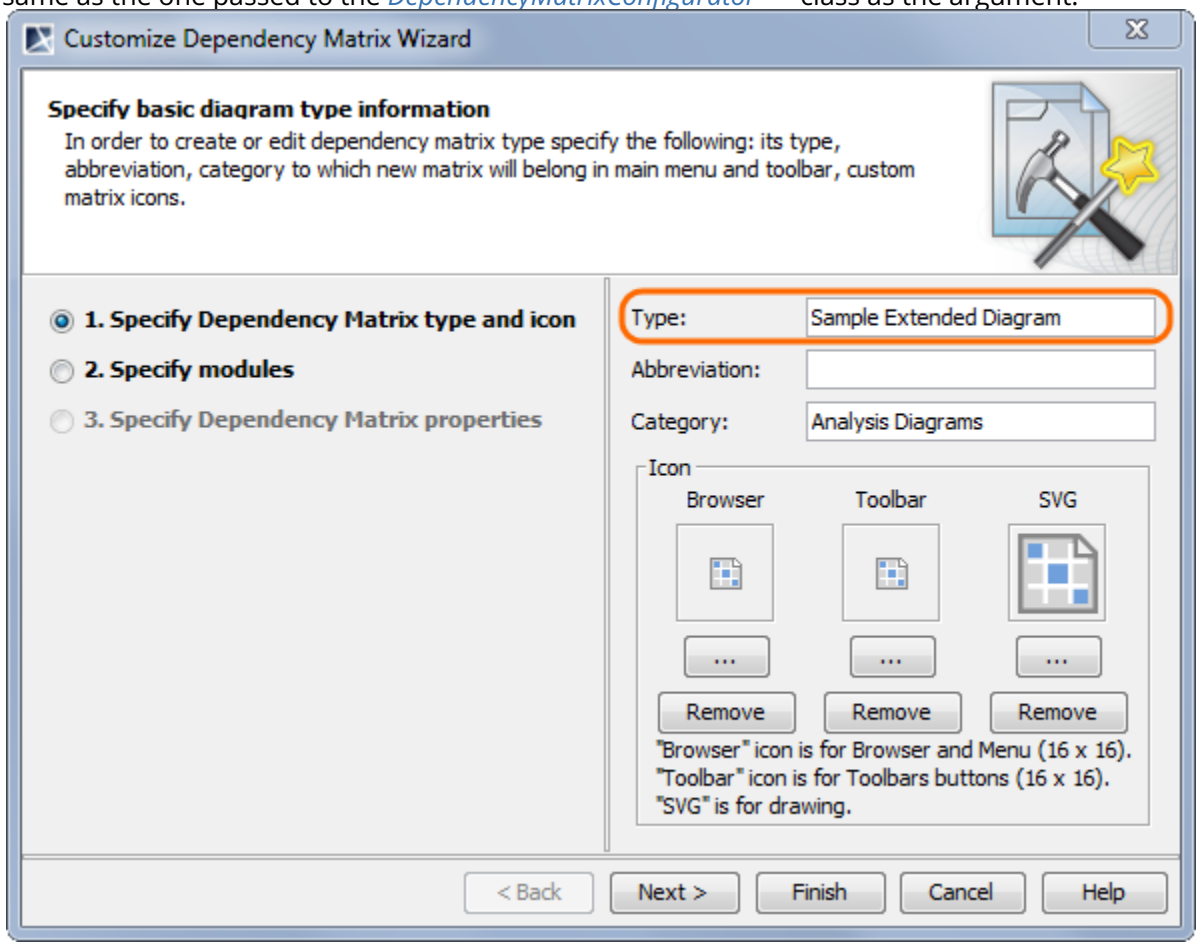
901 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/dependencymatrix/configuration/DependencyMatrixConfigurator.html

Registering a new dependency matrix type in UI

Once the plugin (see page 2157)is created, you have to register the new dependency matrix type by using the **Customize Dependency Matrix Wizard** dialog. Only then it will be added to program toolbars and menus.

To register a new dependency matrix type

1. Start a modeling tool, for example, MagicDraw.
2. On the **Diagrams** menu, click **Customize**. The **Customize Diagrams** dialog opens.
3. Click **Create** and then select **Dependency Matrix Type**. The **Customize Dependency Matrix Wizard** dialog opens.
4. In the **Type** box, specify the name of the new dependency matrix type. The value must be the same as the one passed to the *DependencyMatrixConfigurator*[1020]class as the argument.



Specifying name of new Dependency Matrix type

1020 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/dependencymatrix/configuration/DependencyMatrixConfigurator.html

5. In the **Abbreviation** box, specify the abbreviation of the new dependency matrix type.

6. Click **Finish**.

7. Restart your modeling tool.

You can now create a custom dependency matrix.

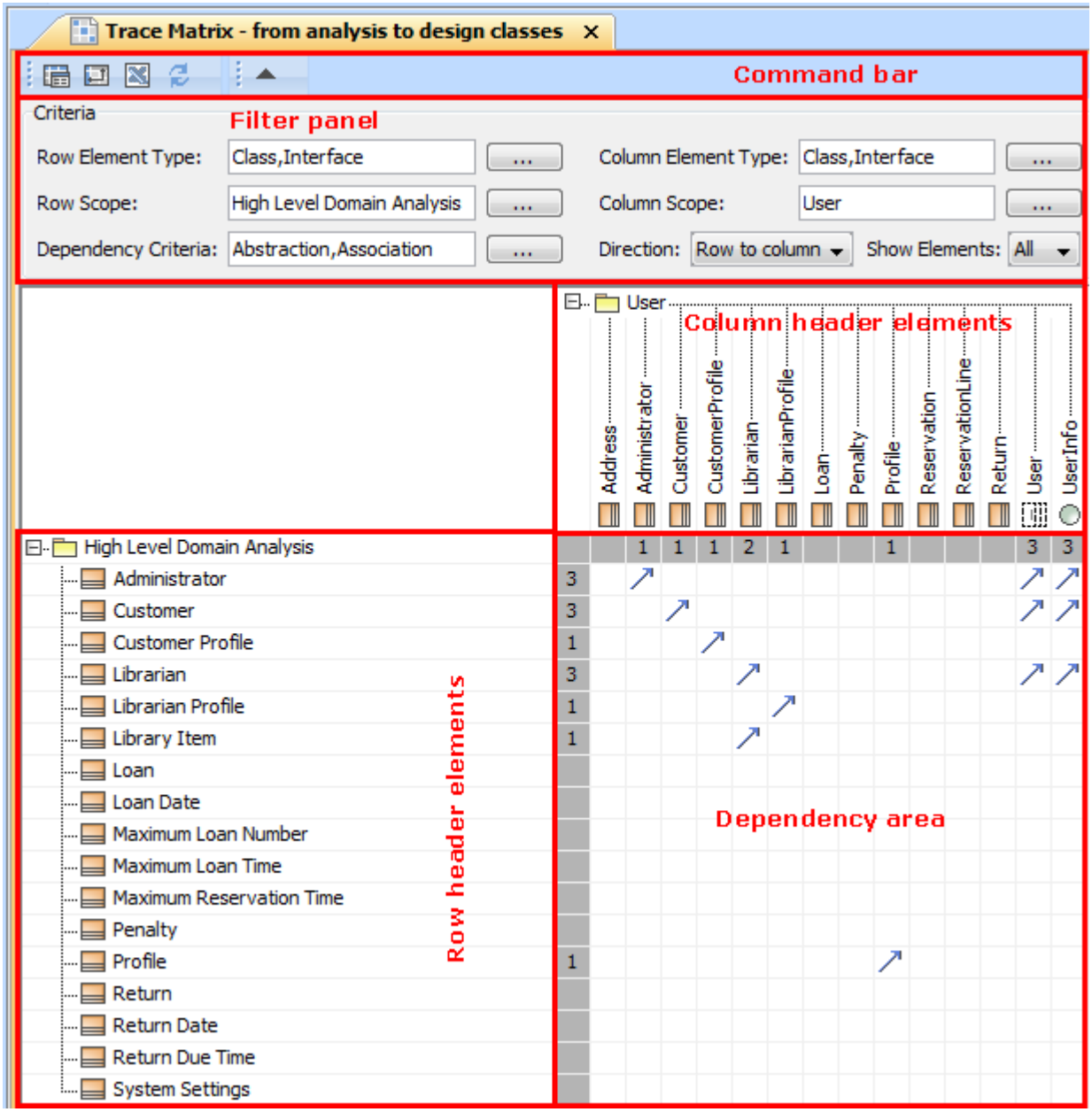
> For more information on how to customize a new dependency matrix type by using the **Customize Dependency Matrix Wizard** dialog, **see "Creating New Dependency Matrix Type" in the MagicDraw UMLProfiling&DSL UserGuide.pdf**.

Customizing a dependency matrix

The topic explains how to configure the following components of the dependency matrix:

- Command bar
- Shortcut keys
- Shortcut menus of the row or column header element, cell, and filter panel
- Cell entries (dependencies)
- Appearance of cells and headers of rows and columns

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

The names of the components differ in other documentation as follows:
- "Toolbar" is used for "Command bar"
- "**Criteria** area" is used for "Filter panel"
- "Row elements" is used for "Row header elements"

Related pages

Configuring the command bar and shortcut keys

You can add custom commands to the command bar of a dependency matrix as well as assign custom shortcut keys for the commands.

To add a new command to the command bar, add an instance of the *com.nomagic.actions.AMConfigurator*[1021] interface to the *com.nomagic.magicdraw.actions.ActionsConfiguratorsManager*[1022] class, as is shown in the following example:

```
    ActionsConfiguratorsManager.getInstance().addDiagramCommandBarConfigurator(
            DependencyMatrixConfigurator.DEFAULT_DEPENDENCY_MATRIX_DIAGRAM_TYPE, new
AMConfigurator()
    {
        @Override
        public void configure(ActionsManager mngr)
        {
            mngr.addCategory(new ActionsCategory("ID", "My Category"));
        }

        @Override
        public int getPriority()
        {
             return AMConfigurator.HIGH_PRIORITY;
        }
    });
```

When customizing the default dependency matrix type, provide the name defined in the com.nomagic.magicdraw.dependencymatrix.configuration.DependencyMatrixConfigurator.DEFAULT_DEPENDENCY_MATRIX_DIAGRAM_TYPE[1023] constant. And when customizing a custom dependency matrix type, provide the name defined in the instance of the *DependencyMatrixConfigurator*[1024]class as well as in the **Customize Dependency Matrix Wizard** dialog.

1021 http://jdocs.nomagic.com/190beta/com/nomagic/actions/AMConfigurator.html
1022 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/actions/ActionsConfiguratorsManager.html
1023 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/dependencymatrix/configuration/DependencyMatrixConfigurator.html#DEFAULT_DEPENDENCY_MATRIX_DIAGRAM_TYPE
1024 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/dependencymatrix/configuration/DependencyMatrixConfigurator.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

The same code example can be used for the shortcut key configuration:

```
ActionsConfiguratorsManager.getInstance().addDiagramShortcutsConfigurator(...)
```

Configuring shortcut menus of a row or column header element and a filter panel

To add a custom command to the shortcut menu of the row or column header element, or to the one of the filter panel, use the *com.nomagic.magicdraw.dependencymatrix.configuration.DependencyMatrixActionRegistry*[1025] class.

The *com.nomagic.magicdraw.dependencymatrix.configuration.DependencyMatrixAMConfigurator*[1026] interface has a single method that adds custom commands. The location of a new command depends on the parameters, such as the following:

- If the collection of element nodes is empty, the command will be added to the shortcut menu of the filter panel.
- If the collection of element nodes is not empty and *forRow* is *true*, the command will be added to the shortcut menu of the row header.
- If the collection of element nodes is not empty and *forRow* is *false* the command will be added to the shortcut menu of the column header element.

Configuring cell entries - dependencies

In order to get the dependencies between row and column elements in the dependency matrix, components of the *com.nomagic.magicdraw.dependencymatrix.datamodel.cell.DependencyExtractor*[1027] interface are used. These components allow to

- Define custom dependencies that can be created in a cell.
- Add shortcut menu commands for navigation (other than opening the Specification window(see page 913) of the element and selecting the element in the Containment tree).
- Construct smart listener configurations that define cases when some element dependencies should be updated instantly (without the full rebuild of the matrix).

Use the *com.nomagic.magicdraw.dependencymatrix.configuration.DependencyMatrixConfigurator.configureDependencyHandlers(java.util.Collection*<[1028]DependencyExtractor[1029]>, java.util.Collection<[1030]DependencyEditor[1031]>

1025 http://jdocs.nomagic.com/190_sp3/com/nomagic/magicdraw/dependencymatrix/configuration/DependencyMatrixActionRegistry.html

1026 http://jdocs.nomagic.com/190_sp3/com/nomagic/magicdraw/dependencymatrix/configuration/DependencyMatrixAMConfigurator.html

1027 http://jdocs.nomagic.com/190_sp3/com/nomagic/magicdraw/dependencymatrix/datamodel/cell/DependencyExtractor.html

1028 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/dependencymatrix/configuration/DependencyMatrixConfigurator.html#configureDependencyHandlers-java.util.Collection-java.util.Collection-

1029 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/dependencymatrix/datamodel/cell/DependencyExtractor.html

1030 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/dependencymatrix/configuration/DependencyMatrixConfigurator.html#configureDependencyHandlers-java.util.Collection-java.util.Collection-

) method to register a custom dependency extractor. The method can remove the default dependency extractor from the collection and leave only a custom extractor.

The most important method is com.nomagic.magicdraw.dependencymatrix.datamodel.cell.DependencyExtractor.getDependencies(ElementNode, ElementNode)[1032]. It is called to add dependencies in the matrix cell. By using custom algorithms you can create additional instances of the *com.nomagic.magicdraw.dependencymatrix.datamodel.cell.DependencyEntry*[1033] class for each cell.

Configuring the shortcut menu of a cell

All commands of the shortcut menu of a cell, are available only from the components of the *com.nomagic.magicdraw.dependencymatrix.datamodel.editing.DependencyEditor*[1034] interface. These components allow for

- Defining if new dependency can be created in the cell.
- Defining if current cell value can be edited.
- Adding custom actions for creating or editing dependencies.

Use the *DependencyMatrixConfigurator.configureDependencyHandlers(java.util.Collection<*[1035] DependencyExtractor[1036]>, java.util.Collection<[1037]DependencyEditor[1038]>*)* method to register a custom dependency editor. The method implementation can remove the default dependency editor from the collection and leave only a custom editor.

Dependency matrix cells with the default renderer can be displayed either as editable or as read-only. If it is possible to create or delete at least one dependency in the cell, the cell is displayed as editable, that is, the cell's background is white.

1032 http://jdocs.nomagic.com/190_sp3/com/nomagic/magicdraw/dependencymatrix/datamodel/cell/DependencyExtractor.html#getDependencies-com.nomagic.magicdraw.dependencymatrix.datamodel.grouping.ElementNode-com.nomagic.magicdraw.dependencymatrix.datamodel.grouping.ElementNode-

1033 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/dependencymatrix/datamodel/cell/DependencyEntry.html

1034 http://jdocs.nomagic.com/190_sp3/com/nomagic/magicdraw/dependencymatrix/datamodel/editing/DependencyEditor.html

1035 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/dependencymatrix/configuration/DependencyMatrixConfigurator.html#configureDependencyHandlers-java.util.Collection-java.util.Collection-

1036 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/dependencymatrix/datamodel/cell/DependencyExtractor.html

1037 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/dependencymatrix/configuration/DependencyMatrixConfigurator.html#configureDependencyHandlers-java.util.Collection-java.util.Collection-

1038 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/dependencymatrix/datamodel/editing/DependencyEditor.html

When adding a new command, the *DependencyEditor.canCreate(*[1039]PersistenceManager[1040], ElementNode[1041], ElemenNode[1042], AbstractMatrixCell[1043]*)* method is first called and if it returns *true*, the *DependencyEditor.createAddActions(*[1044]PersistenceManager[1045], ElementNode[1046], ElementNode[1047], AbstractMatrixCell[1048], ActionsCategory[1049], ActionsCategory[1050], ActionsCategory[1051]*)* method is called to add the command.

[1039] http://jdocs.nomagic.com/190_sp3/com/nomagic/magicdraw/dependencymatrix/datamodel/editing/DependencyEditor.html#canCreate-com.nomagic.magicdraw.dependencymatrix.persistence.PersistenceManager-com.nomagic.magicdraw.dependencymatrix.datamodel.ElementNode-com.nomagic.magicdraw.dependencymatrix.datamodel.ElementNode-com.nomagic.magicdraw.dependencymatrix.datamodel.cell.AbstractMatrixCell-

[1040] http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/dependencymatrix/persistence/PersistenceManager.html

[1041] http://jdocs.nomagic.com/190_sp3/com/nomagic/magicdraw/dependencymatrix/datamodel/ElementNode.html

[1042] http://jdocs.nomagic.com/190_sp3/com/nomagic/magicdraw/dependencymatrix/datamodel/ElementNode.html

[1043] http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/dependencymatrix/datamodel/cell/AbstractMatrixCell.html

[1044] http://jdocs.nomagic.com/190_sp3/com/nomagic/magicdraw/dependencymatrix/datamodel/editing/DependencyEditor.html#createAddActions-com.nomagic.magicdraw.dependencymatrix.persistence.PersistenceManager-com.nomagic.magicdraw.dependencymatrix.datamodel.ElementNode-com.nomagic.magicdraw.dependencymatrix.datamodel.ElementNode-com.nomagic.magicdraw.dependencymatrix.datamodel.cell.AbstractMatrixCell-com.nomagic.actions.ActionsCategory-com.nomagic.actions.ActionsCategory-com.nomagic.actions.ActionsCategory-

[1045] http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/dependencymatrix/persistence/PersistenceManager.html

[1046] http://jdocs.nomagic.com/190_sp3/com/nomagic/magicdraw/dependencymatrix/datamodel/ElementNode.html

[1047] http://jdocs.nomagic.com/190_sp3/com/nomagic/magicdraw/dependencymatrix/datamodel/ElementNode.html

[1048] http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/dependencymatrix/datamodel/cell/AbstractMatrixCell.html

[1049] http://jdocs.nomagic.com/190beta/com/nomagic/actions/ActionsCategory.html

[1050] http://jdocs.nomagic.com/190beta/com/nomagic/actions/ActionsCategory.html

[1051] http://jdocs.nomagic.com/190beta/com/nomagic/actions/ActionsCategory.html

When editing a command, the *DependencyEditor.canEdit(*[1052]PersistenceManager[1053], ElementNode[1054], ElementNode[1055], AbstractMatrixCell[1056]*)* method is first called and if it returns *true*, the DependencyEditor.createEditActions([1057]PersistenceManager[1058],[1059]ElementNode[1060],[1061]

[1052] http://jdocs.nomagic.com/190_sp3/com/nomagic/magicdraw/dependencymatrix/datamodel/editing/DependencyEditor.html#canEdit-com.nomagic.magicdraw.dependencymatrix.persistence.PersistenceManager-com.nomagic.magicdraw.dependencymatrix.datamodel.ElementNode-com.nomagic.magicdraw.dependencymatrix.datamodel.ElementNode-com.nomagic.magicdraw.dependencymatrix.datamodel.cell.AbstractMatrixCell-

[1053] http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/dependencymatrix/persistence/PersistenceManager.html

[1054] http://jdocs.nomagic.com/190_sp3/com/nomagic/magicdraw/dependencymatrix/datamodel/ElementNode.html

[1055] http://jdocs.nomagic.com/190_sp3/com/nomagic/magicdraw/dependencymatrix/datamodel/ElementNode.html

[1056] http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/dependencymatrix/datamodel/cell/AbstractMatrixCell.html

[1057] http://jdocs.nomagic.com/190_sp3/com/nomagic/magicdraw/dependencymatrix/datamodel/editing/DependencyEditor.html#createEditActions-com.nomagic.magicdraw.dependencymatrix.persistence.PersistenceManager-com.nomagic.magicdraw.dependencymatrix.datamodel.ElementNode-com.nomagic.magicdraw.dependencymatrix.datamodel.ElementNode-com.nomagic.magicdraw.dependencymatrix.datamodel.cell.AbstractMatrixCell-com.nomagic.actions.ActionsCategory-com.nomagic.actions.ActionsCategory-

[1058] http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/dependencymatrix/persistence/PersistenceManager.html

[1059] http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/dependencymatrix/datamodel/editing/DependencyEditor.html#createEditActions-com.nomagic.magicdraw.dependencymatrix.persistence.PersistenceManager-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.magicdraw.dependencymatrix.datamodel.cell.AbstractMatrixCell-com.nomagic.actions.ActionsCategory-com.nomagic.actions.ActionsCategory-

[1060] http://jdocs.nomagic.com/190_sp3/com/nomagic/magicdraw/dependencymatrix/datamodel/ElementNode.html

[1061] http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/dependencymatrix/datamodel/editing/DependencyEditor.html#createEditActions-com.nomagic.magicdraw.dependencymatrix.persistence.PersistenceManager-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.magicdraw.dependencymatrix.datamodel.cell.AbstractMatrixCell-com.nomagic.actions.ActionsCategory-com.nomagic.actions.ActionsCategory-

ElementNode[1062],[1063]AbstractMatrixCell[1064],[1065]ActionsCategory[1066],[1067]ActionsCategory[1068])[1069] method is called to edit the command.

Specifying the appearance of cells and the row or column headers

To customize the appearance of a dependency matrix, you have to register cell renderer components. There are three types of the renderers:

- A cell renderer
- A column header cell renderer
- A row header cell renderer

To register a cell renderer component, use the instance of the *com.nomagic.magicdraw.dependencymatrix.configuration.DependencyMatrixConfigurator*[1070] class, created while registering a custom dependency matrix. All these renderers should implement the standard *javax.swing.table.TableCellRenderer* interface.

The cell renderer is used to represent dependencies in a dependency matrix. The renderer component of a single cell gets either the value of the *Integer* class or the instance of the *com.nomagic.magicdraw.dependencymatrix.datamodel.cell.MatrixCellView*[1071] class. The *java.lang.Integer*

1062 http://jdocs.nomagic.com/190_sp3/com/nomagic/magicdraw/dependencymatrix/datamodel/ElementNode.html

1063 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/dependencymatrix/datamodel/editing/DependencyEditor.html#createEditActions-com.nomagic.magicdraw.dependencymatrix.persistence.PersistenceManager-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.magicdraw.dependencymatrix.datamodel.cell.AbstractMatrixCell-com.nomagic.actions.ActionsCategory-com.nomagic.actions.ActionsCategory-

1064 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/dependencymatrix/datamodel/cell/AbstractMatrixCell.html

1065 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/dependencymatrix/datamodel/editing/DependencyEditor.html#createEditActions-com.nomagic.magicdraw.dependencymatrix.persistence.PersistenceManager-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.magicdraw.dependencymatrix.datamodel.cell.AbstractMatrixCell-com.nomagic.actions.ActionsCategory-com.nomagic.actions.ActionsCategory-

1066 http://jdocs.nomagic.com/190beta/com/nomagic/actions/ActionsCategory.html

1067 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/dependencymatrix/datamodel/editing/DependencyEditor.html#createEditActions-com.nomagic.magicdraw.dependencymatrix.persistence.PersistenceManager-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.magicdraw.dependencymatrix.datamodel.cell.AbstractMatrixCell-com.nomagic.actions.ActionsCategory-com.nomagic.actions.ActionsCategory-

1068 http://jdocs.nomagic.com/190beta/com/nomagic/actions/ActionsCategory.html

1069 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/dependencymatrix/datamodel/editing/DependencyEditor.html#createEditActions-com.nomagic.magicdraw.dependencymatrix.persistence.PersistenceManager-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.magicdraw.dependencymatrix.datamodel.cell.AbstractMatrixCell-com.nomagic.actions.ActionsCategory-com.nomagic.actions.ActionsCategory-

1070 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/dependencymatrix/configuration/DependencyMatrixConfigurator.html

class represents dependency count. And the *MatrixCellView*[1072]class has a reference to the *com.nomagic.magicdraw.dependencymatrix.datamodel.cell.AbstractMatrixCell*[1073] class, which has all the information about the dependencies in the cell.

The row and column cell renderers works the same way. To get the particular row or column header element inside the cell, use the *com.nomagic.magicdraw.dependencymatrix.ui.table.renderer.RendererHelper.getHeaderElement(javax.swing.JTable, int, int)*[1074] method.

Accessing dependency matrix data

Internal data of a dependency matrix can be used outside the dependency matrix component. You can export the data to any required format.

Use the *com.nomagic.magicdraw.dependencymatrix.configuration.MatrixDataHelper*[1075] class to get the data component of a dependency matrix. You will get all row and column elements and access all cell information. You can also check, if the dependency matrix is empty.

The *MatrixDataHelper*[1076]class has three methods that are to be used to access data of the matrix. First of all you must check, if the matrix is already built (building the matrix means collecting all the needed data from the model). In some cases, when settings are changed or a lot of model changes occur, you have to rebuild the matrix to get the up-to-date information. Thus the best pattern to use is as follows:

```
MatrixData matrixData;
if (MatrixDataHelper.isRebuildNeeded(matrix))
{
    matrixData = MatrixDataHelper.buildMatrix(matrix);
}
else
{
    matrixData = MatrixDataHelper.getMatrixData(matrix);
}
```

> ⚠ The duration of the build process depends on both the *com.nomagic.magicdraw.dependencymatrix.datamodel.cell.DependencyExtractor*[1077] implementation and the size of the matrix.

1072 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/dependencymatrix/datamodel/cell/MatrixCellView.html
1073 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/dependencymatrix/datamodel/cell/AbstractMatrixCell.html
1074 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/dependencymatrix/ui/table/renderer/RendererHelper.html#getHeaderElement-javax.swing.JTable-int-int-
1075 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/dependencymatrix/configuration/MatrixDataHelper.html
1076 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/dependencymatrix/configuration/MatrixDataHelper.html
1077 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/dependencymatrix/datamodel/cell/DependencyExtractor.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Annotating the elements

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Using the Open API you can add an annotation to any base element (a model element or a symbol on the diagram pane). Annotations are used to highlight and adorn elements in the Containment tree and on the diagram pane. For example validation engine is using annotations for showing invalid elements.

Annotation has the following properties:

- severity - like an error, warning, info, debug, fatal
- kind - a string representing the annotation short name
- text - a string representing the annotation text
- target - a target base element
- actions - an optional list of actions. They are shown in a diagram on a symbol's smart manipulator.

To add or remove annotations, use *com.nomagic.magicdraw.annotation.AnnotationManager*[1078].

Do not forget to call *AnnotationManager.update()*[1079], when you are done with adding or removing in order to refresh UI.

More information is available in javadoc[1080].

You can find the code examples in
- *<programinstallation directory>\openapi\examples\annotations*

Creating validation rules

Validation rules and validation suites specify what will be validating and how. They also specify how a problem that is found by a validation rule can be solved.

A **validation rule** is a constraint with the applied stereotype «UML Standard Profile::Validation Profile::validationRule». Validation rules can validate model elements and non model elements (for example, presentation elements) as well. The UML metaclass specified as a constrained element defines the validation rule that validates elements of the specified metaclass. The stereotype specified as a constrained element specifies the validation rule that validates elements having the stereotype applied. A classifier specified as a constrained element specifies the validation rule that validates instances of the specified classifier. Validation rule's implementation can be OCL2.0 based or binary. Binary validation rules can be implemented in Java. OCL2.0 based validation rules are described using the OCL2.0 language. The validation rule can be global or local one. Global validation rules are executed only once per validation session. Local validation rules are executed for each model element. The validation rule is treated as global if:

- it is the OCL2.0 based validation rule, the OCL2.0 specification is valid, and the OCL2.0 specification does not use the "self" variable (explicitly or implicitly by using the property of a constrained element only)
- it is binary based, has the specified implementation class name, and it has no specified constrained elements.

1078 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/annotation/AnnotationManager.html
1079 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/annotation/AnnotationManager.html#update--
1080 http://jdocs.nomagic.com/190beta/

A **validation suite** is a package with the stereotype «UML Standard Profile::Validation Profile::validationSuite». The validation suite organizes several validation rules into a unit that can be used for performing the validation. Validation rules can be added or imported into the validation suite.

An **active validation suite** is a package with the stereotype «UML Standard Profile::Validation Profile::activeValidationSuite». Active validation rules can be checked constantly or on the model elements change. OCL2.0 or binary validation rules can be used in the active validation. We suggest to prefer binary, because they give the better performance.

An **annotation** defines the validation result. It contains information about what is incorrect, severity of the problem, and possible actions that can be used to solve the problem.

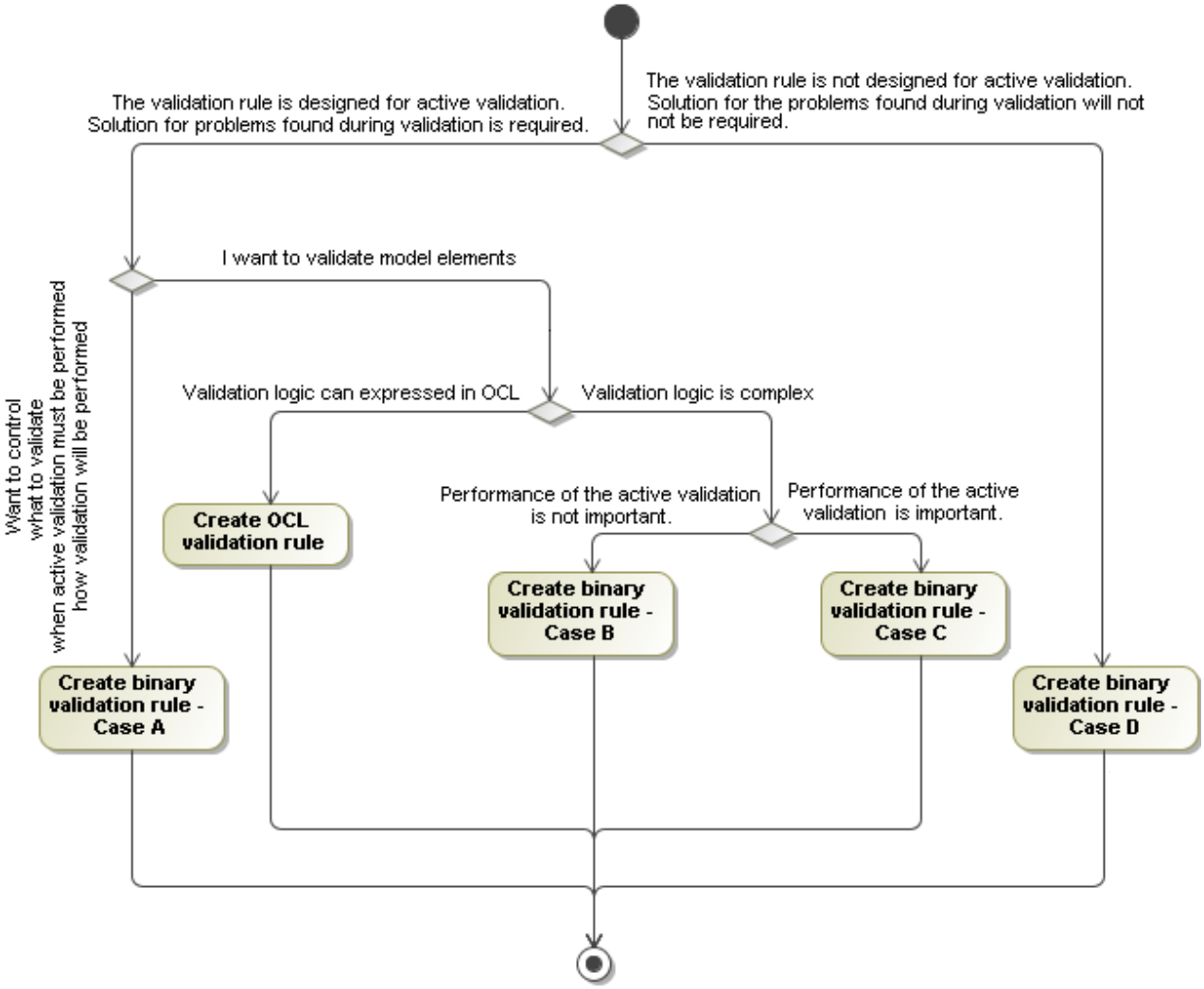
Related pages

- Annotating the elements(see page 2332)

Validation rule developer's roadmap

Validation rule developer’s roadmap allows faster understanding of steps required for creating a validation rule.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Related pages

- Creating validation rules
  (see page 2333)

Create OCL2.0 validation rule

The OCL2.0 validation rule describes the validation logic using OCL2.0.

To create the OCL2.0 validation rule

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

1. Create a constraint.
2. Set the stereotype «UML Standard Profile::Validation Profile::validationRule» for the validation rule.
3. Set the severity level, error message, and abbreviation.
4. Specify constrained element(s).
5. Specify the specification language OCL2.0
6. Enter the OCL2.0 expression as a body of the specification
7. Add/import the created validation rule to a validation suite

Related pages

- Creating validation rules(see page 2333)

The OCL2.0 validation rule can be used in an active validation suite. In this case the validation rule will be executed on any change of constrained elements that are from the validation scope. Executing of the validation rule might be triggered even if no properties important to the validation rule actually changed and this can slow down the active validating speed. In order to avoid degradation of the performance you can create a binary validation rule that uses the OCL2.0 expression to validate model elements but also provides additional information for the modeling tool that allows to optimize triggering of executing after model elements change. For example, we want to check whether all classes have names. This can be done by creating the OCL2.0 validation rule and specifying the OLC2.0 expression:

```
name <> ''
```

The problem is that such validation rule actually is interested in a class property name value change but it will be executed on every property value of a class change. How we can fix this problem and inform the modeling tool to execute the validation rule only on the class property name change:

1. Create a constraint.
2. Set the stereotype «UML Standard Profile::Validation Profile::validationRule» for the validation rule.
3. Set the severity level, error message, and abbreviation.
4. Specify constrained element(s).
5. Specify the specification language OCL2.0
6. Enter the OCL2.0 expression as a body of the specification
7. Create a Java class that extends the *com.nomagic.magicdraw.validation.DefaultValidationRuleImpl*[1081] class
8. Override the method DefaultValidationRuleImpl.getListenerConfigurations()[1082]

1081 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/validation/DefaultValidationRuleImpl.html
1082 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/validation/DefaultValidationRuleImpl.html#getListenerConfigurations--

```
        public Map<Class<? extends Element>, Collection<SmartListenerConfig>>
getListenerConfigurations()
        {
            Map<Class<? extends Element>, Collection<SmartListenerConfig>> configs =
                        new HashMap<Class<? extends Element>,
Collection<SmartListenerConfig>>();
            Collection<SmartListenerConfig> configsForClass = new
 ArrayList<SmartListenerConfig>();
            configsForClass.add(SmartListenerConfig.NAME_CONFIG);
            configs.put(com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class.class,
configsForClass);
            return configs;
        }
```

9. Specify the validation rule's implementation property value - a qualified name of the class

10. Add/import the created validation rule to a validation suite

See the *MyOCLBasedValidationRuleImpl.java* example in the *<program installation directory>\openapi\examples\validation* directory.

> ⚠ The implementation class must be in the classpath of the modeling tool or if the implementation class is in a plugin then the plugin's classloader must be registered to the validation system (see Binary validation rule in the plugin(see page 2345)).

Binary validation rule

Sometimes it is hard to specify the rule for some advanced concepts in OCL. For example, the string manipulation is rather weak in OCL2.0, hence the writing rule for string formats, parsing strings is very hard in OCL. In such cases the modeling tool offers possibility to write rules in Java language and invoke them when validating.

Common steps for creating a binary validation rule:

1. Create a constraint
2. Set the stereotype «UML Standard Profile::Validation Profile::validationRule» for the validation rule
3. Set the severity level, error message, and abbreviation
4. Specify constrained element(s)
5. Specify the specification language binary
6. Create a Java implementation class (see all possible cases(see page 2340))
7. Compile Java code, resulting in one or several classes.
8. Ensure that your modeling tool can find & load these classes. You can place them in the program classpath or use the modeling tool plugins mechanism. The same rules as writing the code for the open API are used here.

These steps are common to all possible binary validation cases and will not be repeated in each case description.

See the *MyBinaryValidationRuleImpl.java* example in the *<program installation directory>\openapi\examples\validation* directory.

The implementation class must be in the classpath of the program or if the implementation class is in a plugin then the plugin's classloader must be registered to the validation system (see Binary validation rule in the plugin(see page 2345)).

Related pages

- Creating validation rules(see page 2333)

Implementing a binary validation rule

Implementing Custom Validation Behaviour(see page 2340)

Implementing Rule to Validate Model Elements(see page 2340)

Optimizing Rule Execution for Active Validation(see page 0)

Implementing Validation Method (Passive Validation Only)(see page 2343)

Implementing Rule to Validate Presentation Elements (Symbols)(see page 2345)

Implementing Custom Validation Behaviour

If you want to control when the validation rule has to be executed and you want to control what must be validated and how, then you have to create a Java class and implement the interface *com.nomagic.magicdraw.validation.ValidationRule*[1083]. The class must have the public *no-arg* constructor. A value of the implementation property of the validation rule must be a qualified name of the implemented class.

This case is not recommended for validating model elements because the validation rule provider must implement not only validating of model elements but also other details that are provided by the modeling tool validation engine. In this case, the validation rule provider must correctly implement validating of model elements that are from the validation scope defined in project options, should take care that implementation would respect the project property *Exclude element from used read-only project*, and is responsible for implementing when the validation has to be executed.

Related pages

- Creating validation rules
  (see page 2333)

Implementing Rule to Validate Model Elements

1083 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/validation/ValidationRule.html

If you want to concentrate on implementation of validating model elements and delegate a task of detecting model element changes and project property changes to the validation engine then you have to create a Java class and implement *com.nomagic.magicdraw.validation.ElementValidationRuleImpl*[1084] and the interface. The class must have the public *no-arg* constructor. The value of the implementation property of the validation rule must be a qualified name of the implemented class. Note that If this rule is used in active validation, then it will be repeatedly executed after every single change in the model.

This example shows how to add custom solver action and set dynamic error message to validated element:

```
public Set<Annotation> run(Project project, Constraint constraint, Collection<?
extends Element> elements)
{
   Set<Annotation> result = new HashSet<>();

   for (Element element : elements)
   {
      if (!isValid(element))
      {
         List<NMAction> actions = new ArrayList<>();
         actions.add(new MyAction("MY_ACTION_ID", "Test", null));
         result.add(new Annotation(element, constraint, getErrorText(element),
actions));
      }
   }

   return result;
}

private static String getErrorText(Element element)
{
   return "Name for [" + element.getID() + "] should be longer than " +
MIN_NAME_LENGTH + " symbols.";
}

private static boolean isValid(Element element)
{
   NamedElement el = (NamedElement) element;
   return el.getName().length() > MIN_NAME_LENGTH;
}
```

Optimizing Rule Execution for Active Validation

If the validation rule is designed for the active validation then the validation performance becomes very important. In this case we want that the validation rule would be executed only when it is actually required, so the implementation of the validation rule must somehow inform the active validation engine which model element property changes are important to the validation rule. In order to create such validation rule you have implement *ElementValidationRuleImpl*[1085]and *com.nomagic.magicdraw.validation.SmartListenerConfigurationProvider*[1086]interfaces. The class must have

1084 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/validation/ElementValidationRuleImpl.html
1085 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/validation/ElementValidationRuleImpl.html
1086 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/validation/SmartListenerConfigurationProvider.html

the public *no-arg* constructor. The value of the implementation property of the validation rule must be a qualified name of the implemented class.

The *SmartListenerConfigurationProvider*[1087]interface defines a single method:

```
    Map<Class<? extends Element>, Collection<SmartListenerConfig>>
getListenerConfigurations()
```

The implementation of this method should return a map of classes derived from the *com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element*[1088] class to collection(s) of com.nomagic.uml2.ext.jmi.smartlistener.SmartListenerConfig[1089] objects. Another way would be to extend the *com.nomagic.magicdraw.validation.DefaultValidationRuleImpl*[1090] class and override the following method:

```
    Map<Class<? extends Element>, Collection<SmartListenerConfig>>
getListenerConfigurations()
```

In the Constraint Specification dialog box, the Constrained Element property should be specified. A constrained element is the element for which the validation rule is created.

Example of listener configuration for the validation rule that is interested in name changes of Class and Interface model elements:

```
    public Map<Class<? extends Element>, Collection<SmartListenerConfig>>
getListenerConfigurations()
    {
        Map<Class<? extends Element>, Collection<SmartListenerConfig>> configMap =
                    new HashMap<Class<? extends Element>,
Collection<SmartListenerConfig>>();
        Collection<SmartListenerConfig> configs = new
 ArrayList<SmartListenerConfig>();
        configs.add(SmartListenerConfig.NAME_CONFIG);
        configMap.put(com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class.class,
configs);
        configMap.put(com.nomagic.uml2.ext.magicdraw.classes.mdinterfaces.Interface.c
lass, configs);
        return configMap;
    }
```

1087 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/validation/SmartListenerConfigurationProvider.html
1088 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html
1089 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/jmi/smartlistener/SmartListenerConfig.html
1090 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/validation/DefaultValidationRuleImpl.html

For the given example, In the specification dialog of the **Constraint** element the Constrained Element property should be set to Class and Interface metaclasses from the UML Standard Profile.

Example of listener configuration for the validation rule that is interested in Activity parameters and Activity parameter node changes

```
    public Map<Class<? extends Element>, Collection<SmartListenerConfig>>
getListenerConfigurations()
    {
        Map<Class<? extends Element>, Collection<SmartListenerConfig>> configMap =
                    new HashMap<Class<? extends Element>,
Collection<SmartListenerConfig>>();
        Collection<SmartListenerConfig> configs = new
 ArrayList<SmartListenerConfig>();
        SmartListenerConfig parameterConfig = new SmartListenerConfig();
        Collection<String> parameterPropertiesList = new ArrayList<String>();
        parameterPropertiesList.add(PropertyNames.DIRECTION);
        parameterPropertiesList.add(PropertyNames.TYPE);
        parameterPropertiesList.add(PropertyNames.OWNED_TYPE);

parameterConfig.listenToNested(PropertyNames.OWNED_PARAMETER).listenTo(parameterPr
opertiesList);
        SmartListenerConfig cfg = new SmartListenerConfig();
        Collection<String> argumentCftList = new ArrayList<String>();
        argumentCftList.add(PropertyNames.PARAMETER);
        argumentCftList.add(PropertyNames.TYPE);
        argumentCftList.add(PropertyNames.OWNED_TYPE);
        cfg.listenTo(argumentCftList);
        SmartListenerConfig argumentConfig = new SmartListenerConfig();
        argumentConfig.listenTo(PropertyNames.NODE, cfg);
        configs.add(parameterConfig);
        configs.add(argumentConfig); configMap.put(Activity.class, configs); return
 configMap;
    }
```

See the *JavaConstantNameValidationRuleImpl.java* example in *<program installation directory>\openapi\examples\validation*.

Case D

If you want to create a validation rule that will not be used in the active validation and you do not need to provide solving actions then you can create a Java class that contains the static method with the signature:

```
public static Boolean <method_name>(<Element_type> param)
```

and specify *<qualifiedClassName>.<method_name>* as a body value of the validation rule's specification.

The referred java method must be a static method and it must take exactly one parameter. Several validating methods can be placed into the one Java class or use the one class per validating method – this is irrelevant.

The type of the parameter MUST match the type of the constrained element of the validation rule. For validation rules on metaclasses, use the appropriate class from the *com.nomagic.uml2.ext.magicdraw.** package. For example, to write the rule for the metaclass *DataType*, use com.nomagic.uml2.ext.magicdraw.classes.mdkernel.DataType[1091] as a type of the parameter.

For the validation rules on stereotypes, use the same class as you would use when specifying the validation rule for a metaclass of that stereotype. For validation rules on the classifiers of the model, use com.nomagic.uml2.ext.magicdraw.classes.mdkernel.InstanceSpecification[1092] as the type of the parameter.

The Java code example of a trivial validation rule, which always returns *true* (that is, all elements are valid):

```
package com.nomagic.magicdraw.validation;
import com.nomagic.uml2.ext.magicdraw.classes.mdkernel.DataType;
public class TestRule
{
    public static Boolean testMeta(DataType exp)
    {
        return Boolean.TRUE;
    }
}
```

Compile such code into the Java bytecode, locate it where the modeling tool can load it (a classpath, plugin), and then you can use it for the validation:

1. Create the validation rule in our model.
2. Select the *DataType* metaclass as the constrained element of the rule.
3. Select the *Binary* language for the expression of the rule.
4. Specify *com.nomagic.magicdraw.validation.TestRule.testMeta* as the body of the expression of the rule.

Run the validation suite with the validation rule included. The method will be invoked for each datatype model element that is found in the scope of the validation. For each element, where this method returns *false*, an entry will be placed in the validation results table.

Implementing Rule to Validate Presentation Elements (Symbols)

In case you need a validation rule that would check and annotate individual presentation elements (symbols) in diagrams, then a Java class extending com.nomagic.magicdraw.validation.DiagramValidator[1093] should be implemented. This rule can be

1091 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/DataType.html
1092 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/InstanceSpecification.html

used for example to annotate symbols with overlapping bounds, or to detect multiple representations of the same model element in the same diagram in case only a single representation is desired.

Instead of receiving model element to validate as a parameter, use *DiagramValidator.getDiagrams()*[1094] method to get diagrams that are in the currently validated scope. To annotate individual symbols, create *com.nomagic.magicdraw.annotation.Annotation*[1095] object with the target being the com.nomagic.magicdraw.uml.symbols.PresentationElement[1096] which does not pass your validation conditions.

> ⓘ Extend a subclass com.nomagic.magicdraw.validation.OpenedDiagramValidator[1097] instead for your rule to always restrict the validated diagrams to the currently opened ones only.

Also see: Presentation elements (symbols)(see page 2276)

Binary validation rule in the plugin

The validation engine must be able to load the validation rule implementation class. The validation engine is able to load a class if the implementation class exists in the program classpath or if the validation rule implementation class is in the plugin classpath. The plugin must register the plugin's classloader to the validation system as follows:

```
EvaluationConfigurator.getInstance().registerBinaryImplementers(<PluginClassName>.
class.getClassLoader());
```

Related pages

- Creating validation rules
  (see page 2333)

Script validation rule

JRuby, Groovy, Javascript, or Jython scripting language can be used to create a validation rule which are used to validate elements in your model. The advantage of using script validation rules is that it is not necessary to compile and store them in a classpath. Oftentimes, the code is compact and simple.

How to create a a script validation rule

1. Create a constraint(see page 690).

1094 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/validation/DiagramValidator.html#getDiagrams--
1095 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/annotation/Annotation.html
1096 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/symbols/PresentationElement.html
1097 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/validation/OpenedDiagramValidator.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

2. Set the stereotype «UML Standard Profile::Validation Profile::validationRule» for the validation rule.
3. Set the severity level, error message, and abbreviation.
4. Specify constrained element(s).
5. Specify the specification language (JRuby, Groovy, Javascript, or Jython).
6. Enter the validation code in the selected language.

Here is an example validation rule written in JRuby:

```
if $THIS.isAbstract()
 return $THIS.getOwner().isAbstract()
end
true
```

The rule validates operations in the model. If an operation is abstract, but its owning classifier isn't, the rule fails. "$THIS" variable references the element being validated. It should be used at least once in the code or in a comment.

Related pages

- Validation(see page 1306)
- Creating validation rules(see page 2333)

How to provide a solution for a problem found during validation?

The validation rule returns annotations as validation results to the validation engine. Each annotation can contain a list of action objects that implements how a particular problem found by the validation rule can be solved. In order to create an action for solving, the validation rule's provider must create a Java class that extends the com.nomagic.actions.NMAction[1098] class and implement the *actionPerformed(java.awt.event.ActionEvent)*[1099] method. A user will be able to invoke the action from the validation results table or from the browser. In order to enable performing the action on multiple targets, the action class must implement the com.nomagic.magicdraw.annotation.AnnotationAction[1100] interface.

See *MyAction.java*, *FixJavaConstantNamesAction.java*, *MyBinaryValidationRuleImpl.java* and *JavaConstantNameValidationRuleImpl.java* examples in <program installation directory>\openapi\examples\validation.

Related pages

- Creating validation rules(see page 2333)
- Annotating the elements(see page 2332)

1098 http://jdocs.nomagic.com/190beta/com/nomagic/actions/NMAction.html
1099 http://jdocs.nomagic.com/190beta/com/nomagic/actions/NMAction.html#actionPerformed-java.awt.event.ActionEvent-
1100 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/annotation/AnnotationAction.html

Custom Legend Item Implementation

It is possible to create Java implementation describing what model element or presentation element should a Legend Item adorn in a diagram.

1. The Java class should be implemented following the very same rules as when creating implementation for a validation rule(see page 2338).
   Each com.nomagic.magicdraw.annotation.Annotation[1101] object returned by the rule will cause the targeted model or presentation element to be adorned in the diagram (but only if the Legend is used in that diagram).
2. Specify the full class name as the value of "Implementation" property for the Legend Item in the modeling application.
3. Compile Java code, resulting in .class files possibly contained by a .jar file.
4. Ensure that your modeling tool can find and load these classes. You can place them in the program classpath or use the modeling tool plugins mechanism.

Example
See *AdornComponentsOfInterestInDiagramValidator.java* example implementation in the *<program installation directory>\openapi\examples\legendItem* directory.

Also see: Legends(see page 530)

Custom elements numbering

The generic numbering open API provides two interfaces of different complexity. They allow a user to modify how the numbering process executes.

- The IJavaNumberPart(see page 2345) interface produces a single part of a number generated by the framework. It does not interfere with the execution of the framework, but simply provides a little piece of information included in the numbering.
- The INumberingAction(see page 2345) allows for generating complete numbers for sets of elements. All sorting and numbering is delegated to the implementing class. The framework will only register the generated numbers internally.

Classes that implement these interfaces can be set as the expression value of an Expression NumberPart. (see Numbering Scheme property values(see page 2143)).

These classes have to be specified with the complete package and class name, for example, *com.nomagic.magicdraw.autoid.sample.SquareNumberPart*.

Related pages

1101 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/annotation/Annotation.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

IJavaNumberPart interface

The provided NumberParts for Numeric and Character implement the numbering based on the natural ordering of integers or the alphabetic ordering of strings. Examples for this would be P1.1, P1.2, P1.3 or P1.A, P1.B, P1.C, etc. What if we wanted to create numbers of the following kind: P1.1, P1.4, P1.9, P1.16, etc. with all intermediate numbers omitted?

The *com.nomagic.magicdraw.autoid.IJavaNumberPart*[1102] interface provides a way to create a slice for a generated number that does not follow these natural sequences.

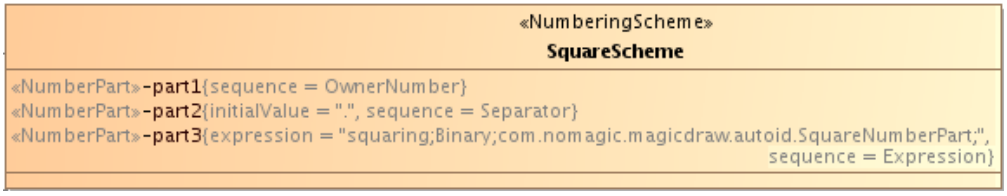
The parameter that the *com.nomagic.magicdraw.autoid.IJavaNumberPart.generateNextId*[1103] method receives, namely *lastNumberPart*, is the value that this method is generated as a result in the previous call. If the method has not been called before this *lastNumberPart* value equals an empty string.

> ⚠ This method should adhere to the contract that equal inputs must produce equal outputs. If, for example, the input is ABC and the method computes a value of XYZ then the return value for ABC must always be XYZ.

The return value of this method will be incorporated by the Numbering Framework to create a complete number together with all other *NumberParts* that are part of the same *NumberingScheme*.

Example

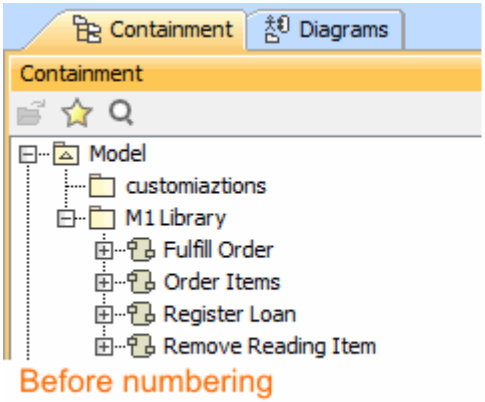
Let's assume the *SquareScheme* is used as target for UML activities. According to this *NumberingScheme*, the first part of a number will be taken from a numbered parent (in this case, the package *sample* as shown in the following figure), then a dot is inserted and the final slice is generated by the *SquareNumberPart* expression.

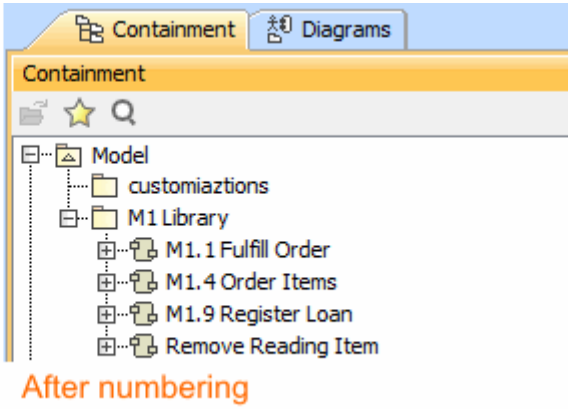


We have created a few activities inside the package named *sample*. The owning package is also customized and has a generated number P1. If we number the activities with this *NumberingScheme*, the *SquareNumberPart* class, being set as *part3* of the type expression, will create the following sequence: 1, 4, 9, 16, 25... These pieces are then added to the results from the other *NumberParts*, resulting in the activities being numbered as shown in the following figure.

1102 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/autoid/IJavaNumberPart.html
1103 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/autoid/IJavaNumberPart.html#generateNextId-java.lang.String-

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.





The source code for this simple implementation should be self-explaining. It will simply try to transform the string value of *lastNumberPart* into an integer value called *square*. Subsequently, the following mathematical operation is performed and returned $nextSquare = (\sqrt{square} + 1)^2$.

```
public class SquareNumberPart implements IJavaNumberPart
{
    private final int initialValue = 1;
    @Override
    public String generateNextId(String lastNumberPart)
    {
        int nextSquare = initialValue;
        try
        {
            if ("".equals(lastNumberPart))
            {
                lastNumberPart = "0";
            }
            int square = Integer.valueOf(lastNumberPart);
            int number = (int) Math.sqrt(square) + 1;
            nextSquare = (int) Math.pow(number, 2);
        }
        catch (NumberFormatException ex)
        {
            // already set nextSquare as initialValue
        }
        return Integer.toString(nextSquare);
    }
}
```

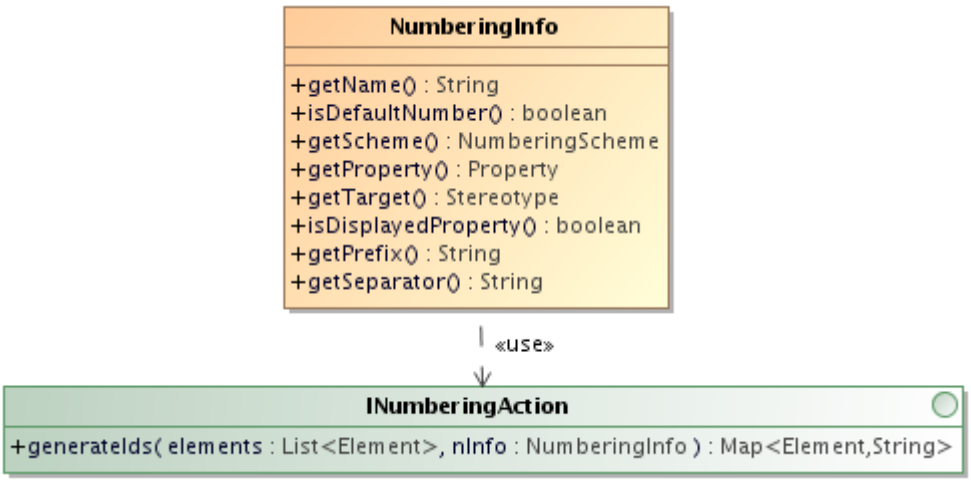
INumberingAction interface

What if we want to customize the numbering of elements even further? Let's say we need to number uml:Activities depending on their *getStructuredNode()* value. By implementing the *com.nomagic.magicdraw.autoid.INumberingAction*[1104] interface, it is possible to create the numbering

1104 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/autoid/INumberingAction.html

that follow arbitrary conditions, this comes of course at the cost of increased complexity. The implementation must calculate the values for the numbers all by itself.

There are two parts to be explained that pertain to this interface. The actual interface that provides access to customized numbering and an immutable support class which encapsulates the values as defined in the *AutoNumber* and the associated *NumberingScheme* objects.



The INumberingAction[1105]**Interface**. The interface has one method, which is invoked whenever a new *Element* is created or when **Create** or **Renumber** buttons are clicked in the **Element Numbering** dialog. This applies of course only if the Element type equals the type for which the *AutoNumber* has been customized. So at the time when a new element is created, this method receives a *java.util.List* with a single entry. On the other hand, when the user navigates the **Element Numbering** dialog, the method receives all elements that are currently listed in the right partition of the dialog. This implies that all elements have the same owner. Since this will give access to the complete Model Tree, it is possible to collect necessary information from other parts of the model and incorporate it. This method must return a mapping of each element together with the value that was generated for it. There is no need to number every element that the method receives. The Numbering Framework will register the generated values internally and set the property value for each element.

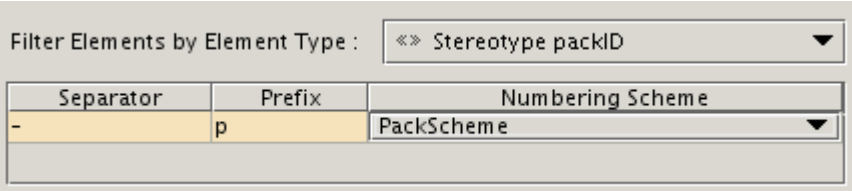
The *com.nomagic.magicdraw.autoid.NumberingInfo*[1106] **support object.** This is an immutable Java Bean object which provides the implementer of the *INumberingAction*[1107]interface with all the information as defined in the *AutoNumber* and *NumberingScheme* objects:

- getName()[1108] returns the name of the *AutoNumber*.
- isDefaultNumber()[1109] returns the *AutoNumber.defaultNumber* value.
- *getScheme()*[1110]returns the *AutoNumber.numberingScheme* value, which is the *NumberingScheme* used to number this type of element.
- getProperty()[1111] returns the *AutoNumber.numberedProperty* value, which is the *Property* that will hold the value of the calculated number.

1105 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/autoid/INumberingAction.html
1106 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/autoid/NumberingInfo.html
1107 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/autoid/INumberingAction.html
1108 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/autoid/NumberingInfo.html#getName--
1109 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/autoid/NumberingInfo.html#isDefaultNumber--
1110 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/autoid/NumberingInfo.html#getScheme--
1111 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/autoid/NumberingInfo.html#getProperty--

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- getTarget()[1112] returns the *AutoNumber.getTarget()* value, which is the type of the elements to be numbered.
- isDisplayedProperty()[1113] returns true, if the value of the above *getProperty()*[1114] is displayed in the GUI. An element type can have more than one numbered *Property*, but only one can be displayed on symbols or in the Element Tree.
- getPrefix()[1115] returns the prefix as modified by the user in the **Element Numbering** dialog, which can be different from the *AutoNumber.getPrefix()* value (see the following figure).
- getSeparator()[1116] returns the separator as modified by the user in the **Element Numbering** dialog. In the implemented Numbering Framework, this value will override all the *NumberParts* of a type separator (see the following figure).



For more information on how to create Numbering customizations with «AutoNumber», «NumberingScheme», and «NumberPart» elements, see Working with Generic Numbering Mechanism(see page 1041).

Constraints

A *NumberingScheme* that references an expression which implements this interface, can have one and only one *NumberPart*. Namely a *NumberPart* that references an *INumberingAction*[1117].

Example #1

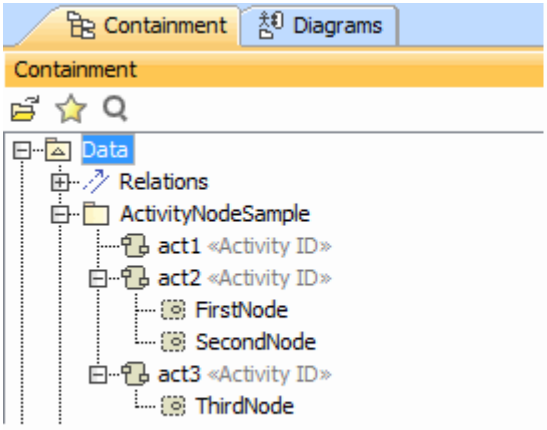
We shall now proceed to show a Java class that generates numbers for UML activities depending on their *getStructuredNode()* value. Since this class implements the *INumberingAction*[1118]interface, it should be set as the expression in the only *NumberPart* of this *NumberingScheme*.

We have created a few activities inside a sample *Package*. With the creation of each new activity, an ID is generated and assigned to it. We then add some *StructuredActivityNodes* into two of the given activities. Now, in order to assign *Numbers* to these *Activities* that includes the number of *StructuredActivityNodes*

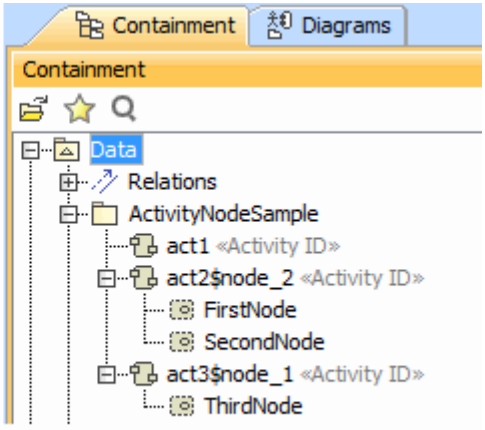
1112 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/autoid/NumberingInfo.html#getTarget--
1113 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/autoid/NumberingInfo.html#isDisplayedProperty--
1114 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/autoid/NumberingInfo.html#getProperty--
1115 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/autoid/NumberingInfo.html#getPrefix--
1116 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/autoid/NumberingInfo.html#getSeparator--
1117 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/autoid/INumberingAction.html
1118 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/autoid/INumberingAction.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

children, we have to open the **Element Numbering** dialog(see page 1046) and renumber the Activities. This will generate new numbers that reflect the ownership relation.



Before numbering



After numbering

Let's have a look at the source code. This is a very simple example, which differentiates between receiving one or many elements, and calculates the values accordingly. This class can be found in *com.nomagic.magicdraw.autoid.sample*.

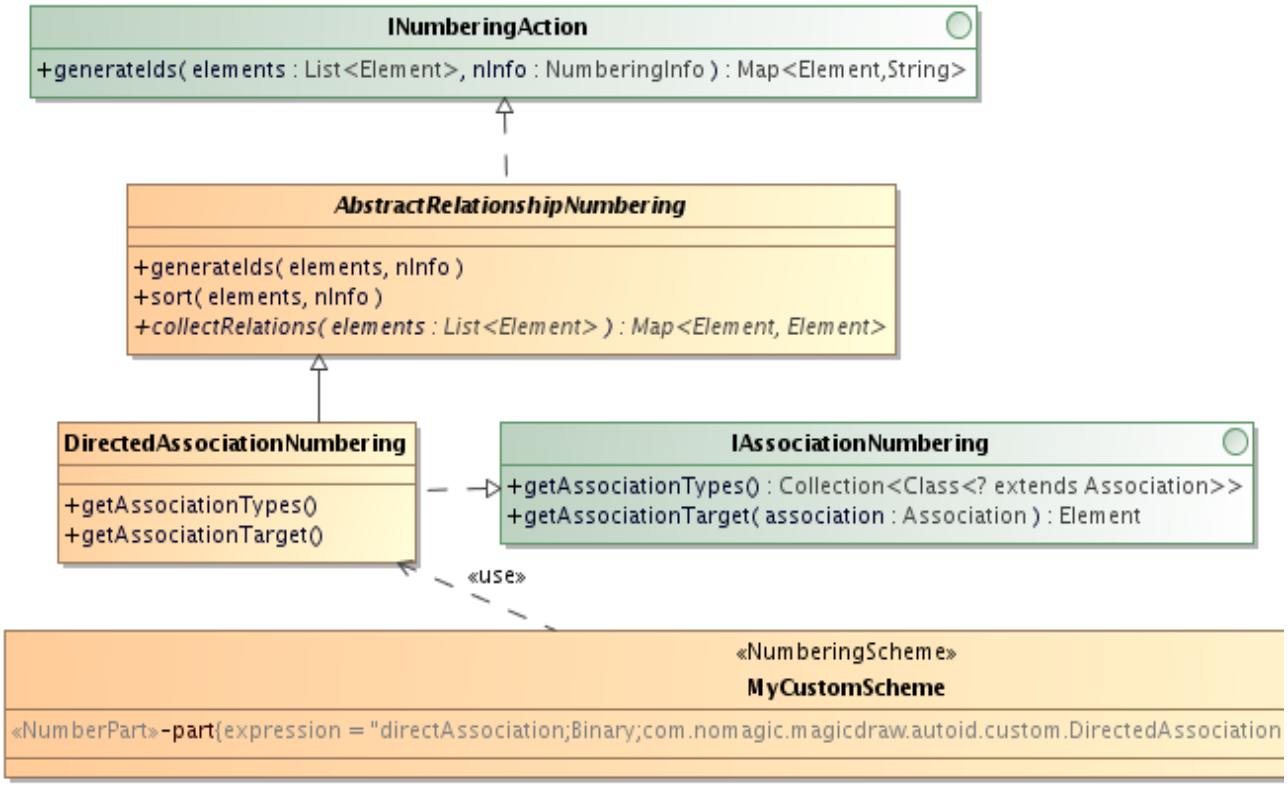
```
    public class ActivityNodeNumbering implements INumberingAction {
        @Override
        public Map<Element, String> generateIds(List<Element> elements, NumberingInfo
nInfo) {
            Map<Element, String> idMap = new HashMap<Element, String>();
            if (! elements.isEmpty()) {

                // the NumberingInfo object contains the values set in the
customization
                String baseId = nInfo.getPrefix() + nInfo.getSeparator();

                // when creating an Activity, this will be called with a single
element
                if (elements.size() == 1) {
                    Element e = elements.iterator().next();
                    Class<?>[] types = new Class<?>[]{Activity.class};
                    Collection<? extends Element> acts =
ModelHelper.getElementsOfType(e.getOwner(), types, false, true);
                    String id = baseId + (acts.size());
                    idMap.put(e, id);


                    // when renumbering in the dialog this will be called with
multiple elements
                } else {
                    int counter = 1;

                    // we sort so that renumber will get the same order
                    // normally this would ask for a custom comparator class
                    // but it is left out for brevity's sake
                    Collections.sort(elements);
                    for (Element e : elements) {
                        if (e instanceof Activity) {
                            Activity act = (Activity) e;
                            String id = baseId + counter;
                            counter ++;
                            Collection<StructuredActivityNode> nodes =
act.getStructuredNode();

                            // we simply attach the suffix '$node_' and the node
count
                            if (nodes != null && !nodes.isEmpty()) {
                                id += "$node_" + nodes.size();
                            }
                            idMap.put(e, id);
```

```
                        }
                    }
                }
            }
            // returning the mapping of elements and ids, so that the framework can
register them
            return idMap;
        }
    }
```

Example #2

A more complicated example can be found the OpenAPI packages *com.nomagic.magicdraw.autoid* and *com.nomagic.magicdraw.autoid.custo*. It shows an implementation that creates numbers for elements connected through UML Relationships. The UML diagram shows the *com.nomagic.magicdraw.autoid.custom.DirectedAssociationNumbering*[1119] class, which is customized to number element types that are connected by «DirectedAssociation» links.

The interested Reader should consult the source code, in order to learn how such an algorithm has been implemented.



Example of relationship numbering (fragment)

1119 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/autoid/custom/DirectedAssociationNumbering.html

Creating Use Case Scenario

In modeling tools, No Magic Inc. provides API for creating and manipulating a Use Case Scenario(see page 882) for a use case:

- The *com.nomagic.magicdraw.usecasescenarios.scenarios.Scenario*[1120] class represents a Use Case Scenario. It contains methods for manipulating it.
- The *com.nomagic.magicdraw.usecasescenarios.scenarios.ScenarioFactory*[1121] class is used for creating use case scenarios.
- The *com.nomagic.magicdraw.usecasescenarios.scenarios.ScenarioManager*[1122] class contains utility methods for creating and manipulating scenarios.
- The scenario management calls should be wrapped with the session management calls.

See an example of how to create the use case scenario with the basic, alternative, and exceptional flows. After the use case scenario is created, the Activity diagram is opened. In the Activity diagram, the use case scenario is represented as an action flow.

1120 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/usecasescenarios/scenarios/Scenario.html
1121 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/usecasescenarios/scenarios/ScenarioFactory.html
1122 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/usecasescenarios/scenarios/ScenarioManager.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

```
SessionManager.getInstance().createSession(...);
// Creates a use case scenario.
Scenario scenario = ScenarioManager.createScenario(useCase);
// Sets the scenario name.
scenario.setName("Extract money from ATM.");

// Adds a basic flow step.
FlowStep flowStep1 = scenario.addFlowStep();
// Sets a name for the basic flow step.
flowStep1.setName("Insert card");

FlowStep flowStep2 = scenario.addFlowStep();
flowStep2.setName("Enter pin");

FlowStep flowStep3 = scenario.addFlowStep();
flowStep3.setName("Good bye");

// Adds an alternative condition for the basic flow step.
AlternativeCondition condition = scenario.addAlternativeCondition(flowStep2);
// Sets a condition guard.
condition.setIfCondition("Pin correct");

// Sets a name for the alternative flow step.
FlowStep flowStep = condition.getAlternativeFlowSteps().get(0);
flowStep.setName("Extract money");
// Adds an exception type to the basic flow step.
ExceptionType exceptionType = scenario.addExceptionType(flowStep2);

// Sets a name for the exception type.
exceptionType.setName("Card expired");
// Sets a name for the exceptional flow step.
FlowStep exceptionalFlowStep = exceptionType.getExceptionalFlowSteps().get(0);
exceptionalFlowStep.setName("Inform customer about expired card");
SessionManager.getInstance().closeSession();

// Opens and layouts the Activity diagram.
ScenarioManager.displayScenario(scenario, true, true, "Open ATM Scenario");
```

More information is available in javadoc[1123].

> You can find the code examples in
> - *<programinstallation directory>\openapi\examples\scenario*

For more information about the use case scenario, see Use Case Scenario in MagicDraw UserManual.pdf.

Merging and differencing

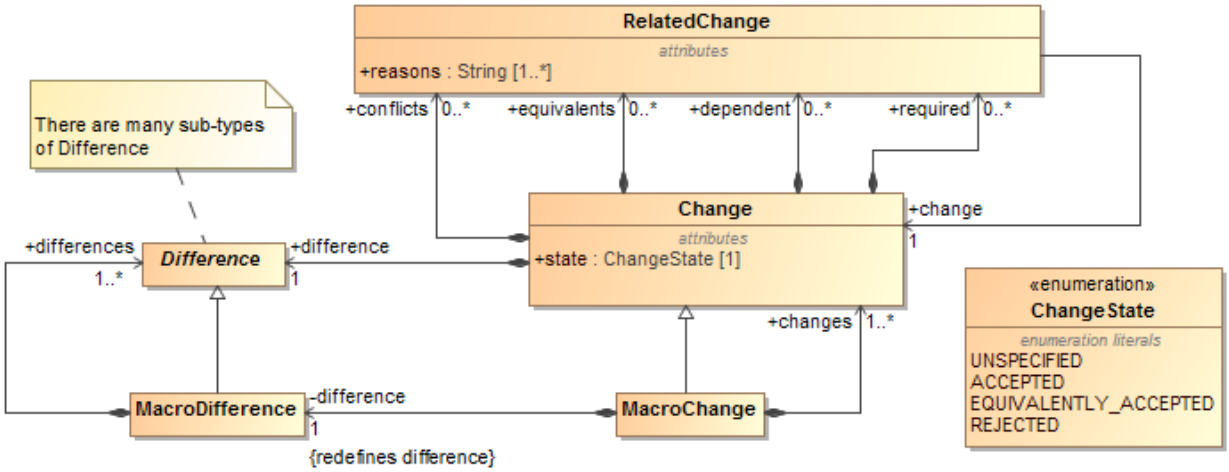
Modeling tools developed by No magic Inc. provide the API for the project merging (and differencing). The project merging consists of three steps:

1123 http://jdocs.nomagic.com/190beta/

1. **Discovering project changes (differences)**. The ancestor (in a 3-way merge only) and source projects are loaded (if they are not loaded yet) for comparing.
2. **Accepting / rejecting changes**. All target changes that do not conflict source changes are applied (source changes conflicting with target changes are rejected). A 2-way merge produces source changes only.
3. **Applying accepted changes**. Changes are applied on the ancestor project (target and ancestor projects are the same in a 2-way merge).

There are the API for the standard and advanced project merging. The standard API encapsulates all three steps into a single call, while the advanced merge API allows for accessing the project differences and controlling what changes are accepted or rejected.

The following figure shows the domain model of the merging.



More information is available in javadoc[1124].

> You can find the code examples in
> - *<programinstallation directory>\openapi\examples\merge*

Code engineering using API

You can perform Code Engineering using the Open API.

Code engineering allows generation of the source code from the specific model, and source code reversing into model elements. API is provided for:

- Code engineering sets creation for particular programming languages.
- Automatic component creation for every class involved in forward engineering and every file involved into reverse engineering.
- The specification of either working, or output, or temporary directories for processing source code files. The destination of the code reverse operation output can be any model package.

1124 http://jdocs.nomagic.com/190beta/

In the following topics we will review how to manage code engineering and how to do reverse and forward engineering.

More information is available in javadoc[1125].

> You can find the code examples in
> - *<programinstallation directory>\openapi\examples\codeengineering*

Related pages

Code engineering set

A *com.nomagic.magicdraw.ce.CodeEngineeringSet*[1126] object represents a code-engineering object as a resource for forward or reverse engineering.

Each *CodeEngineeringSet*[1127]provides properties, which can be read:

- **Name**. Represents a name of the code engineering set, visible in the Browser.
- **Model elements**. Represents elements that are added into the code engineering set from the model.
- **Working directory**. Represents a code engineering directory where generated/reversed files are located.

Forward Engineering

To perform the code generation, elements should be added to the *CodeEngineeringSet*[1128]object. Use the CodeEngineeringSet.addElementsToCodeEngineeringSet(java.util.List<[1129]BaseElement[1130]>)[1131] method to add a list of model elements to the code engineering set.

> Model elements should be in the working package, otherwise the model element is not be added to the code engineering set. The working package is set when creating a code engineering set.

Reverse Engineering

1125 http://jdocs.nomagic.com/190beta/
1126 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ce/CodeEngineeringSet.html
1127 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ce/CodeEngineeringSet.html
1128 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ce/CodeEngineeringSet.html
912 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ce/CodeEngineeringSet.html#addElementsToCodeEngineeringSet-java.util.List-
1130 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/BaseElement.html
113 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ce/CodeEngineeringSet.html#addElementsToCodeEngineeringSet-java.util.List-

Source code files are required to perform the Reverse Engineering. There are 2 methods available for adding files into the code engineering set:

1. CodeEngineeringSet.addFilesToCodeEngineeringSet(java.util.List<java.io.File>)[1132]; This method adds the given list of files to the code engineering set.
2. CodeEngineeringSet.addAllFilesRecursivelyToCES(java.io.File)[1133]; This method adds all specific source code files to the code engineering set, starting from the given directory.

⚠ Source code files should be in the working directory, in order to have the successful reverse.

Managing code engineering sets

The *com.nomagic.magicdraw.ce.CodeEngineeringManager*[1134] class provides static API methods for creating, removing, reversing, generating, and adjusting code-engineering settings. The *CodeEngineeringManager*[1135]class is used for managing code engineering sets.

When creating the code engineering set, the following fields are required:

- Language and dialect IDs. All languages and dialects IDs are stored in the *com.nomagic.magicdraw.ce.CodeEngineeringConstants*[1136] class.
- The name of the code engineering set(see page 2356).
- The project, where code engineering set will be added.
- The working package.
- The working directory.

New *com.nomagic.magicdraw.ce.CodeEngineeringSet*[1137] can be created directly via *CodeEngineeringManager* [1138]using the *CodeEngineeringManager.createCodeEngineeringSet(java.lang.String, java.lang.String, java.lang.String,*[1139]Project[1140], Package[1141], java.lang.String)[1142] method.

The language and dialect can be selected from the *CodeEngineeringConstants*[1143]class. The dialect is required for DDL and C++ languages.

1132 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ce/CodeEngineeringSet.html#addFilesToCodeEngineeringSet-java.util.List-
1133 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ce/CodeEngineeringSet.html#addAllFilesRecursivelyToCES-java.io.File-
1134 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ce/CodeEngineeringManager.html
1135 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ce/CodeEngineeringManager.html
1136 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ce/CodeEngineeringConstants.html
1137 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ce/CodeEngineeringSet.html
1138 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ce/CodeEngineeringManager.html
1139 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ce/CodeEngineeringManager.html#createCodeEngineeringSet-java.lang.String-java.lang.String-java.lang.String-com.nomagic.magicdraw.core.Project-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Package-java.lang.String-
1140 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Project.html
1141 http://jdocs.nomagic.com/190beta/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Package.html
1142 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ce/CodeEngineeringManager.html#createCodeEngineeringSet-java.lang.String-java.lang.String-java.lang.String-com.nomagic.magicdraw.core.Project-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Package-java.lang.String-
1143 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ce/CodeEngineeringConstants.html

- null *workingPackage* means, that the code engineering working package is *Model* by default
- null *workingDirectory* means, that the code engineering working directory is the modeling tool install root

To remove the instance of *CodeEngineeringSet*[1144]use the *CodeEngineeringManager.removeCodeEngineeringSet(*[1145]CodeEngineeringSet[1146]*)* method.

All Code Engineering Sets can be retrieved by the *CodeEngineeringManager.getAllCodeEngineeringSets(*[1147]Project[1148]*)* method.

To perform generation of the CodeEngineeringSet[1149]object use the *CodeEngineeringManager.generate(*[1150]Project[1151]*, boolean)*[1152] method.

To perform the reverse engineering use the *CodeEngineeringManager.reverse(*[1153]Project[1154]*, boolean)*[1155] method.

Language Specific Options

com.nomagic.magicdraw.ce.JavaCodeEngineeringManager[1156] provides methods to set Java specific options.These options applie to all code engineering sets in the project.

To add a classpaths for the java code engineering sets, use *JavaCodeEngineeringManager.setJavaClasspath(*[1157]Project[1158], java.lang.String[])[1159]

Setting a new classpath will override the old one.

1144 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ce/CodeEngineeringSet.html
1145 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ce/CodeEngineeringManager.html#removeCodeEngineeringSet-com.nomagic.magicdraw.ce.CodeEngineeringSet-
1146 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ce/CodeEngineeringSet.html
1147 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ce/CodeEngineeringManager.html#getAllCodeEngineeringSets-com.nomagic.magicdraw.core.Project-
1148 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Project.html
1149 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ce/CodeEngineeringSet.html
1150 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ce/CodeEngineeringManager.html#generate-com.nomagic.magicdraw.ce.CodeEngineeringSet-
1151 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Project.html
1152 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ce/CodeEngineeringManager.html#generate-com.nomagic.magicdraw.ce.CodeEngineeringSet-
1153 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ce/CodeEngineeringManager.html#reverse-com.nomagic.magicdraw.ce.CodeEngineeringSet-boolean-
1154 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Project.html
1155 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ce/CodeEngineeringManager.html#reverse-com.nomagic.magicdraw.ce.CodeEngineeringSet-boolean-
1156 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ce/JavaCodeEngineeringManager.html
1157 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ce/JavaCodeEngineeringManager.html#setJavaClasspath-com.nomagic.magicdraw.core.Project-java.lang.String:A-
1158 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Project.html
1159 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ce/JavaCodeEngineeringManager.html#setJavaClasspath-com.nomagic.magicdraw.core.Project-java.lang.String:A-

To get applied classpaths for java code engineering sets, use *JavaCodeEngineeringManager.getJavaClasspath(*[1160]Project[1161]*)*.

To resolve collection generics when reversing the java code, *CodeEngineeringManager.setResolveCollectionGenerics(*[1162]Project[1163], boolean)[1164]

Samples of the forward and reverse engineering

- Performing the forward engineering
- Performing the reverse engineering

Performing the forward engineering

The example shows how to perform a simple java code generation.

Step #1. Creating *CodeEngineeringSet*

```
    Project project = Application.getInstance().getProject();
    String name = "sample CE project";
    String workingDir = OPENAPI_DATA_DIRECTORY_PATH;

    // create a working package
    ElementsFactory ef = project.getElementsFactory();
    Package workingPackage = ef.createPackageInstance();
    workingPackage.setName("my working package");
    workingPackage.setOwner(project.getModel());

    // creating a code engineering set
    CodeEngineeringSet javaGenerationSet =
CodeEngineeringManager.createCodeEngineeringSet(
                CodeEngineeringConstants.Languages.JAVA, null, name, project,
workingPackage, workingDir);
```

Step #2. Adding model elements to *CodeEngineeringSet*

[1160] http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ce/JavaCodeEngineeringManager.html#getJavaClasspath-com.nomagic.magicdraw.core.Project-
[1161] http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Project.html
[1162] http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ce/CodeEngineeringManager.html#setResolveCollectionGenerics-com.nomagic.magicdraw.core.Project-boolean-
[1163] http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Project.html
[1164] http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/ce/CodeEngineeringManager.html#setResolveCollectionGenerics-com.nomagic.magicdraw.core.Project-boolean-

```
Project project = Application.getInstance().getProject();

// create a new element
ElementsFactory ef = project.getElementsFactory();
Class classA = ef.createClassInstance();
classA.setName("ClassA");
classA.setOwner(project.getModel());
List<BaseElement> modelsForSample = new ArrayList<BaseElement>();
modelsForSample.add(classA);
javaGenerationSet.addElementsToCodeEngineeringSet(modelsForSample);
```

Step #3. Performing the *CodeEngineeringSet* generation

```
CodeEngineeringManager.generate(javaGenerationSet);
```

Performing the reverse engineering

The example shows how to perform a simple java code reverse.

Step #1. Creating *CodeEngineeringSet*

```
    Project project = Application.getInstance().getProject();
    String name = "sample CE project";
    String workingDir = OPENAPI_DATA_DIRECTORY_PATH;// e.g C:\myworkingPackage

    // create a working package
    ElementsFactory ef = project.getElementsFactory();
    Package workingPackage = ef.createPackageInstance();
    workingPackage.setName("my working package");
    workingPackage.setOwner(project.getModel());

    // creating a code engineering set
    CodeEngineeringSet ces = CodeEngineeringManager.createCodeEngineeringSet(
                CodeEngineeringConstants.Languages.JAVA, null, name, project,
workingPackage, workingDir);
```

Here the null dialect is used for the java language, because java doesn't have any dialect.

Step #2. Adding the source code to *CodeEngineeringSet*

```
    ces.addAllFilesRecursivelyToCES(new File(workingDir + File.separator + "test
directory")); // starting from C:\myworkingPackage\test directory\
```

This sets the given instance of the code engineering set working directory and adds all files from that directory. Set java classpaths for the project:

```
    String[] claspath = new String[] { path1, path2, path3, path4 };
    JavaCodeEngineeringManager.setJavaClasspath(project, claspath);
```

Step #3. Performing the reverse of *CodeEngineeringSet*

```
     CodeEngineeringManager.reverse(ces, false);
```

Oracle DDL generation and customization

The Oracle DDL script generation is based on the Velocity engine. It provides an ability to change the generated DDL script by changing the velocity template. We will introduce how the Oracle DDL generation works in modeling tools developed by No Magic Inc., how to change the template for some specific things.

Knowledge of the Velocity Template Language is necessary for understanding, editing, or creating templates. The Velocity documentation can be downloaded from http://click.apache.org/docs/velocity/VelocityUsersGuide.pdf.

For more information about Oracle DDL 11g, see http://docs.oracle.com/cd/E25054_01/server.1111/e25789/sqllangu.htm.

The modeling tool Oracle DDL generation consists of the following three components:

- The Velocity engine. Oracle DDL generation is performed by the Velocity engine. The engine collects context variables and mergers them into a template.
- The template. The template is a user-defined document that provides the Velocity Template Language (VTL) syntax. The VTL syntax is used to manipulate the context variables for generating the text script. The template file can be changed in the **CG Properties Editor** dialog box (in the Oracle DDL set shortcut menu, choose the **Properties** command). The default Oracle DDL template is stored in *<program installation folder>\data\DB_engineering\Oracle_template* folder.
- Context variables. Context variables are models (retrieved from OpenAPI), code engineering set information, and user-defined variables.

To identify the Oracle object, check applied Stereotypes and Tagged Values.

Related pages

Understanding the Oracle DDL template structure

The default Oracle DDL template is stored in *<program installation folder>\data\DB_engineering\Oracle_template* folder. The template consists of many macros - velocity functions. Each macro is dedicated for a particular object generation.

This example shows the velocity macro for the Oracle VIEW object generation:

```
    #macro ( generateView $data )
    #set ( $QUERY_RESTICTION = "query restriction")
    #set ( $force = false)
    #set ( $force = $oracleHelper.getBooleanValueFromDefaultProfile($data,
$VIEW_STEREOTYPE, $FORCE_TAG ) )
    #set ( $sqlrestriction = false)
    #set ( $sqlrestriction = $ oracleHelper.getFirstPropertyValueFromProfile($data,
$VIEW_STEREOTYPE, $QUERY_RESTICTION ) )
    #set ( $sql = false)
    #set ( $sql = $ oracleHelper.getFirstPropertyValueFromProfile($data,
$VIEW_STEREOTYPE, $QUERY_TAG ))
    #set ( $columns = false)
    #set ( $columns = $ oracleHelper.getViewColumnList( $data ) )
    #if ($columns)
        #set ( $columns = " ($columns)" )
    #else
        #set ( $columns = $nospace )
    #end
    #writeDocumentation ( $data $nospace )
    #writeLine( "$ oracleHelper.getCreate($data)
    #if ( $oracleHelper.getPropertiesListFromDefaultProfile($data, $VIEW_STEREOTYPE,
$FORCE_TAG).size()>0 )
        #if( $force ) FORCE
        #else NO FORCE
        #end
    #end
    VIEW $oracleHelper.getQualifiedName( $data )$columns AS" $nospace)
    #if ( $sql )
        #writeLine( $sql $tab )
    #end
    #writeText( ${sqlrestriction} ${space}${nospace})${semicolon}
    #end
```

A model element with the stereotype “view” is passed to a macro function, and then all element information is retrieved by the *$oracleHelper* utility class. The Oracle *View* stereotype has 3 properties. These tagged values store information about the force, query restriction, and query statements.

```
    Method $oracleHelper.getFirstPropertyValueFromProfile( $data,$VIEW_STEREOTYPE,
$QUE- RY_RESTICTION) gets query restriction statement from element $data.
```

Customizing templates

To change the generation of the particular Oracle Object, add a new functionality to its macro in the velocity template. The Helper utility class will assist in retrieving model information.

We suggest to make a back up of the default template. The default template is stored in *<program instal lation folder>\ data\DB_engineering\Oracle_template* folder. The template file can be changed in the **CG Properties Editor** dialog box (in the Oracle DDL set shortcut menu, choose the **Properties** command).

When generating DDL, objects are passed from the code engineering set(see page 2356) to the velocity engine. Objects, that are passed to the Oracle DDL velocity engine are presented in the following table:

Object name	Object type	Description
`$CESList`	`List<NamedEleme nt>`	The list of model elements that are added to the code engineering set(see page 2356).
`$dropRequired`	`boolean`	Flag to notify if the drop objects is required. This is a drop setting option which generates the Drop statement.
`$oracleHelper`	`Java.lang.Class`	This is a helper class for retrieving information from model elements. Use this class to check the element for applied stereotypes, get tagged values and element's owned information.
`$newLineBracket`	`boolean`	The generation Option to generate a bracket in a new line.
`$genDocumentation`	`boolean`	The language option for the documentation generation.

Utility class

The utility class helps to retrieve information from model elements. Use these commands to get particular information:

```
$oracleHelper.hasStereotype($element, $stereotypeName)
```

Returns *true* if a given element has an applied given stereotype.

	Name	Type	Description
Parameters	`$element`	`com.nomagic.uml2 .ext.magicdraw. c lasses.mdkernel. Element`	An element to check.
	`$stereotypeN ame`	`java.lang.String`	A stereotype name to be checked.
Return	–	`boolean`	*true* if elements have an applied stereotype with a given name.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

```
$oracleHelper.getBooleanValueFromDefaultProfile($element, $stereotypeName, $propertyName)
```

From the given element, stereotype name, and tag name returns a tag value as *Boolean* from the Oracle profile.

	Name	Type	Description
Parameters	`$element`	`com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element`	An element to check.
	`$stereotypeName`	`java.lang.String`	A stereotype name (from the default profile) that should be applied.
	`$propertyName`	`java.lang.String`	A property name which value will be retrieved.
Return	-	`boolean`	A boolean value of property (tag).

```
$oracleHelper.getPropertiesListFromDefaultProfile($element, $stereotypeName, $propertyName)
```

Returns a list of given property values, that exist on the given element with the applied stereotype from the Oracle profile.

	Name	Type	Description
Parameters	`$element`	`com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element`	An element to be tested.
	`$stereotypeName`	`java.lang.String`	A stereotype name which should be applied.
	`$propertyName`	`java.lang.String`	A property (tag) name to get values from.
Return	-	`java.util.List`	A list of property values.

```
$oracleHelper.getFirstPropertyValueFromProfile($element, $stereotypeName, $propertyName)
```

Returns teh first given tag property value from the given element, which has the applied given stereotype.

	Name	Type	Description
Parameters	`$element`	`com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element`	An element to be tested.
	`$stereotypeName`	`java.lang.String`	A stereotype name that should be applied to the element.
	`$propertyName`	`java.lang.String`	A property name where to check for values.
Return	-	`java.lang.String`	The first property value in a string representation.

```
$oracleHelper.getDefaultValueAsBoolean($property)
```

From the given property returns the default value as boolean.

	Name	Type	Description
Parameter	`$property`	`com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Property`	A boolean property to be check for the default value.
Return	-	`boolean`	The default boolean value of a property (tag).

```
$oracleHelper.getFirstPropertyValueFromGivenProfile($element, $profileName, $stereotypeName, $propertyName)
```

Returns first given tag property value from given element, which has applied given stereotype from profile.

	Name	Type	Description
Parameters	`$element`	`com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element`	An element to be tested.

	Name	Type	Description
	`$profileName`	`java.lang.String`	A profile name where the stereotype exists.
	`$stereotypeName`	`java.lang.String`	A stereotype name.
	`$propertyName`	`java.lang.String`	A property (tag) name, which value will be checked.
Return	–	`java.lang.String`	The first value in a property list in *String*.

```
$oracleHelper.getPropertiesListFromProfile($element, $profileName, $stereotypeName, $propertyName)
```

Returns a list of given property values, that exist on a given element with an applied stereotype from the given profile.

	Name	Type	Description
Parameters	`$element`	`com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element`	An element to be tested.
	`$profileName`	`java.lang.String`	A profile name where the stereotype exists.
	`$stereotypeName`	`java.lang.String`	A stereotype name.
	`$propertyName`	`java.lang.String`	A property (tag) name, which value will be checked.
Return	–	`java.lang.List`	A list of property values.

```
$oracleHelper.getStringValue($object)
```

From the given tag value returns the *String* representation.

	Name	Type	Description
Parameter	`$object`	`java.lang.Object`	An objectto be tested.
Return	-	`java.lang.String`	A string representation of the object.

```
$oracleHelper.isDataType($element)
```

Returns *true* if element is data type.

	Name	Type	Description
Parameter	`$element`	`com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element`	An element to be tested.
Return	-	boolean	*true* if the element is the datatype.

```
$oracleHelper.getType($type, $modifier)
```

From the given type and modifier returns it's description.

	Name	Type	Description
Parameters	`$type`	`com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Type`	A type of an element.
	`$modifier`	`java.lang.String`	A modifier of the type.
Return	–	`java.lang.String`	A type definition for Oracle DDL.

```
$oracleHelper.getTypeModifier($element)
```

Returns the Type modifier for the given element.

	Name	Type	Description
Parameters	`$element`	`com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element`	An element to be tested.
Return	–	`java.lang.String`	The Type modifier description.

```
$oracleHelper.getParameters($operation)
```

Returns the list of operation parameters.

	Name	Type	Description
Parameter	`$element`	`com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Operation`	An operation to be tested.
Return	–	`java.util.List`	A list of operation parameters.

```
$oracleHelper.getColumnConstraint($column)
```

Returns the given property constraint.

	Name	Type	Description
Parameter	`$column`	`com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Property`	A column to check for the constraint.
Return	–	`java.util.List`	A constraint definition.

```
$oracleHelper.getCreate($element)
```

Returns the CREATE statement for the given element.

	Name	Type	Description
Parameter	`$element`	`com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element`	An element to be tested.
Return	–	`Java.lang.String`	A description - CREATE or CREATE OR REPLACE.

```
$oracleHelper.getReturnParameter($operation, $createIfNeeded)
```

Returns the Return type parameter of a given Operation.

	Name	Type	Description
Parameters	`$operation`	`com.nomagic.uml2.ext.magicdraw.classes.mdkernel.BehavioralFeature`	An operation to check.
	`$createIfNeeded`	`boolean`	A flag to create a return parameter if it does not exist.
Return	–	`com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Parameter`	Return the parameter of a given Operation.

```
$oracleHelper.getIndexNameDefinition($index)
```

Returns a name of a given index.

	Name	Type	Description
Parameter	`$index`	`com.nomagic.uml2.ext.magicdraw.classes.mdkernel.BehavioralFeature`	BehavioralFeature as an Oracle index to be checked for a name.
Return	–	`java.lang.String`	A name of an index.

```
$oracleHelper.getTableConstraintDefinition($dependency)
```

From the given dependency, returns a table constraint definition.

	Name	Type	Description
Parameter	`$dependency`	`com.nomagic.uml2.ext.magicdraw.classes.ddependencies.Dependency`	A dependency to be tested.
Return	–	`java.lang.String`	A definition of the table constraint.

```
$oracleHelper.isObjectPackage($element)
```

Checks if a given element is a package.

	Name	Type	Description
Parameter	`$element`	`com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element`	An element to be tested.
Return	–	`boolean`	*true* if the element is a package.

```
$oracleHelper.isPackageDatabase($package)
```

Returns *true,* if a given package is a database.

	Name	Type	Description
Parameter	`$package`	`com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Package`	A package to be tested.
Return	–	`boolean`	*true* if a package is a database.

```
$oracleHelper.isPublic($element)
```

Returns *true,* if a given element has the public visibility.

	Name	Type	Description
Parameter	`$element`	`com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element`	An element to be tested.
Return	–	`boolean`	*true* if an element has the public visibility.

```
$oracleHelper.reverseList($list)
```

Reverses a given list.

	Name	Type	Description
Parameter	`$list`	`Java.util.List`	A list to be reversed.
Return	–	`Java.util.List`	A reversed list.

```
$oracleHelper.getRefName($element)
```

Returns a reference name description for the element with the "Ref:Element" tag.

	Name	Type	Description
Parameter	`$element`	`com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element`	An element to be tested.
Return	–	`java.lang.String`	A reference element name.

Example

In this sample, we will extend the current Oracle View DROP statement. In the Default template, we have the Oracle view drop function. In this sample, the simple macro is presented, and it generates the following text:

```
    DROP VIEW view_name;
```

See the following sample:

```
    #macro (dropView $data )
    #writeLine( "DROP VIEW $utils.getQualifiedName( $data )${semicolon}" $nospace)
    #end
```

We will add a new tag to identify the “CASCADE CONSTRAINTS” clause in the DROP VIEW statement. The following script will be generated:

```
    DROP VIEW view_name CASCADE CONSTRAINTS;
```

There are three steps to do this:

Step #1. Creating a new stereotype and tags

We need to create a new stereotype with the Boolean property, or extend the default “View” stereotype with a new property. Let's name a new property “cascade_clause”, apply the stereotype to the View object, and set value to *true*.

We added a new tag to the view stereotype in the default profile.

Step #2. Changing the template file

Add the following lines to the template file:

```
    #set($cascadeOption =false)
    ##this line is required to for setting a new variable to the false state.
    #set($cascadeOption = $oracleHelper.getBooleanValueFromDefaultProfile($data,
$VIEW_STEREOTYPE, "cascade_clause") )
```

This line retrieves a boolean value from the given tag in the given stereotype “$view_stereotype” and sets it to a newly created $cascadeOption value.

Then add a new checking clause into the drop statement. See the final method:

```
    #macro (dropView $data )
    ## sets a value to a new variable
    #set($cascadeOption = $oracleHelper.getBooleanValueFromDefaultProfile($data,
$VIEW_STEREOTYPE, "cascade_clause") )
    #writeLine( "DROP VIEW $utils.getQualifiedName( $data )
    #if ($cascadeOption) CASCADE CONSTRAINTS
    #end${semicolon}" $nospace)
    #end
```

Running programs in batch mode

There are cases when custom task must be executed in the batch (command line) mode. For example, if you want to run MagicDraw or other modeling tool application, open some project, execute code generation and exit application.

API provides two ways to run the tool in the batch mode depending on the required functionality.

1. Functionality is provided by an application core. You need to extend the *com.nomagic.magicdraw.commandline.CommandLine* class.
2. Functionality is provided by a plugin. Plugin loading is managed by the application core so it is undesirable to add core and plugin classes into the same class path. The different approach accessing plugin functionality is used. You need to implement the *com.nomagic.magicdraw.commandline.CommandLineAction* interface.

> You can find the code examples in
> 1. *<modeling tool installation directory>\openapi\examples\imagegenerator*. It takes a project file and destination directory as arguments and generates images for all diagrams.
> 2. *<modeling tool installation directory>\openapi\examples\commandlineplugin*. Just prints the program arguments to the application log file.

> The application cannot be run on the headless device even in a batch mode. The graphical environment is required.

> For more information on the launching batch mode programs, see Specifying batch mode program classpath(see page 2374).

More information is available in javadoc[1165].

Related pages

[1165] http://jdocs.nomagic.com/190beta/

Specifying batch mode program classpath and required system properties

MagicDraw or other modeling tool run in OSGI Framework[1166]. This implies that batch mode programs or any applications that start MagicDraw or other modeling tool have to run in OSGI Framework as well.

In order to launch the program, the launcher is started using a *java* command line which then starts the program inside the framework. The initial classpath have to include only jars needed to start the framework launcher. Here is the list of required jars on the classpath:

- `lib/com.nomagic.osgi.launcher-<latest version>.jar`
- `lib/bundles/org.eclipse.osgi_<latest version>.jar`
- `lib/bundles/com.nomagic.magicdraw.osgi.fragment_<latest version>.jar`
- `lib/md_common_api.jar`
- `lib/md_common.jar`
- `lib/md_api.jar`
- `lib/md.jar`
- `lib/jna.jar`

The rest of batch mode program is configured using system properties. MagicDraw or other modeling tool jars are specified using the system property *md.class.path*. The preferred way to specify this classpath is as follows:

```
file: <absolute or relative URL to MagicDraw or other modeling tool configuration
properties file> [?base=<path to MagicDraw or other modeling tool installation
directory>]#<classpath property name>
```

In case the current directory is the MagicDraw or other modeling tool installation directory, the `base` parameter does not have to be specified. The classpath property name for MagicDraw or other modeling tool property file is `CLASSPATH`.

There are other mandatory properties that have to be specified as well:

- `-Dcom.nomagic.osgi.config.dir=configuration`
- `-Desi.system.config=data/application.conf`
- `-Dlogback.configurationFile=data/logback.xml`

They here are specified as paths relative to *<modeling tool installation directory>* therefore you either have to make your current directory before launching your program or to specify absolute paths for above listed properties.

The main class to run is *com.nomagic.osgi.launcher.ProductionFrameworkLauncher* which is the OSGI framework launcher that starts the program in OSGI Framework.

The following system properties depends on the type of the batch mode program.

1. Core related batch program:

1166 https://en.wikipedia.org/wiki/OSGi

- `md.additional.class.path` - the jar(s) constituting batch mode program specific classpath is specified using this system property. The jar's are separated using the platform specific path separator in the same way as for a regular classpath.
- `com.nomagic.magicdraw.launcher` - specifies the batch mode program (implementation of *com.nomagic.magicdraw.commandline.CommandLine*) you want to launch, for example, `-Dcom.nomagic.magicdraw.launcher=com.nomagic.magicdraw.examples.imagegenerator.ExportDiagramImages`

2. Plugin related batch program:
   - `-Dcom.nomagic.magicdraw.launcher=com.nomagic.magicdraw.commandline.CommandLineActionLauncher`
   - `com.nomagic.magicdraw.commandline.action` - specifies the batch mode action (implementation of *com.nomagic.magicdraw.commandline.CommandLineAction*) you want to execute, for example, `-Dcom.nomagic.magicdraw.commandline.action=com.nomagic.magicdraw.examples.commandlineplugin.CommandLineActionExample`

The examples of complete command lines that launches the batch mode program:

Core batch program command line for Windows

```
C:\Program Files\MagicDraw>java -Xmx1200M -Xss1024K -XX:PermSize=60M
-XX:MaxPermSize=200M ^
-cp lib/com.nomagic.osgi.launcher-17.0.5-SNAPSHOT.jar;^
lib/bundles/org.eclipse.osgi_3.10.1.v20140909-1633.jar;^
lib/bundles/com.nomagic.magicdraw.osgi.fragment_1.0.0.201512211944.jar;^
lib/md_api.jar;lib/md_common_api.jar;lib/md.jar;lib/md_common.jar;lib/jna.jar ^
-Dmd.class.path=file:bin/magicdraw.properties#CLASSPATH ^
-Dcom.nomagic.osgi.config.dir=configuration ^
-Desi.system.config=data/application.conf ^
-Dlogback.configurationFile=data/logback.xml ^
-Dmd.additional.class.path=openapi/examples/imagegenerator/imagegenerator.jar ^
-Dcom.nomagic.magicdraw.launcher=com.nomagic.magicdraw.examples.imagegenerator.Export
DiagramImages ^
com.nomagic.osgi.launcher.ProductionFrameworkLauncher project_file=project.mdzip
destination_dir=out
```

Plugin batch program command line for Windows

```
C:\Program Files\MagicDraw>java -Xmx1200M -Xss1024K -XX:PermSize=60M
-XX:MaxPermSize=200M ^
-cp lib/com.nomagic.osgi.launcher-17.0.5-SNAPSHOT.jar;^
lib/bundles/org.eclipse.osgi_3.10.1.v20140909-1633.jar;^
lib/bundles/com.nomagic.magicdraw.osgi.fragment_1.0.0.201512211944.jar;^
lib/md_api.jar;lib/md_common_api.jar;lib/md.jar;lib/md_common.jar;lib/jna.jar ^
-Dmd.class.path=file:bin/magicdraw.properties#CLASSPATH ^
-Dcom.nomagic.osgi.config.dir=configuration ^
-Desi.system.config=data/application.conf ^
-Dlogback.configurationFile=data/logback.xml ^
-Dcom.nomagic.magicdraw.launcher=com.nomagic.magicdraw.commandline.CommandLineActionL
auncher ^
-Dcom.nomagic.magicdraw.commandline.action=com.nomagic.magicdraw.examples.commandline
plugin.CommandLineActionExample ^
com.nomagic.osgi.launcher.ProductionFrameworkLauncher argument1 argument2
```

Core batch program command line for Mac OS-X and Linux

```
> cd MagicDraw_183_no_install
> java -Xmx1200M -Xss1024K -XX:PermSize=60M -XX:MaxPermSize=200M \
       -Dmd.class.path=file:bin/magicdraw.properties#CLASSPATH \
       -Dcom.nomagic.osgi.config.dir=configuration \
       -Desi.system.config=data/application.conf \
       -Dlogback.configurationFile=data/logback.xml \

-Dcom.nomagic.magicdraw.launcher=com.nomagic.magicdraw.examples.imagegenerator.Export
DiagramImages \
       -cp lib/com.nomagic.osgi.launcher-17.0.5-SNAPSHOT.jar:\
`         `lib/bundles/org.eclipse.osgi_3.10.1.v20140909-1633.jar:\
`         `lib/bundles/com.nomagic.magicdraw.osgi.fragment_1.0.0.201512211944.jar:\
`         `lib/md_api.jar:lib/md_common_api.jar:lib/md.jar:lib/md_common.jar:lib/
jna.jar \
       -Dmd.additional.class.path=openapi/examples/imagegenerator/imagegenerator.jar
\
       com.nomagic.osgi.launcher.ProductionFrameworkLauncher
project_file=project.mdzip destination_dir=out
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Plugin batch program command line for Mac OS-X and Linux

```
> cd MagicDraw_183_no_install
> java -Xmx1200M -Xss1024K -XX:PermSize=60M -XX:MaxPermSize=200M \
       -Dmd.class.path=file:bin/magicdraw.properties#CLASSPATH \
       -Dcom.nomagic.osgi.config.dir=configuration \
       -Desi.system.config=data/application.conf \
       -Dlogback.configurationFile=data/logback.xml \

-Dcom.nomagic.magicdraw.launcher=com.nomagic.magicdraw.commandline.CommandLineActionL
auncher \

-Dcom.nomagic.magicdraw.commandline.action=com.nomagic.magicdraw.examples.commandline
plugin.CommandLineActionExample \
       -cp lib/com.nomagic.osgi.launcher-17.0.5-SNAPSHOT.jar:\
`         `lib/bundles/org.eclipse.osgi_3.10.1.v20140909-1633.jar:\
`         `lib/bundles/com.nomagic.magicdraw.osgi.fragment_1.0.0.201512211944.jar:\
`         `lib/md_api.jar:lib/md_common_api.jar:lib/md.jar:lib/md_common.jar:lib/
jna.jar \
       com.nomagic.osgi.launcher.ProductionFrameworkLauncher argument1 argument2
```

In case the command line program is not started from MagicDraw, the command line becomes a little bit more complex. The exemplary scripts that can be used to start the command line program are shown in the following script examples. Additionally, the scripts can be used with multiple MagicDraw versions since they construct a startup class path dynamically. In order to use them, the MagicDraw installation location has to be specified by setting the `MAGICDRAW_HOME` environment variable.

The shell script that starts a command line program on Mac OS-X, Linux, and Windows cygwin/msys:

Command line program shell script that is able to launch program from any directory

```
#!/bin/bash

if [ -z "$MAGICDRAW_HOME" ]; then
    echo "MAGICDRAW_HOME environment variable not set, please set it to the MagicDraw
installation folder"
    exit 1
fi

if [ "$OS" = Windows_NT ]; then
    md_home_url_leader=$(echo "$MAGICDRAW_HOME" | sed -e 's/^/\//' -e 's/ /%20/g' -e
's/\\/\//g')
    md_home_url_base=$(echo "$MAGICDRAW_HOME" | sed -e 's/:/%3A/g' -e 's/ /%20/g' \
                                                    -e 's/\//%2F/g' -e 's/\\/%5C/g')
    cp_delim=";"
else
    md_home_url_leader=$(echo "$MAGICDRAW_HOME" | sed -e 's/ /%20/g')
    md_home_url_base=$(echo "$MAGICDRAW_HOME" | sed -e 's/ /%20/g')
    cp_delim=":"
fi

md_cp_url=file:$md_home_url_leader/bin/magicdraw.properties?
base=$md_home_url_base#CLASSPATH

OSGI_LAUNCHER=$(echo "$MAGICDRAW_HOME"/lib/com.nomagic.osgi.launcher-*.jar)
OSGI_FRAMEWORK=$(echo "$MAGICDRAW_HOME"/lib/bundles/org.eclipse.osgi_*.jar)
MD_OSGI_FRAGMENT=$(echo "$MAGICDRAW_HOME"/lib/bundles/
com.nomagic.magicdraw.osgi.fragment_*.jar)

CP="${OSGI_LAUNCHER}${cp_delim}${OSGI_FRAMEWORK}${cp_delim}${MD_OSGI_FRAGMENT}$
{cp_delim}\
`   `$MAGICDRAW_HOME/lib/md_api.jar${cp_delim}$MAGICDRAW_HOME/lib/md_common_api.jar$
{cp_delim}\
`   `$MAGICDRAW_HOME/lib/md.jar${cp_delim}$MAGICDRAW_HOME/lib/md_common.jar${cp_delim}
\
`   `$MAGICDRAW_HOME/lib/jna.jar"

java -Xmx1200M -Xss1024K \
     -Dmd.class.path=$md_cp_url \
     -Dcom.nomagic.osgi.config.dir="$MAGICDRAW_HOME/configuration" \
     -Desi.system.config="$MAGICDRAW_HOME/data/application.conf" \
     -Dlogback.configurationFile="$MAGICDRAW_HOME/data/logback.xml" \

-Dcom.nomagic.magicdraw.launcher=com.nomagic.magicdraw.examples.imagegenerator.Export
DiagramImages \
     -cp "$CP" \
     -Dmd.additional.class.path="$MAGICDRAW_HOME/openapi/examples/imagegenerator/
imagegenerator.jar" \
     com.nomagic.osgi.launcher.ProductionFrameworkLauncher "$@"
```

A Windows batch file that starts a command line program on Windows:

Command line program Windows batch file that is able to launch program from any directory

```
@echo off
setlocal EnableExtensions

if "%MAGICDRAW_HOME%" == "" (
    echo MAGICDRAW_HOME environment variable not set, please set it to the MagicDraw
installation folder
    exit 1
)

setlocal enableDelayedExpansion
set md_home_url_leader=!MAGICDRAW_HOME: =%%20!
set md_home_url_leader=!md_home_url_leader:\=/!
set md_home_url_leader=/!md_home_url_leader!
set md_home_url_base=!MAGICDRAW_HOME::=%%3A!
set md_home_url_base=!md_home_url_base: =%%20!
set md_home_url_base=!md_home_url_base:/=%%2F!
set md_home_url_base=!md_home_url_base:\=%%5C!
setlocal disableDelayedExpansion

set md_cp_url=file:%md_home_url_leader%/bin/magicdraw.properties?
base=%md_home_url_base%#CLASSPATH

for /F %%a in ('dir /b "%MAGICDRAW_HOME%\lib\com.nomagic.osgi.launcher-*.jar"') do ^
set OSGI_LAUNCHER=%MAGICDRAW_HOME%\lib\%%a
for /F %%a in ('dir /b "%MAGICDRAW_HOME%\lib\bundles\org.eclipse.osgi_*.jar"') do ^
set OSGI_FRAMEWORK=%MAGICDRAW_HOME%\lib\bundles\%%a
for /F %%a in ('dir /b "%MAGICDRAW_HOME%
\lib\bundles\com.nomagic.magicdraw.osgi.fragment_*.jar"') do ^
set MD_OSGI_FRAGMENT=%MAGICDRAW_HOME%\lib\bundles\%%a

set CP=%OSGI_LAUNCHER%;%OSGI_FRAMEWORK%;%MD_OSGI_FRAGMENT%;^
%MAGICDRAW_HOME%\lib\md_api.jar;%MAGICDRAW_HOME%\lib\md_common_api.jar;^
%MAGICDRAW_HOME%\lib\md.jar;%MAGICDRAW_HOME%\lib\md_common.jar;^
%MAGICDRAW_HOME%\lib\jna.jar

java -Xmx1200M -Xss1024K ^
     -Dmd.class.path=%md_cp_url% ^
     -Dcom.nomagic.osgi.config.dir="%MAGICDRAW_HOME%\configuration" ^
     -Desi.system.config="%MAGICDRAW_HOME%\data\application.conf" ^
     -Dlogback.configurationFile="%MAGICDRAW_HOME%\data\logback.xml" ^

-Dcom.nomagic.magicdraw.launcher=com.nomagic.magicdraw.examples.imagegenerator.Export
DiagramImages ^
     -cp "%CP%" ^
     -Dmd.additional.class.path="%MAGICDRAW_HOME%
\openapi\examples\imagegenerator\imagegenerator.jar" ^
     com.nomagic.osgi.launcher.ProductionFrameworkLauncher %*
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Starting MagicDraw or other modeling tool as part of another application

To integrate MagicDraw or other modeling tool into your application you should launch your application in the OSGI Framework provided by MagicDraw or other modeling tool installation. This would allow you to interact with MagicDraw or other modeling tool classes directly.

In order to launch your application in the OSGI Framework you should create you application launcher extending *com.nomagic.magicdraw.commandline.CommandLine* class. See Running programs in batch mode(see page 2374) for further details. Starting MagicDraw or other modeling tool is achieved by invoking one of *com.nomagic.magicdraw.core.Application.start(...)* methods from your application code.

For details on how to run your application see Specifying batch mode program classpath and required system properties(see page 2375).

MagicDraw file format

File format in MagicDraw 3.6-17.0

MagicDraw (or other modeling tool developed by No Magic Inc.) file was a plain xmi file with extensions used to save diagrams information and project structure information (a mount table). The mdzip file was a compressed xmi file. The code engineering information was saved in the different file with extension "mdr".

File format in MagicDraw 17.0.1 and up

The inner project structure as well as the file format has been changed in version 17.0.1. An XMI version is upgraded to 2.4 with corresponding changes required by this standard.

A project is a collection of xmi, text, and binary resources. A native file format contains all resources saved as separate files. All these files are collected to one zip file.

Zip Based File Format (mdzip, xml.zip)

A zip file (xml.zip, mdzip) contains at least the following entries:

- Meta information about entries records.properties.
- Project identifier starts with PROJECT-, for example, PROJECT-f41072fa-d1aa-4e90-b7d9-fdd8862ed8af.
- Project structure information:

  com.nomagic.ci.metamodel.project.

- Shared and not shared uml model information:

  com.nomagic.magicdraw.uml_model.shared_model,

  com.nomagic.magicdraw.uml_model.model

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Proxies information backup of other external resources. Starts with proxy.
- Project options ends with

  com.nomagic.magicdraw.core.project.options.personalprojectoptions and

  com.nomagic.magicdraw.core.project.options.commonprojectoptions.

- Diagrams and other “binary” resources, entry names do not contain any meaningful information, just resource id, like 80a7058a-91b1-41fb-9a3a-cd7a8c6be52d.

Plain Text File Format (xml, mdxml)

Export to a non zip file formats like .xml or .mdxml is a combination of all zip file entries into the one plain xmi file. In this file, a uml model is saved as it was in earlier versions, while other information is saved as xmi:Extension.

Changes Related to Diagrams

Until version 17.0.1, diagrams were saved as an xmi extension. All diagrams were saved under one extension <mdOwnedDiagrams>. Both a diagram model and diagram symbols were saved in this extension. The <mdOwnedDiagrams> extension was saved outside a uml:Model element.

Version 17.0.1 saves the diagram element as a separate element extension named <modelExtension>. This extension is stored in a place where the diagram element is serialized (inside a diagram owner). If one owner has more than one diagram, they may be written in the one extension. In this extension, a diagram element is saved as a normal model element, and all its attributes and references are saved according xmi rules. A diagram element contains DiagramRepresentationObject, which contains information about a diagram type, required features, and so on. DiagramRepresentationObject contains DiagramContentsDescriptor, which describes used elements in a diagram and contains BinaryObject. BinaryObject has the attribute “streamContentID” which identifies a zip entry for diagram symbols. All information is stored in <mdOwnedViews>. The following schema presents finding diagram symbols entry id:

Diagram > DiagramRepresentationObject > DiagramContentsDescriptor > BinaryObject > BinaryObject. streamContentID

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Genmodel for DiagramRepresentationObject and DiagramContentsDescriptor

```
<?xml version="1.0" encoding="UTF-8"?>
<genmodel:GenModel xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI" xmlns:ecore="http://www.eclipse.org/emf/2002/
Ecore"
xmlns:genmodel="http://www.eclipse.org/emf/2002/GenModel" modelDirectory="/
com.nomagic.magicdraw.foundation/src"
modelPluginID="com.nomagic.magicdraw.foundation" modelName="Diagram
updateClasspath="false"
rootExtendsInterface="org.eclipse.emf.cdo.CDOObject"
rootExtendsClass="org.eclipse.emf.internal.cdo.CDOObjectImpl"
reflectiveDelegation="true" importerID="org.eclipse.emf.importer.ecore"
featureDelegation="Reflective"
containmentProxies="true" complianceLevel="5.0" copyrightFields="false"
usedGenPackages="../../com.nomagic.ci.metamodel.project/model/binary.genmodel#//
binary"
classNamePattern="">
    <foreignModel>diagram.ecore</foreignModel>
    <modelPluginVariables>CDO=org.eclipse.emf.cdo</modelPluginVariables>
    <genPackages prefix="Diagram" basePackage="com.nomagic.magicdraw.foundation"
    disposableProviderFactory="true"
    ecorePackage="diagram.ecore#/">
        <genClasses image="false" ecoreClass="diagram.ecore#//
        AbstractDiagramRepresentationObject">
            <genFeatures createChild="false" ecoreFeature="ecore:EAttribute
            diagram.ecore#//AbstractDiagramRepresentationObject/ID"/>
            <genFeatures createChild="false" ecoreFeature="ecore:EAttribute
            diagram.ecore#//AbstractDiagramRepresentationObject/type"/>
            <genFeatures createChild="false" ecoreFeature="ecore:EAttribute
            diagram.ecore#//AbstractDiagramRepresentationObject/umlType"/>
            <genFeatures createChild="false" ecoreFeature="ecore:EAttribute
            diagram.ecore#//AbstractDiagramRepresentationObject/diagramProperties"/>
            <genFeatures createChild="false" ecoreFeature="ecore:EAttribute
            diagram.ecore#//AbstractDiagramRepresentationObject/initialFrameSizeSet"/
>
            <genFeatures createChild="false" ecoreFeature="ecore:EAttribute
            diagram.ecore#//AbstractDiagramRepresentationObject/requiredFeature"/>
            <genFeatures property="None" children="true" createChild="true"
            ecoreFeature="ecore:EReference diagram.ecore#//
            AbstractDiagramRepresentationObject/diagramContents"/>
            <genFeatures createChild="false" ecoreFeature="ecore:EAttribute
            diagram.ecore#//AbstractDiagramRepresentationObject/diagramStyleID"/></
genClasses>
            <genClassesecoreClass="diagram.ecore#//DiagramContentsDescriptor">
            <genFeatures property="None" notify="false" createChild="false"
            ecoreFeature="ecore:EReference diagram.ecore#//DiagramContentsDescriptor/
representation"/>
            <genFeatures createChild="false" ecoreFeature="ecore:EAttribute
            diagram.ecore#//DiagramContentsDescriptor/exporterName"/>
            <genFeatures createChild="false" ecoreFeature="ecore:EAttribute
```

```
            diagram.ecore#//DiagramContentsDescriptor/exporterVersion"/>
            <genFeatures createChild="false" ecoreFeature="ecore:EAttribute
diagram.ecore#//DiagramContentsDescriptor/usedElements"/>
            <genFeatures property="None" children="true" createChild="true"
ecoreFeature="ecore:EReference diagram.ecore#//DiagramContentsDescriptor/
binaryObject"/>
            <genFeatures createChild="false" ecoreFeature="ecore:EAttribute
diagram.ecore#//DiagramContentsDescriptor/contentHash"/>
        </genClasses>
    </genPackages>
</genmodel:GenModel>
```

Jython Scripting

Modeling tools allow you to access open API using the Jython scripting language.

Jython is an implementation of the high-level, dynamic, and object-oriented language Python which is seamlessly integrated with the Java platform.

Using Jython you may access all java API and MagicDraw (or other modeling tool developed by No Magic Inc.) open API. This allows to avoid compilation and to get the same results without a java code. Using scripting you may do everything that you can achieve using a java plugin, and even more: you may change your code without recompiling and restarting an application.

More information about Jython you can find at http://www.jython.org[1167]. Information about the python language you can find at http://www.python.org[1168].

On every startup, program checks for scripts in *plugins/com.nomagic.magicdraw.jpython/scripts/*. If there are subdirectories, each of them are scanned for the *script.xml* file. This file provides information about the script in this directory. A file is similar to a plugin descriptor described in Plugin descriptor(see page 2162). If *script.xml* contains valid information, the script file specified in *script.xml* is executed. The script file should contain a valid Jython 2.5.3 script.

Related pages

Creating script

Let's describe creating a script in an example. We will create a script showing a message on a program startup. The creation process consists of three steps:

1. Creating a directory
2. Writing a script descriptor
3. Writing a script code

1167 http://www.jython.org/
1168 http://www.python.org/

Step #1: Create Directory

In the *plugins/com.nomagic.magicdraw.jpython* folder, create a *scripts* sub-folder, and then a folder for the particular script. For example, *plugins/com.nomagic.magicdraw.jpython/scripts/example*

Step #2: Write Script Descriptor

A script descriptor is a file written in XML and named *script.xml*. The script descriptor provides information about a script file to run, version of script, ID, and other.

In the created directory, create a *script.xml* file:

```
<?xml version="1.0" encoding="UTF-8"?>
<script
    id="example 1"
    name="Simple menu item"
    version="1.0"
    provider-name="No Magic"
    script-file="main.py"
    requiresApi="1.0">
</script>
```

The following table describes the *script.xml* file structure:

Element	Description	
script	**Attributes**	
	Name	**Description**
	id	A scrip ID, should be unique. Used to identify a script. Example: “my.first.script.0”
	name	A script name. No strict rules applied to this attribute. Example: “Example script”
	version	A script version. Allows numbers separated with one dot value. Examples: “1.0”, “0.1”
	provider-name	A script provider name. A company or an author name. Example: "No Magic"

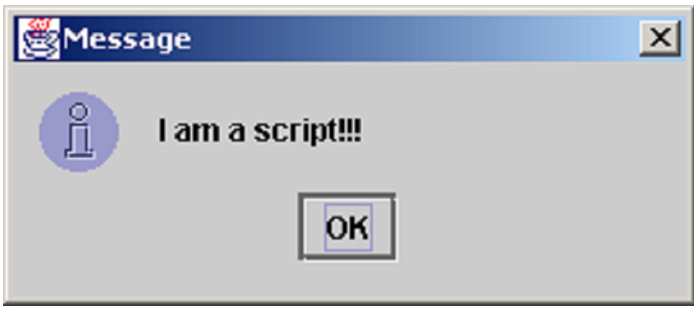
Element	Description	
	script-file	A relative path to a script file. This file will be executed. Example:"main.py"
	requires-api	A program API version required by a script. Example:"1.0"

Step #3: Write Script Code

Then in the same directory, create the *main.py* file:

```
from javax.swing import JOptionPane

# Script starts here
print "Starting script, descriptor", pluginDescriptor
JOptionPane.showMessageDialog( None, "I am a script!!!")
```

After saving files, restart your modeling tool. On a program startup, a message dialog should appear.



Variables Passed to Script

The program passes the one variable to the script *pluginDescriptor*. This variable contains information from the parsed *script.xml* file. A variable is an instance of a *com.nomagic.magicdraw.jpython.PythonPluginDescriptor*[1169] class.

A script can retrieve the script directory and other necessary information from the *pluginDescriptor* variable. There is no need to change any other fields for this variable.

1169 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/jpython/PythonPluginDescriptor.html

Plugins migration

We are keeping Open API stable as much as possible, but in some cases API changes in a newer application version. For example this happens if support for a new UML version is added or some very serious refactoring is done in components.

Sometimes it is enough to recompile plugin code with new libraries and adjust some minor API changes. Sometimes more serious migration is needed.

⚠ We recommend to recompile a plugin with new libraries to see the incompatibilities if any.

Related pages

Plugins migration to MagicDraw 18.2 and later Open API

The 18.2 version Open API changes are described here[1170].

Plugins migration to MagicDraw 18.1 and later Open API

18.1 Open API changes are described here[1171].

Plugins migration to MagicDraw 18.0 and later Open API

18.0 Open API changes are described here[1172].

Plugins migration to MagicDraw 17.0.1 and later Open API

If you are migrating from earlier than MagicDraw 15.0 version, please read the Plugins migration to MagicDraw 15.0 and later Open API(see page 2387) chapter first.

The changes that have been made to Open API are the following:

- **UML metamodel changes**. UML 2.4.1 specification introduced few changes in the UML metamodel itself. Some metamodel classes were added, some of them were removed, some metaclasses properties were changed. Most of these changes are not in a core UML, so they will affect you if your plugin is oriented to complex things in UML. For the details of changes, see Supported UML Specification Changes from Version 2.3 to 2.4.1.[1173]

1170 http://www.nomagic.com/news/new-noteworthy/magicdraw-noteworthy/magicdraw-18-2-fr/magicdraw-18-2-fr-all.html#api

117 http://www.nomagic.com/news/new-noteworthy/magicdraw-noteworthy/magicdraw-18-1-fr/magicdraw-18-1-fr-all.html#API_changes

217 http://www.nomagic.com/news/new-noteworthy/magicdraw-noteworthy/magicdraw-18-0-ltr/magicdraw-18-0-ltr-all.html#Open_API_Changes

1173 http://www.nomagic.com/supported-uml-specification-changes.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Project (decomposition) structure API changes(see page 2387).
- Changes in methods:
  - *com.nomagic.magicdraw.core.options.ProjectOptionsConfigurator*[1174]**.** The method *afterLoad(*[1175]ProjectOptions[1176]*)* has been added. An empty implementation of this method must be added to a custom ProjectOptionsConfigurator [1177]implementation.
  - *com.nomagic.magicdraw.core.project.ProjectEventListener*[1178]**.** The following methods have been added:

    projectActivatedFromGUI([1179]Project[1180])[1181]

    projectCreated([1182]Project[1183])[1184]

    projectOpenedFromGUI([1185]Project[1186])[1187]

    projectPreActivated([1188]Project[1189])[1190]

    projectPreClosed([1191]Project[1192])[1193]

    projectPreDeActivated([1194]Project[1195])[1196]

1174 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/options/ProjectOptionsConfigurator.html
1175 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/options/ProjectOptionsConfigurator.html#afterLoad-com.nomagic.magicdraw.core.options.ProjectOptions-
1176 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/options/ProjectOptions.html
1177 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/options/ProjectOptionsConfigurator.html
1178 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectEventListener.html
1179 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectEventListener.html#projectActivatedFromGUI-com.nomagic.magicdraw.core.Project-
1180 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Project.html
1181 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectEventListener.html#projectActivatedFromGUI-com.nomagic.magicdraw.core.Project-
1182 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectEventListener.html#projectCreated-com.nomagic.magicdraw.core.Project-
1183 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Project.html
1184 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectEventListener.html#projectCreated-com.nomagic.magicdraw.core.Project-
1185 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectEventListener.html#projectOpenedFromGUI-com.nomagic.magicdraw.core.Project-
1186 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Project.html
1187 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectEventListener.html#projectOpenedFromGUI-com.nomagic.magicdraw.core.Project-
1188 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectEventListener.html#projectPreActivated-com.nomagic.magicdraw.core.Project-
1189 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Project.html
1190 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectEventListener.html#projectPreActivated-com.nomagic.magicdraw.core.Project-
1191 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectEventListener.html#projectPreClosed-com.nomagic.magicdraw.core.Project-
1192 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Project.html
1193 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectEventListener.html#projectPreClosed-com.nomagic.magicdraw.core.Project-
1194 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectEventListener.html#projectPreDeActivated-com.nomagic.magicdraw.core.Project-
1195 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Project.html
1196 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectEventListener.html#projectPreDeActivated-com.nomagic.magicdraw.core.Project-

projectPreReplaced([1197]Project[1198],[1199]Project[1200])[1201]

projectPreSaved([1202]Project[1203], boolean)[1204]

projectPreClosedFinal([1205]Project[1206])[1207]

An empty implementation of these methods must be added to the custom *ProjectEventListener* [1208]implementation. The code change is not required, if *com.nomagic.magicdraw.core.project.ProjectEventListenerAdapter*[1209] is extended.

- Changes in **libraries jars** have been made. New jar files have been introduced in the *lib* folder. Do not forget to update your classpath(see page 2160).

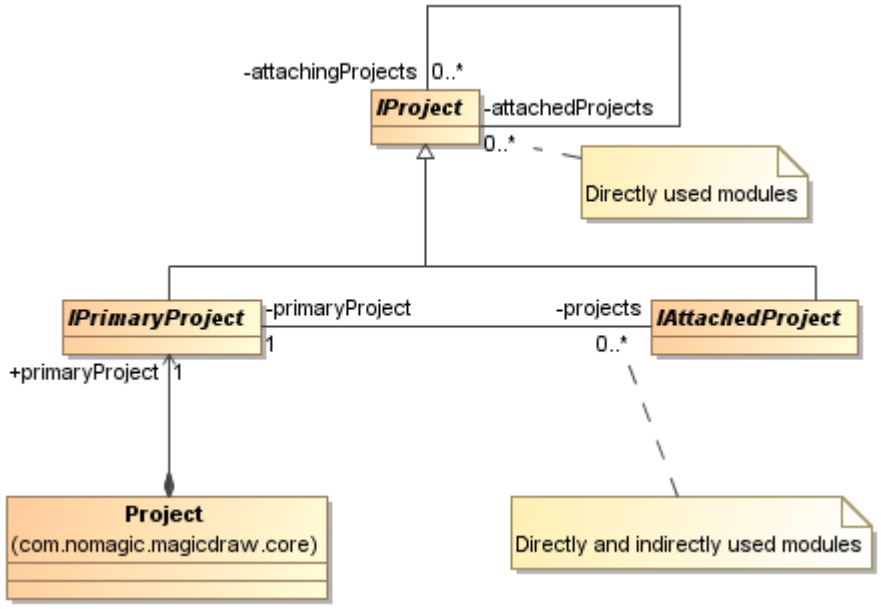
Related pages

Project (decomposition) structure API changes

The following figure shows the domain model of the project decomposition.

1197 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectEventListener.html#projectPreReplaced-com.nomagic.magicdraw.core.Project-com.nomagic.magicdraw.core.Project-
1198 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Project.html
1199 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectEventListener.html#projectPreReplaced-com.nomagic.magicdraw.core.Project-com.nomagic.magicdraw.core.Project-
1200 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Project.html
1201 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectEventListener.html#projectPreReplaced-com.nomagic.magicdraw.core.Project-com.nomagic.magicdraw.core.Project-
1202 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectEventListener.html#projectPreSaved-com.nomagic.magicdraw.core.Project-boolean-
1203 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Project.html
1204 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectEventListener.html#projectPreSaved-com.nomagic.magicdraw.core.Project-boolean-
1205 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectEventListener.html#projectPreClosedFinal-com.nomagic.magicdraw.core.Project-
1206 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Project.html
1207 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectEventListener.html#projectPreClosedFinal-com.nomagic.magicdraw.core.Project-
1208 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectEventListener.html
1209 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/project/ProjectEventListenerAdapter.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The core change is an introduction of com.nomagic.ci.persistence.IProject[1210] - com.nomagic.ci.persistence.IPrimaryProject[1211] and com.nomagic.ci.persistence.IAttachedProject[1212]. *IPrimaryProject*[1213]is a single instance in the scope of a project. *IPrimaryProject*[1214]can have attached (used) any number of *IAttachedProject*[1215](used projects), while *IAttachedProject*[1216]itself can attach other used projects (*IAttachedProject*[1217]).

The earlier than 17.0.1 project structure API made deprecated in version 17.0.1. The deprecated classes from the *com.nomagic.magicdraw.core.modules* package are:

Committable

IProject

ModuleDescriptor

ModulesManager

MountHelper

MountInfo

MountTable

MountTableListener

1210 http://jdocs.nomagic.com/190beta/com/nomagic/ci/persistence/IProject.html
1211 http://jdocs.nomagic.com/190beta/com/nomagic/ci/persistence/IPrimaryProject.html
1212 http://jdocs.nomagic.com/190beta/com/nomagic/ci/persistence/IAttachedProject.html
1213 http://jdocs.nomagic.com/190beta/com/nomagic/ci/persistence/IPrimaryProject.html
1214 http://jdocs.nomagic.com/190beta/com/nomagic/ci/persistence/IPrimaryProject.html
1215 http://jdocs.nomagic.com/190beta/com/nomagic/ci/persistence/IAttachedProject.html
1216 http://jdocs.nomagic.com/190beta/com/nomagic/ci/persistence/IAttachedProject.html
1217 http://jdocs.nomagic.com/190beta/com/nomagic/ci/persistence/IAttachedProject.html

ShareInfo

ShareTable

There is no straight one to one mapping, but the set of deprecated classes are now generally replaced by the set of the following classes:

com.nomagic.magicdraw.core.ProjectUtilities[1218]

com.nomagic.ci.persistence.IProject[1219]

com.nomagic.ci.persistence.IPrimaryProject[1220]

com.nomagic.ci.persistence.IAttachedProject[1221]

com.nomagic.ci.persistence.decomposition.ProjectAttachmentConfiguration[1222]

com.nomagic.ci.persistence.mounting.IMountPoint

com.nomagic.ci.persistence.sharing.ISharePoint

ProjectUtilities[1223]and com.nomagic.magicdraw.core.modules.ModulesService[1224] utility classes are introduced to help working with a new project decomposition structure and should replace *MountHelper*.

ModuleDescriptor is replaced with a combination of *IAttachedProject*[1225]and com.nomagic.ci.persistence.decomposition.ProjectAttachmentConfiguration[1226]. There is separate *ModuleDescriptor* for every (same) project usage in MagicDraw 17.0 or earlier versions, while there is a single *IAttachedProject*[1227]for the used project, but a separate *ProjectAttachmentConfiguration*[1228]for every project usage in the current version.

The example of the MagicDraw 17.0 and earlier version code:

1218 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/ProjectUtilities.html
1219 http://jdocs.nomagic.com/190beta/com/nomagic/ci/persistence/IProject.html
1220 http://jdocs.nomagic.com/190beta/com/nomagic/ci/persistence/IPrimaryProject.html
1221 http://jdocs.nomagic.com/190beta/com/nomagic/ci/persistence/IAttachedProject.html
1222 http://jdocs.nomagic.com/190beta/com/nomagic/ci/persistence/decomposition/ProjectAttachmentConfiguration.html
1223 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/ProjectUtilities.html
1224 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/modules/ModulesService.html
1225 http://jdocs.nomagic.com/190beta/com/nomagic/ci/persistence/IAttachedProject.html
1226 http://jdocs.nomagic.com/190beta/com/nomagic/ci/persistence/decomposition/ProjectAttachmentConfiguration.html
1227 http://jdocs.nomagic.com/190beta/com/nomagic/ci/persistence/IAttachedProject.html
1228 http://jdocs.nomagic.com/190beta/com/nomagic/ci/persistence/decomposition/ProjectAttachmentConfiguration.html

```
    final Collection<ModuleDescriptor> projectModules =
project.getModulesManager().getMountTable().getModules()
    for (ModuleDescriptor moduleDescriptor : projectModules)
    {
        // properties of the project usage
        final AutoLoadKind autoLoadKind = moduleDescriptor.getAutoLoadType();
        final boolean loadIndex = moduleDescriptor.isLoadIndex();
        final boolean editable = moduleDescriptor.isEditable();
    }
```

The example of a new MagicDraw 17.0.2 code:

```
    final IPrimaryProject primaryProject = project.getPrimaryProject();
    final Collection<IAttachedProject> projectAttachedProjects =
ProjectUtilities.getAttachedProjects(primaryProject);
    for (IAttachedProject attachedProject : projectAttachedProjects)
    {
        final ProjectAttachmentConfiguration attachmentConfiguration =
ProjectUtilities.getAttachment(primaryProject, attachedProject);
        // properties of the project usage
        final AutoLoadKind autoLoadKind =
ProjectUtilities.getAutoLoadKind(attachmentConfiguration);
        final boolean loadIndex =
ProjectUtilities.isLoadIndex(attachmentConfiguration);
        final boolean editable = !attachmentConfiguration.isReadOnly();
    }
```

Plugins migration to MagicDraw 15.0 and later Open API

Open API in version 15.0 and later versions has changed together with changes in the UML 2.1.2 (2.2) specification. You should read this chapter if you want to migrate your plugin created for earlier MagicDraw version.

Thereby we made a cleaner UML metamodel implementation API after the UML specification has been updated. There are no big changes in Open API, so migration will not be a long and complicate task for you.

The changes that have been made to Open API are the following:

- UML metamodel changes(see page 2393).
- Some **deprecated methods** have been removed:

Removed deprecated method	Substitution
`BaseElement.getProject()`	`Project.getProject(BaseElement)`
`MainFrame.getDialogParent()`	`MDDialogParentProvider.getProvider().getDialogParent()`
`MainFrame.setDialogParent(Frame)`	`MDDialogParentProvider.getProvider().setDialogParent(Frame)`
`ElementListProperty.setStereotype(Stereotype)`	`ElementListProperty.setSelectableRestrictedElements(Collection)`

- Changes in **libraries jars** have been made. Open API classes now are packaged into *md_api.jar* and *md_common_api.jar* (was *md.jar* and *md_common.jar*). If your plugin was using API classes from some program build-in plugin, a plugin jar is also renamed with a "*ProgramInitials_api.jar*" pattern.
- Changes in **package names for build-in plugins** have been made. In earlier program versions, some build-in plugins exposed their Open API, for example, patterns, transformations, emf, and etc. We have changed a package name pattern. Earlier API versions used a *com.nomagic.magicdraw.plugins.impl.***** pattern, now these plugins are moved to *com.nomagic.magicdraw.*.*,* for example *com.nomagic.magicdraw.patterns.**. A simple import statement change in your java source will solve this migration issue.

Related pages

UML metamodel changes

There are two types of changes in UML metamodel API.

UML specification changes

The UML 2.1.2 (2.2) specification introduced few changes in the UML metamodel itself. Some metamodel classes were added, some of them were removed, some metaclasses properties were

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

changed. Most of these changes are not in a core UML, so they will affect you if your plugin is oriented to complex things in UML.

Most changes are made in UML Templates, Simple Time, Interactions. We suggest to look at UML 2.1.2 (2.2) specification for such changes if you see some compile errors.

UML metamodel API implementation changes

Earlier UML metamodel API used several interface layers for every compatibility level described in the UML specification. All these layers were merged into one using package merging, this is why earlier API had so many interfaces for every metamodel element in a different package. Such structure was very complicated and hardly understandable for new UML users.

MagicDraw version 15.0 and later UML metamodel API provides just one final merged layer. All intermediate layers were dropped from API. This change reduced a UML metamodel API size in few times.

Interfaces that are removed form the following packages:

- *com.nomagic.uml2.ext.omg*
- *com.nomagic.uml2.magicdraw*
- *com.nomagic.uml2.omg*

We left the same package name for merged interfaces like in a previous API version, that is *com.nomagic.uml2.ext.magicdraw.***** . You do not need to make any changes in your code if you were using interfaces from this layer. Otherwise you need to import a statement in your java source files, for example, *import com.nomagic.uml2.omg.kernel.Element* to *import com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element*.

UML metamodel interfaces use Java 5 generics, so it is much easier to understand types of various collections in the metamodel.

Getting licensing information

The *com.nomagic.magicdraw.license.utils.LicenseUtils*[1229] may be used to check some licensing information (edition, is it a trial or evaluation version).

Multi-threading

The majority of the application code (including OpenAPI) is not thread safe. Only the UML model is thread safe for reading (the UML model can be queried safely by concurrent threads), but still the UML model writing (modification) is not thread safe.

Related pages

- Running action with progress(see page 2208)

1229 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/license/utils/LicenseUtils.html

Idle job service

The *com.nomagic.magicdraw.job.IdleJobService*[1230] allows to add *com.nomagic.magicdraw.job.Job(s)*[1231] which can be executed when a program is idle. The program decides when it is the most appropriate to execute the Job[1232]without getting the program into an inconsistent state. The example of usage:

```
Job job = new Job()
{
    public boolean needsExecute()
    {
        return true;
    }

    public void execute(ProgressStatus progressStatus) throws Exception
    {
        // Do necessary work in the session.
    }

    public void finished()
    {
        // Cleanup after job is finished.
    }

    public String getName()
    {
        return "My Job";
    }
};

IdleJobService.getInstance().addJob(job);

// Remove the job when it is no longer necessary.
IdleJobService.getInstance().removeJob(job);
```

For more information please refer to *<programinstallation directory>\openapi\examples\JobExample*.

Related pages

- Multi-threading(see page 2394)

1230 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/job/IdleJobService.html
1231 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/job/Job.html
1232 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/job/Job.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Application environment

Use com.nomagic.magicdraw.core.Application.runtime()[1233] to retrieve a *com.nomagic.magicdraw.core.Application.Runtime*[1234] object. It contains an application name, version, and file format information.

Use Application.environment()[1235] to retrieve an *com.nomagic.magicdraw.core.Application.Environment*[1236] object. It contains application environment directory information like install, configuration, data, templates, profiles, etc.

Attached Files

Use com.nomagic.magicdraw.fileattachments.FileAttachmentsHelper[1237] to create, remove or save attached files to disk.

> You can find the code examples in
> - *<programinstallation directory>\openapi\examples\fileattachments*

Element Referencing in Texts

Open API provides the *com.nomagic.magicdraw.elementreferenceintext.ElementReferencingInTexts*[1238] class for creating text with element references, setting, and getting text from elements.

> **Examples**
> You can find the code examples in *<modeling_tool_installation_directory>\openapi\examples\elementreferencingintext.*

Creating text that refers an element

To create text that refers an element, use:

- ElementReferencingInTexts.createReferencingText([1239]BaseElement[1240], java.lang.String)[1241],

---

1233 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Application.html#runtime--
1234 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Application.Runtime.html
1235 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Application.html#environment--
1236 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/core/Application.Environment.html
1237 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/fileattachments/FileAttachmentsHelper.html
1238 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/elementreferenceintext/ElementReferencingInTexts.html
1239 1241 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/elementreferenceintext/ElementReferencingInTexts.html#createReferencingText-com.nomagic.magicdraw.uml.BaseElement-java.lang.String-
1240 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/BaseElement.html

- ElementReferencingInTexts.createReferencingText([1242]BaseElement[1243], java.lang.String,[1244]DisplayMode[1245])[1246],
- *ElementReferencingInTexts.createReferencingText*[1247](BaseElement[1248], java.lang.String,[1249] DisplayMode[1250], UpdateMode[1251])

```
//creates a text that references an element with default options
ElementReferencingInTexts.createReferencingText(element, "Link");

//creates a text that is updated automatically and is displayed using its
representation text.
//This means the elements representation text will be displayed rather than "Link"
ElementReferencingInTexts.createReferencingText(element, "Link",
DisplayMode.REPRESENTATION_TEXT,
UpdateMode.AUTOMATIC_UPDATE);

//create a referencing text with a provided display mode and with a default update
mode
ElementReferencingInTexts.createReferencingText(element, "Link",
DisplayMode.REPRESENTATION_TEXT);
```

Setting text with element references

Some element properties support text with references, e.g., a comment body, constraint specification, opaque behavior body, etc. In order to set the text with element references, you must convert it using com.nomagic.text.TextUtils.toLightHtml(java.lang.String)[1252].

1242 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/elementreferenceintext/ElementReferencingInTexts.html#createReferencingText-com.nomagic.magicdraw.uml.BaseElement-java.lang.String-com.nomagic.magicdraw.elementreferenceintext.DisplayMode-
1243 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/BaseElement.html
1244 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/elementreferenceintext/ElementReferencingInTexts.html#createReferencingText-com.nomagic.magicdraw.uml.BaseElement-java.lang.String-com.nomagic.magicdraw.elementreferenceintext.DisplayMode-
1245 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/elementreferenceintext/DisplayMode.html
1246 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/elementreferenceintext/ElementReferencingInTexts.html#createReferencingText-com.nomagic.magicdraw.uml.BaseElement-java.lang.String-com.nomagic.magicdraw.elementreferenceintext.DisplayMode-
1247 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/elementreferenceintext/ElementReferencingInTexts.html#createReferencingText-com.nomagic.magicdraw.uml.BaseElement-java.lang.String-com.nomagic.magicdraw.elementreferenceintext.DisplayMode-com.nomagic.magicdraw.elementreferenceintext.UpdateMode-
1248 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/uml/BaseElement.html
1249 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/elementreferenceintext/ElementReferencingInTexts.html#createReferencingText-com.nomagic.magicdraw.uml.BaseElement-java.lang.String-com.nomagic.magicdraw.elementreferenceintext.DisplayMode-com.nomagic.magicdraw.elementreferenceintext.UpdateMode-
1250 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/elementreferenceintext/DisplayMode.html
1251 http://jdocs.nomagic.com/190beta/com/nomagic/magicdraw/elementreferenceintext/UpdateMode.html
1252 http://jdocs.nomagic.com/190beta/com/nomagic/text/TextUtils.html#toLightHtml-java.lang.String-

Setting text for comment body

```
//create a text with an element reference
String constructedText = "This text refers an element: " +
createReferencingText(elementToRefer);
//convert it to a light html. Light html can be used in a comment body, constraint
specification, etc. to show th etext containing references
String textForCommentBody = TextUtils.toLightHtml(constructedText);

Project project = Project.getProject(comment);
SessionManager.getInstance().createSession(project, "Set comment text with reference")
;
comment.setBody(textForCommentBody);
SessionManager.getInstance().closeSession(project);
```

Code Engineering

Demos

- Code Generation[1253]
- Code Reverse[1254]

Abbreviations

- **UML** Unified Modeling Language
- **GUI** Graphical User Interface
- **CE** Code Engineering
- **CES** Code Engineering Set
- **AST** Abstract Syntax Tree
- **JLS** Java Language Specification
- **JVM** Java Virtual Machine
- **RT** Round-Trip forward and backward code engineering without code loss CLR Common Language Runtime
- **DSL** Domain Specific Language

> **Functionality Availability**
> Code Engineering is available in Professional, Architect and Enterprise editions (see page 2398)only.

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.

1253 http://www.magicdraw.com/files/viewlets/MD_100_viewlets_CodeGeneration_viewlet_swf.html
1254 http://www.magicdraw.com/files/viewlets/MD_100_viewlets_CodeReverse_viewlet_swf.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

There are several cases where code engineering may be very useful:

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

The tool may generate code from models or create models out of code. Changes in the existing code can be reflected in the model, and you can see model changes in your code. You can also merge independent changes to a model and code without destroying data in the code or model.

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 you can create UML models from the source code written in those languages. Additionally, the tool supports reversing from Java Bytecode and CIL.

Related pages:

- Code Engineering Sets(see page 2400)
- Generating Code(see page 2404)
- Reverse Options(see page 2407)
- Global options for Code Engineering(see page 2431)
- Files of Properties(see page 2442)
- Java Code Engineering(see page 2443)
- C++ Code Engineering(see page 2470)
- C# Code Engineering(see page 2622)
- CORBA IDL Mapping To UML(see page 2680)
- WSDL(see page 2683)

Code Engineering Sets

You can 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 as well as instruments for managing those sets.

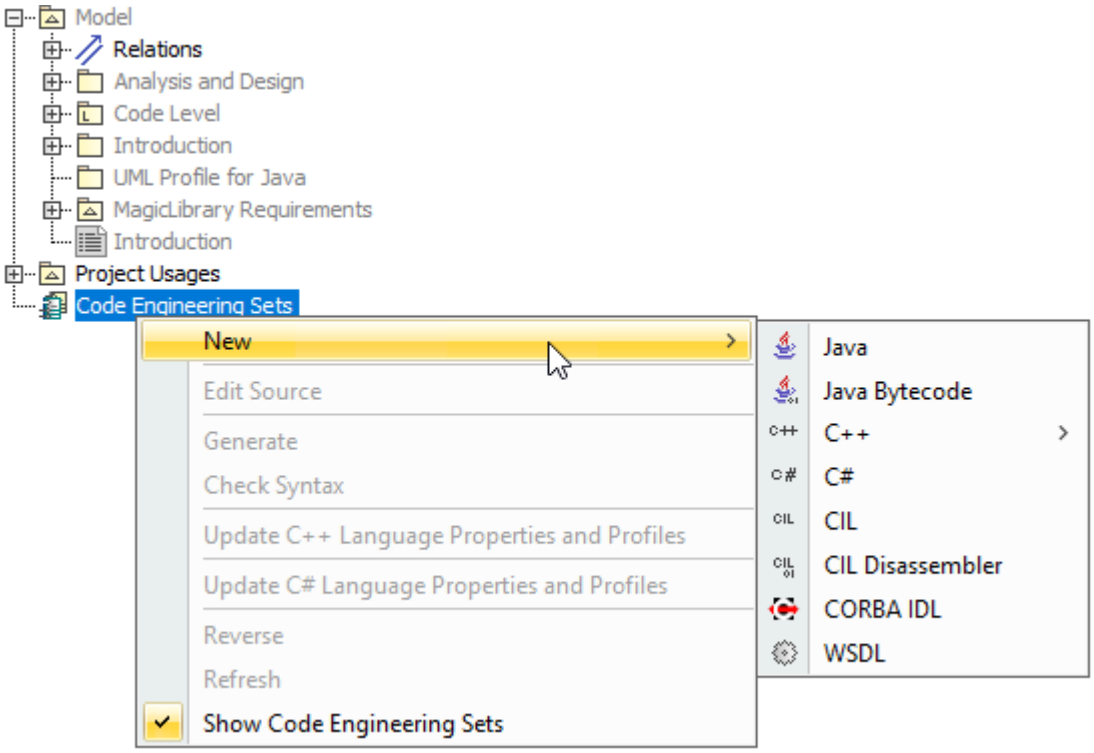
To add a new set

---

1. Choose **New** from the **Code Engineering Sets** shortcut menu.
2. Choose the language you want:
    - Java
    - Java Bytecode
    - C++
    - C#
    - CIL

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- CIL Disassembler
- CORBA IDL
- DDL (Cloudscape, DB2, Microsoft Access, Microsoft SQL Server, MySQL, Oracle, Pervasive, Pointbase, PostgreSQL, Sybase) For DDL, the Cameo Data Modeler plugin[1255] is required
- XML Schema, the Cameo Data Modeler plugin[1256] is required
- WSDL



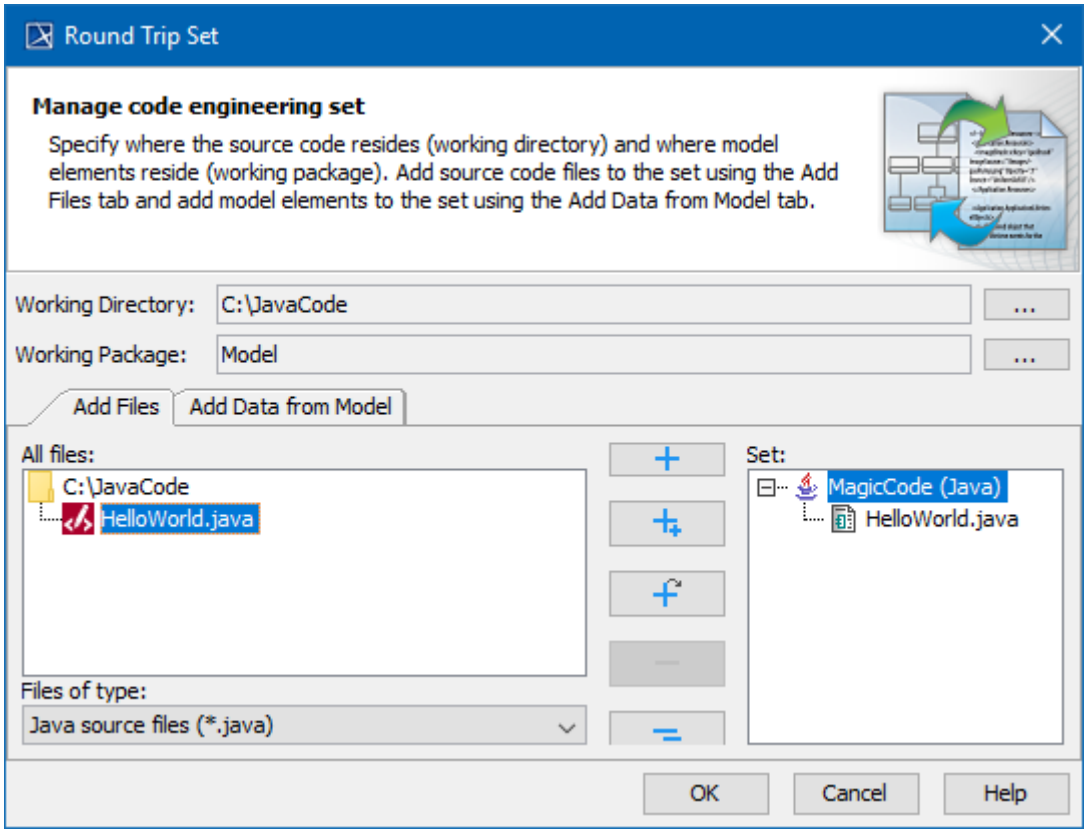
Edit sets in the **Round Trip Set** dialog box.

To open the **Round Trip Set** dialog box

- Right-click the selected set and choose **Edit** from the set shortcut menu.
- Select the set and press Enter.

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

1255 https://docs.nomagic.com/display/CDMP190SP4
1256 https://docs.nomagic.com/display/CDMP190SP4



Round Trip Set dialog box. Add files tab

Specify the **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 browse in the directory tree by clicking the ... button.

The **Working Package** box allows you to define any package for reverse output or code generation. A model is reversed or code generated from this specified package.

> You can only select or change the working package prior to the addition of files from the working directory to the code engineering set.

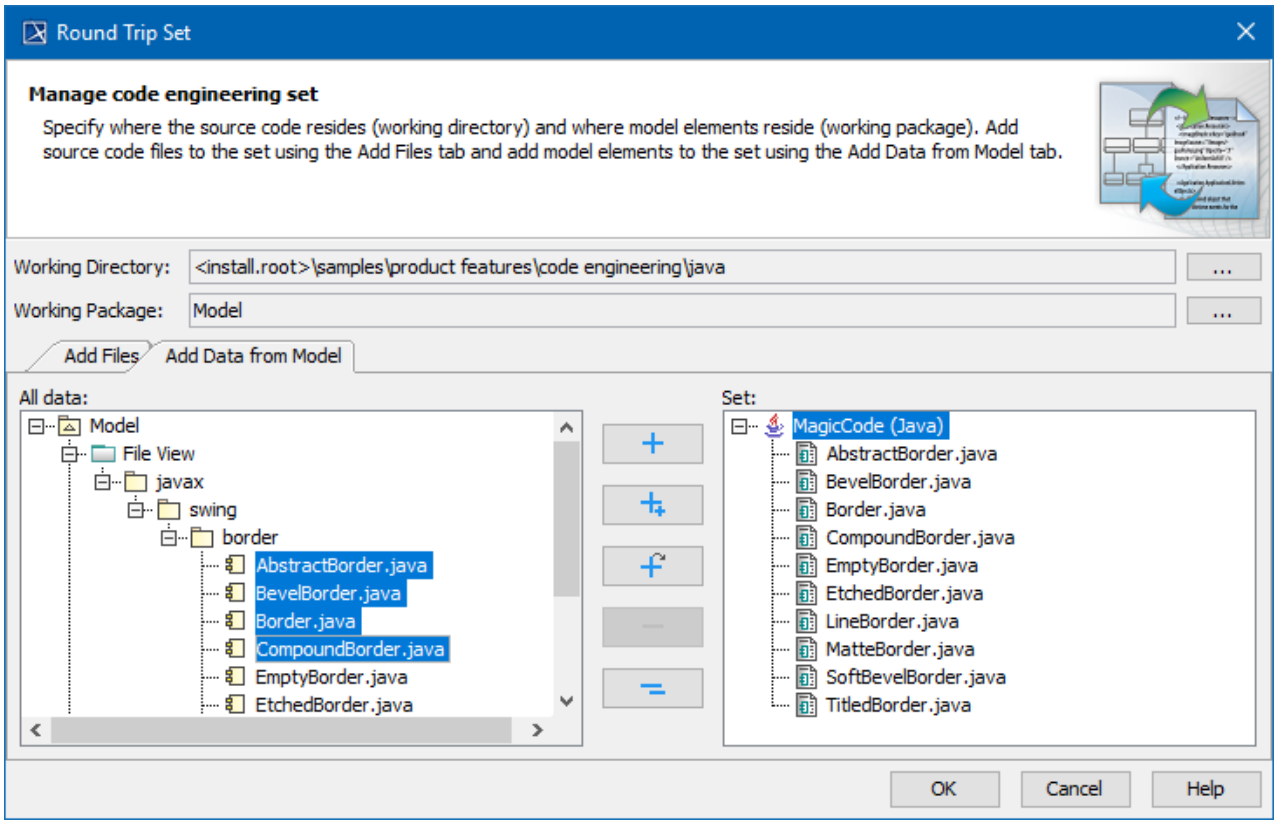
The **Round Trip Set** dialog box has two tabs: **Add Files** and **Add Data from Model**.

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

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

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

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



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. 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.

Related Pages:

- Code Engineering(see page 2398)
  - Code Engineering Sets(see page 2399)
  - Generating Code(see page 2404)
  - Reverse Options(see page 2407)
  -

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Global options for Code Engineering(see page 2431)
- Files of Properties(see page 2442)
- Java Code Engineering(see page 2443)
- C++ Code Engineering(see page 2470)
- C# Code Engineering(see page 2622)
- CORBA IDL Mapping To UML(see page 2680)
- WSDL(see page 2683)

Generating Code

Demo

- Code Generation[1257]

You can generate code for the selected and prepared set as well as directly for model elements.

Code Generation for a Set

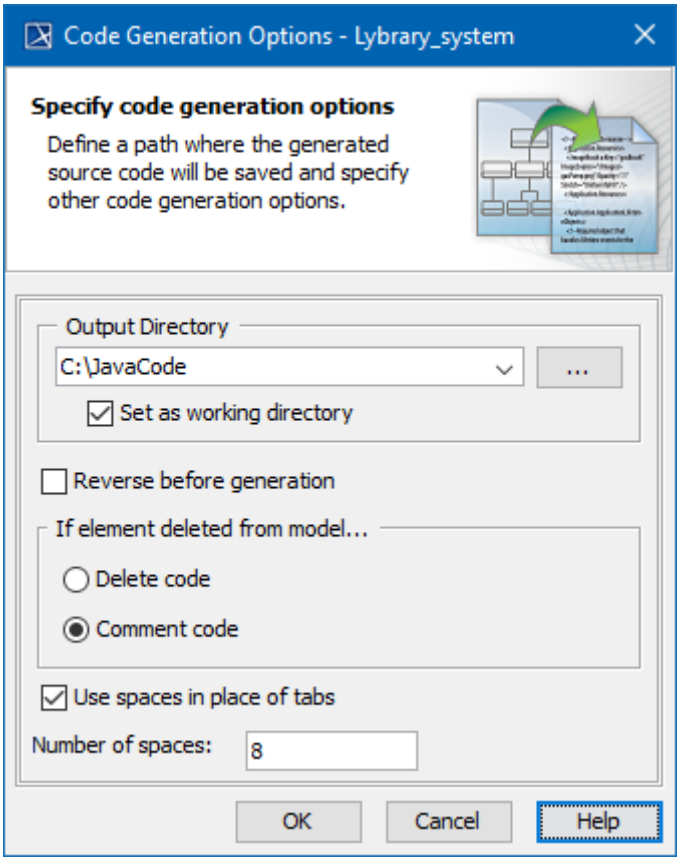
Start code generation once the set or sets are prepared. For more details about creating and editing sets, see Code Engineering Sets(see page 2399).

- 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.

1257 http://www.magicdraw.com/files/viewlets/MD_100_viewlets_CodeGeneration_viewlet_swf.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Code Generation Options dialog box.

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

Once you specify generating options 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.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Box Name	Function
Reverse before generation	Changes your model according to changes in the existing code. 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 these cases, you will lose some of your work.
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 about 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 a 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 to generate in the browser Data package and click Generate in the class shortcut menu. For packages and components, you can 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 choose 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 can view the results with your favorite file viewer or programming environment. Look for the files in the directory you specified as your **Working directory** in the **Round Trip Set** dialog(see page 2399) box or in the **Project Options** dialog(see page 2220) box. Additional sub-directories could be created.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related Pages:

- Code Engineering(see page 2398)
  - Code Engineering Sets(see page 2399)
  - Generating Code(see page 2403)
  - Reverse Options(see page 2407)
  - Global options for Code Engineering(see page 2431)
  - Files of Properties(see page 2442)
  - Java Code Engineering(see page 2443)
  - C++ Code Engineering(see page 2470)
  - C# Code Engineering(see page 2622)
  - CORBA IDL Mapping To UML(see page 2680)
  - WSDL(see page 2683)

Reverse Options

Demo

- Code Reverse[1258]

On this page

- Importing JAVA .JAR packages(see page 2426)

A reverse is the opposite of code generation. Existing code can be converted to UML models with the help of the MagicDraw reverse feature. In order to perform a reverse operation, prepare the sets the same way you did for code generation, see Code Engineering Sets(see page 2399).

- Choose **Reverse** from the **Code engineering sets** item shortcut menu.
- Choose **Reverse** from the selected set shortcut menu.

The UML model for the component can be reversed in the same way. Simply select the component you are interested in from the browser and click Reverse on its shortcut menu. Models can be reversed without creating a set.

To reverse a model without creating a set

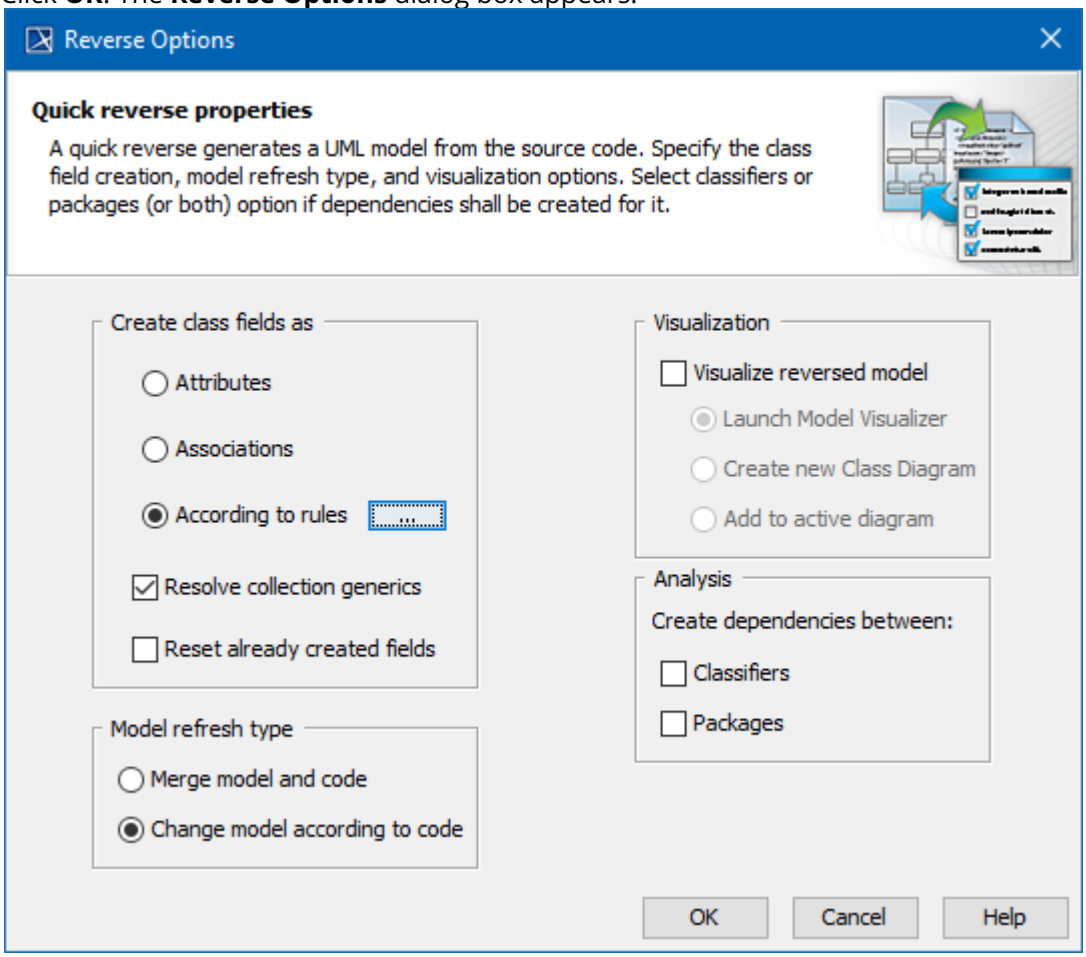
---

1. Choose **Quick Reverse** from the **Tools** menu. The **Round Trip Set** dialog box(see page 2399) appears. Quick Reverse is available only in Professional and Enterprise editions(see page 2406).
2. Select the files from the **Round Trip Set** dialog box, **Add Files** tab.

1258 http://www.magicdraw.com/files/viewlets/MD_100_viewlets_CodeReverse_viewlet_swf.html

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

3. Click **OK**. The **Reverse Options** dialog box appears.



The following table outlines Dialog descriptions:

Area	Element name	Function
Create class fields as	**Attributes**	Class (see page 666)fields are represented in a model as attributes(see page 672).
	Associations	Class fields are represented in a model as association ends(see page 654).
	According to rules	Association (see page 651)or Attribute creation on reverse is the ability to enter rules (see page 2407)that help decide if an association or attribute must be created on reverse.

Area	Element name	Function
	Resolve collection generics	Reverse engineering can create associations when one class has a collection of other classes and uses Java generics (for example, *List<String>*). If selected, types of collections will be resolved (property type will not be a collection, but a real type). Predefined container types in Java language properties will be appended by all the same containers in this form: *·java.util.List<$$type$$>* where *$$type$$* replaced by the value of "type" property when the code is generated.
	Reset already created fields	Select this option if you want to keep previously 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	The model will be created strictly by code. Entities in the model that do not match entities in the code will be discarded.	
Vizualization	**Visualize reserved model**	Classes created while reversing can be added to a diagram.
	Launch Model Visualizer	After reversing, the **Model Visualizer** dialog box(see page 1346) appears. It will assist you in creating a Class diagram(see page 297) or Sequence diagram(see page 304) (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.

Area	Element name	Function
Analysis. **Create dependencies between**	**Classifiers**	Dependencies between classes will be analyzed and created.
	Packages	Dependencies only between packages will be created.

> Method bodies are not parsed on dependency search. Only static information is used.

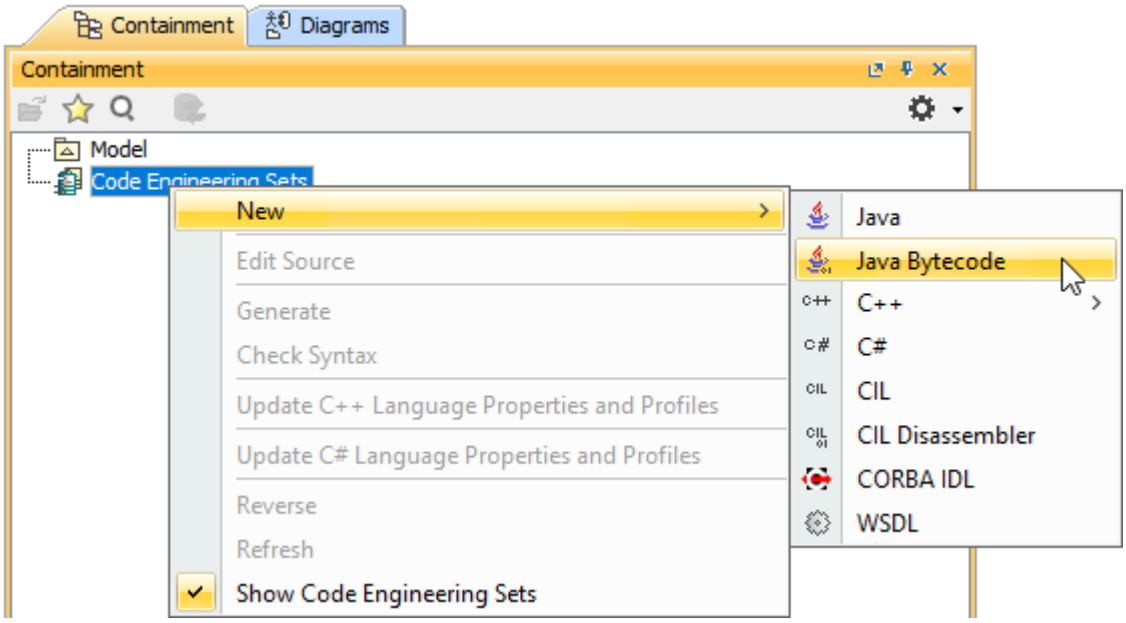
If you have a code set combined from several files, you may see changes you wish to model without reversing all the code. Only changed files should be reversed. You can do this type of reversing by clicking the **Refresh** button on the set shortcut menu, or by performing model refresh from the Code Engineering Sets dialog box(see page 2399).

Importing JAVA .JAR packages

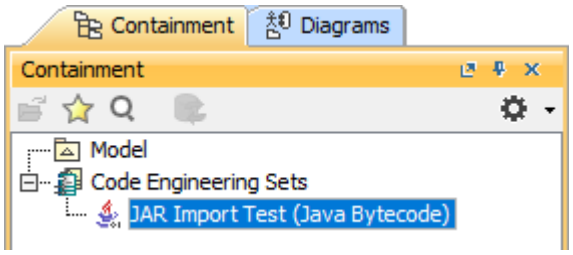
Apart from reversing code operation, **Code Engineering Sets** also allows you to import JAVA .JAR files and their Packages using **Java Bytecode** on the **Code Engineering Sets** shortcut menu.

To include (or import) a JAVA .JAR package with Java code generation

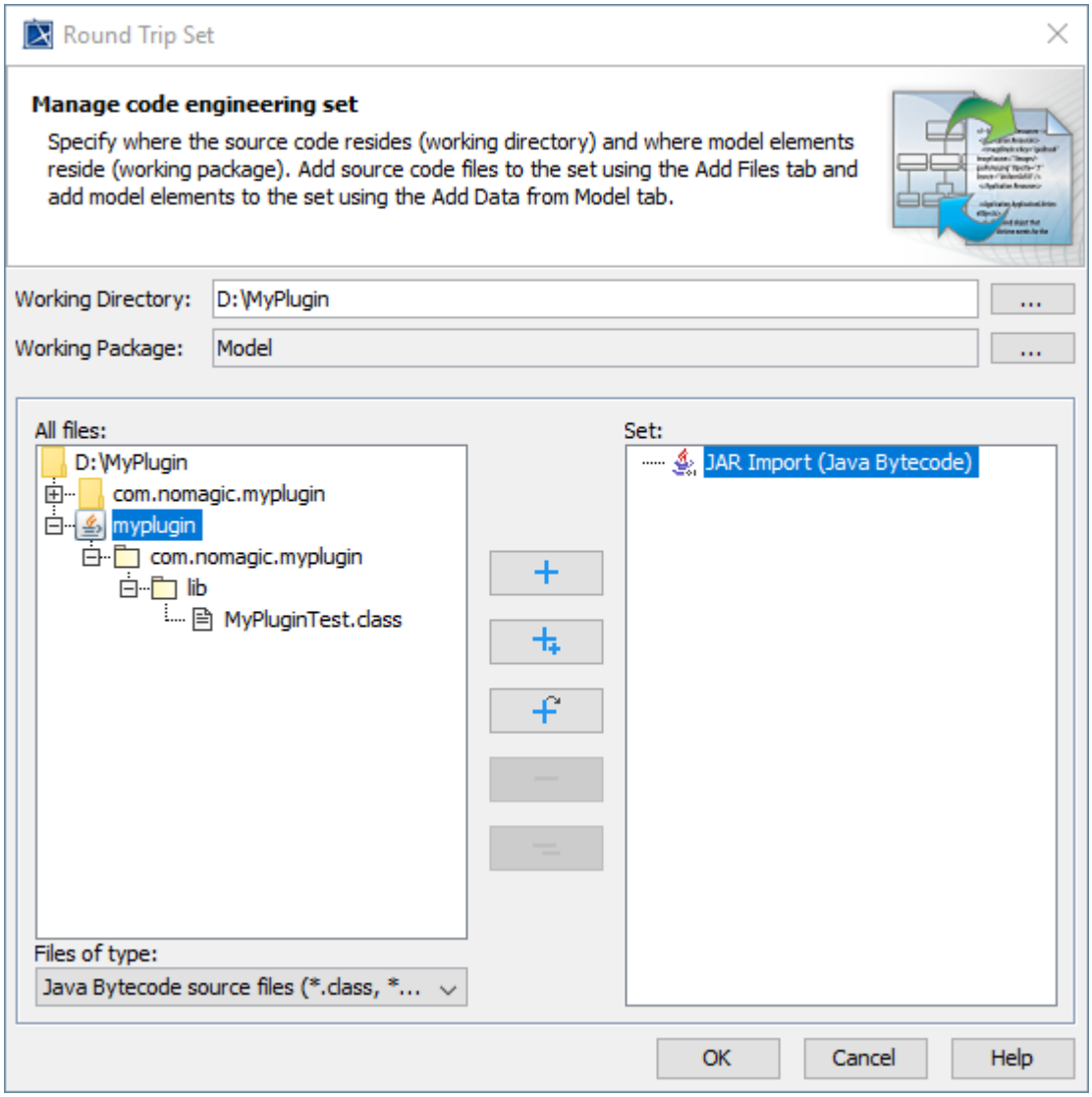
1. Create a new UML project or open an existing one.
2. In the Containment tree, right-click **Code Engineering Sets**. Select **New** > **Java Bytecode**.



3. Name your new Java Bytecode engineering set and click **OK**. The new Java Bytecode engineering set is created under **Code Engineering Sets** in the Containment tree.



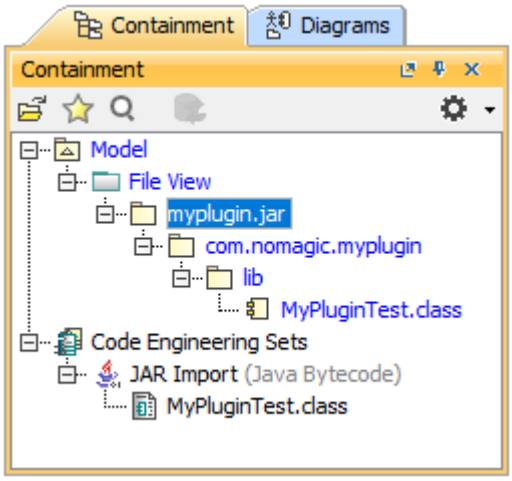
4. Right-click your newly created Java Bytecode engineering set. Select **Edit**.
5. The **Round Trip Set** dialog(see page 2399) opens. In the **Working Directory** box, specify the root directory of the .JAR file that you want to import.



6. In the **All files** list, select the .JAR file you want to import.

> **Note**
> The file hierarchy of the imported .JAR file in the Containment tree will start from the working directory you set. For example, if you set the working directory as the MagicDraw installation directory, the whole file hierarchy that the imported .JAR file is in the installation directory will be also imported into the Containment tree.

7. Select and click **OK**. The imported .JAR file is now in the Containment tree under the **Model** Package, **File View** Package, and the rest of the file hierarchy of the imported .JAR file.



Related Pages:

- Rules of association or attribute creation on reverse(see page 2428)
- Code Engineering(see page 2398)
  - Code Engineering Sets(see page 2399)
  - Generating Code(see page 2403)
  - Reverse Options(see page 2406)
  - Global options for Code Engineering(see page 2431)
  - Files of Properties(see page 2442)
  - Java Code Engineering(see page 2443)
  - C++ Code Engineering(see page 2470)
  - C# Code Engineering(see page 2622)
  - CORBA IDL Mapping To UML(see page 2680)
  - WSDL(see page 2683)

Rules of association or attribute creation on reverse

Association(see page 651) or Attribute(see page 672) creation on reverse is the ability to enter rules to help decide if an association or attribute must be created on reverse.

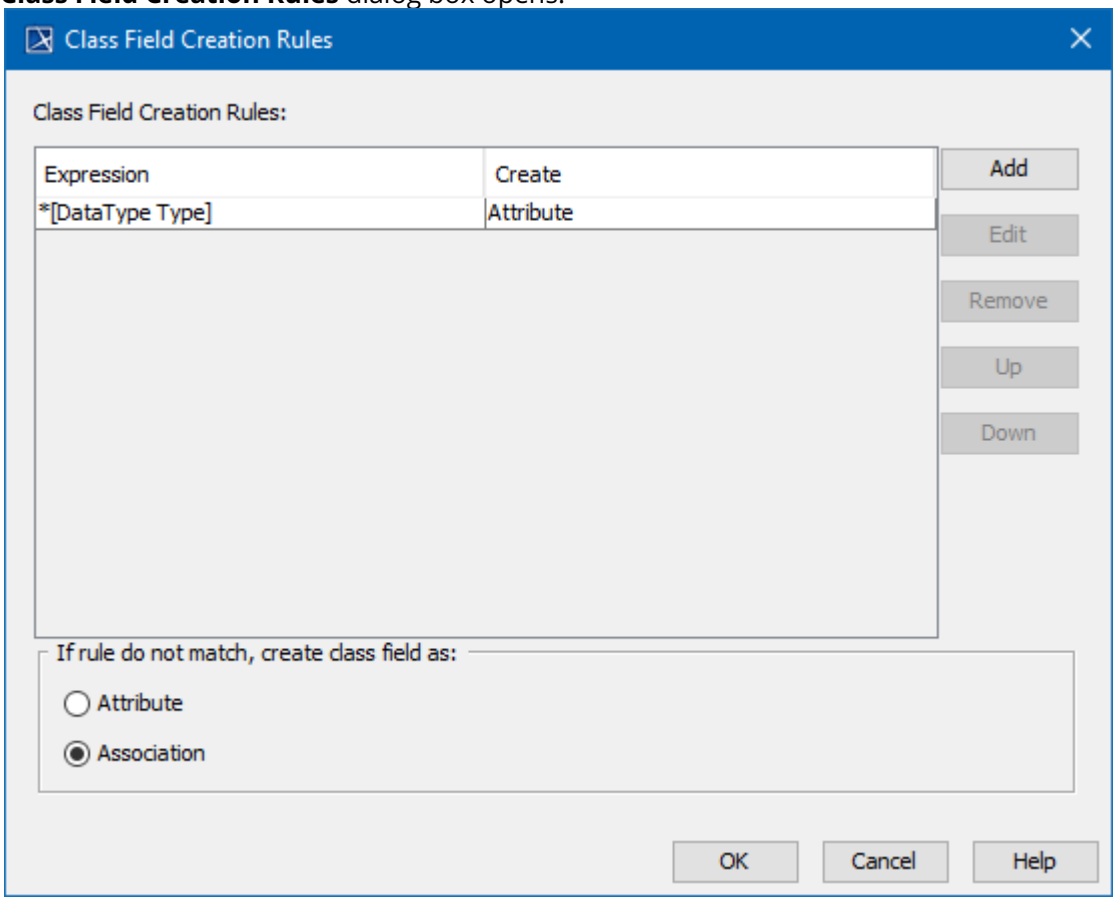
To create association or attribute creation rules on reverse

---

1. In the Model Browser(see page 29), select the code engineering set for reverse.
2. Right-click the set and choose the **Reverse** command. The **Reverse Options** dialog box(see page 2406) opens.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

3. In the **Create class fields as** area, select **According to rules** and click the button. The **Class Field Creation Rules** dialog box opens.



4. Click **Add** to add a rule.

Managing association or attribute creation on reverse rules

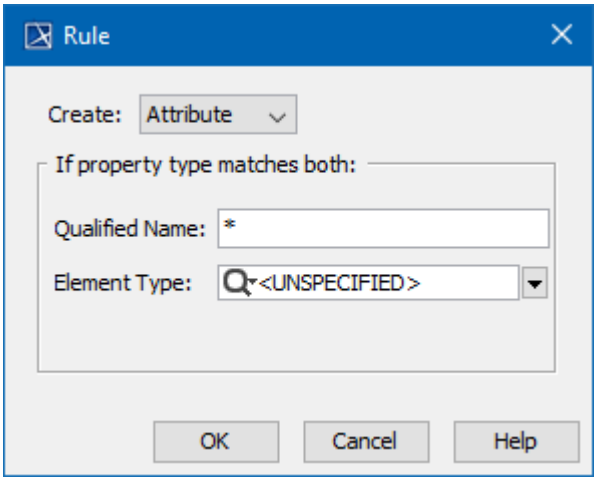
The **Class Field Creation Rules** dialog box lists the described rules. If a type qualified name matches any rule, a specified type is created. Rules described in this dialog are executed in order from top to bottom. If one rule matches, no others are executed. The dialog allows you to add a new rule,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

- If no rules are matched, an attribute or association is created according the radio button choice at the bottom of the dialog.
- Click **Add** to add an expression.
- Click **Edit** to edit a selected rule.

To define rules

You can define rules in the **Rules** dialog box. Click the **Add** or **Edit** button in the **Class Field Creation** dialog box to open this dialog box.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



The **Rules** dialog box description:

- In the **Create** drop-down list, choose **Attribute** or **Association** on reverse if a property matches a defined criterion.
- The **Qualified Name** text box supports simple search patterns, as well as '*' and '?' symbols.
- The **Qualified Name** pattern can contain a qualified name of the target element including a separator of a UML separator style (::). The qualified name is counted to the first package with the «ModelLibrary» stereotype.
- The **Element Type** drop-down list contains a subset of UML Type names: Any, Class, Interface, DataType, Primitive, Enumeration, Stereotype values are displayed.
- If the **<UNSPECIFIED>** value is selected in the **Element Type** drop-down list, the element type is matched by the qualified name expression.

Sample of the association creation on reverse
These two samples show how to create associations among reversed classes, while avoiding creating associations to the used libraries (java).
If you place your classes inside the package "*com::myProduct*", you must follow these Creation rules:

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

If you place your classes inside two packages "*myClient*" and "*myServer*", the Creation rules should be:

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

Related Pages:

- Reverse Options(see page 2406)
  - Rules of association or attribute creation on reverse(see page 2412)
- Code Engineering(see page 2398)
  -

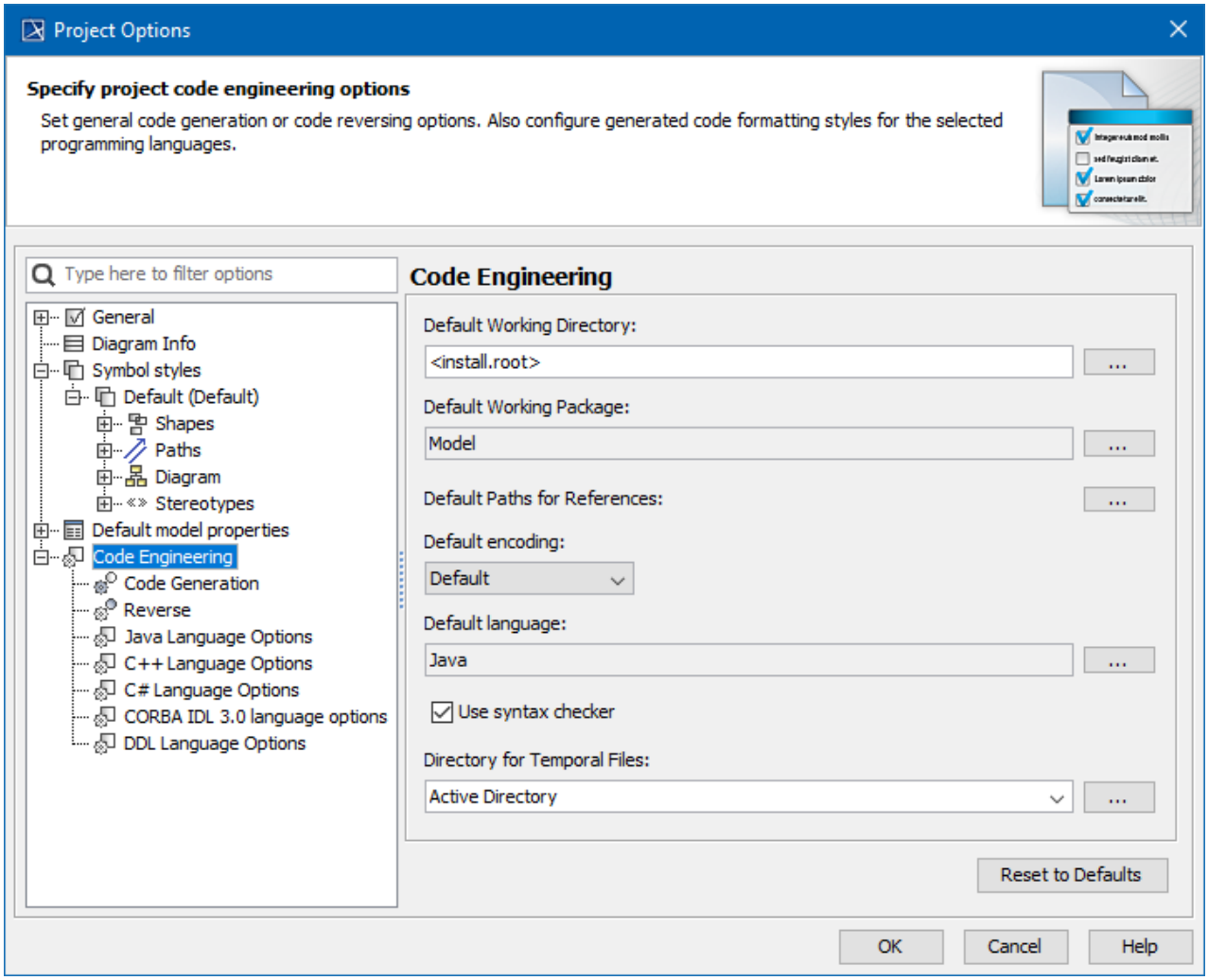
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Code Engineering Sets(see page 2399)
- Generating Code(see page 2403)
- Reverse Options(see page 2406)
- Global options for Code Engineering(see page 2431)
- Files of Properties(see page 2442)
- Java Code Engineering(see page 2443)
- C++ Code Engineering(see page 2470)
- C# Code Engineering(see page 2622)
- CORBA IDL Mapping To UML(see page 2680)
- WSDL(see page 2683)

Global options for Code Engineering

Code Engineering Options for all Sets in your Project

Choose **Project** from the **Options** menu. The Project Options dialog box appears. You can set the Code Engineering options in the **Project Options** dialog(see page 214). The **Project Options** dialog box has two main collections of customizable options, represented by the hierarchy tree on the left side of the dialog box. The figure below illustrates **Code Engineering** related options.



Code Engineering options in the Project Options dialog.

In the option list, select **Code engineering** and specify the following items:

Item name	Description
Default Working Directory	Enter the name of your working directory.
Default Working Package	Define any package for the reverse output or code generation. Your model will be reversed or the code will be generated from this specified package.
Default Paths for References	Add specific profiles, used projects, or libraries to define where to search for paths for references during reverse/code generation.
Default Encoding	Click to open a list of available encodings.
Default language	When selected, the syntax checker of the selected language runs while Code Engineering is executed.

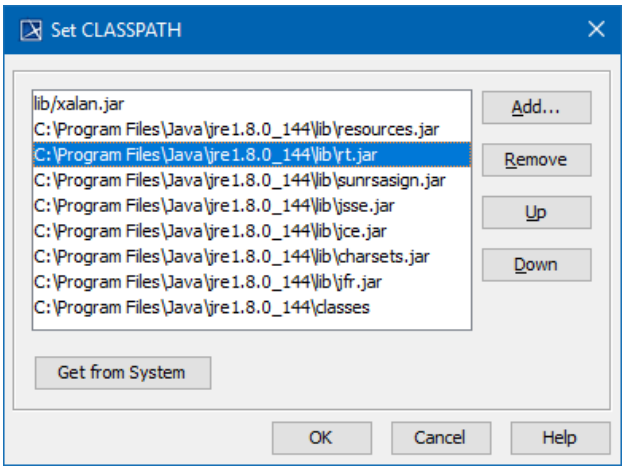
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Item name	Description
Use Syntax Checker	When selected, the syntax checker runs while Code Engineering is executed.
Directory for Temporal Files	You can specify Active Directory, System, or you can select another directory.

The following table describes other **Code Engineering** options:

Option	Element 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 with the same functionality as the Code generations options dialog box. See Generating Code(see page 2403).
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 Options(see page 2406) area contains boxes that have the same functionality as in the Reverse options dialog box.
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 being generated.
	Generate spaces	Generate 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 referenced in the generated class.
	Class count to generate import on demand	Specify the number of classes imported from a single package until all statements importing a single class are substituted with a statement importing an entire package.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Option	Element Name	Description
	Documentation Processor	After selecting the Java Doc processor, click ... to open the Documentation Properties dialog box(see page 2439).
	Style	Two styles are available for documentation.
	Use CLASSPATH	The ... button is activated. Search a classpath for importing sentence generation in the Set classpath dialog box.  Click **Get from System** to get the CLASSPATH variable defined by the operating system or click **Add** and select the classpath directory.
	Java Source	Available choices are 1.4, 5.0, 7.0, or 8.0.
	Header	Add a specific header to all your code files. Click ... and enter the header text in the **Header** dialog. You can also define $DATE, $AUTHOR, and $TIME in the header.

Option	Element Name	Description
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 the class in the model has no documentation.
	Generate methods body into class	Select check box to generate methods body into class.
	Generate field initialization into class	Generates the value initialization for the attribute.
	Automatically remove unnecessary usage relationship	Tracks the use of classes and removes the unnecessary include directives.
	Documentation Style	Four styles are available for documentation.
	Use include path	Click ... and then specify the path for the #include directives in the **Set Include Path** dialog box.
	Use explicit macros	Select the check box. The ... button is activated. Click it and use a set of predefined macros in the **C++ Macros** dialog box.
	Header	Add the specific header to all your code files. Click ... and enter the header text in the **Header** screen. You can 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 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Option	Element Name	Description
	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 the class in the model has no documentation.
	Generate imports	Generation of "import" statements for classes referenced in the generated class.
	Generate preprocessor directives	Generates pre-processors directives.
	Generate end comments for composite elements	Generates end comments with composite element names.
	Generate operation parameters on the new line	Generates operation parameters on the new line.
	Documentation Style	Three styles are available for documentation.
	Header "..."	Add the specific header to all your code files. Click ... and enter the header text in the **Header** dialog. You can also define $DATE, $AUTHOR, and $TIME in the header. Since MagicDraw version 17.0, the variable $DOCUMENTATION can be used to export the documentation of the file component (from the File View package) as a header of the IDL file. **Functionality Availability** The variable $DOCUMENTATION is available with MagicDraw 17.0 Service Pack 4 and later versions.

Option	Element Name	Description
	Set Include Path	Specify the path for the #include directives. Click ... to open the **Select Folder** dialog box.
DDL Language Options	**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 documentation	Comment brackets are placed in your code, unless the 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** dialog. You may also define $DATE, $AUTHOR, and $TIME in the header.
C# Language Options	**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 the 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.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Option	Element Name	Description

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

	Documentation: • **Processor** • **Style**	• Use C# XMI processor then generates c# xmi documentation for commenting the code. • Select one of the three available documentation styles.
	Header	Adds the specific header to all your code files. Click ... and type the header text in the **Header** dialog box. You can 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.

Related Pages:

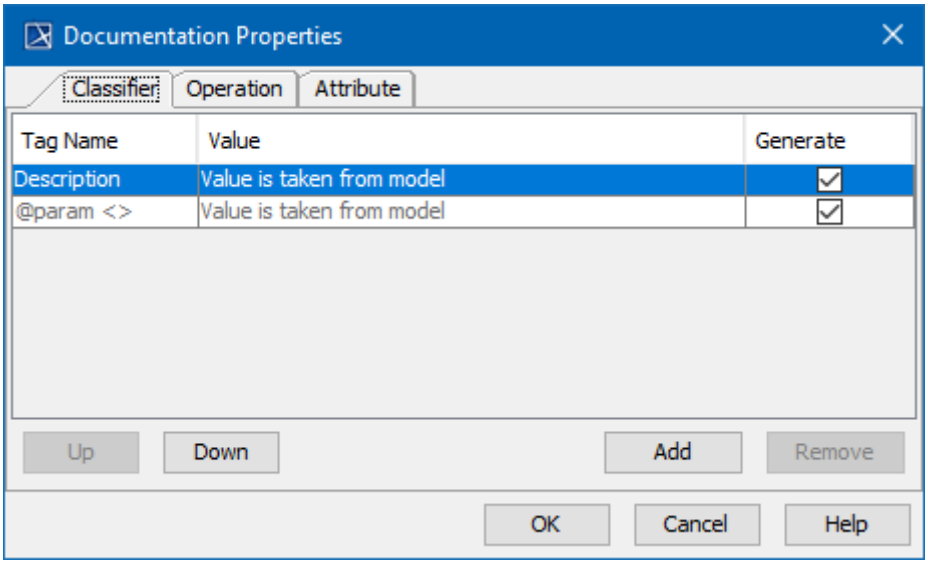
- Java Documentation Properties dialog box(see page 2439)
- Round Trip(see page 2440)
- Type Mapping Table(see page 2441)
- Code Engineering(see page 2398)
  - Code Engineering Sets(see page 2399)
  - Generating Code(see page 2403)
  - Reverse Options(see page 2406)
  - Global options for Code Engineering(see page 2415)
  - Files of Properties(see page 2442)
  - Java Code Engineering(see page 2443)
  - C++ Code Engineering(see page 2470)
  - C# Code Engineering(see page 2622)
  - CORBA IDL Mapping To UML(see page 2680)
  - WSDL(see page 2683)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Java Documentation Properties dialog box

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

1. Open the **Project Options** dialog box(see page 214).
2. In the Options list, expand **Code Engineering** and click **Java Language Options**.
3. In the **Java Language Options** area, select **Java Doc** as the **Processor** for **Documentation**.
4. Click ... to open the **Documentation Properties** dialog.



Documentation Properties dialog box

Box Name	Function
Tag Name	Type the tag name.
Value	Type the value of the tag.
Generate	The selected tag will be placed in the generated code as a comment before a classifier (class or interface), operation or attribute.
Up	Moves the selected item up the list.
Down	Moves the selected item down the list.
Add	Adds a new item in the list.
Remove	Removes the selected from the list.
OK	Saves changes and closes the dialog box.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Box Name	Function
Cancel	Closes the dialog box without saving changes.
Help	Displays MagicDraw Help.

Related Pages:

- Global options for Code Engineering(see page 2415)
  - Java Documentation Properties dialog box(see page 2423)
  - Round Trip(see page 2440)
  - Type Mapping Table(see page 2441)
- Code Engineering(see page 2398)
  - Code Engineering Sets(see page 2399)
  - Generating Code(see page 2403)
  - Reverse Options(see page 2406)
  - Global options for Code Engineering(see page 2415)
  - Files of Properties(see page 2442)
  - Java Code Engineering(see page 2443)
  - C++ Code Engineering(see page 2470)
  - C# Code Engineering(see page 2622)
  - CORBA IDL Mapping To UML(see page 2680)
  - WSDL(see page 2683)

Round Trip

The Round Trip feature keeps your code and model synchronized. Because Round Trip traces all model and code changes, you can freely change entity specifications without discarding code changes made outside the tool.

For example, Round Trip prevents a job from being damaged by code editions or changes when the following steps are performed:

1. A Class *Base* is created within the tool.
2. The Operation *getInstance* is added to the Class.
3. Code is generated.
4. With an external tool, a programmer adds some code to that operation.
5. With a modeling tool, a name of the operation is changed to *Instance*.
6. 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, you are not allowed to refresh the UML model. The **Refresh** co mmand from the set shortcut menu is unavailable.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- 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 reverse to the UML models. If the **Code Generation Options** dialog box appears when you attempt to generate code, you can 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.

Related Pages:

- Global options for Code Engineering(see page 2415)
  - Java Documentation Properties dialog box(see page 2423)
  - Round Trip(see page 2424)
  - Type Mapping Table(see page 2441)
- Code Engineering(see page 2398)
  - Code Engineering Sets(see page 2399)
  - Generating Code(see page 2403)
  - Reverse Options(see page 2406)
  - Global options for Code Engineering(see page 2415)
  - Files of Properties(see page 2442)
  - Java Code Engineering(see page 2443)
  - C++ Code Engineering(see page 2470)
  - C# Code Engineering(see page 2622)
  - CORBA IDL Mapping To UML(see page 2680)
  - WSDL(see page 2683)

Type Mapping Table

Languages supported in our modeling tools have built-in types. One language's type could either have no matches or multiple matches in another language. Additionally, some names are interpreted differently in different languages. Therefore, while performing code generation, problems may occur when switching between different languages. To avoid this, type-mapping tables are used to manage the mapping between distinct languages. The map describes the rules how one language's built-in types are converted to those of another language.

Related Pages:

- Global options for Code Engineering(see page 2415)
  - Java Documentation Properties dialog box(see page 2423)
  - Round Trip(see page 2424)
  - Type Mapping Table(see page 2425)
- Code Engineering(see page 2398)
  - Code Engineering Sets(see page 2399)
  - Generating Code(see page 2403)
  - Reverse Options(see page 2406)
  - Global options for Code Engineering(see page 2415)
  - Files of Properties(see page 2442)
  - Java Code Engineering(see page 2443)
  -

C++ Code Engineering(see page 2470)
- C# Code Engineering(see page 2622)
- CORBA IDL Mapping To UML(see page 2680)
- WSDL(see page 2683)

Files of Properties

Code can be generated out of prepared UML models. Mapping should be implemented between the identifiers used in the UML model and the language to which the model is being generated. 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.

A separate *prop* file is created for every language supported in modeling tools developed by No Magic, Inc. Files are located in the *<modeling tool installation directory>\data* folder. The file name pattern is *lang.prop*, where lang stands for the name of the programming language.

The following table lists supported languages:

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 can edit existing entities, add new ones, and change the default values.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

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

Related Pages:

- Code Engineering(see page 2398)
  - Code Engineering Sets(see page 2399)
  - Generating Code(see page 2403)
  - Reverse Options(see page 2406)
  - Global options for Code Engineering(see page 2415)
  - Files of Properties(see page 2426)
  - Java Code Engineering(see page 2443)
  - C++ Code Engineering(see page 2470)
  - C# Code Engineering(see page 2622)
  - CORBA IDL Mapping To UML(see page 2680)
  - WSDL(see page 2683)

Java Code Engineering

This chapter describes how our modeling tool maps Java language elements to UML, helps choose which profiles to use, and describes Java code engineering properties.

This chapter features the following:

1. Java Mapping to UML(see page 2427) describes general rules our modeling tool uses to map each Java element to UML and which profile is used. You will find an example and corresponding model in MagicDraw with marked properties used in Java to describe mapping rules.
2. Java CE Options(see page 2465) introduces specific Java options.
3. Method Implementation Reverse(see page 2470) describes a functionality that switches Java from a Sequence Diagram to our modeling tool.

References

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

Java support in modeling tools developed by No Magic, Inc.

You can perform the following actions:

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

Related pages:

- Java Mapping to UML(see page 2428)
- Java CE Options(see page 2465)
- Method Implementation Reverse(see page 2470)
- Code Engineering(see page 2398)
  - Code Engineering Sets(see page 2399)
  - Generating Code(see page 2403)
  - Reverse Options(see page 2406)
  - Global options for Code Engineering(see page 2415)
  - Files of Properties(see page 2426)
  - Java Code Engineering(see page 2427)
  - C++ Code Engineering(see page 2470)
  - C# Code Engineering(see page 2622)
  - CORBA IDL Mapping To UML(see page 2680)
  - WSDL(see page 2683)

Java Mapping to UML

Java Profile

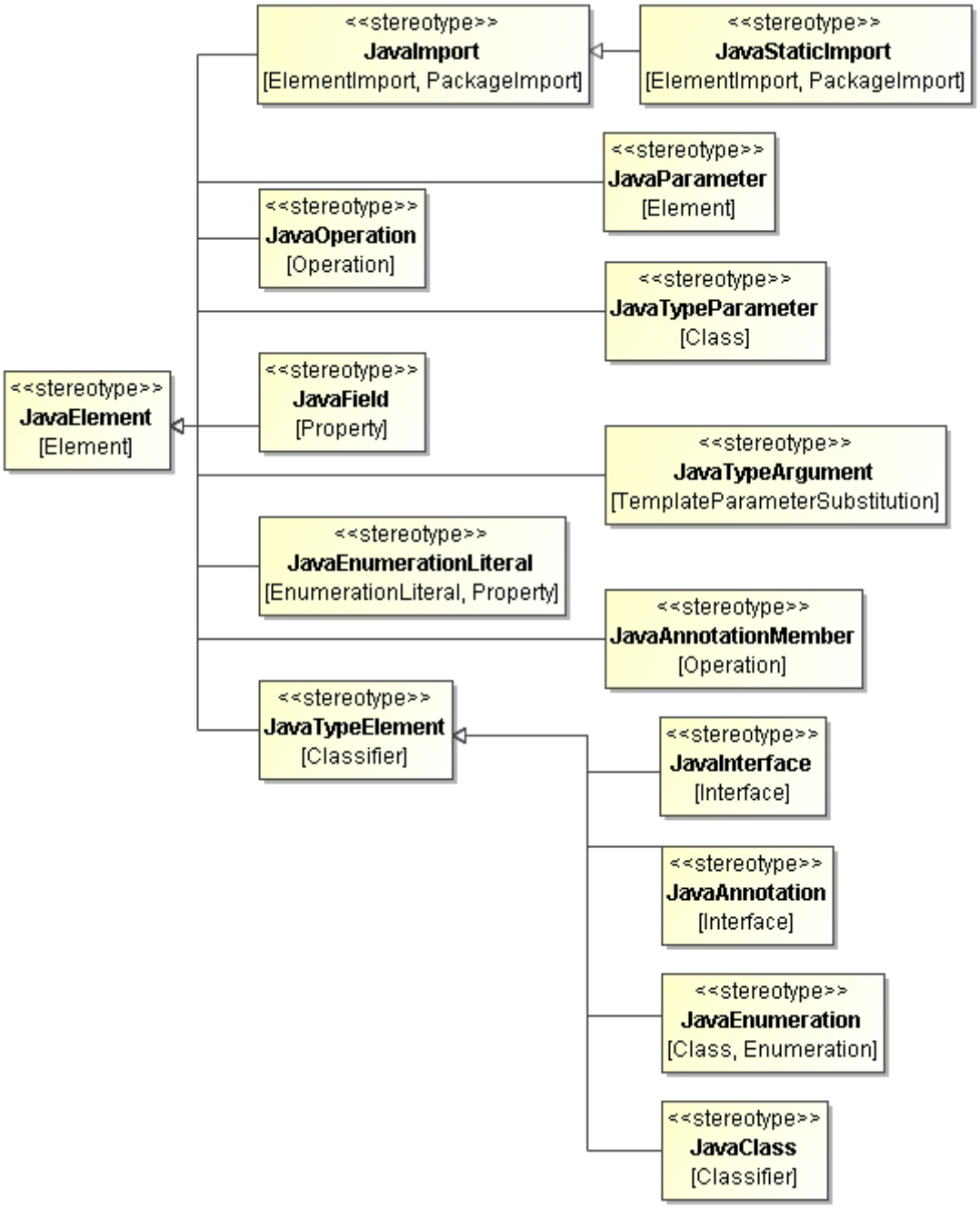
Java Profile is used to map the Java language to UML.

UML specification does not provide elements to fully cover JLS; thus, a modeling tool uses UML stereotypes to mark a UML class or interface to be a specific Java element.

Java stereotypes are provided in the *Java_Profile.xml file* in the profiles directory. Some stereotypes have tagged values used for mapping special Java language elements or keywords that are not mapped to 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 stereotype is used to represent an appropriate Java element:

- «JavaImport» represents Java Imports(see page 2464),
- «JavaOperation» represents Java Operation(see page 2428), and
- «JavaClass» represents Java Class(see page 2428) type.



Stereotypes.

Related Pages

- Java Referenced Types(see page 2429)
- Mapping to UML Rules(see page 2429)
- Code Engineering(see page 2398)
  - Code Engineering Sets(see page 2399)
  - Generating Code(see page 2403)
  -

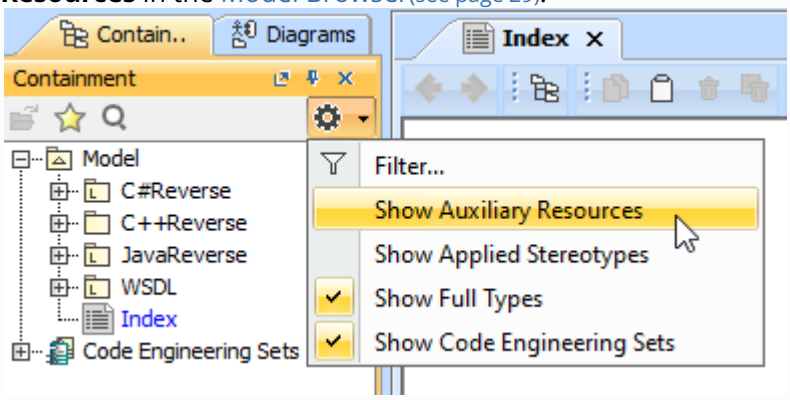
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Reverse Options(see page 2406)
- Global options for Code Engineering(see page 2415)
- Files of Properties(see page 2426)
- Java Code Engineering(see page 2427)
- C++ Code Engineering(see page 2470)
- C# Code Engineering(see page 2622)
- CORBA IDL Mapping To UML(see page 2680)
- WSDL(see page 2683)

Java Referenced Types

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

Show Auxiliary Resources
The UML Standard Profile by default is hidden. If you want to see it, select **Show Auxiliary Resources** in the Model Browser(see page 29).



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

A modeling tool resolves referenced classes from the specified class path (by default, the class path is the boot class path taken from the JVM on which the modeling tool is started). It creates an appropriate package structure as described below in **Sorting reversed classes according to the classpath**.

If a referenced class is not found in the model or the class path, the reference is created in the *Default* package.

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

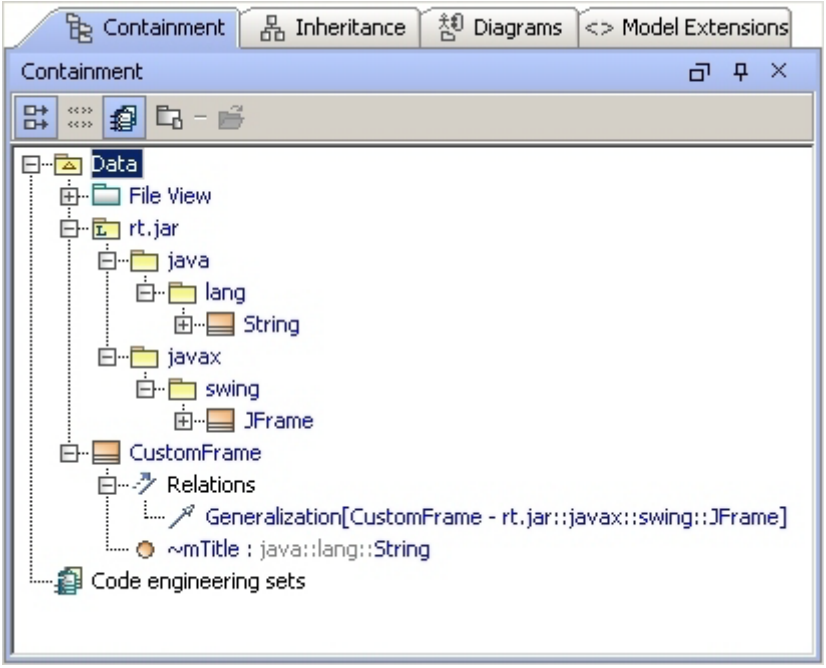
Sorting reversed classes according to the classpath

References are sorted to packages according to the classpath entry.

The figure below shows how the reverse result of the following java code is represented:

Example
Java code

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



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

Related Pages:

- Java Mapping to UML(see page 2428)
  - Java Referenced Types(see page 2430)
  - Mapping to UML Rules(see page 2431)

Mapping to UML Rules

This section describes the general rules and profile our modeling tool uses to map each Java element is mapped to UML. Please find examples and corresponding models in MagicDraw with marked properties used in Java to describe mapping rules.

Related Pages:

- Java Package(see page 2431)
- Java Class(see page 2431)
- Java Field(see page 2431)
- Java Operation(see page 2431)
-

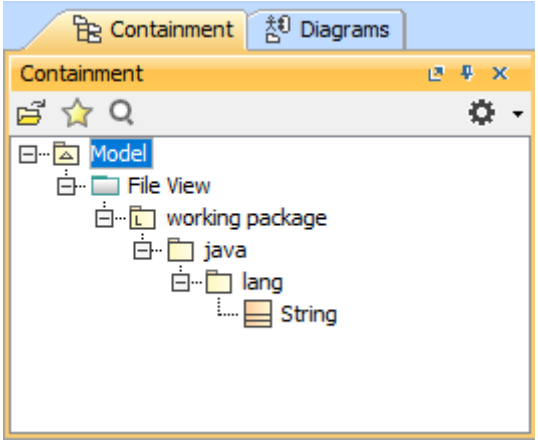
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Java Interface(see page 2431)
- Java Enumeration(see page 2432)
- Java Enumeration Literal(see page 2432)
- Java Annotation Type(see page 2432)
- Java Annotation Member(see page 2432)
- Java Annotations Usage(see page 2432)
- Java Type Variables(see page 2461)
- Java Parameterized Type(see page 2463)
- Java Imports(see page 2464)

- Code Engineering(see page 2398)
  - Code Engineering Sets(see page 2399)
  - Generating Code(see page 2403)
  - Reverse Options(see page 2406)
  - Global options for Code Engineering(see page 2415)
  - Files of Properties(see page 2426)
  - Java Code Engineering(see page 2427)
  - C++ Code Engineering(see page 2470)
  - C# Code Engineering(see page 2622)
  - CORBA IDL Mapping To UML(see page 2680)
  - WSDL(see page 2683)

Java Package

A Java package is mapped to the general UML Package(see page 800). It does not have any specific stereotypes (see page 856)and properties. However, if the UML package represents the Java package, it must not have a «modelLibrary» stereotype from a standard UML profile. The «modelLibrary» stereotype is used to show the root package, where the Java package tree ends and no parent packages, including «modelLibrary», are part of the Java package structure. The figure below shows *java.lang.String* added into the *working package*. Packages *java* and *lang* represent general Java packages, but *working package* with *L* is stereotyped with «modelLibrary» and is not part of the Java package.



Package Structure

Related Pages:

-

Mapping to UML Rules(see page 2431)
- Java Package(see page 2432)
- Java Class(see page 2433)
- Java Field(see page 2433)
- Java Operation(see page 2433)
- Java Interface(see page 2433)
- Java Enumeration(see page 2433)
- Java Enumeration Literal(see page 2433)
- Java Annotation Type(see page 2433)
- Java Annotation Member(see page 2433)
- Java Annotations Usage(see page 2433)
- Java Type Variables(see page 2433)
- Java Parameterized Type(see page 2463)
- Java Imports(see page 2464)

Java Class

A Java class is mapped directly to the UML Class (see page 666)with the stereotype «JavaClass». This stereotype is optional and if the UML Class has no stereotype, Java CE treats it as a Java class. If no appropriate property is found in UML, class modifiers are mapped into UML Class properties or to the Java language properties for Class.

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 Documentation	UML Class Documentation
Class extends Clause	Mapped to the UML Generalization relationship, where the supplier is an extended class and the client is a mapped class
Class Implementation Clause	Mapped to the UML Interface Realization relationship, where the supplier is an extended class and the client is a mapped class.

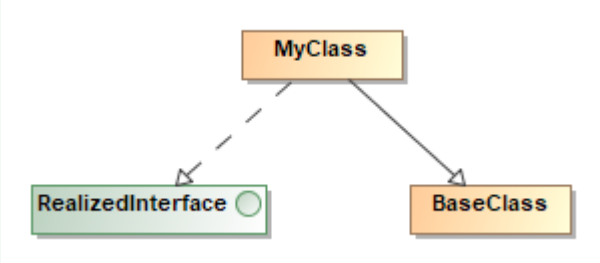
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Java Element	MagicDraw-UML element
Visibility Modifier	UML Class “Visibility” property
Abstract Modifier	UML Class “Is Abstract” property
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
{
}
```

UML model



Related Pages:

- Mapping to UML Rules(see page 2431)
  - Java Package(see page 2432)
  - Java Class(see page 2433)
  - Java Field(see page 2434)
  - Java Operation(see page 2434)
  - Java Interface(see page 2434)
  - Java Enumeration(see page 2434)
  - Java Enumeration Literal(see page 2434)
  - Java Annotation Type(see page 2434)
  - Java Annotation Member(see page 2434)
  - Java Annotations Usage(see page 2434)
  - Java Type Variables(see page 2434)
  - Java Parameterized Type(see page 2463)
  - Java Imports(see page 2464)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Java Field

A Java field is mapped directly to the UML Attribute(see page 672) with the stereotype «JavaField». This stereotype is optional and if a UML Class has no stereotype, Java CE treats it as a Java field. If no appropriate property is found in UML, field modifiers are mapped into the UML Property properties or to the Java language properties.

Note that Java field type modifiers are mapped to the 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	Mapped to the UML Type property. It is a reference to the UML Classifier; its package structure and name represent the referenced Java class.
Field Type Modifiers	Mapped to the specified property *Type Modifier*
Visibility Modifier	UML Property *Visibility* property
Final Modifier	UML Property *Is Read Only* property
Static Modifier	UML Property *Is Static* property
Transient Modifier	Java Language property *Transient modifier*

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

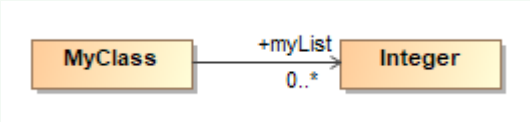
Java Element	MagicDraw-UML Element
Volatile Modifier	Java Language property *Volatile modifier*

Examples

Java source code

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

UML model



Related Pages:

- Mapping to UML Rules(see page 2431)
  - Java Package(see page 2432)
  - Java Class(see page 2433)
  - Java Field(see page 2435)
  - Java Operation(see page 2452)
  - Java Interface(see page 2455)
  - Java Enumeration(see page 2456)
  - Java Enumeration Literal(see page 2457)
  - Java Annotation Type(see page 2458)
  - Java Annotation Member(see page 2459)
  - Java Annotations Usage(see page 2460)
  - Java Type Variables(see page 2461)
  - Java Parameterized Type(see page 2463)
  - Java Imports(see page 2464)

Java Operation

The Java Operation mapped directly to the UML Operation is done with the stereotype «JavaOperation». This stereotype is optional, so if the UML class has no stereotype Java CE treats it as a Java Operation. If no appropriate property is found in UMLOperation, modifiers are mapped into UML Property properties or to the Java language properties. The Java Operation return type is mapped to the UML Type property of the UML Parameter with the “Return“ direction kind. Java Operation parameters are mapped to the UML Parameters. The direction kind is set “In“ for “final” parameters. Note that Java

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

parameter type modifiers are mapped to the UML specific property “Type Modifier,“ but not to the UML Multiplicity.

Operation mapping table

Java Element	UML Element
Operation declaration	UML Operation, owned by UML Class, with stereotype «JavaOperation» (optional)
Operation Name	UML Operation Name
Operation documentation	UML Operation Documentation
Parameters list	UML Operation “Parameters” list
Return type	Mapped to the UML Parameter type with “return“ direction kind, i.e., it resides in the UML Operation parameters list. It is a reference to the UML Classifier; its package structure and name represent the referenced Java class
Return type modifiers	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 a list of references to the UML Classes; its package structure and name represent the referenced Java exception class.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Java Element	UML Element
Native modifier	Java Language property “Native modifier”
Strictfp modifier	Java Language property “Strictfp modifier”

Parameter mapping table

Java Element	UML Element
Parameter declaration	UML Parameter, owned by UML Operation, with stereotype «JavaParameter» (optional)
Parameter name	UML Parameter Name
Parameter documentation	When used in JavaDoc preprocessing, it is mapped to the UML Parameter Documentation. If not, it is part of the UML Operation Documentation.
Return type	Mapped to the UML Type property. It is a reference to the UML Classifier; its package structure and name represent the referenced Java exception class.
Return type modifiers	Mapped to the MagicDraw specified property “Type Modifier“
Final modifier	Direction kind “in“ of the UML Parameter

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Examples

Java Source Code

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

UML Model

Class with an operation.

Related Pages:

- Mapping to UML Rules(see page 2431)
  - Java Package(see page 2432)
  - Java Class(see page 2433)
  - Java Field(see page 2435)
  - Java Operation(see page 2436)
  - Java Interface(see page 2455)
  - Java Enumeration(see page 2456)
  - Java Enumeration Literal(see page 2457)
  - Java Annotation Type(see page 2458)
  - Java Annotation Member(see page 2459)
  - Java Annotations Usage(see page 2460)
  - Java Type Variables(see page 2461)
  - Java Parameterized Type(see page 2463)
  - Java Imports(see page 2464)

Java Interface

The Java interface is mapped directly to the UML Interface with the stereotype «JavaInterface». This stereotype is optional, and if the UML Interface has no stereotype, Java CE treats it as a Java interface. If no appropriate property is found in the UMLinterface, modifiers are mapped into UML Interface properties or to the Java language properties for the interface. All mapping rules used in Java class(see page 2433) mapping apply to the Java interface.

Related Pages:

- Mapping to UML Rules(see page 2431)
  - Java Package(see page 2432)
  - Java Class(see page 2433)
  - Java Field(see page 2435)
  -

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Java Operation(see page 2436)
- Java Interface(see page 2439)
- Java Enumeration(see page 2456)
- Java Enumeration Literal(see page 2457)
- Java Annotation Type(see page 2458)
- Java Annotation Member(see page 2459)
- Java Annotations Usage(see page 2460)
- Java Type Variables(see page 2461)
- Java Parameterized Type(see page 2463)
- Java Imports(see page 2464)

Java Enumeration

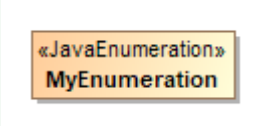
Java enumeration is mapped directly to the UML Class with the stereotype «JavaEnumeration». If no appropriate property is found in UML, enumeration modifiers are mapped into UML Class properties or to the Java language properties for the interface. Java enumeration literals are mapped to the UML Property with the 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(see page 2433) mapping apply to the Java enumeration.

Example

Java source code

```
enum MyEnumeration
{
}
```

UML model



Example of Enumeration

Related Pages:

- Mapping to UML Rules(see page 2431)
  - Java Package(see page 2432)
  - Java Class(see page 2433)
  - Java Field(see page 2435)
  - Java Operation(see page 2436)
  - Java Interface(see page 2439)
  - Java Enumeration(see page 2440)
  - Java Enumeration Literal(see page 2457)
  - Java Annotation Type(see page 2458)
  - Java Annotation Member(see page 2459)
  - Java Annotations Usage(see page 2460)
  - Java Type Variables(see page 2461)
  - Java Parameterized Type(see page 2463)
  -

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Java Imports(see page 2464)

Java Enumeration Literal

Java Enumeration Literal is mapped directly to the UML Property with the stereotype «JavaEnumerationLiteral». It does not require specific modifiers.

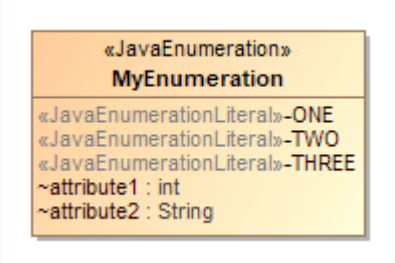
Java Element	UML Element
Literal declaration	UML Parameter with stereotype «JavaEnumerationLiteral», owned by UML Class with stereotype «JavaEnumeration»
Literal name	UML Property Name
Literal documentation	When using JavaDoc preprocessing, it is mapped to the UML Parameter Documentation; otherwise, it is part of the UML Operation Documentation.

Example

Java source code

```
enum MyEnumeration
{
ONE, TWO, THREE;
int attribute1;
String attribute2;
}
```

UML model



UML Class representing Java enumeration with enumeration literals

Related Pages:

- Mapping to UML Rules(see page 2431)
  - Java Package(see page 2432)
  - Java Class(see page 2433)
  -

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Java Field(see page 2435)
- Java Operation(see page 2436)
- Java Interface(see page 2439)
- Java Enumeration(see page 2440)
- Java Enumeration Literal(see page 2441)
- Java Annotation Type(see page 2458)
- Java Annotation Member(see page 2459)
- Java Annotations Usage(see page 2460)
- Java Type Variables(see page 2461)
- Java Parameterized Type(see page 2463)
- Java Imports(see page 2464)

Java Annotation Type

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

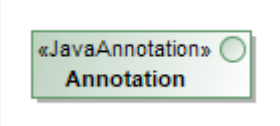
Annotation mapping table

Java Element	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
{
}
```

UML model



UML Interface representing Java annotation type

Related Pages:

-

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Mapping to UML Rules(see page 2431)
- Java Package(see page 2432)
- Java Class(see page 2433)
- Java Field(see page 2435)
- Java Operation(see page 2436)
- Java Interface(see page 2439)
- Java Enumeration(see page 2440)
- Java Enumeration Literal(see page 2441)
- Java Annotation Type(see page 2442)
- Java Annotation Member(see page 2443)
- Java Annotations Usage(see page 2443)
- Java Type Variables(see page 2443)
- Java Parameterized Type(see page 2443)
- Java Imports(see page 2443)

Java Annotation Member

A Java annotation member is mapped directly to the UML Operation owned by the interface stereotyped as «JavaAnnotation». This operation, by itself, can have the stereotype «JavaAnnotationMember». However, it is optional, unless specify a default value for it.

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

Annotation member mapping table:

Java Element	UML element
Annotation member declaration	UML Operation with the stereotype «JavaAnnotationMember» (Optional), owned by a UML Interface with the stereotype «JavaAnnotation».
Annotation member name	UML Operation Name
Annotation member documentation	UML Operation Documentation
Annotation member type	Mapped to the UML Parameter type with the “return“ direction kind (resides in UML Operation parameters list)
Annotation member type modifier	Mapped to the MagicDraw specified property “Type Modifier“

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Java Element	UML element
Default value	{JavaAnnotationMemberDefaultValue} tagged value of a «JavaAnnotationMember» stereotype. The stereotype is set on UML Operation

Example
Java source code

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

UML model

Related Pages:

- Mapping to UML Rules(see page 2431)
  - Java Package(see page 2432)
  - Java Class(see page 2433)
  - Java Field(see page 2435)
  - Java Operation(see page 2436)
  - Java Interface(see page 2439)
  - Java Enumeration(see page 2440)
  - Java Enumeration Literal(see page 2441)
  - Java Annotation Type(see page 2442)
  - Java Annotation Member(see page 2443)
  - Java Annotations Usage(see page 2460)
  - Java Type Variables(see page 2461)
  - Java Parameterized Type(see page 2463)
  - Java Imports(see page 2464)

Java Annotations Usage

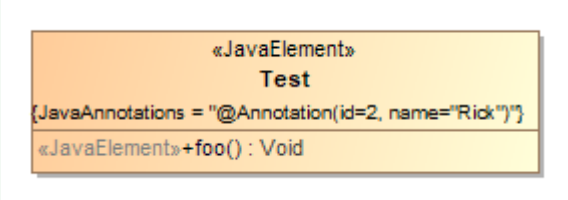
You can annotate Java elements. This annotation is mapped on top of the {JavaAnnotations} tagged value of the stereotype «JavaElement». «JavaElement» is a base stereotype for all stereotypes used in Java mapping, and can be used directly or with any other stereotype derived from it. Annotation is mapped as a simple string value.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example
Java source code

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

UML Model



Java annotation usage

Related Pages:

- Mapping to UML Rules(see page 2431)
  - Java Package(see page 2432)
  - Java Class(see page 2433)
  - Java Field(see page 2435)
  - Java Operation(see page 2436)
  - Java Interface(see page 2439)
  - Java Enumeration(see page 2440)
  - Java Enumeration Literal(see page 2441)
  - Java Annotation Type(see page 2442)
  - Java Annotation Member(see page 2443)
  - Java Annotations Usage(see page 2444)
  - Java Type Variables(see page 2461)
  - Java Parameterized Type(see page 2463)
  - Java Imports(see page 2464)

Java Type Variables

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

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Type variable mapping table

Java Element	UML element
Type variable declaration	Mapped to the UML Template Parameter. This UML Template Parameter is of the Class type from the UML Metamodel. The UML Template Parameter has the property “Parametered Element“ of the UML Template Parameter. The metamodel type “Class“ is taken from the “UML Standard profile/UML 2 Metamodel”
Type variable name	Mapped to the UML Class name. This UML Class is “Parametered Element“
Type bounds	If a bound type is a Java class, it is mapped to the General class of the “Parametered Element“. If a 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

UML Class with template parameter E, representing Java type variable

In this example, the Template Parameter “E” is a subtype of the Cloneable class (e.g. “java.util.Cloneable“). “E” is used to type the “Test” public class attribute.

Template parameters in UML Class specification dialog

Related Pages:

- Mapping to UML Rules(see page 2431)
  - Java Package(see page 2432)
  - Java Class(see page 2433)
  - Java Field(see page 2435)
  - Java Operation(see page 2436)
  - Java Interface(see page 2439)
  - Java Enumeration(see page 2440)
  - Java Enumeration Literal(see page 2441)
  - Java Annotation Type(see page 2442)
  - Java Annotation Member(see page 2443)
  - Java Annotations Usage(see page 2444)
  - Java Type Variables(see page 2445)
  - Java Parameterized Type(see page 2446)
  - Java Imports(see page 2446)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Java Parameterized Type

Parameterized types are mapped to the general UML Classifier connected with a UML Template Binding to the UML Classifier. The supplier of this binding link is the UML Classifier with UML Template Parameters and represents the Java generic type with type parameters. The client of the UML Template Binding is the UML Classifier of the same UML type as the supplier. 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
Parametrized type	This is called bounded elements in UML, and is connected to the UML Classifier by the UML Template Binding as a client. The client must be of the same type as the supplier. The supplier must have at least one UML Template Parameter.
Parameter for type	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. The type of the type parameter is any reference to the UML Classifier from the model set as “Actual“ value of the UML Template Parameter Substitution.
Type modifiers of the parameter	Mapped to the MagicDraw property “Type modifiers” of the UML Template Parameter Substitution
Wildcard	A UML Class with the name “?“ from the Java Profile is used as an “Actual” value in the UML Template Parameter Substitution.
Wildcard with bounds	Java bounding type is mapped to the UML Classifier and used as the “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 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example
When you have a type with type variables represented in the model, you can create a parameterized type for Java. For this you must create an empty UML Classifier of the same type as the template classifier (type with type variables). Then create a UML Template Binding and create a UML Template Parameter Substitution for the UML Template Parameters.
To create a “java.util.List<String>” type, you must create a UML Interface first, with a 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 it with the List interface with UML Template Binding.
Note that to draw a UML Template Binding, the client element must have at least one UML Template Parameter created

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

You must open the UML Template Binding specification dialog and create a UML Template Parameter Substitution for the appropriate UML Template Parameter.

For a created substitution element, you need to select actual value. In the current example, it is a UML class representing “java.lang.String”.

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

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

Related Pages:

- Mapping to UML Rules(see page 2431)
  - Java Package(see page 2432)
  - Java Class(see page 2433)
  - Java Field(see page 2435)
  - Java Operation(see page 2436)
  - Java Interface(see page 2439)
  - Java Enumeration(see page 2440)
  - Java Enumeration Literal(see page 2441)
  - Java Annotation Type(see page 2442)
  - Java Annotation Member(see page 2443)
  - Java Annotations Usage(see page 2444)
  - Java Type Variables(see page 2445)
  - Java Parameterized Type(see page 2447)
  - Java Imports(see page 2448)

Java Imports

Java Element	MagicDraw-UML Element
Type Import	Mapped to the UML Element Import with the optional stereotype «JavaImport». The supplier is an imported UML Classifier, representing the imported Java type, and the client is a UML Classifier requiring an imported element.
Package Import	Mapped to the UML Package Import with the optional stereotype «JavaImport». The supplier is an imported UML Package, representing the imported Java package, and the client is a UML Classifier requiring imported elements.
Static import for all static members	Mapped to the UML Element Import with the stereotype «JavaStaticImport». The {JavaImportAll} tag of «JavaStat- icImport» must have a “true“ value. The supplier is an imported UML Classifier owner of imported static members, and the client is a UML Classifier requiring imported elements.
Static import for single static members	Mapped to the UML Element Import with the stereotype «JavaStaticImport». The {JavaImportAll} tag of «JavaStaticImport» must have a “false“ value and its tag {JavaImportedMember} must have reference to the imported member. The supplier is a UML Classifier owner of an imported static member, and the client is a UML Classifier requiring imported elements. If several static members are imported, {JavaImportedMember} can list all of them.

Related Pages:

- Mapping to UML Rules(see page 2431)
  - Java Package(see page 2432)
  - Java Class(see page 2433)
  - Java Field(see page 2435)
  - Java Operation(see page 2436)
  - Java Interface(see page 2439)
  - Java Enumeration(see page 2440)
  - Java Enumeration Literal(see page 2441)
  - Java Annotation Type(see page 2442)
  - Java Annotation Member(see page 2443)
  - Java Annotations Usage(see page 2444)
  - Java Type Variables(see page 2445)
  - Java Parameterized Type(see page 2447)
  - Java Imports(see page 2448)

Java CE Options

Java CE options are visible every time you attempt to reverse source code. This section describes the **Re solve collection generic** option for Java-specific options. Other options are common for all languages and are described in Reverse Options(see page 2406).

Resolve collection generics option

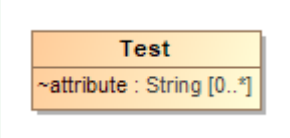
By default, this option is selected in the **Reverse Options** dialog. Java introduced parameterized types in JLS 3 and added type variables to all Java collections. With reverse engineering, it is possible to find which type is in a Java collection and make associations directly to the contained type, instead of the Java collection.

Reverse engineering allows you to find a Java parameterized collection and retrieves the Java type used in a container. This type is set as a *Type* to the UML Property. The container type is set to the UML Property/Java language property "Container" as a simple string.

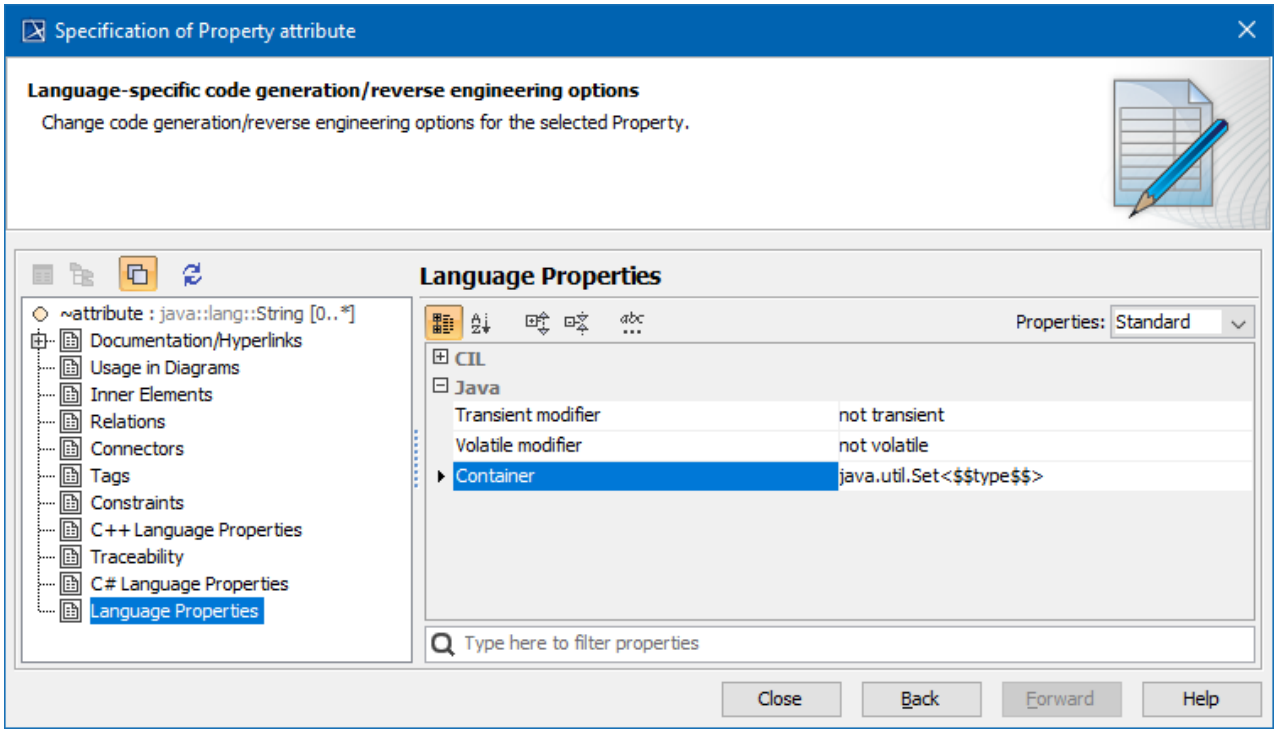
Example
Source code sample

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

UML Model



Attribute type, retrieved from collection



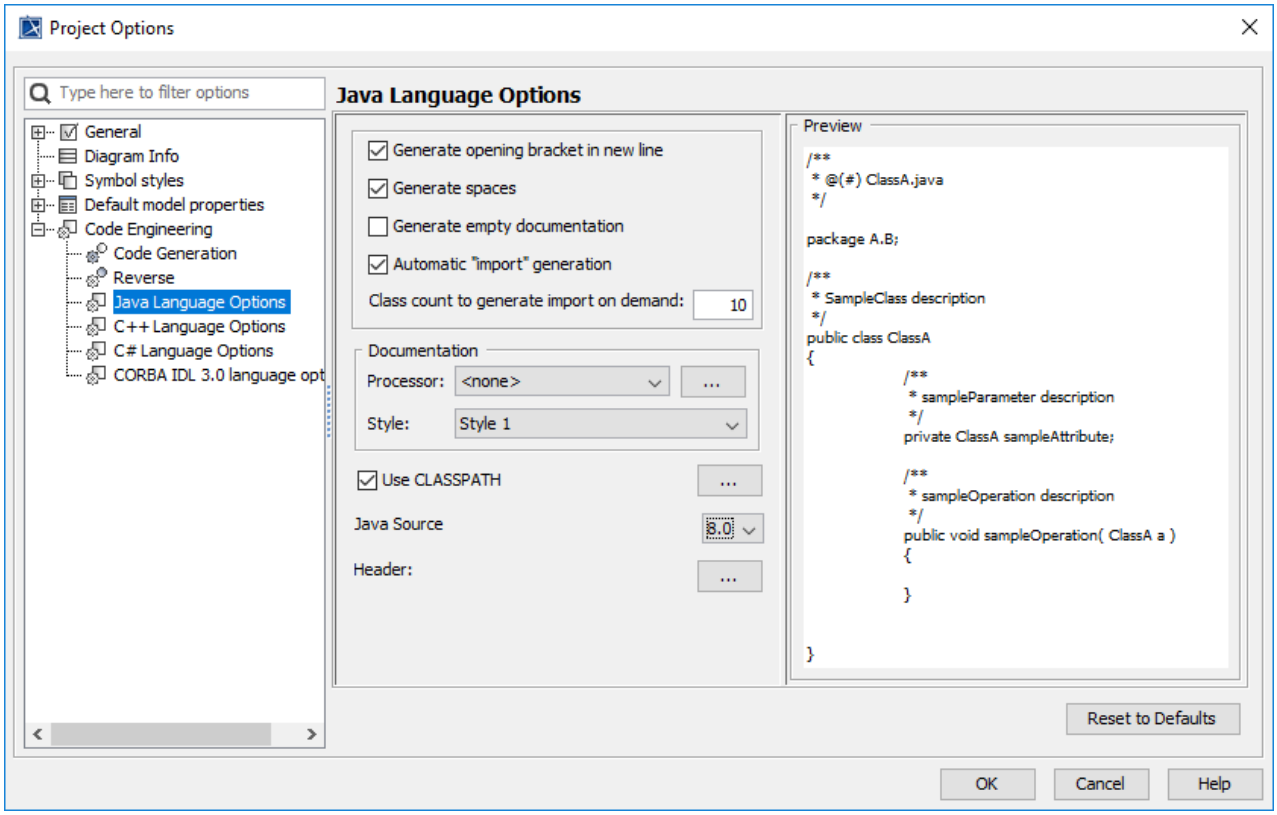
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Collection type in UML Property specification

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

Java Language Options

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



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.

```
/* *
 *
 */
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(see page 2448).

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(see page 2448).

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.

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

Related Pages:

- Java CE Options(see page 2449)
  - Java Language Options(see page 2451)
- Code Engineering(see page 2398)
  - Code Engineering Sets(see page 2399)
  -

Generating Code(see page 2403)
- Reverse Options(see page 2406)
- Global options for Code Engineering(see page 2415)
- Files of Properties(see page 2426)
- Java Code Engineering(see page 2427)
- C++ Code Engineering(see page 2470)
- C# Code Engineering(see page 2622)
- CORBA IDL Mapping To UML(see page 2680)
- WSDL(see page 2683)

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.

Related Pages:

- Java Code Engineering(see page 2427)
  - Java Mapping to UML(see page 2428)
  - Java CE Options(see page 2449)
  - Method Implementation Reverse(see page 2454)
- Sequence Diagram from Java Source Wizard(see page 1352)
- Code Engineering(see page 2398)
  - Code Engineering Sets(see page 2399)
  - Generating Code(see page 2403)
  - Reverse Options(see page 2406)
  - Global options for Code Engineering(see page 2415)
  - Files of Properties(see page 2426)
  - Java Code Engineering(see page 2427)
  - C++ Code Engineering(see page 2470)
  - C# Code Engineering(see page 2622)
  - CORBA IDL Mapping To UML(see page 2680)
  - WSDL(see page 2683)

C++ Code Engineering

References

- https://isocpp.org/std/the-standard
- ECMA-372 C++/CLI Language Specification [1259]

Demo

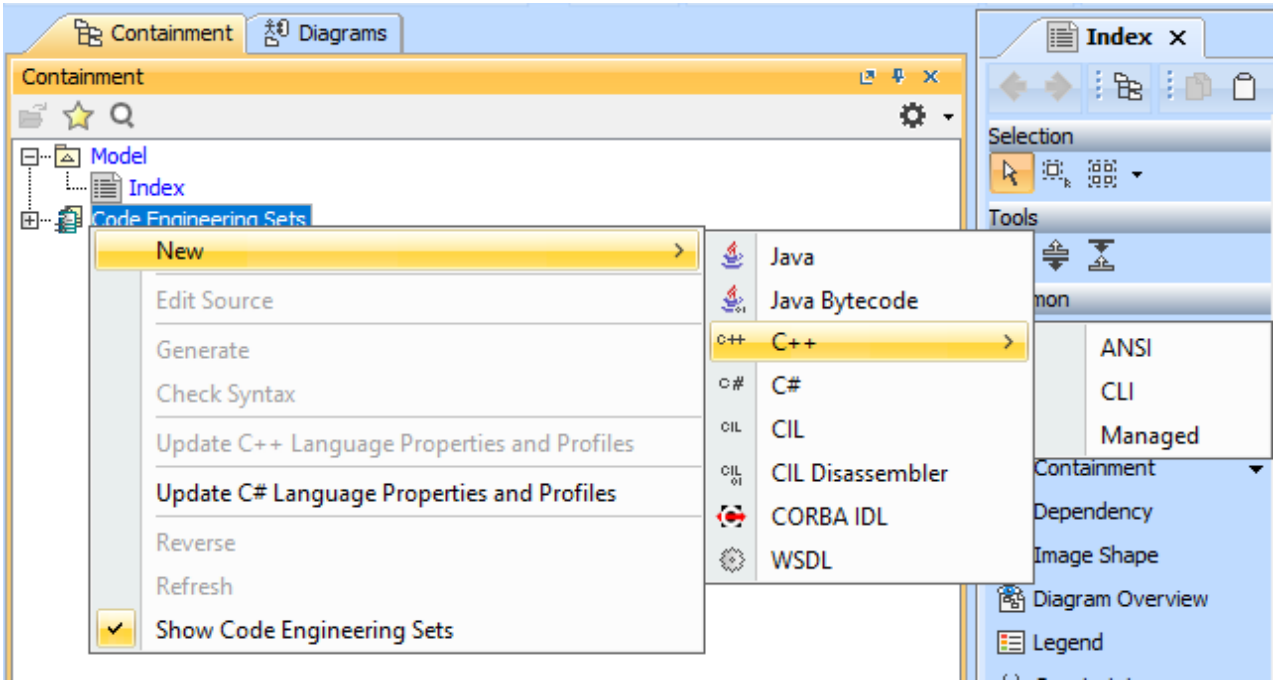
C++ Code Engineering [1260]

Supported dialects

C++ code engineering supports three dialects, set in our modelling tools:

- 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

> **Syntax**
> Syntax under the Managed dialect is deprecated in Microsoft Visual studio 2005.



New dialects in C++ CE

1259 https://www.ecma-international.org/publications/standards/Ecma-372.htm
1260 https://www.nomagic.com/support/demos#cpp

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related pages:

- C++ Code Engineering(see page 2454)
  - C++ ANSI Profile(see page 2472)
  - C++ Mapping to UML(see page 2487)
  - Conversion from an Earlier Project Version(see page 2527)
  - DSL Customization(see page 2557)
  - ANSI Profile Constraints(see page 2562)
  - Project Constraints(see page 2563)
  - Microsoft C++ Profiles(see page 2567)
  - Symbian macro(see page 2619)
  - @see on documentation(see page 2619)
  - Doxygen-after-member documentation(see page 2620)
  - CG Properties Editor(see page 2620)
  - Support C++ dialects(see page 2622)

C++ ANSI Profile

Also in this section:

- C++ Stereotypes(see page 2474)

The ANSI C++ profile is the base for all other C++ profiles. Other specialized C++ profiles must inherit from this profile by creating a generalization in modeling tool.

Data Types

Fundamental types defined by ANSI are mapped to a UML data type. You can state each fundamental type with a modifier (signed, unsigned, short, long) in a different order, but it is the same type. For example, you can use either short int or int short. Only versions defined as *type* by ANSI simple-type-specifiers are created as datatypes in the profile. If an UML synonym dataType is found during the reverse process, a class is created with this name.

Each fundamental type has 3 corresponding cv-qualified versions: const, volatile, and Const Volatile qualified type(see page 2500).

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

These datatypes are specific. In UML 2, only Integer, Boolean, String, and UnlimitedNatural are defined; they are not defined as datatypes, but as primitives. Our modeling tool does not define UnlimitedNatural in any profile and Integer and boolean (b is lowercase) are datatypes.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

PROFILE: UML Standard Profile. MagicDraw Profile.datatypes		
Data Type	**Synonym**	**Notes**
char	Char does not have a synonym	Char types are represented as signed char and unsigned char according to the compiler implementation.
int	Signed, signed int, int signed	
double	Double does not have a synonym	
float	Float does not have a synonym	
void	Void does not have a synonym	

PROFILE: C++ ANSI Profile.datatypes		
Data Type	**Synonym**	**Notes**
signed char	Signed char does not have a synonym	Char types are represented as signed char and unsigned char according to the compiler implementation.
unsigned char	Unsigned char does not have a synonym	
unsigned int	unsigned, int unsigned	
unsigned short int	unsigned short, short unsigned, unsigned int short, short unsigned int, short int unsigned, int short unsigned, int unsigned short	
unsigned long int	unsigned long, long unsigned, unsigned int long, long unsigned int, long int unsigned, int long unsigned, int unsigned long	

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

PROFILE: C++ ANSI Profile.datatypes		
Data Type	**Synonym**	**Notes**
long int	long, int long, signed lon, long signed, signed long int, signed int long, long signed int, long int signed, int signed long,int long signed	
short int	short, int short, signed short, short signed, signed short int, signed int short, short signed int, short int signed, int signed short, int short signed	
long double	Long double does not have a synonym	
bool	Bool does not have a synonym	
wchar_t	wchar_t does not have a synonym	

Related pages

- C++ Code Engineering(see page 2454)
  - C++ ANSI Profile(see page 2456)
  - C++ Mapping to UML(see page 2487)
  - Conversion from an Earlier Project Version(see page 2527)
  - DSL Customization(see page 2557)
  - ANSI Profile Constraints(see page 2562)
  - Project Constraints(see page 2563)
  - Microsoft C++ Profiles(see page 2567)
  - Symbian macro(see page 2619)
  - @see on documentation(see page 2619)
  - Doxygen-after-member documentation(see page 2620)
  - CG Properties Editor(see page 2620)
  - Support C++ dialects(see page 2622)

C++ Stereotypes

All C++ stereotypes are derived from «C++Element».

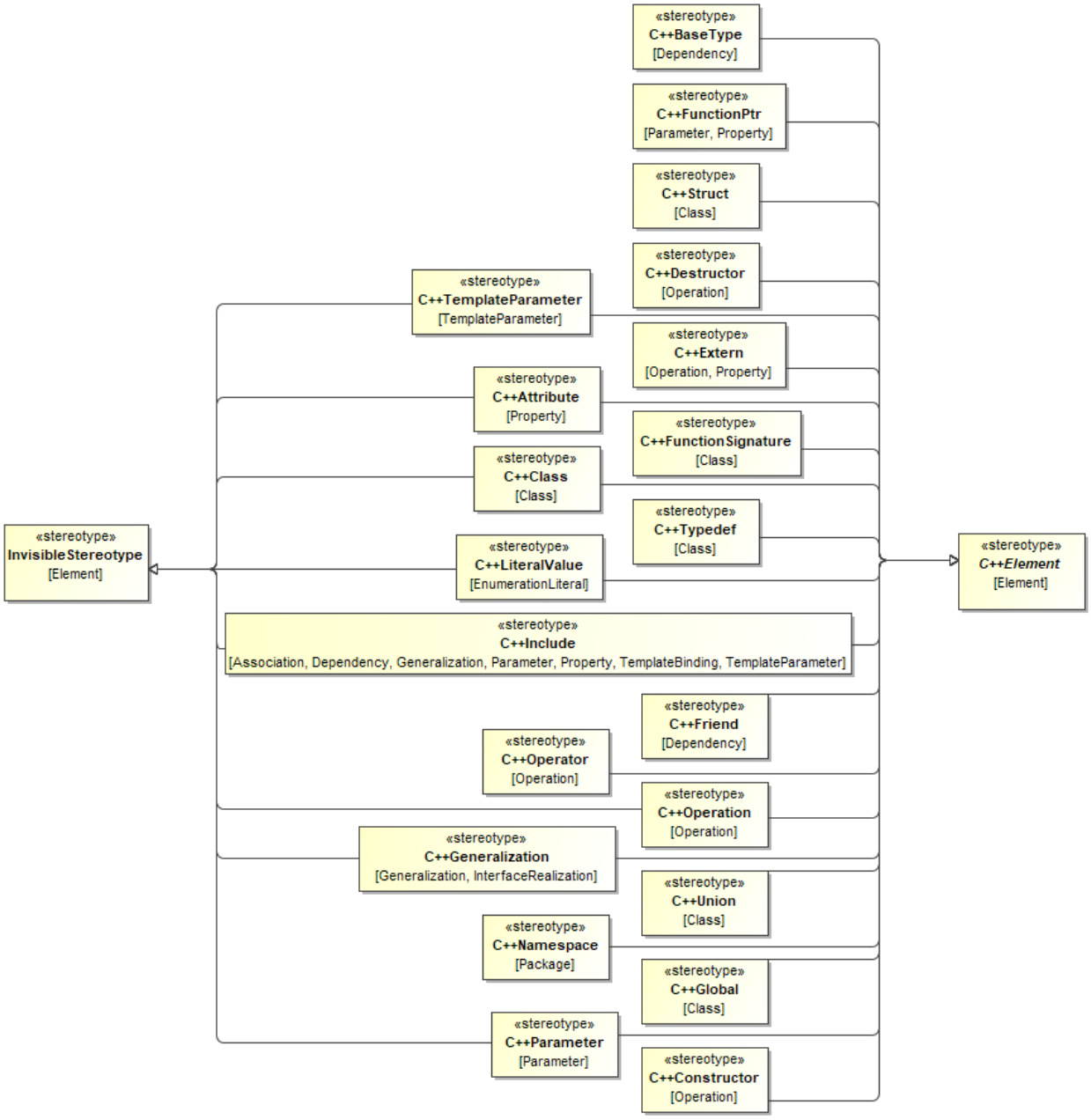
Constraints described in this chapter are for information only; syntax of these constraints must be checked with the Object Constraint Language(see page 1147) interpreter.

Invisible stereotypes *«C++Class»*, *«C++Operation»*, *«C++Parameter»*, *«C++Attribute»*, *«C++LiteralValue»*, *«C++Include»*, *«C++Generalization»*, and *«C++TemplateParameter»* are used only to store C++ language properties. These stereotypes and their tag definitions are used by the Domain Specific Language Customization(see page 2067) framework.

Here are the Stereotypes presented throughout the page:

- C++ Stereotypes(see page 2476)
- C++Operation(see page 2476)
- C++Operator(see page 2478)
- C++Constructor(see page 2478)
- C++Destructor(see page 2479)
- C++Parameter(see page 2479)
- C++ Attribute(see page 2480)
- C++LiteralValue(see page 2481)
- C++Friend(see page 2482)
- C++Struct(see page 2482)
- C++Typedef(see page 2482)
- C++Union(see page 2482)
- C++Global(see page 2482)
- C++Namespace(see page 2483)
- C++Extern(see page 2483)
- C++FunctionPtr(see page 2484)
- C++FunctionSignature(see page 2484)
- C++Class(see page 2484)
- C++BaseType(see page 2484)
- C++Include(see page 2485)
- C++TemplateParameter(see page 2486)
- C++Generalization(see page 2486)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



C++ Stereotypes

C++Operation

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

Name	Meta Class	Constraints

C++Operation	Operation	Function *Const*
		
		`void f() const;`
		Constraint: Only valid for member function (member functions are operators that are declared as members of a class)
		
		`if isQuery then`
		`stereotype->select(name='C++Global')->isEmpty()`

Tag definition	**Type and default value**	**Descriptions**
inline	boolean[1]=false	Function *Inline* `inline a();`

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

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

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

C++Operator

Stereotype *«C++Operator»* is used to define a C++ operator function. This stereotype extends the stereotype «C++Operation».

Name	Meta Class	Constraints
C++Operator	Operation	Function Operator `T& operator+(T& a);` Constraint: the name start with word *operator*

C++Constructor

Stereotype *«C++Constructor»* is used to define C++ Constructor. This stereotype extends the stereotype *«C++Operation»*.

Name	Meta class	Constraints
C++Constructor	Operation	name = owner.name

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Tag definition	Type and default value	Description
explicit	boolean[1]=false	`Constructor` *`Explicit`* `explicit a();`
initialization list	String[0..1]	Constructor initialization: `a() : x(1) {}`

C++Destructor

Stereotype *«C++Destructor»* is used to define C++ destructor. This stereotype extends Stereotype *«C++Operation»*.

Name	Meta Class	Constraints
C++Destructor	Operation	`name = “~”+owner.name`

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 definition	**Type and default value**	**Description**

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

Register	boolean[1]=false	Paramer *Register* `void a(register int x);`

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Array	String[0..1]	C++ *Array* definition `void a(int x[2][2]);`

C++ Attribute

An invisible stereotype *«C++Attribute»* is 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.
		
		`class A { int x = 1; };`
		`if defaultValue.size() > 0 then`
		`owner.stereotype->exists(name='C++Global') or (isStatic = true and`
		`typeModifiers.contains("const"))`

Tag definition	**Type and default value**	**Descriptions**

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

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

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

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;`

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

An invisible stereotype *«C++LiteralValue»* is used to include language properties for any C++ enum field.

Name	Meta Class	Constraints
C++LiteralValue	EnumerationLiteral	
Tag definition	**Type and default value**	**Descriptions**
value	String[0..1]	Value definition of an enum field: (A valid C++ expression) `enum Day {Mon = 2};`

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

C++Friend

Stereotype *«C++Friend»* is used to define C++ friend relationship.

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

Stereotype *«C++Struct»* is used to define C++ struct.

Name	Meta Class
C++Struct	Class

C++Typedef

Stereotype *«C++Typedef»* is used to define C++ typedef.

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

C++Union

Stereotype *«C++Union»* is used to define C++ union.

Name	Meta Class
C++Union	Class

C++Global

Stereotype *«C++Global»* is used to define global functions and variables (functions and variables outside a class/struct/union declaration).

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Name	Meta Class	Constraints

C++Global	Class	Only one «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

Stereotype *«C++Namespace»* is used to define a C++ namespace.

Name	Meta class	
C++Namespace	Package	
Tag definition	**Type and default value**	**Description**
unique namespace name	String[0..1]	Unnamed namespace `namespace {}`

C++Extern

Stereotype *«C++Extern»* is used to define C++ extern variable.

Name	Meta class	Constraints
C++Extern	Operation, Property	`owner.stereotype->exists(name='C++Global')`
Tag definition	**Type and default value**	**Description**
linkage	String[0..1]	Specification *Linkage:* `extern "C"`

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

C++FunctionPtr

Stereotype *«C++FunctionPtr»* is used to define C++ function pointer.

Name	Meta class	
C++FunctionPtr	Parameter, Property	
Tag definition	**Type and default value**	**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

Stereotype *«C++FunctionSignature»* is used as a container to model C++ function pointer.

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

C++Class

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

Name	Meta class
C++Class	Class

C++BaseType

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

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

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

Stereotype «C++Include» is used to keep information about the include type, 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 definition	**Type and default value**	**Description**
header include	String	The value of the tag can be one of the following • None • User Include • System Include • Class Forward
implementation include	String	The value of the tag can be one of the following • None • User Include • System Include • Class Forward

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

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

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

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

C++Generalization

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

Name	**Meta Class**	
C++Generalization	Generalization, InterfaceRealization	
Tag definition	**Type and default value**	**Description**
Inheritance Visibility	C++GeneralizationVisibility[1] = none	The value of tag can be one of the following • None • public • protected • private Example: class A : private B {};
Virtual Inheritance	boolean[1]=false	

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

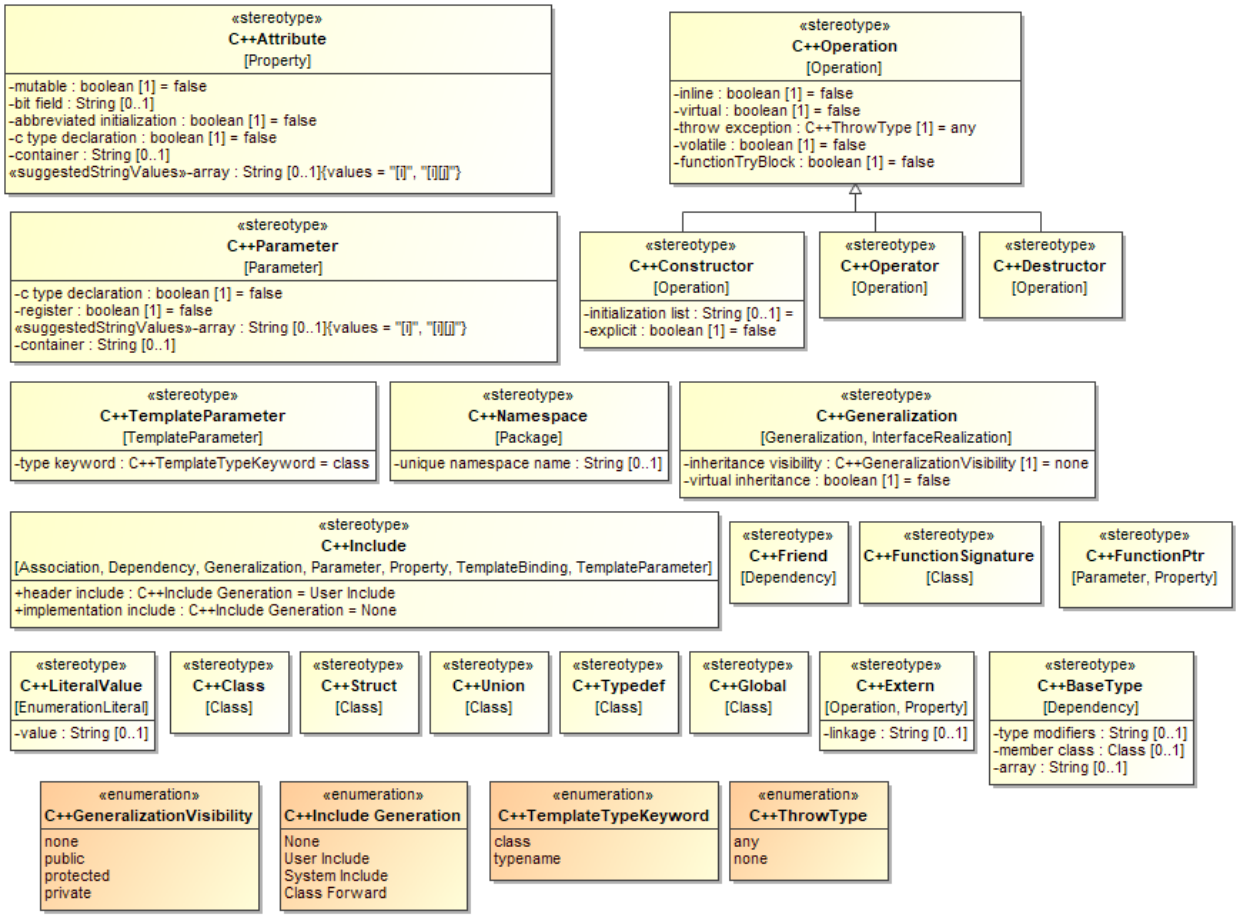
Related pages

- Code Engineering(see page 2398)
  - Code Engineering Sets(see page 2399)
  - Generating Code(see page 2403)
  - Reverse Options(see page 2406)
  - Global options for Code Engineering(see page 2415)
  - Files of Properties(see page 2426)
  - Java Code Engineering(see page 2427)
  - C++ Code Engineering(see page 2454)
  - C# Code Engineering(see page 2622)
  - CORBA IDL Mapping To UML(see page 2680)
  - WSDL(see page 2683)

C++ Mapping to UML

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

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Stereotypes with Tag values

Click here to see what's on this page

- Stereotypes with Tag values(see page 2472)
- C++Class(see page 2489)
  - Base Class Definition(see page 2490)
    - Class relations(see page 2491)
- Class Member Function(see page 2492)
  - C++Constructor(see page 2492)
  - C++Destructor(see page 2493)
- Variable(see page 2494)
  - Pointer and reference (see page 2494)
    - Pointer mapping with property Type Modifier $*(see page 2495)
    - Reference mapping with property Type Modifier $&(see page 2495)
  - Array (see page 2495)
    - <> Tag arrayVar value is 5(see page 2496)
- Mutable variable modifiers(see page 2496)
  - When variable is mutable, its Tag mutable value is true(see page 2497)
- Bit field(see page 2497)
  - Tag bit field with value 2(see page 2498)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Variable Default Value(see page 2499)
  - Variable initial value is mapped to UML attribute's default value(see page 2500)
  - Tag Abbreviated Initialization set to true.(see page 2500)
- Const Volatile Qualified Type(see page 2500)
  - Tag bit field with value 2(see page 2501)
- Function(see page 2501)
  - Return type of function(see page 2502)
- Function Variable-Length Parameter List(see page 2502)
- Void Parameter(see page 2503)
  - Operation with type void(see page 2503)
- Register Parameter(see page 2503)
  - Tag register(see page 2504)
- Function Modifiers(see page 2504)
  - Tag explicit(see page 2505)
- Function Pointer(see page 2505)
  - Tag explicit(see page 2506)
- Function Operator(see page 2506)
- Exception (see page 2507)
  - Property throw exception(see page 2507)
- Visibility (see page 2507)
- Static members(see page 2508)
  - Property Is Static(see page 2509)
- Pure virtual function and abstract class(see page 2509)
  - Operation property Is Abstract(see page 2510)
- Friend Declaration(see page 2510)
- Struct(see page 2512)
- Union(see page 2512)
- Enumeration(see page 2513)
- Typedef(see page 2513)
- Namespace(see page 2514)
- Global Functions And Variables(see page 2514)
- Class Definition(see page 2515)
- Class Template Definition(see page 2516)
  - Template Parameters(see page 2517)
  - Template Parameter Tag type keyword.(see page 2518)
- Function Template Definition(see page 2518)
  - Template Parameters(see page 2519)
- Default Template Parameter(see page 2519)
- Template Instantiation(see page 2519)
- Partial Template Instantiation(see page 2520)
- Template Specialization (see page 2520)
- Forward Class Declaration(see page 2521)
  - Template Parameters(see page 2522)
  - header include value Class forward(see page 2523)
- Include Declaration(see page 2523)
  - User Include(see page 2524)
  - System include(see page 2525)
  - User Include(see page 2526)
  - System include(see page 2527)

C++Class

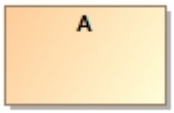
C++ class map to a UML class

Stereotype *«C++Class»* is an invisible stereotype used to include language properties for any C++ variable and belongs to Meta class *Class*.

Example
C++ source code

```
class A {
};
```

UML model:



Base Class Definition

A *base class* is a Class from which other Classes are derived. It facilitates the creation of other Classes that can reuse the code implicitly inherited from the *base class* (except constructors and destructors).

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

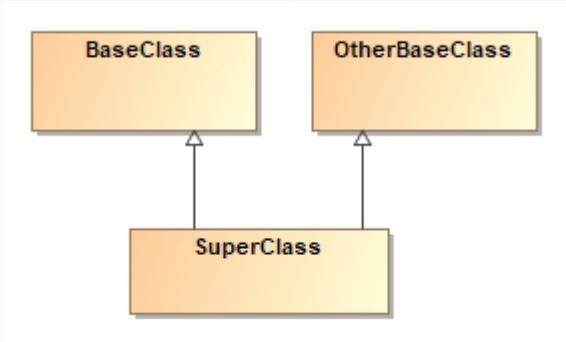
Access visibility (public, protected and private) and virtual properties of the *base class* are mapped to C++ language properties of the UML generalization.

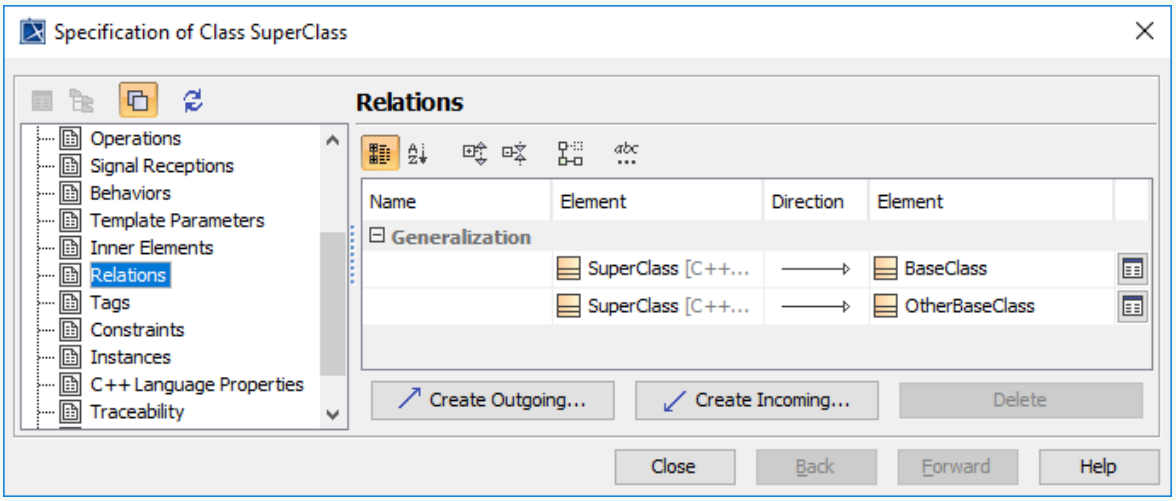
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example
C++ source code

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

UML model:





Class relations

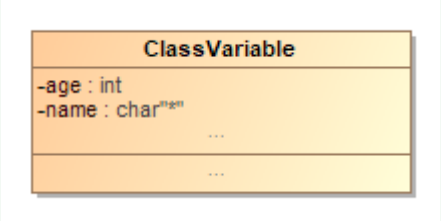
Class Member Variable

Class member variables are mapped to UML attributes. See Variable(see page 2475) for more info.

Example
C++ source code

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

UML model:



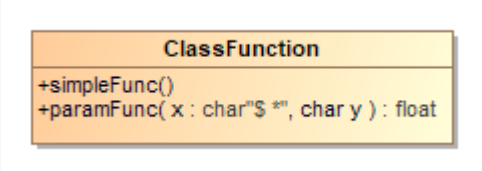
Class Member Function

Class member functions are mapped to UML operations.

Example
C++ source code

```
class ClassFunction {
public:
 void simpleFunc();
 float paramFunc(int x,char y);
};
```

UML model:



Class Constructor and Destructor

C++Constructor

Stereotype *«C++Constructor»* is used to define C++ Constructor. This stereotype extends stereotype *«C++Operation»*.

Name	Meta class	Constraints

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

C++Constructor	Operation	name = owner.name[1261]
Tag definition	**Type and default value**	**Description**
explicit	boolean[1]=false	`Constructor` *`Explicit`* `explicit a();`
initialization list	String[0..1]	Constructor initialization: `a() : x(1) {}`

C++Destructor

Stereotype *«C++Destructor»* is used to define C++ destructor. This stereotype extends stereotype *«C++Operation»*.

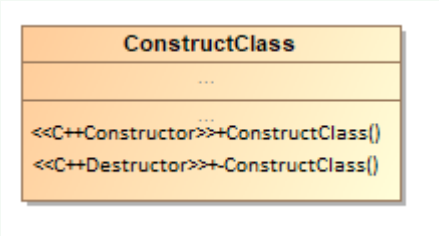
Name	Meta Class	Constraints
C++Destructor	Operation	`name = "~"+owner.name`

C++ class constructor and destructor are mapped to UML operation with stereotypes *«C++Constructor»* and «C++Destructor»:

Example
C++ source code

```
class ConstructClass {
public:
 ConstructClass();
 ~ConstructClass();
}
```

UML model:



1261 http://owner.name

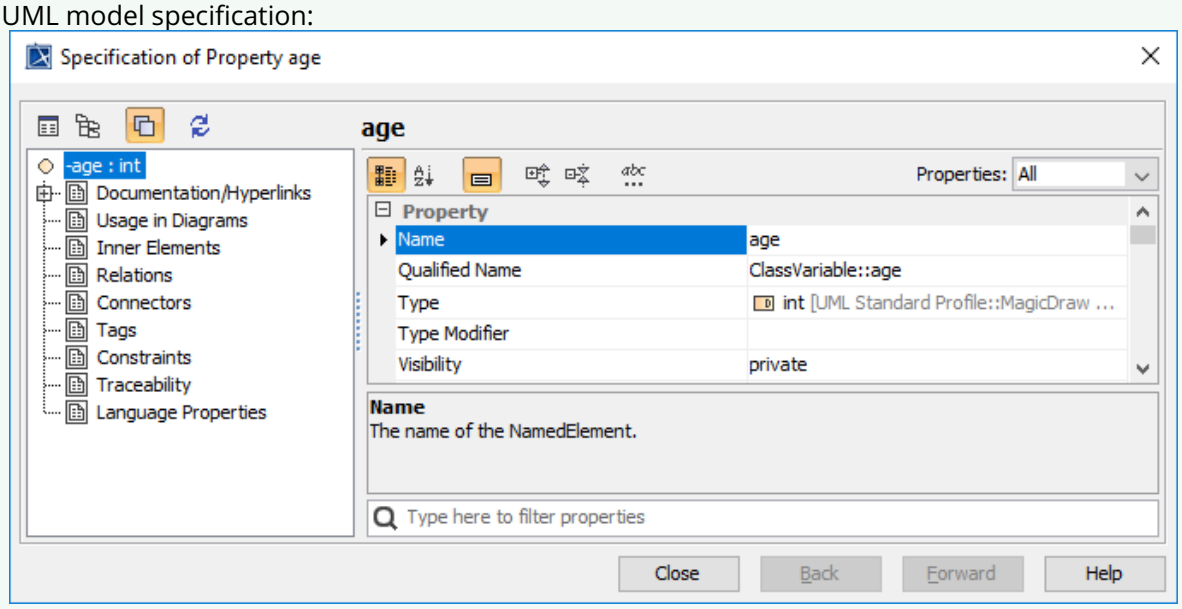
Variable

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

Example
C++ source code

```
int age;
```

UML model specification:



Pointer and reference

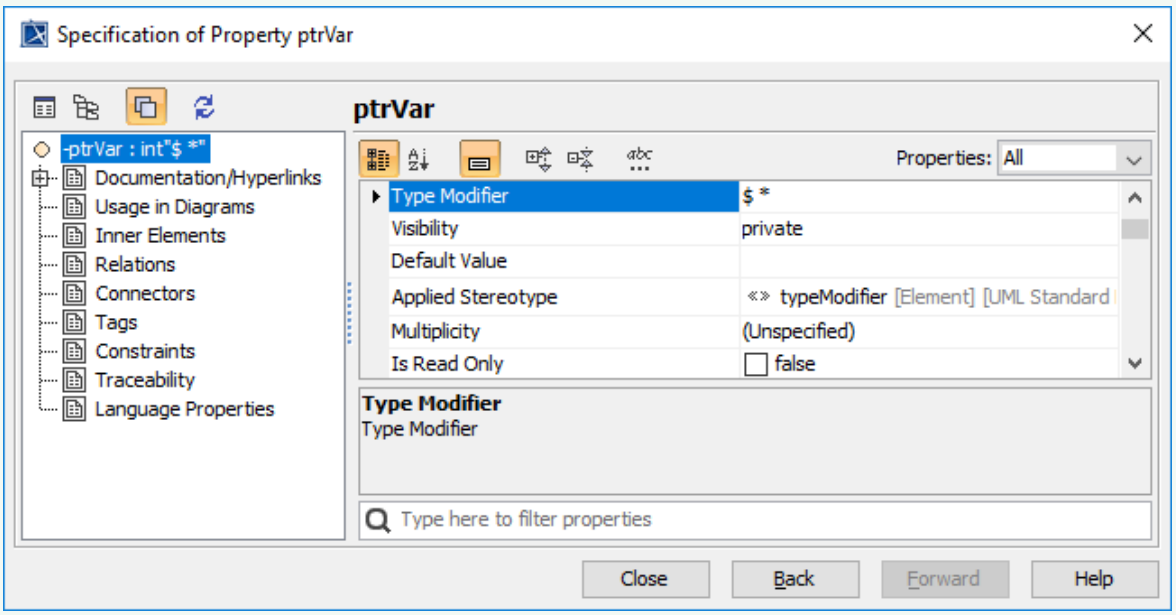
C++ type *pointer* and *reference* is mapped to property *Type Modifier* of the attribute. Character *$* is replaced by the type name.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

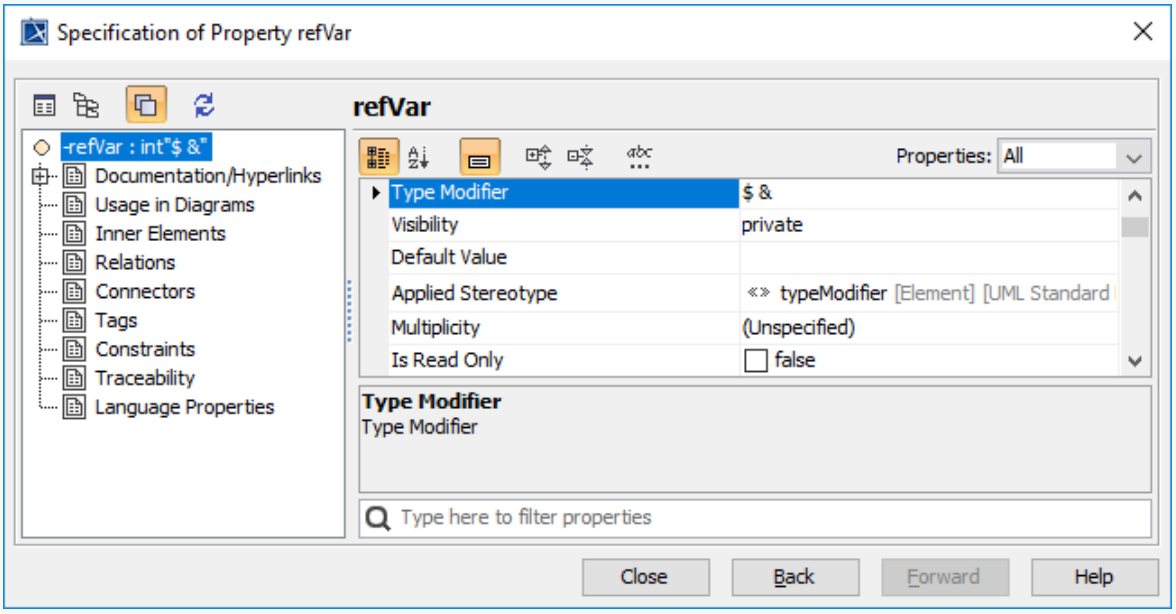
Example
C++ source code

```
int* ptrVar;
int& refVar
```

UML model specification:



Pointer mapping with property *Type Modifier $**



Reference mapping with property *Type Modifier $&*

Array

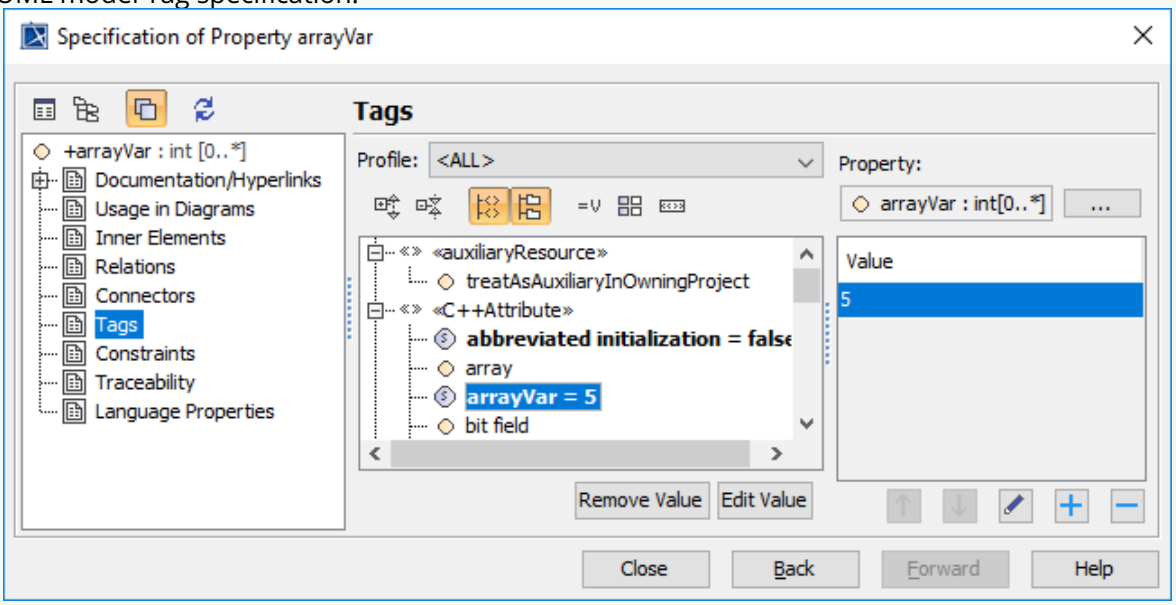
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

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..*]”

Example
C++ source code

```
int arrayVar[5];
```

UML model Tag specification:



<<C++Attribute>> Tag *arrayVar* value is 5

Mutable variable modifiers

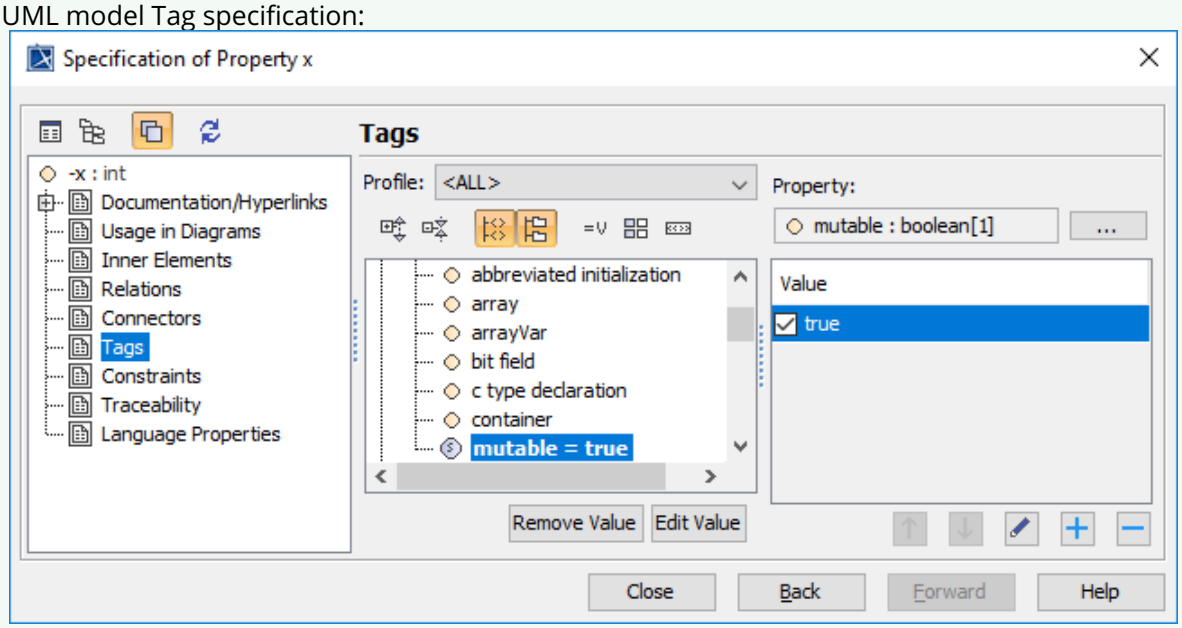
Mutable variable modifiers are mapped to UML attribute’s language property *Mutable*.

Constraint: only member variable can be *Mutable* (*Global variable* cannot be *Mutable).*

Example
C++ source code

```
mutable int x;
```

UML model Tag specification:



When variable is mutable, its Tag *mutable* value is true

Bit field

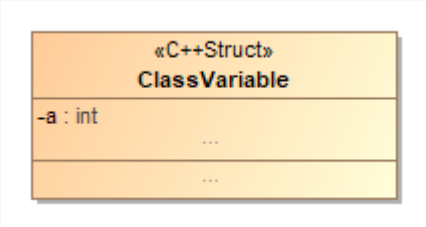
Bit field is mapped to tag value *Bit field*.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

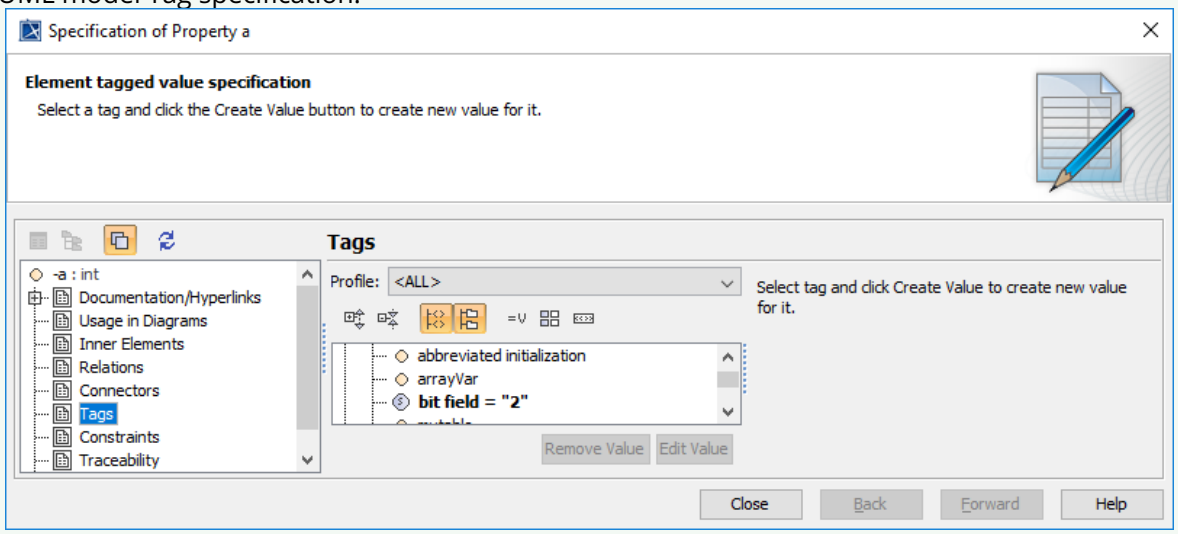
Example
C++ source code

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

UML model:



UML model Tag specification:



Tag *bit field* with value 2

Variable Extern

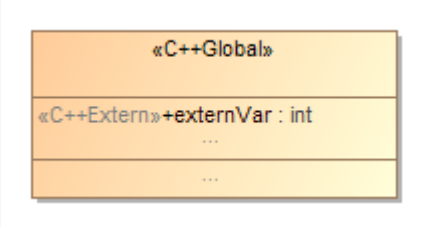
C++ extern variable is mapped to stereotype *«C++Extern»*. Tag *Linkage* value is used to specify the kind of linkage *C* or *C++*, if *linkage* is not specified (or without value), extern variable without *linkage* is generated.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example
C++ source code

```
extern int externVar;
```

UML model:



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 Tag *Abbreviated Initialization* set to true.

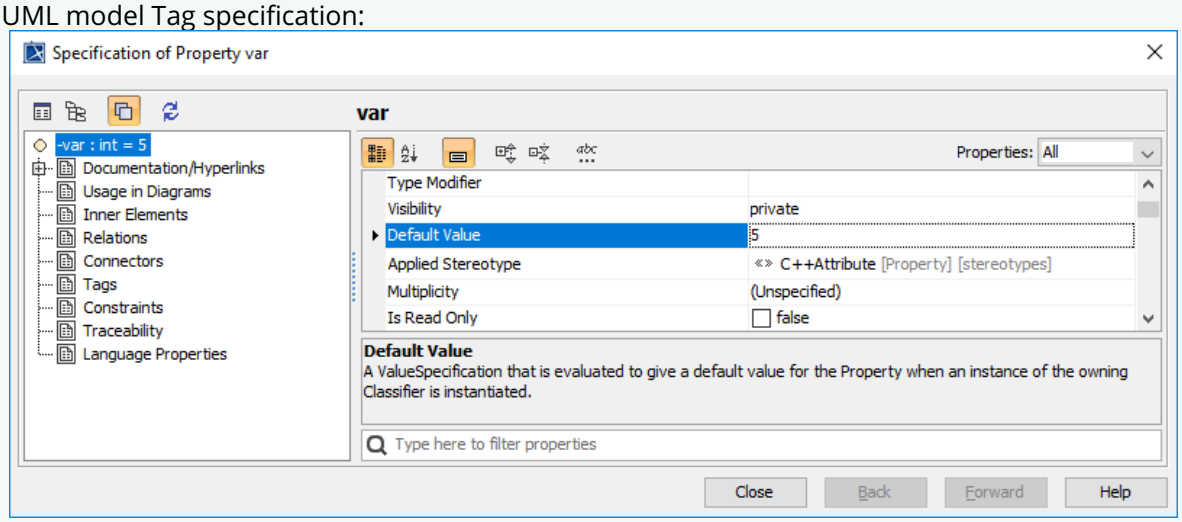
Constraint: only *static const* member variables can be initialized, and they cannot be initialized using function style method.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

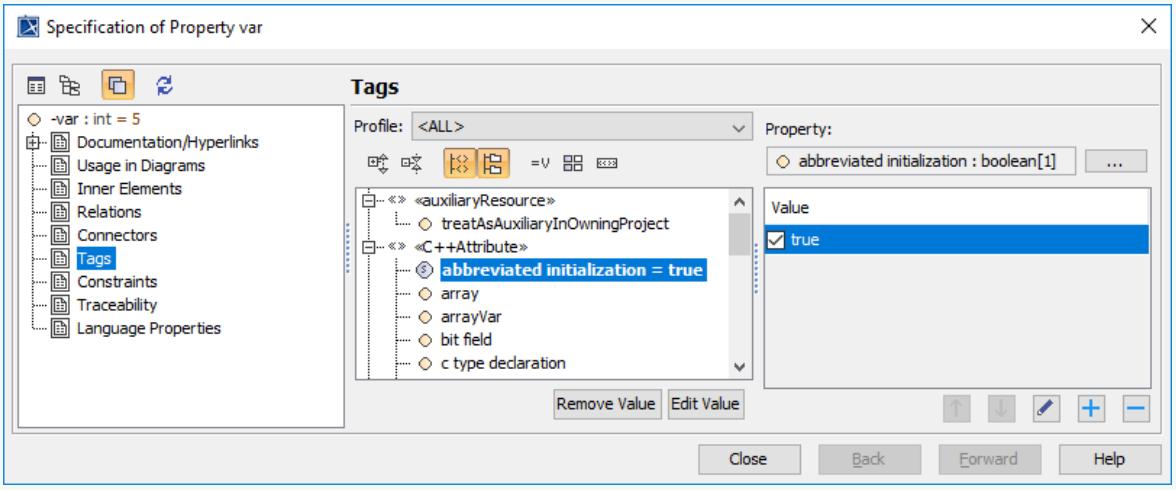
Example
C++ source code

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

UML model Tag specification:



Variable initial value is mapped to UML attribute's default value



Tag *Abbreviated Initialization* set to true.

Const Volatile Qualified Type

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

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

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

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

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.

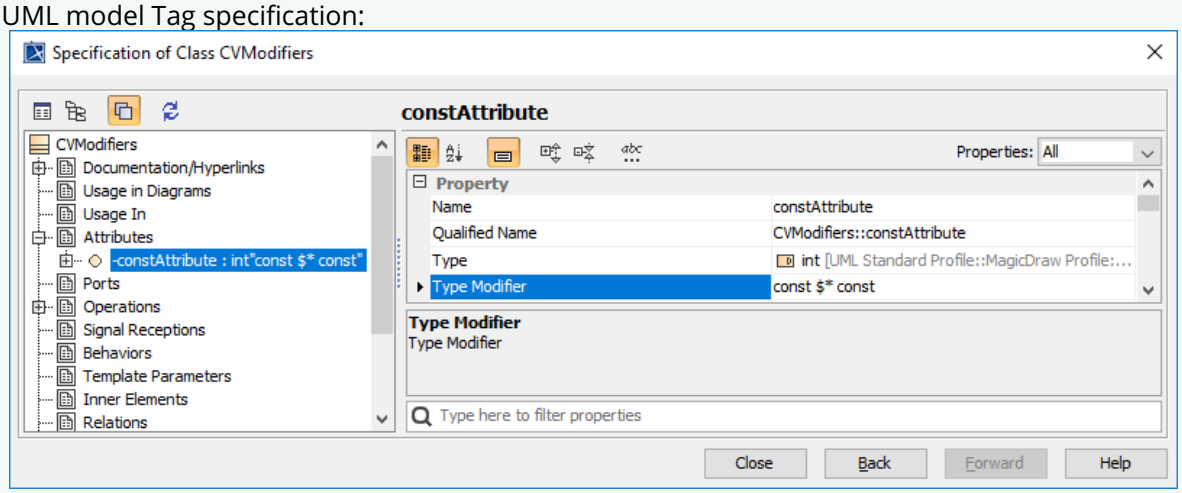
Example

C++ source code

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

UML model:

UML model Tag specification:



Tag *bit field* with value 2

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.

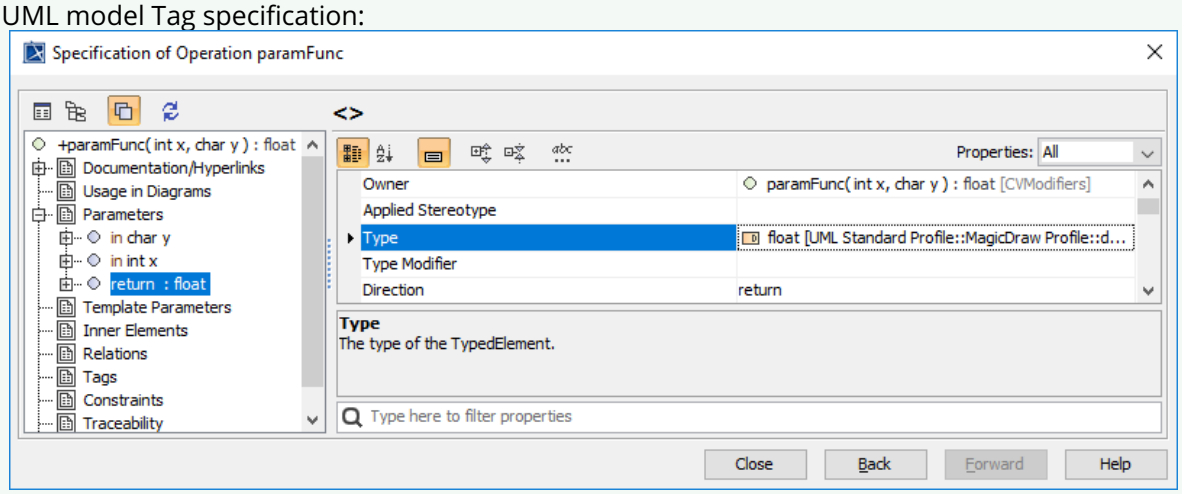
Pointer, reference and array type of parameter are mapped to property **Type Modifier** of parameter.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example
C++ source code

```
float paramFunc(int x, char x);
```

UML model Tag specification:



Return type of function

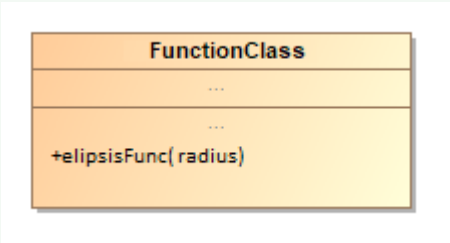
Function Variable-Length Parameter List

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

Example
C++ source code

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

UML model:



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Void Parameter

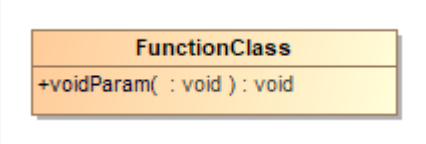
C++ void function parameter is mapped to a UML parameter without name and with type *void*.

Example

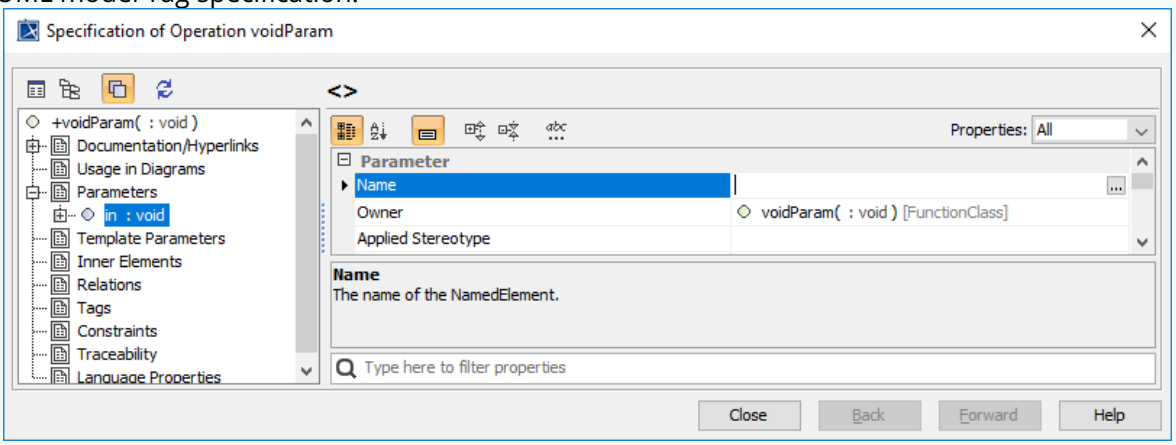
C++ source code

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

UML model:



UML model Tag specification:



Operation with type void

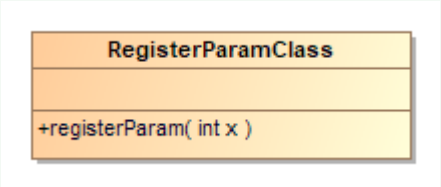
Register Parameter

C++ register parameter is mapped to UML parameter language property **Register Depending** on the compiler, register can be limited on some types (int, char).

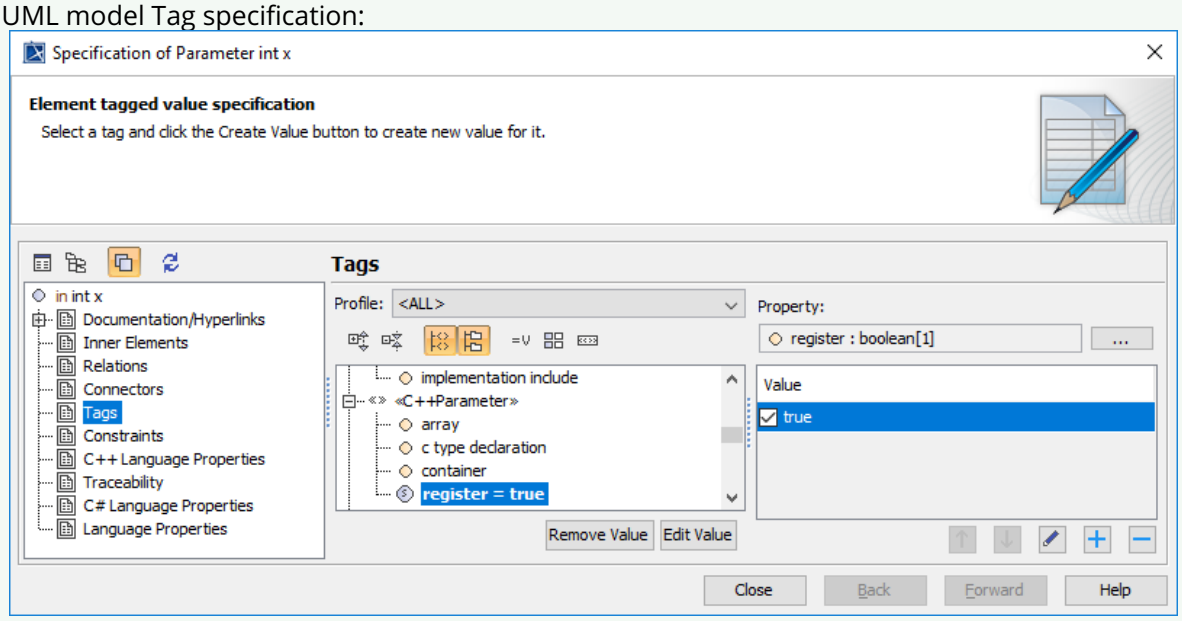
Example
C++ source code

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

UML model:



UML model Tag specification:



Tag *register*

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.

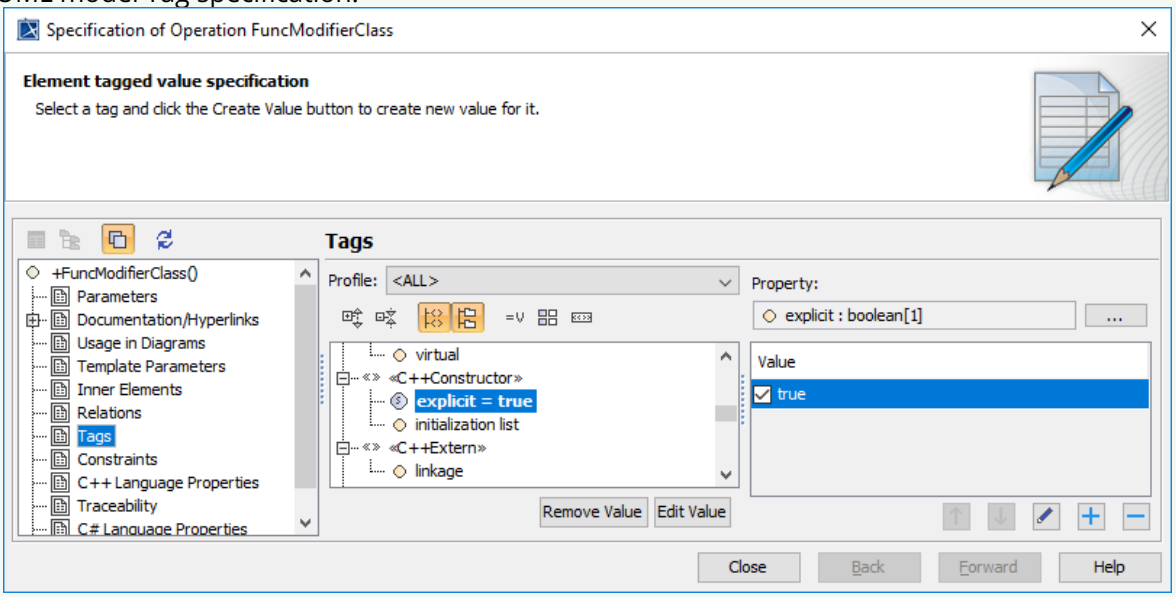
Volatile function is mapped to Tag value *volatile*.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example
C++ source code

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

UML model Tag specification:



Tag *explicit*

Function Pointer

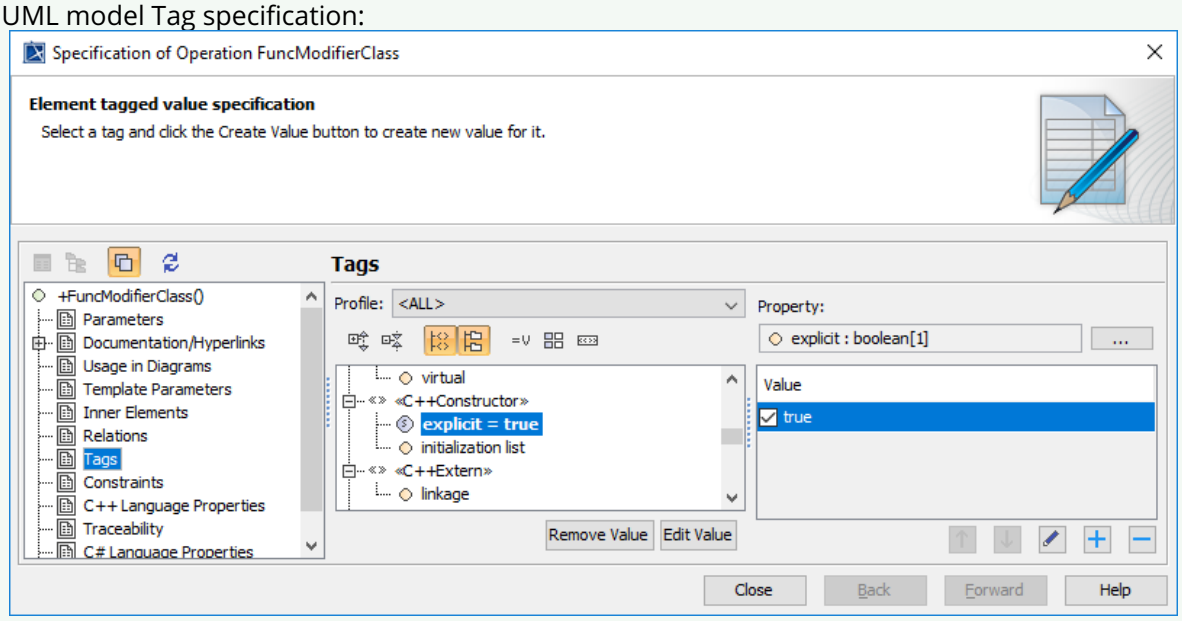
C++ function pointer type is mapped to attribute/parameter with «C++FunctionPtr» stereotype, a dependency with «C++BaseType» stereotype link from the attribute/parameter to the operation in a «C++FunctionSignature» class, and type modifiers of the dependency is set to *$. Member function pointer use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

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example
C++ source code

```
float (A*funcPtr)(int);
```

UML model Tag specification:



Tag *explicit*

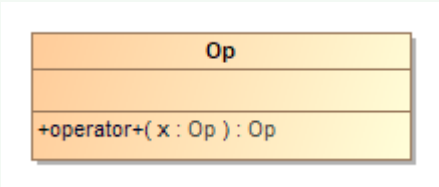
Function Operator

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

Example
C++ source code

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

UML model:



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

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.

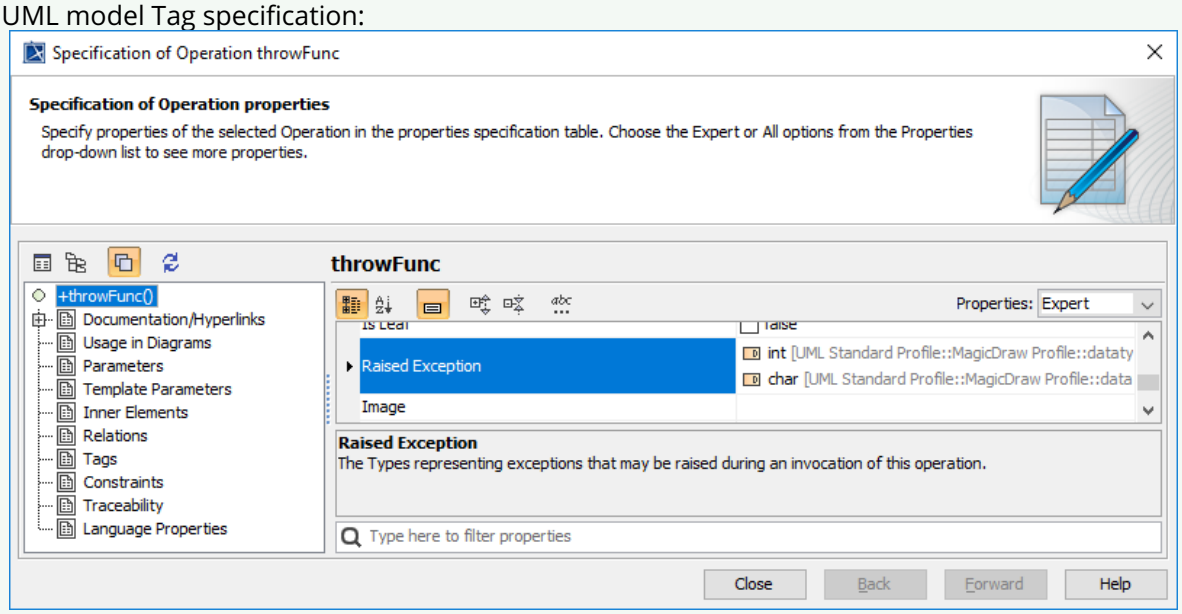
If the tag *throw exception* is not set, then generate specific *raisedExpression*, or do not generate throw if *raisedExpression* is empty.

Example

C++ source code

```
void throwFunc() throw (int,char);
```

UML model Tag specification:



Property *throw exception*

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.

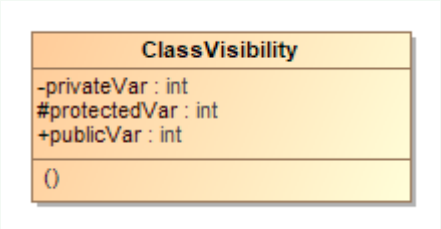
Variables and functions outside a class/struct/union are public.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example
C++ source code

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

UML model:



Static members

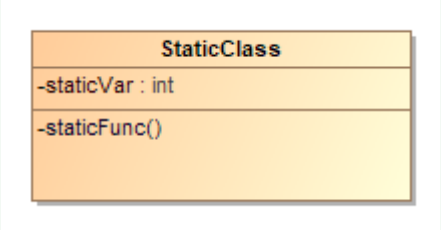
Static variables and functions are mapped to UML *Is Static* property.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

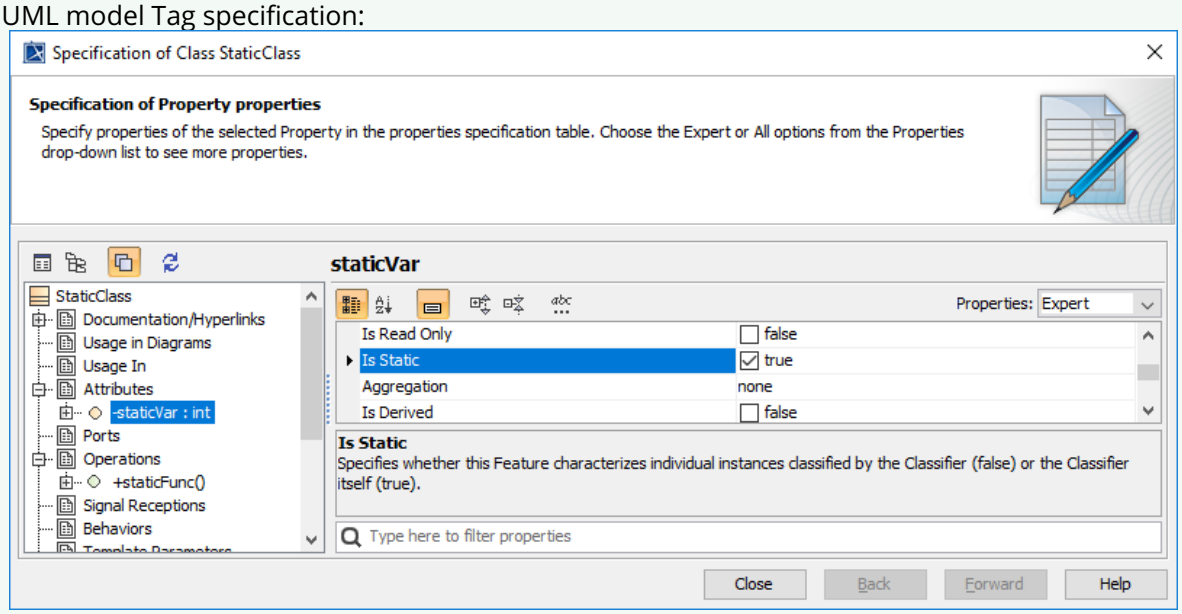
Example
C++ source code

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

UML model:



UML model Tag specification:



Property *Is Static*

Pure virtual function and abstract class

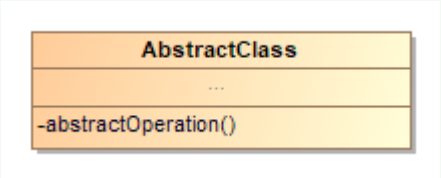
Pure virtual C++ function is mapped to UML operation with property *Is Abstract* set to true. If one or more functions are abstract in a class, the property *Is Abstract* of the UML class is set to true. Constraint: if no operation is abstract, the class cannot be abstract.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

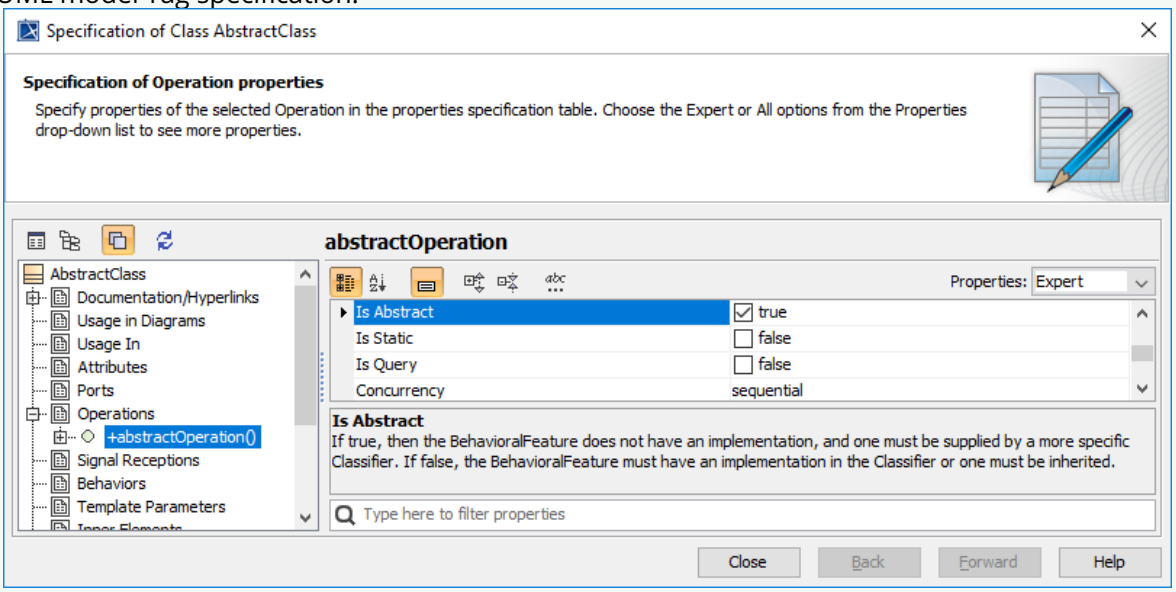
Example
C++ source code

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

UML model:



UML model Tag specification:



Operation property Is Abstract

Friend Declaration

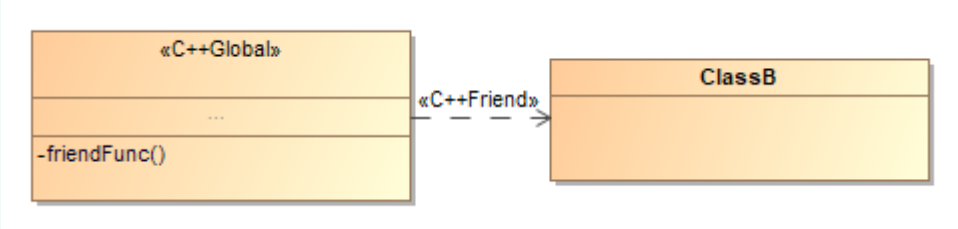
C++ friend function is mapped with a «C++Friend» stereotyped dependency relationship between the function (an UML operation) and the friendClass. This relationship grants the friendship to the friendClass.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example
C++ source code

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

UML model:

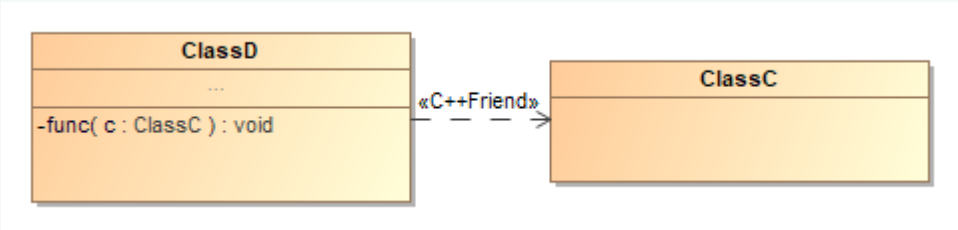


C++ friend member function is mapped with a «C++Friend» stereotyped dependency relationship between the member function and the friend class. This relationship grants the friendship to the friend class.

Example
C++ source code

```
class ClassD {
void func(ClassC c);
};
class ClassC {
    friend void ClassD::func(ClassC
c);
};
```

UML model:



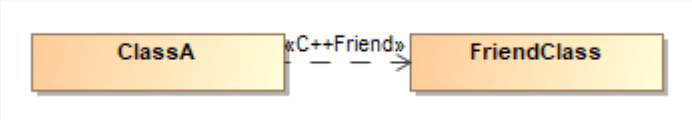
C++ friend class are mapped with a «C++Friend» stereotyped dependency relationship between the class and the friend class. This relationship grants the friendship to the friend class.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example
C++ source code

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

UML model:



Struct

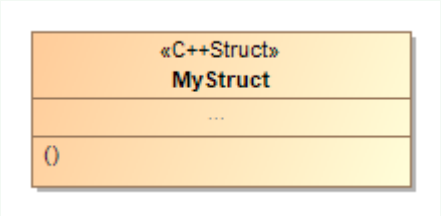
C++ struct are mapped to a UML class with stereotype «C++Struct». See **C++Struct** for more info.

Current Modeling tool
The current version of MD use class's language property "Class Key"

Example
C++ source code

```
struct MyStruc {
};
```

UML model:



Union

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

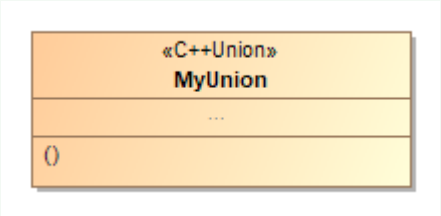
Current Modeling tool
The current version of MD use class's language property "Class Key"

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example
C++ source code

```
union MyUnion {
};
```

UML model:



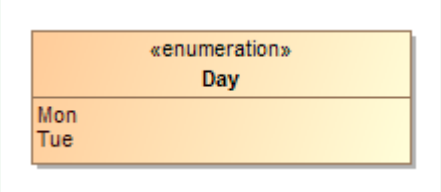
Enumeration

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.

Example
C++ source code

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

UML model:



Typedef

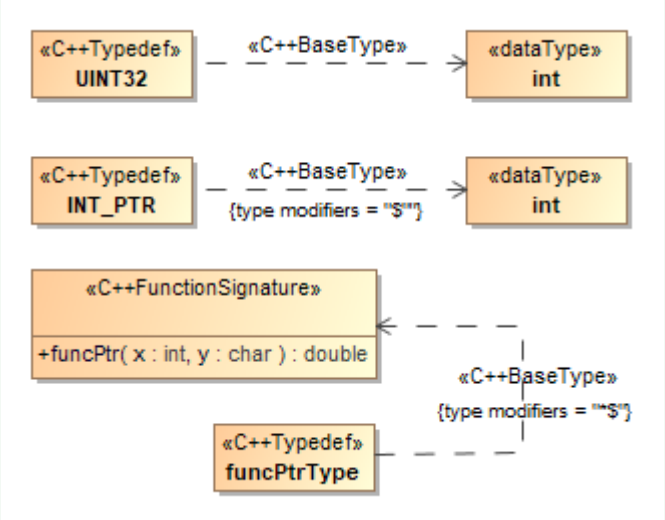
C++ typedef is mapped to a class with «C++Typedef» stereotype. A «C++BaseType» dependency links to the original type. Type modifiers tag of «C++BaseType» dependency is used to define type modifiers. *$* character is replaced by the type name. A typedef on a function pointer is mapped by linking a «C++BaseType» dependency to an operation and type modifiers tag of «C++BaseType» dependency is set to **$*. Operation signature can be stored in a «C++FunctionSignature» class.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example
C++ source code

```
typedef int UINT32;
typedef int* INT_PTR;
typedef double (*funcPtrType)(int, char);
;
```

UML model:



Namespace

C++ namespace is mapped to a UML package with the stereotype «C++Namespace». 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).

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

Global Functions And Variables

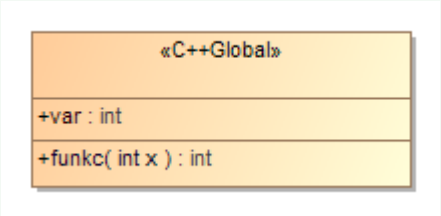
Global functions and variables are mapped to operations and attributes into an unnamed class with stereotype «C++Global». «C++Global» class resides in its respective namespace, or in a top package.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example
C++ source code

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

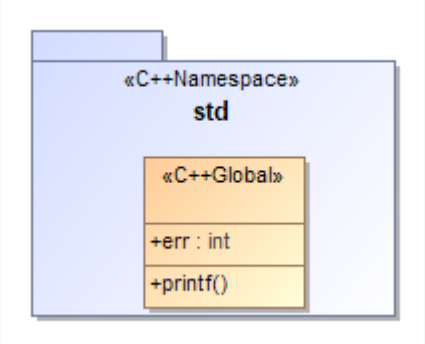
UML model:



Example
C++ source code

```
namespace std {
    int err;
    void printf();
}
```

UML model:



Class Definition

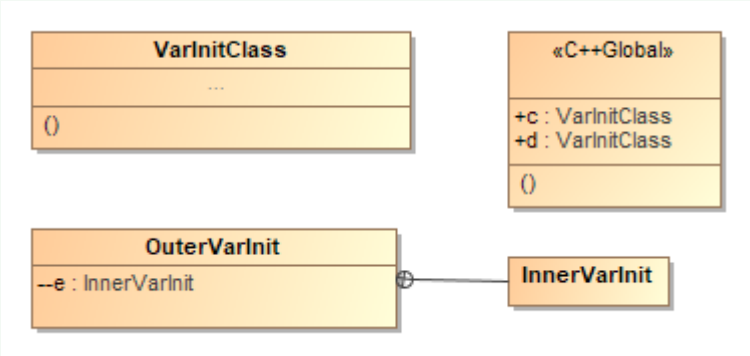
Variables can be created after a class/struct/union declaration. These variables are mapped to UML attribute, and placed in their respective namespace/global/class container.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example
C++ source code

```
class VarInitClass {
} c, d;
class OuterVarInit {
     class InnerVarInit {
     } e;
};
```

UML model:



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.

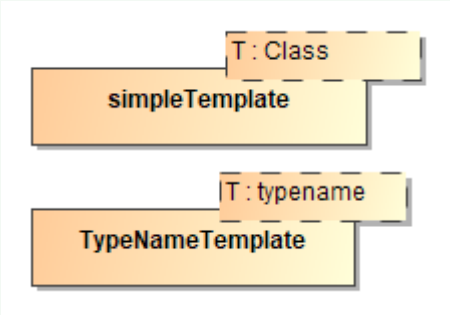
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

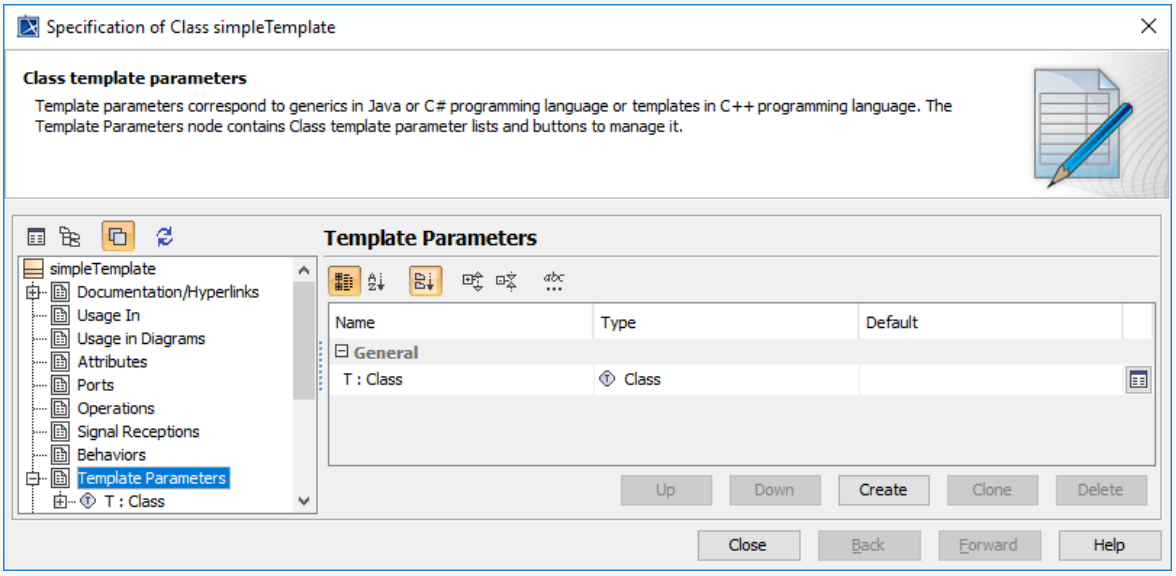
Example

C++ source code

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

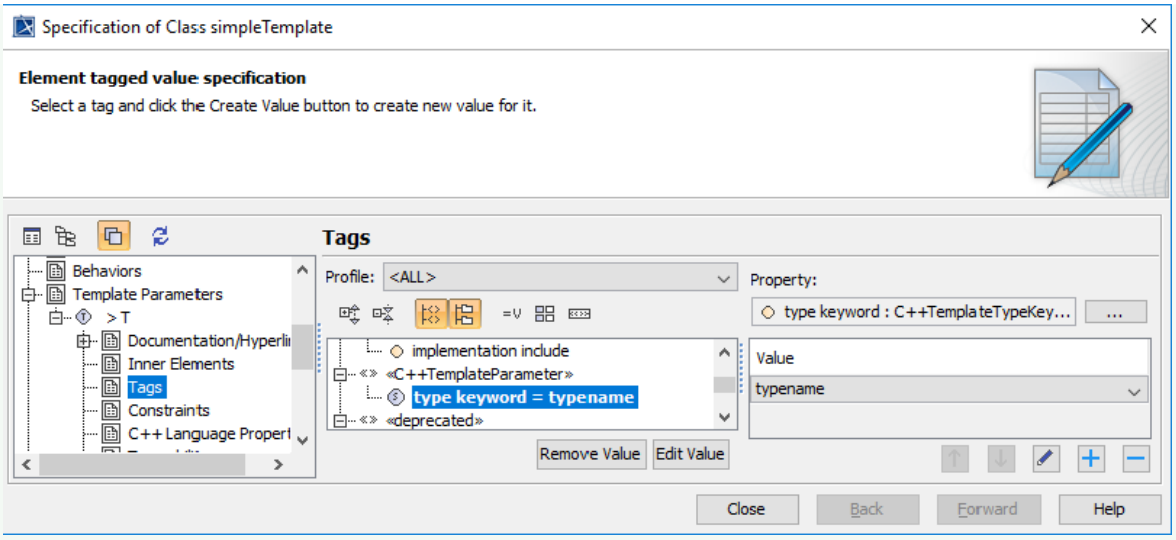
UML model:





Template Parameters

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Template Parameter Tag type keyword.

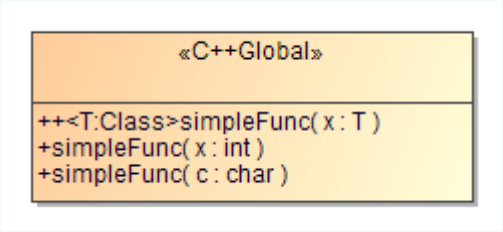
Function Template Definition

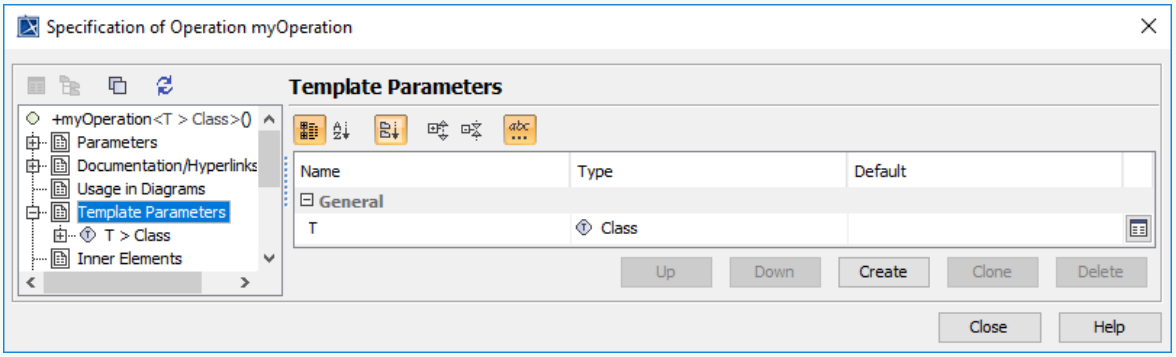
C++ template function is mapped to UML operation with template parameters properties added. C++ template function overload is mapped to a normal function. (the same name with the same number of parameter, but different type of parameter) New style of template function overloading is mapped to a normal 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.

Example
C++ source code

```
template <class T>
void simpleFunc(T x);
// overload old style
void simpleFunc(int x);
// overload new style
template<>
void simpleFunc<char>(char x);
```

UML model:





Template Parameters

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.

```
template <class T=int>
class defaultTemplate {
};
```

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.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

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

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

```
template <class T>
class T1Class {
};
template <class T>
class T2Class {
};
T1Class<T2Class<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.

```
template <class T>
class b {
 template <class T>
 class c {
 };
};
b<int>::c<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.

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.

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

Template Specialization

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

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

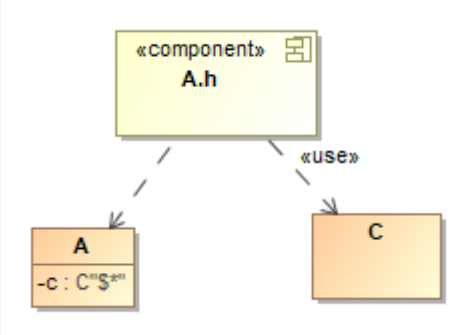
Forward Class Declaration

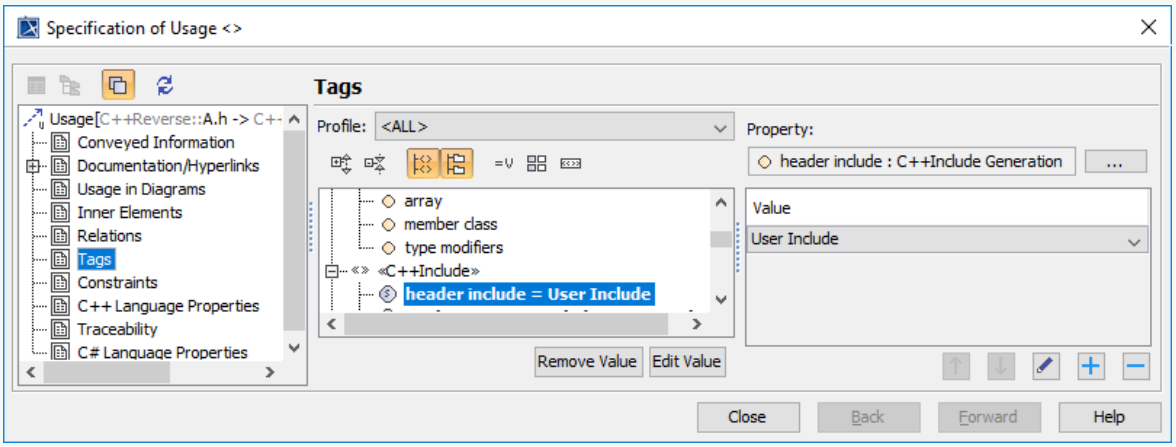
The example code is declared in A.h file. The file component A.h has the «use» association applied by «C++Include» stereotype with *Class Forward* tag value.

Example
C++ source code

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

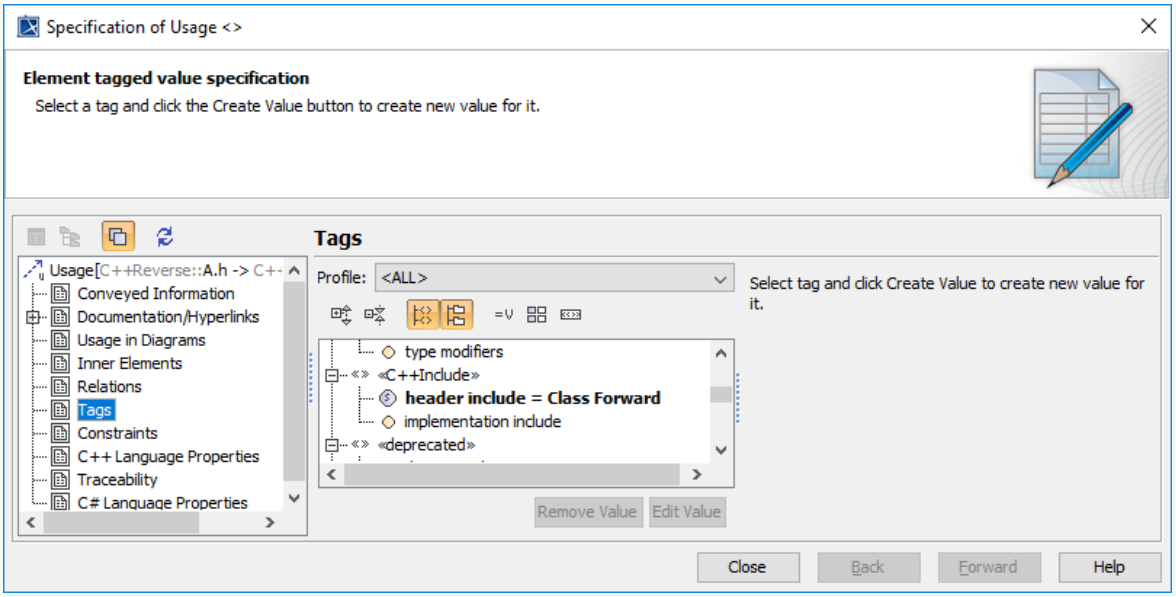
UML model:





Template Parameters

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



header include value *Class forward*

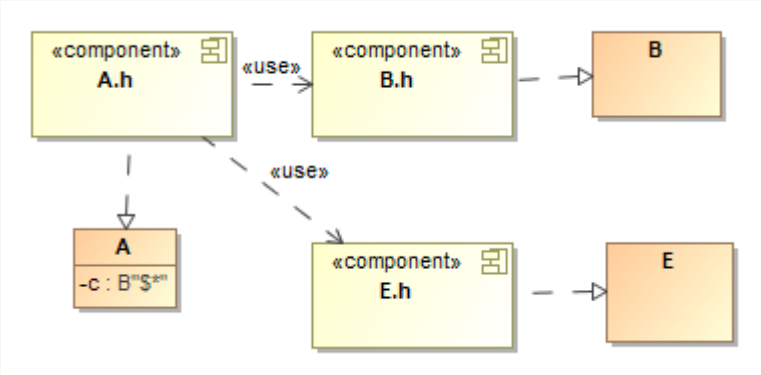
Include Declaration

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

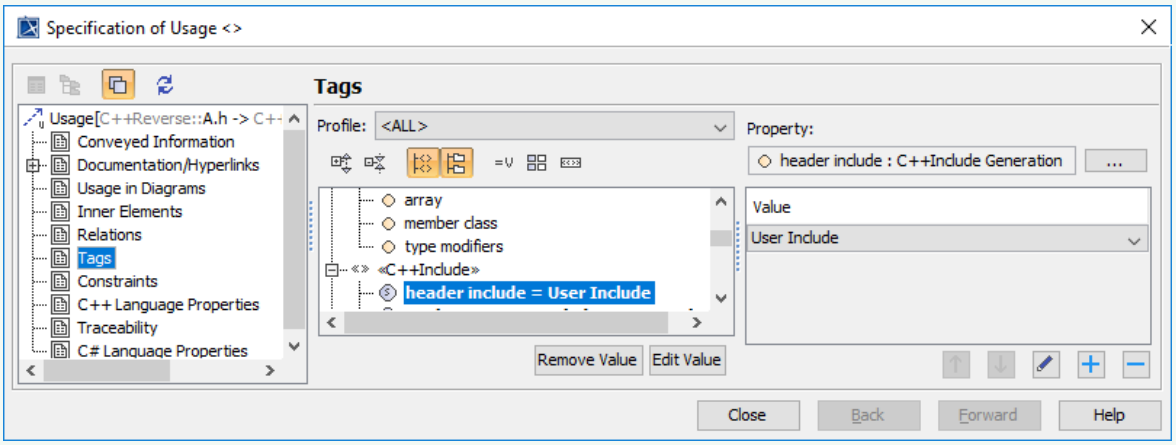
Example
C++ source code

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

UML model:

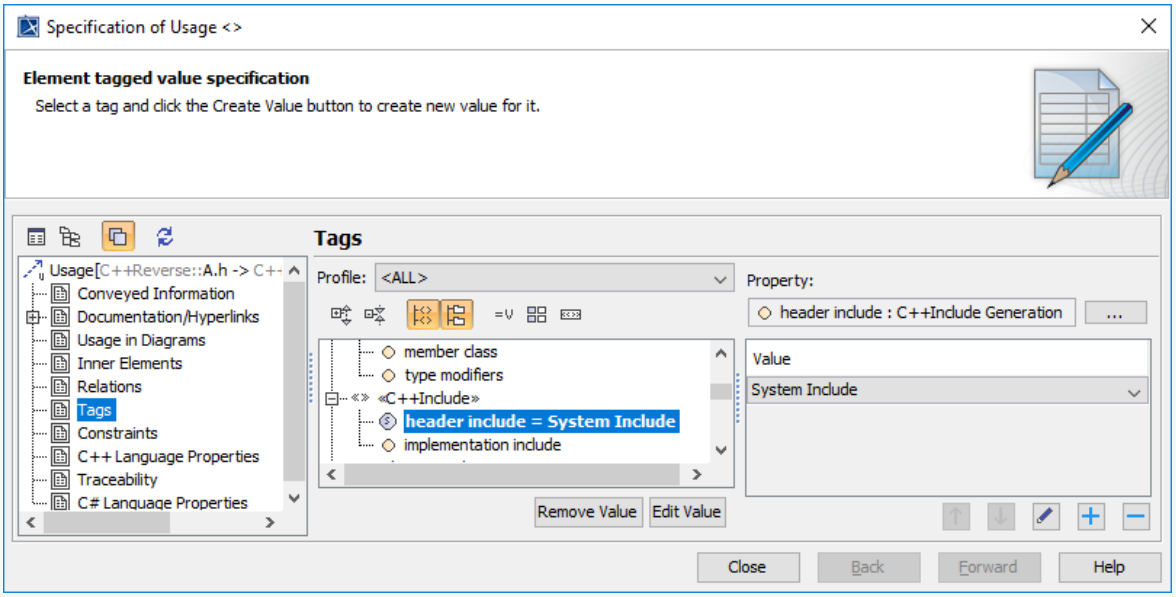


Specification for #include "B.h":



User Include

Specification for #include <E.h>:



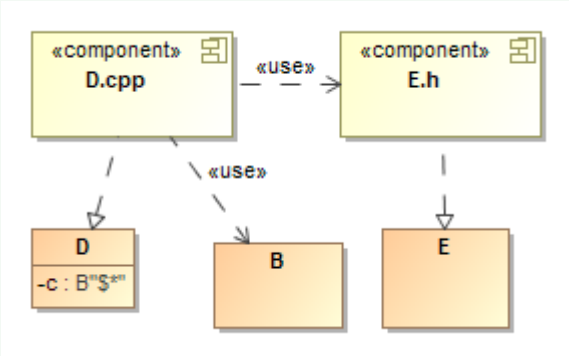
System include

The example code is declared in D.cpp file.

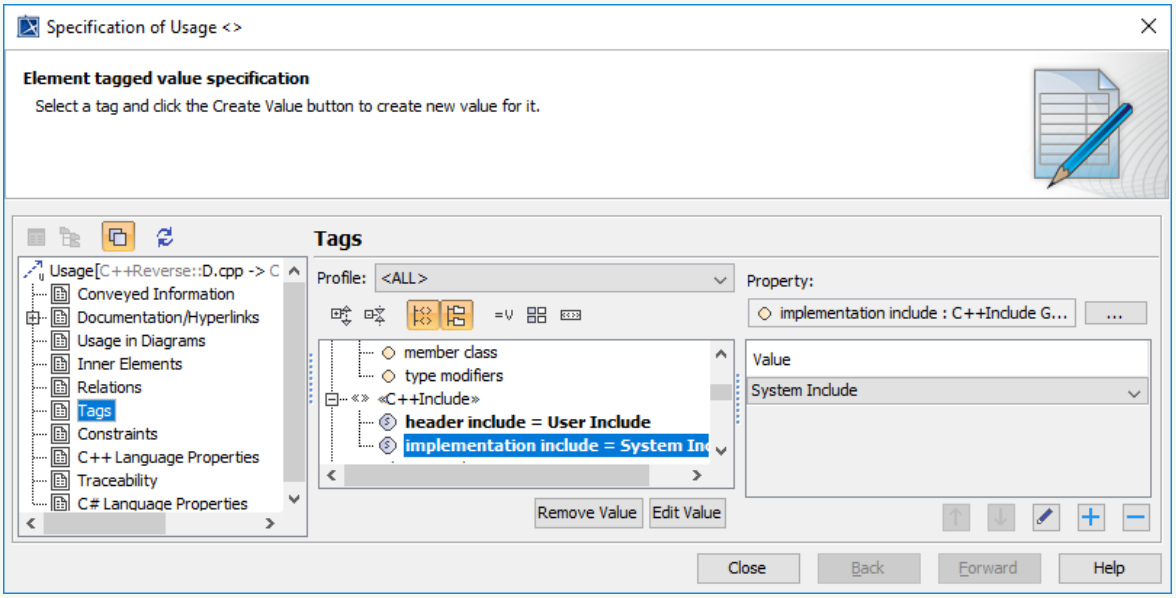
Example
C++ source code

```
#include <E.h>
class B;
class D {
private:B* b;
E* e;
}
```

UML model:

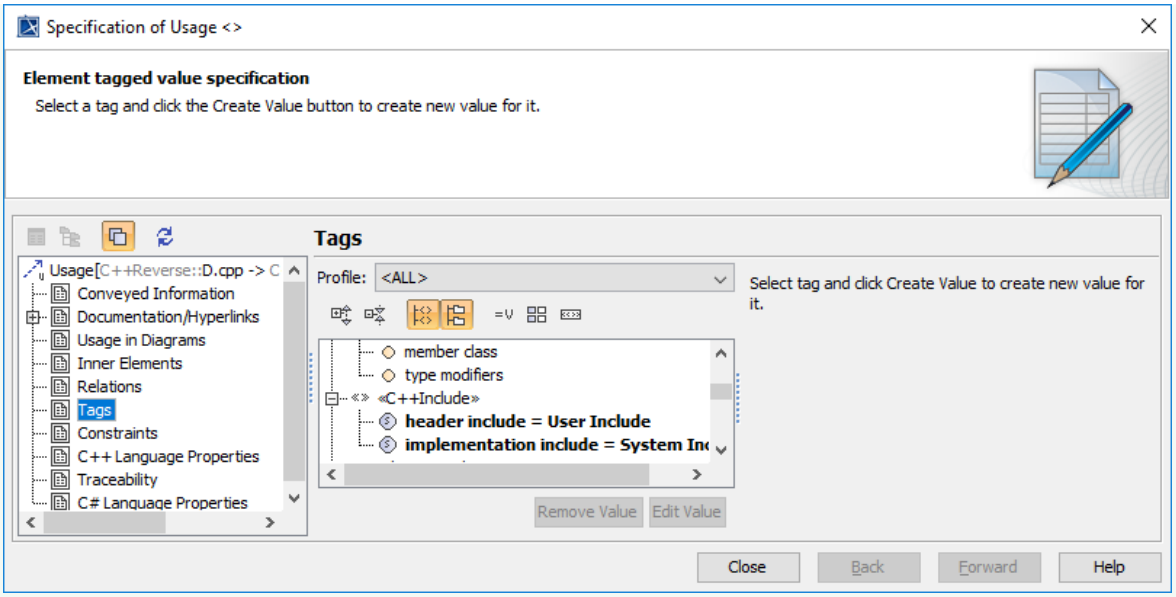


Specification for #include "B.h":



User Include

Specification for #include <E.h>:



System include

Conversion from an Earlier Project Version

This chapter describes the changes applied when loading a MagicDraw project with version 11.6 or earlier.

- Translation Activity Diagram(see page 2511)
- Language Properties(see page 2511)
- Conversion with Array Type Modifiers(see page 2511)
- Stereotypes from Old Project Version(see page 2511)
- Thrown exception tag value translation(see page 2511)
- Constructor and Destructor Name From Old Project Versions(see page 2551)
- Data Type From Old Project Versions(see page 2553)

Related Pages:

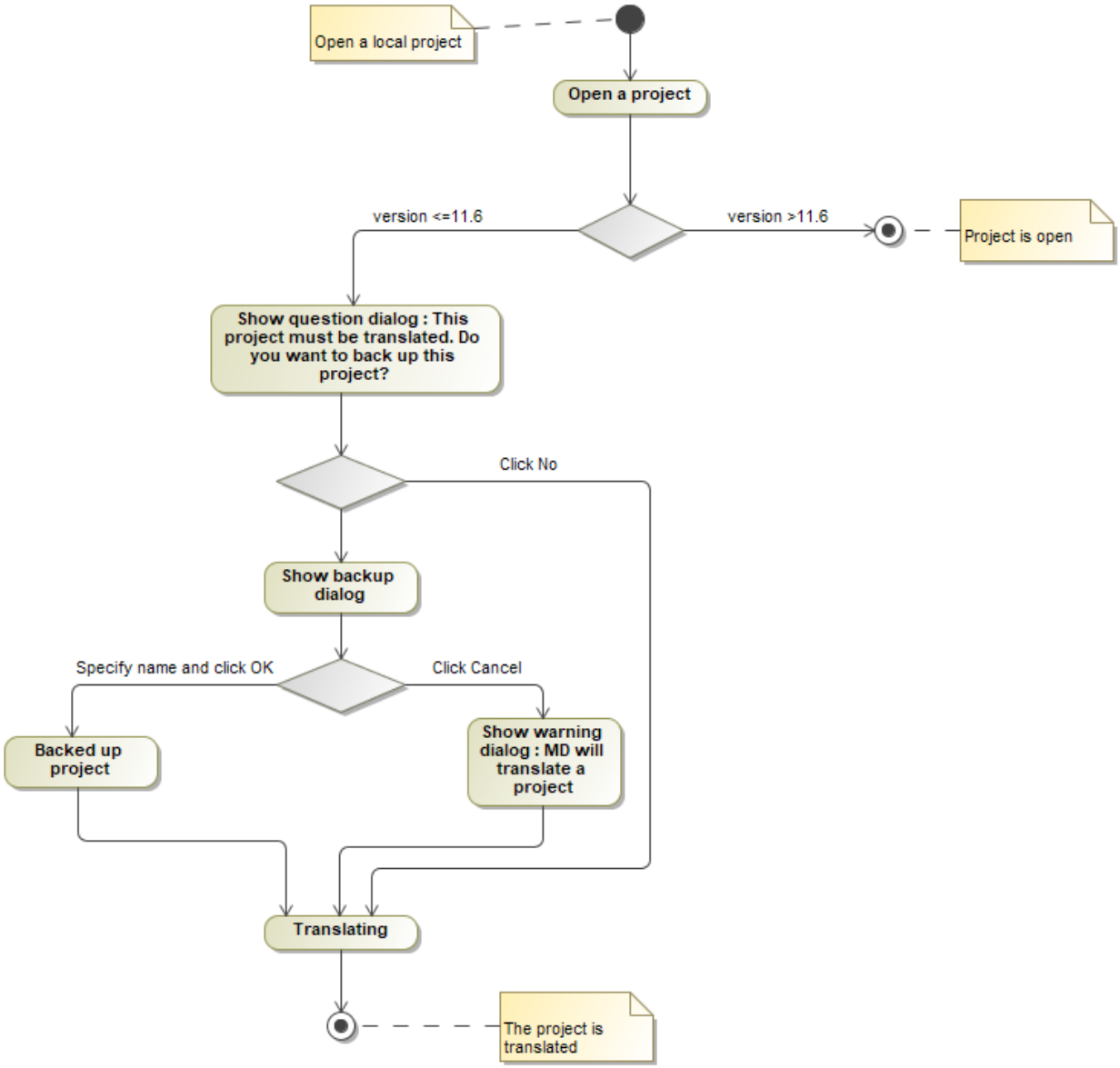
- Code Engineering(see page 2398)
  - Code Engineering Sets(see page 2399)
  - Generating Code(see page 2403)
  - Reverse Options(see page 2406)
  - Global options for Code Engineering(see page 2415)
  - Files of Properties(see page 2426)
  - Java Code Engineering(see page 2427)
  - C++ Code Engineering(see page 2454)
  - C# Code Engineering(see page 2622)
  - CORBA IDL Mapping To UML(see page 2680)
  - WSDL(see page 2683)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Translation Activity Diagram

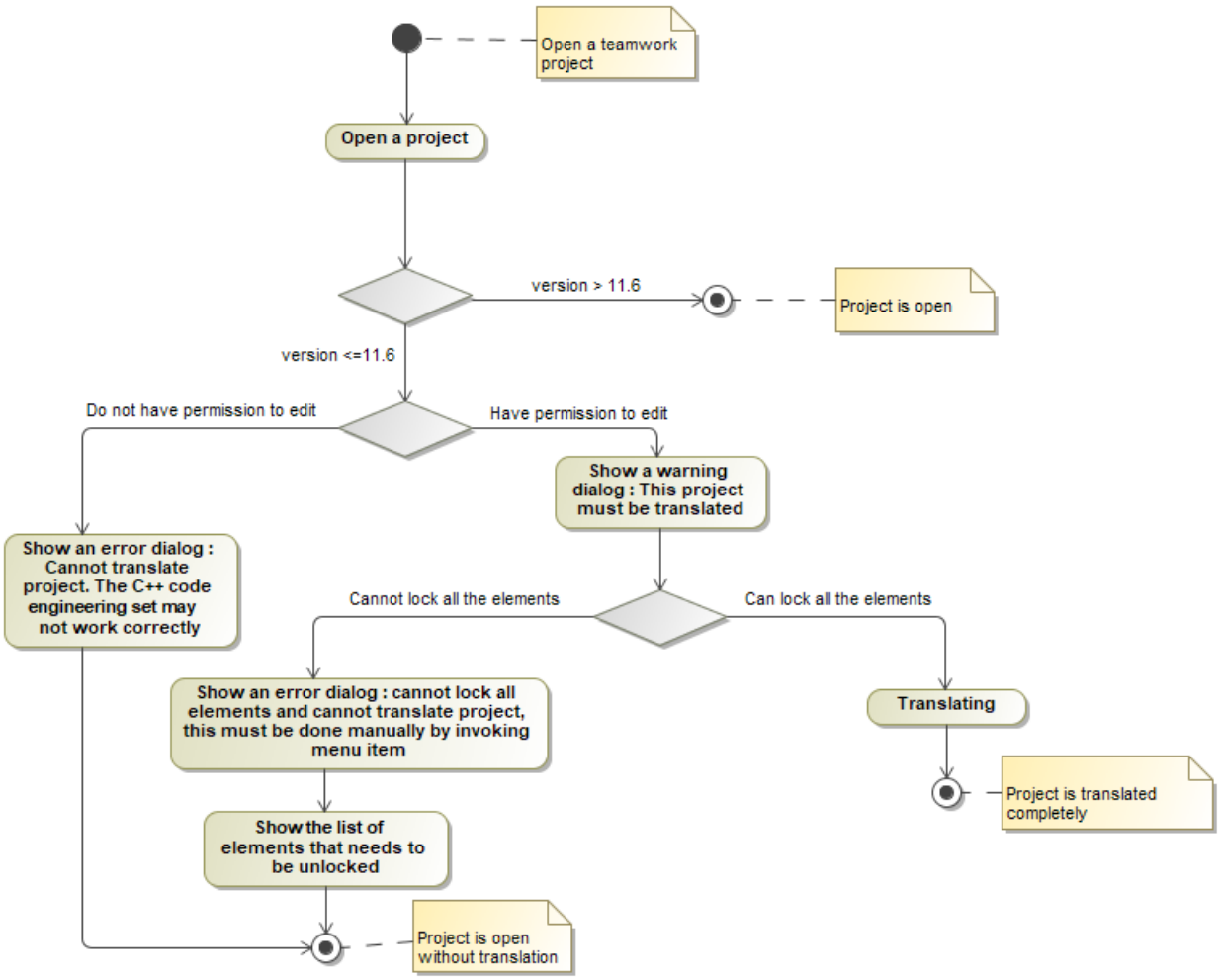
Some projects that use C++ language properties or C++ profile or have type modifier must be translated with version 11.6 (or earlier) of a MagicDraw project, as shown below.

Open Local Project



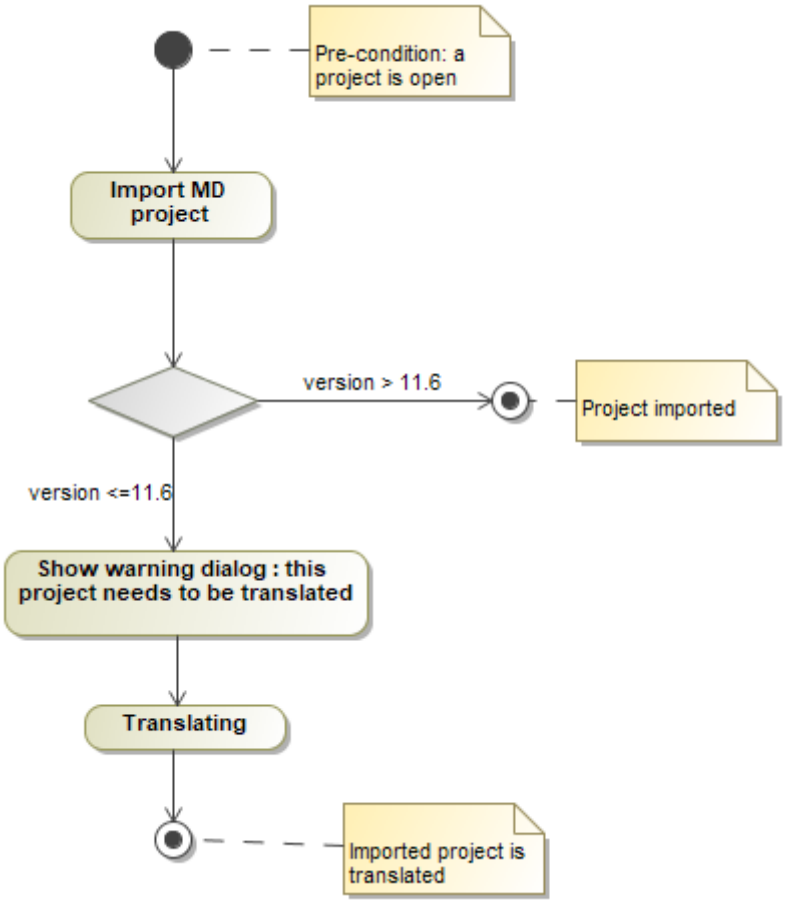
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Open Teamwork Project

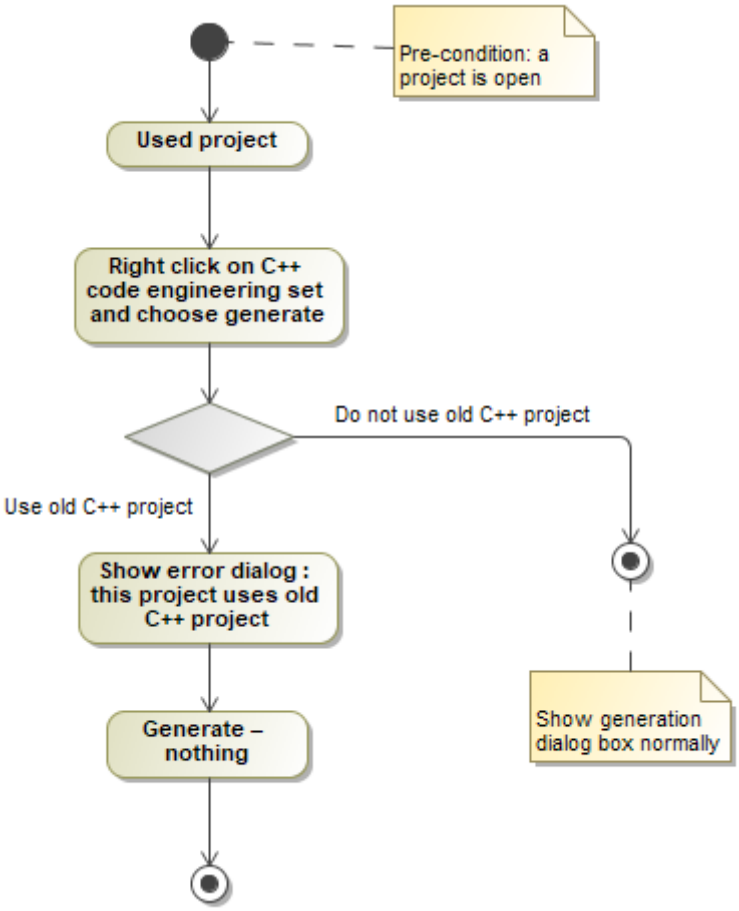


Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Import MagicDraw Project

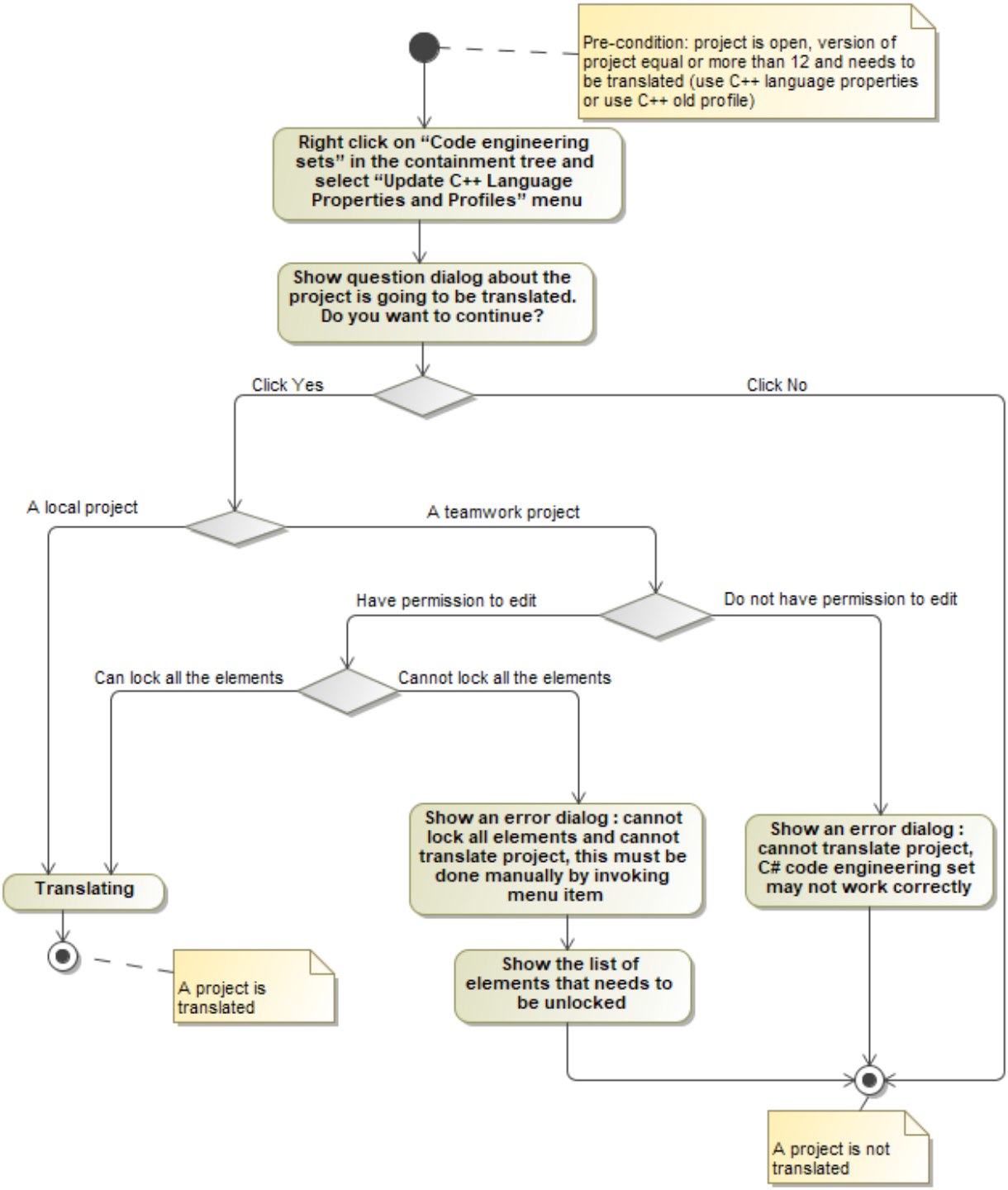


Used Project



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Update C++ Language Properties and Profiles



Related pages

- Conversion from an Earlier Project Version(see page 2511)
-

Translation Activity Diagram(see page 2512)
- Language Properties(see page 2533)
- Conversion with Array Type Modifiers(see page 2544)
- Stereotypes from Old Project Version(see page 2547)
- Thrown exception tag value translation(see page 2550)
- Constructor and Destructor Name From Old Project Versions(see page 2551)
- Data Type From Old Project Versions(see page 2553)

- Code Engineering(see page 2398)
  - Code Engineering Sets(see page 2399)
  - Generating Code(see page 2403)
  - Reverse Options(see page 2406)
  - Global options for Code Engineering(see page 2415)
  - Files of Properties(see page 2426)
  - Java Code Engineering(see page 2427)
  - C++ Code Engineering(see page 2454)
  - C# Code Engineering(see page 2622)
  - CORBA IDL Mapping To UML(see page 2680)
  - WSDL(see page 2683)

Language Properties

In versions earlier than MagicDraw 11.6, language properties are stored in a specific format; however, beginning with MagicDraw version 12, language properties are moved to a stereotype's tag value.

In this section:

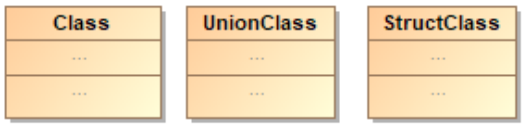
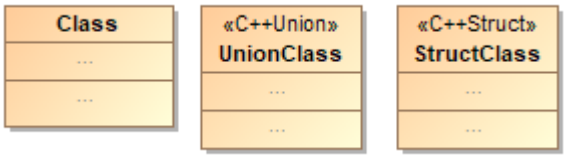
- Class Language Properties(see page 2517)
- Generalization Language Properties(see page 2517)
- Operation Language Properties(see page 2517)

Related Pages:

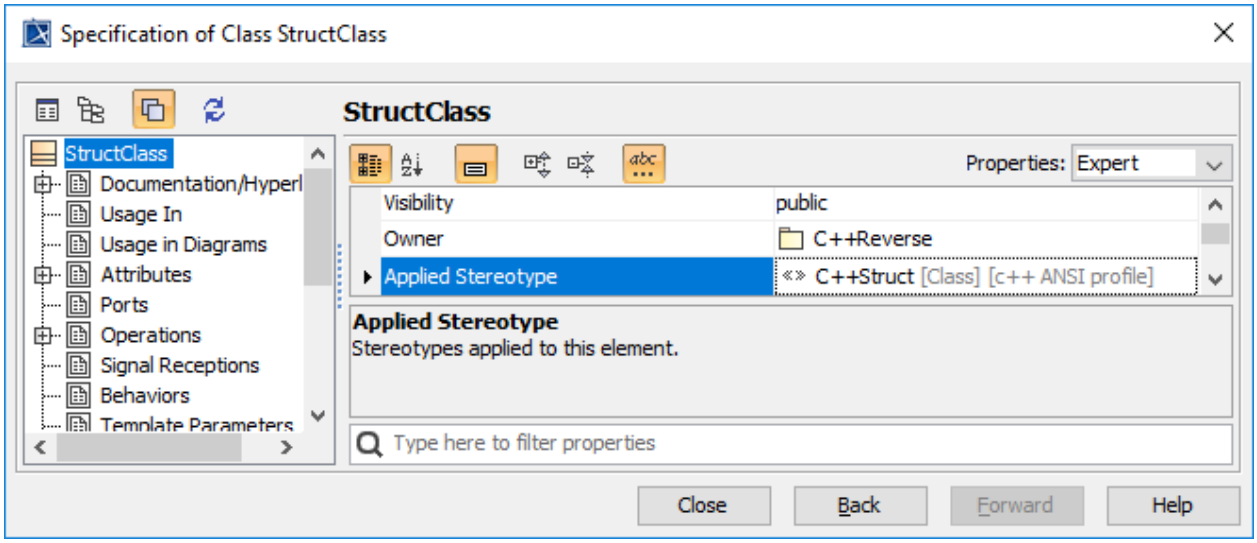
- Code Engineering(see page 2398)
  - Code Engineering Sets(see page 2399)
  - Generating Code(see page 2403)
  - Reverse Options(see page 2406)
  - Global options for Code Engineering(see page 2415)
  - Files of Properties(see page 2426)
  - Java Code Engineering(see page 2427)
  - C++ Code Engineering(see page 2454)
  - C# Code Engineering(see page 2622)
  - CORBA IDL Mapping To UML(see page 2680)
  - WSDL(see page 2683)

Class Language Properties

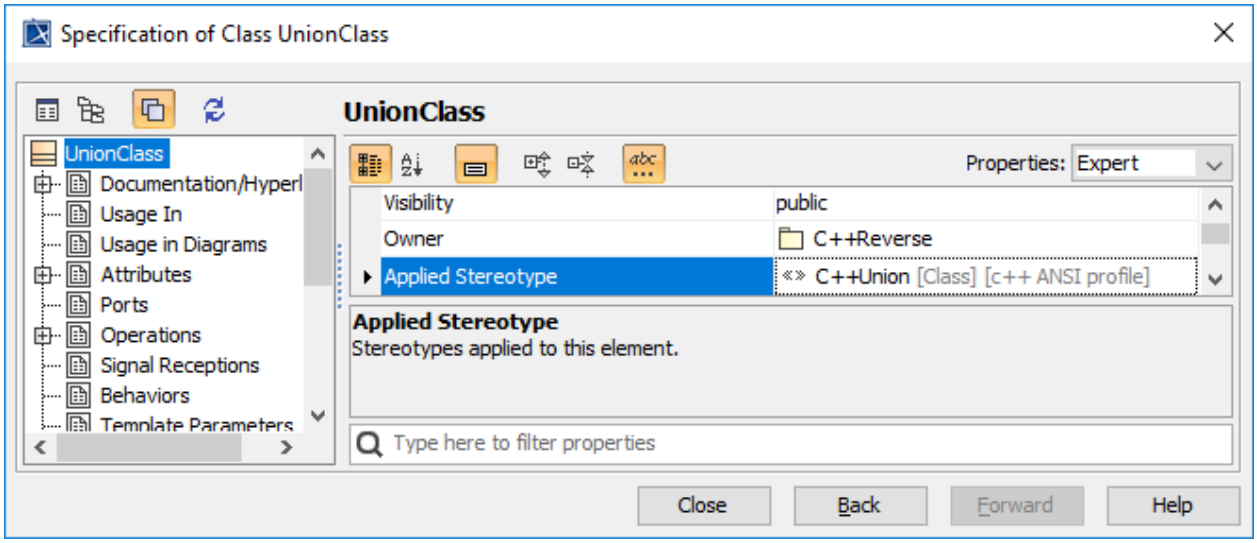
There are class, struct and union class key in Class Language Properties.

Class - Class key	
Old Value	**Translation**
	
`class`	no changes
`struct`	Apply the «C++Struct» stereotype.
`union`	Apply the «C++Union» stereotype.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



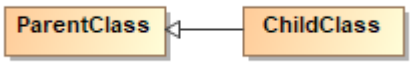
Applied stereotype «C++Struct» stereotype



Applied stereotype «C++Union» stereotype

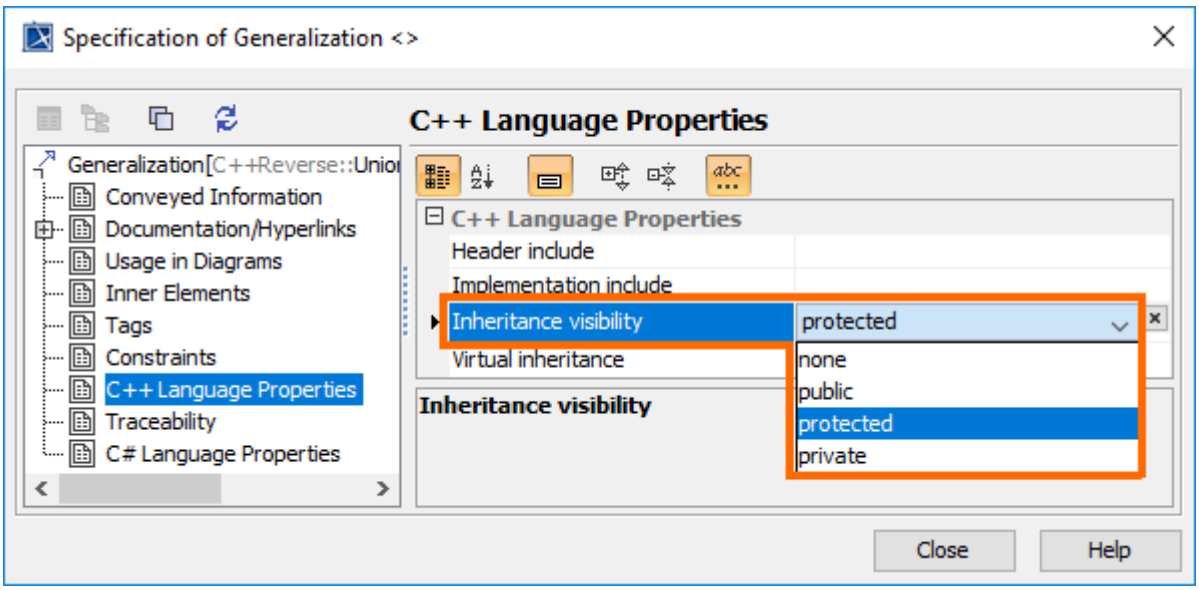
Generalization Language Properties

This is the example of generalization for childClass class that extends from ParentClass class.



Generalization Example in Class Diagram

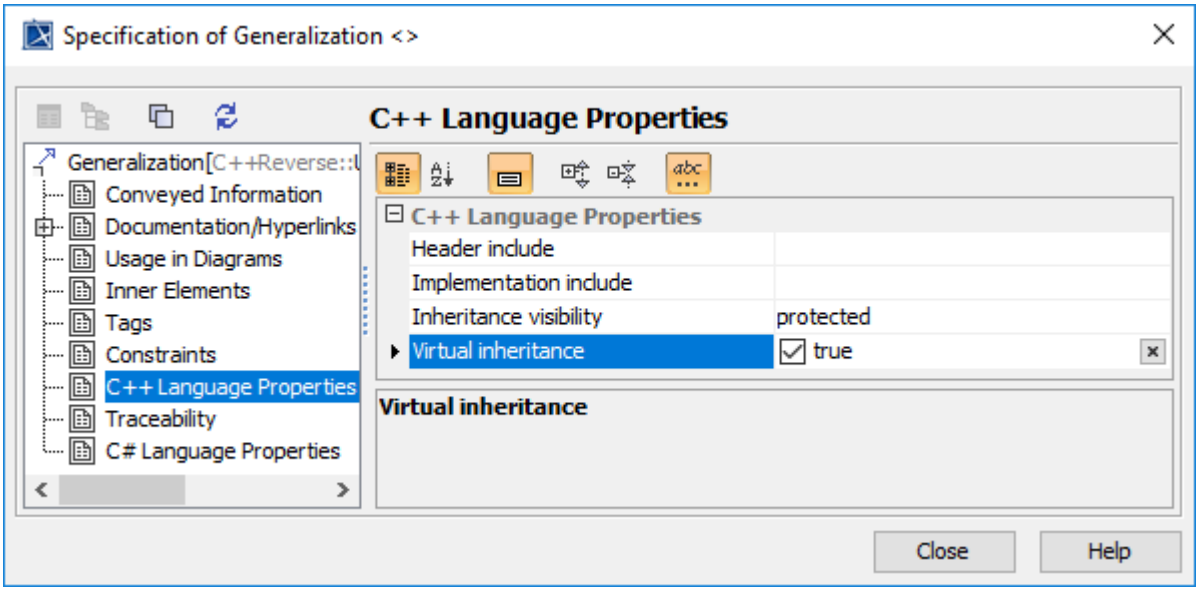
There are Inheritance type and Virtual modifier in the Generalization C++ Language Properties that need to be translated and applied to the «C++Generalization» stereotype.



Generalization Inheritance visibility

Generalization - Inheritance visibility 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.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Generalization Virtual inheritance modifier

Generalization - Virtual inheritance 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.

Operation Language Properties

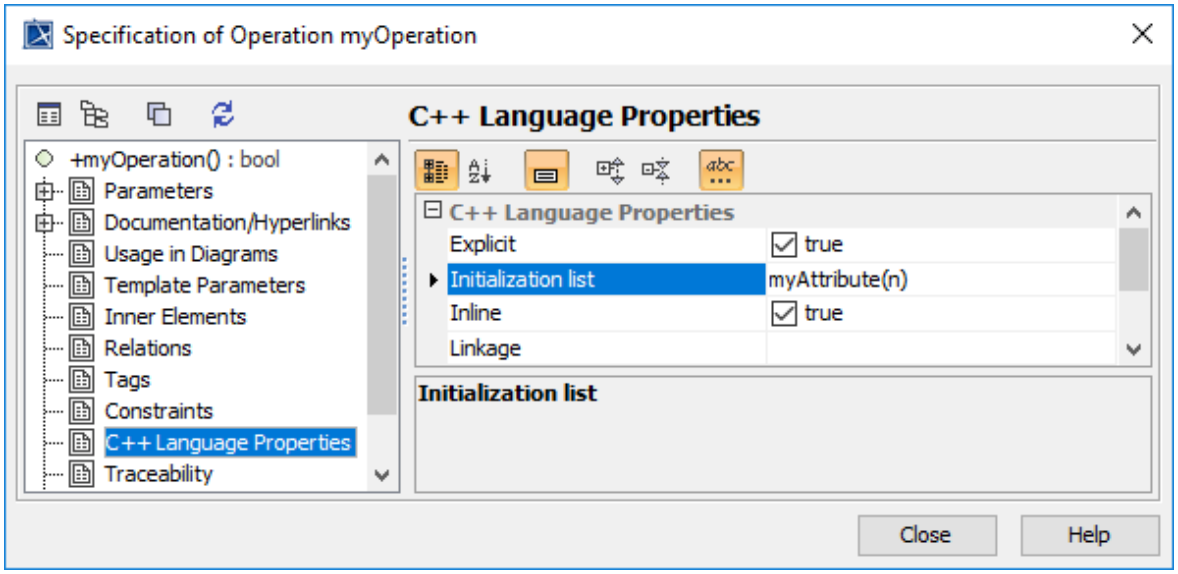
This example is OperationClass class that has operation named OperationClass() which is constructor and operation named myOperation. The Model that is being shown in the figure below is a translation. There are Inline modifier and Virtual modifier in Operation Language Properties that need to be translated and apply the «C++Operation» stereotype. There are Initialization list and Explicit modifier in Operation Language Properties that need to be translated and apply the «C++Constructor» stereotype.

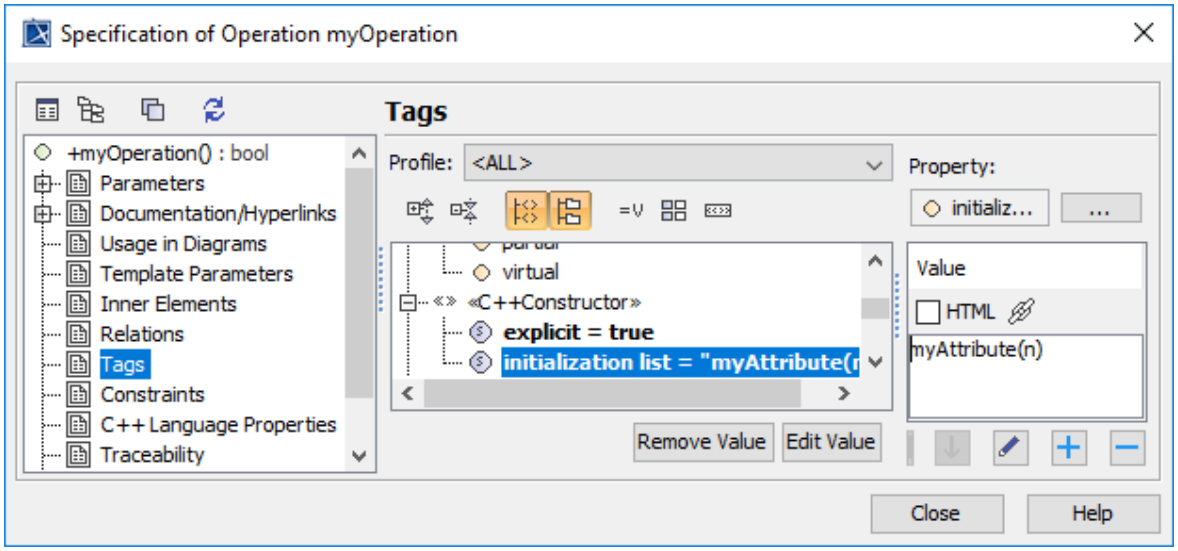
> **Constructor**
> Initialization list and Explicit modifier will be translated when it was set in Constructor. The Constructor is an operation that has the same name as its owner or applies the «constructor» stereotype in UML Standard Profile.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

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).

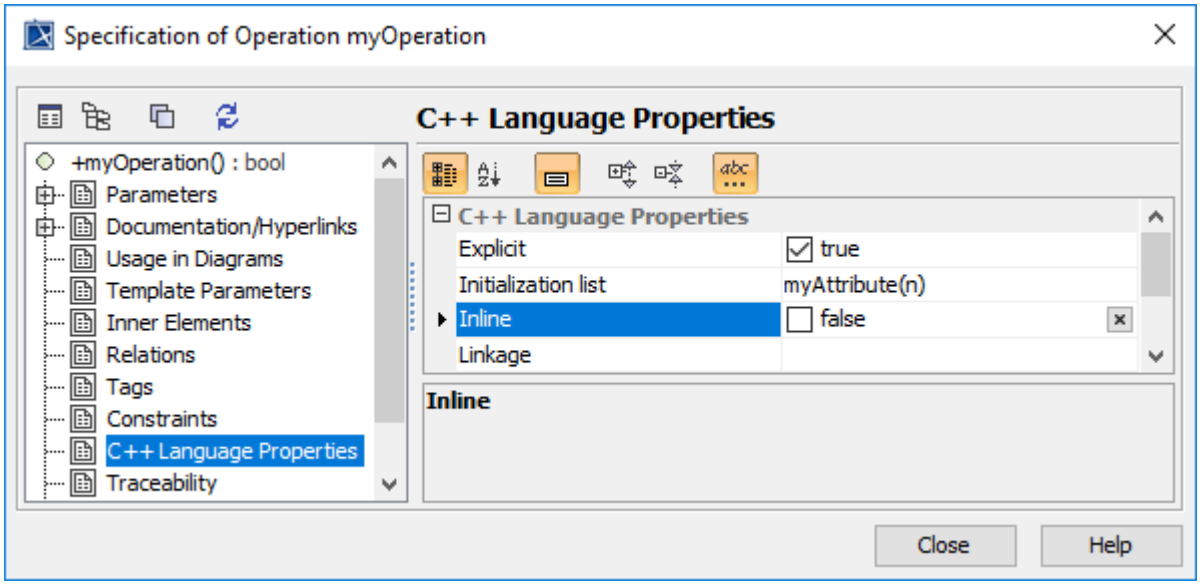




Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Operation - Inline Modifier

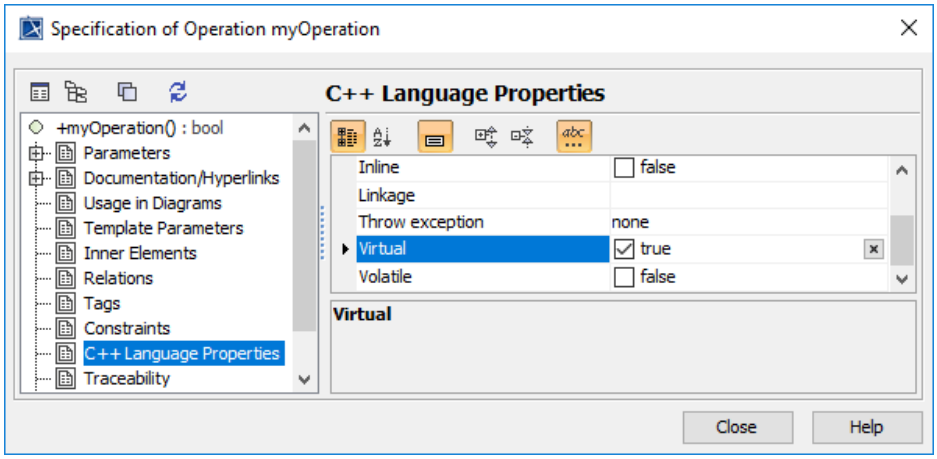
Old Value	Translation
`not inline`	Apply the «C++Operation» stereotype and set inline tag value to false.
`inline`	Apply the «C++Operation» stereotype and set inline tag value to true.



Operation - Virtual Modifier

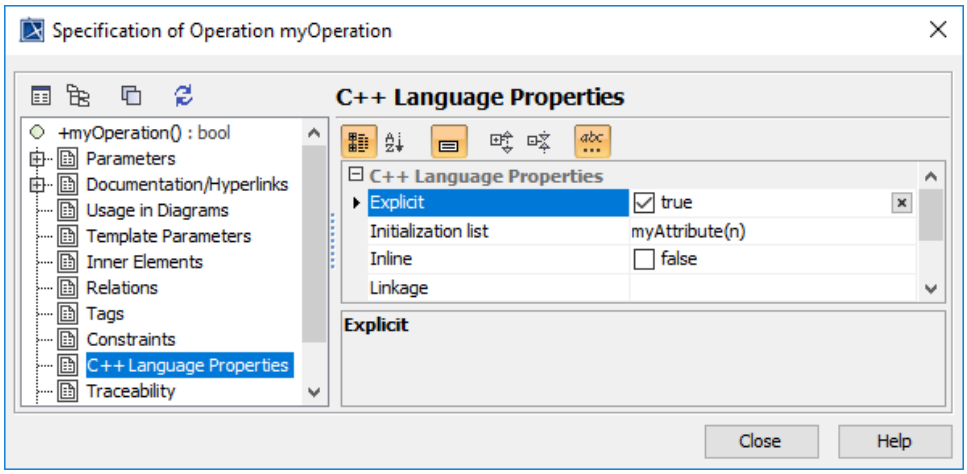
Old Value	Translation
`not virtual`	Apply the «C++Operation» stereotype and set virtual tag value to false.
`virtual`	Apply the «C++Operation» stereotype and set virtual tag value to true.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



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.



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.

Figure 68 -- Return value changeability Example in Class Diagram

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

Figure 69 -- Translated Return value changeability in Class Diagram
The Return value changeability Language Properties is being shown in the figure below.

Old value	Translation
none	no change.
`const, operation doesn't has parameter.`	Create one return type parameter and set Type Modifier to const $.
`const, operation has parameter but does not have return type parameter.`	Create one return type parameter and set Type Modifier to const $.
`const, operation has return type parameter.`	Set Type Modifier in return type parameter to const $.

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.

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.

Figure 73 -- Attribute Language Properties

Figure 74 -- Attribute Mutable

Attribute - Mutable	
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.

Figure 75 -- Attribute Volatile

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

Figure 76 -- Attribute Bit field

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 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Figure 77 -- Attribute Abbreviated Initializer

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.

Figure 78 -- Attribute Pointer to function

Attribute - Pointer to function	
Old value	**Translation**
`false`	no change.
`true`	no change.

Figure 79 -- Attribute Container

Attribute - Container	
Old value	**Translation**
`<empty>`	no change.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

`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, Com- pare>.
`set<Key, Compare>`	Apply the «C++Attribute» stereotype and set container tag value to set<Key, Compare>.

Conversion with Array Type Modifiers

With 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*.

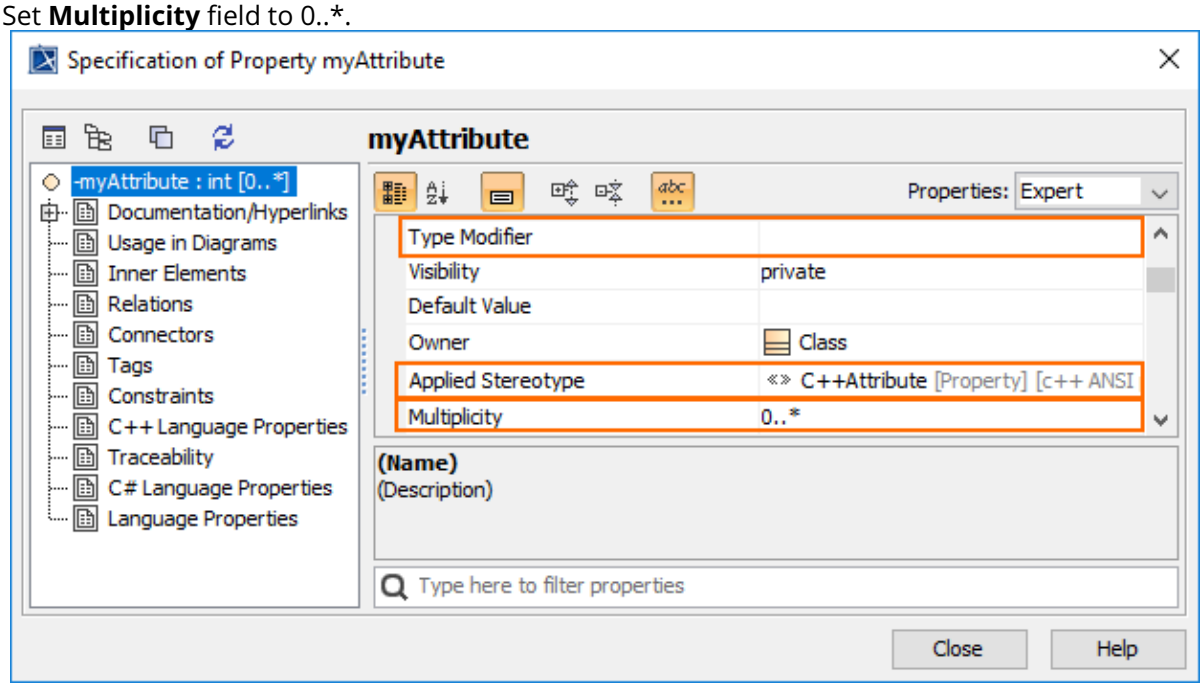
Type Modifier conversion in Attribute

Example of Type Modifier conversion in Attribute

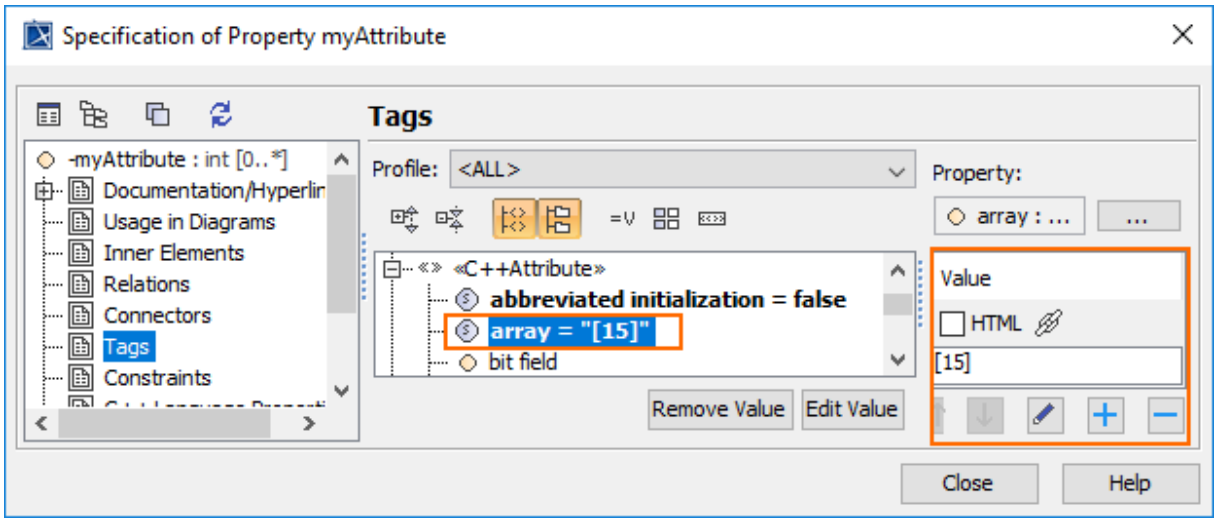
To convert a Type modifier in Attribute(see page 672):

1. Open the Attribute specification window.

2. Remove *[15]* from **Type Modifier** field.
3. Apply the «C++Attribute» stereotype.
4. Set **Multiplicity** field to 0..*.



5. Go to Tags and set Array tag value to *[15]*.



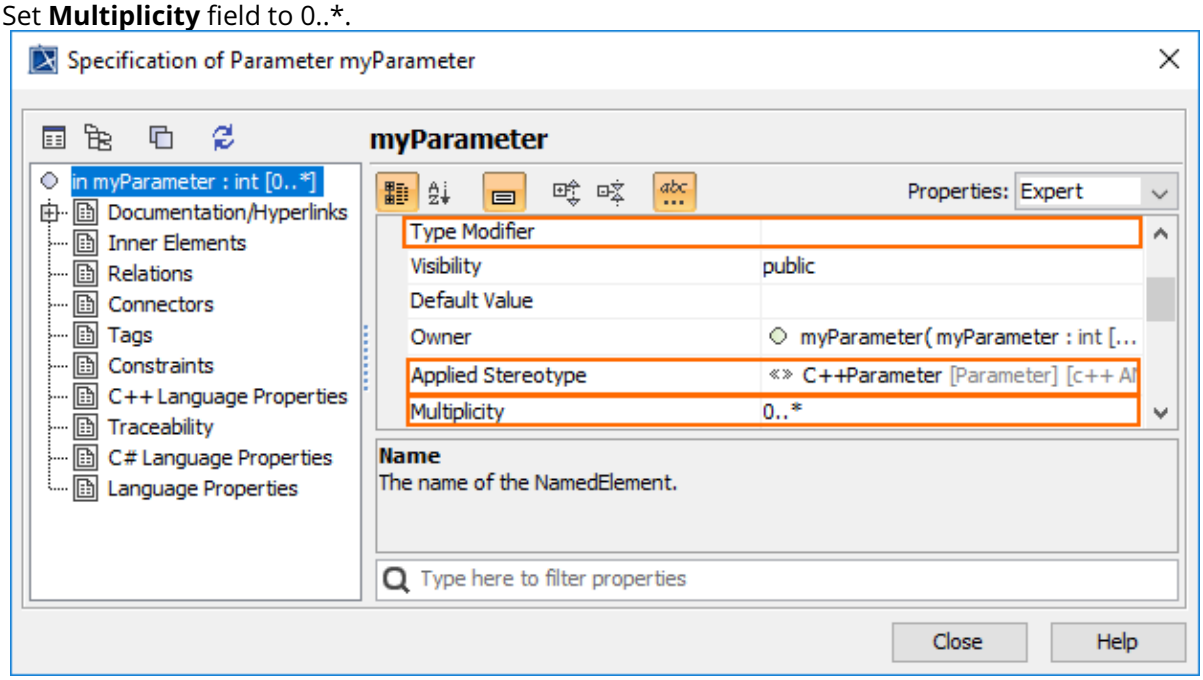
Type Modifier conversion in Parameter

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

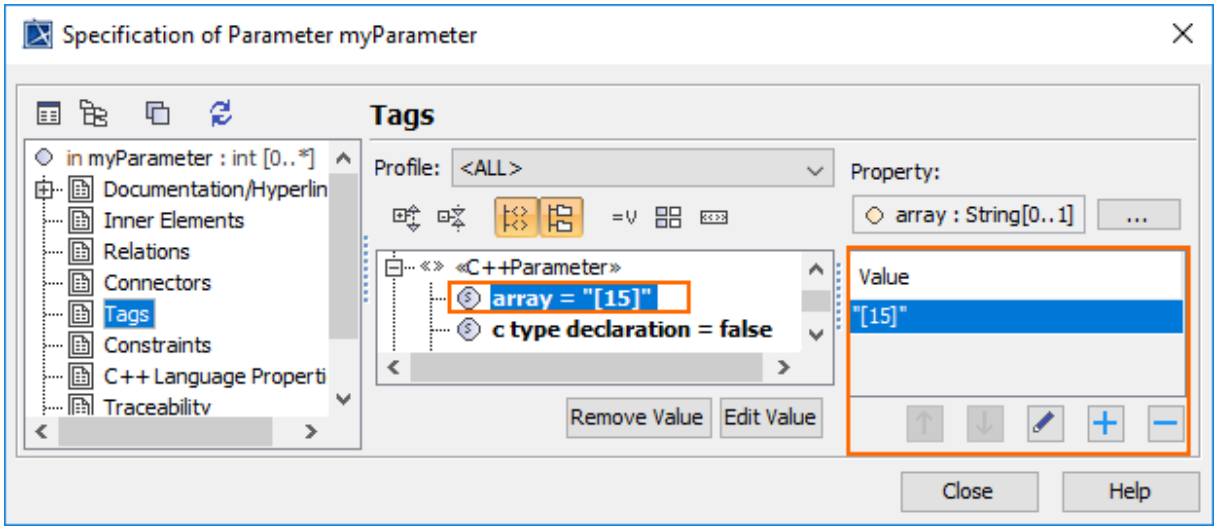
Example of Type Modifier conversion in Parameter

To convert a Type modifier in Parameter:

1. Open the Parameter specification window.
2. Remove *[15]* from **Type Modifier** field.
3. Apply the «C++Parameter» stereotype.
4. Set **Multiplicity** field to 0..*.



5. Go to Tags and set Array tag value to *[15]*.



Related pages:

- Code Engineering(see page 2398)
  - Code Engineering Sets(see page 2399)
  - Generating Code(see page 2403)
  -

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Reverse Options(see page 2406)
- Global options for Code Engineering(see page 2415)
- Files of Properties(see page 2426)
- Java Code Engineering(see page 2427)
- C++ Code Engineering(see page 2454)
- C# Code Engineering(see page 2622)
- CORBA IDL Mapping To UML(see page 2680)
- WSDL(see page 2683)

Stereotypes from Old Project Version

«C++EnumerationLiteral»

This example shows an EnumClass class with a literal value named literal. This literal value applies the «C++EnumerationLiteral» stereotype and sets C++Initializer tag value to 0.

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

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

Figure 87 -- «C++EnumerationLiteral» stereotype

The C++Initializer tag value is 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 the value tag value to 0. Remove the «C++EnumerationLiteral» stereotype and C++Initializer tag value from Enumeration Literal.

The Model shown in the figure below is a translation.

Figure 89 -- Translated Enumeration Literal in Class Diagram

«C++Namespace»

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

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

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

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

Figure 91 -- «C++Namespace» stereotype

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

Figure 92 -- unique namespace name tag value

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

«C++Typename»

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

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 the type keyword tag value to typename. If not, the translation will apply the «C++TemplateParameter» stereotype and set type keyword tag value to class.

Figure 94 -- «C++Typename» stereotype

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

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.
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 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

«constructor» and «destructor» in UML Standard Profile

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

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

«constructor»

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

Figure 97 -- «constructor» stereotype

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).

«destructor»

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

Figure 98 -- «destructor» stereotype

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).

Related Pages:

- Code Engineering(see page 2398)
  - Code Engineering Sets(see page 2399)
  - Generating Code(see page 2403)
  - Reverse Options(see page 2406)
  - Global options for Code Engineering(see page 2415)
  - Files of Properties(see page 2426)
  - Java Code Engineering(see page 2427)
  - C++ Code Engineering(see page 2454)
  - C# Code Engineering(see page 2622)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

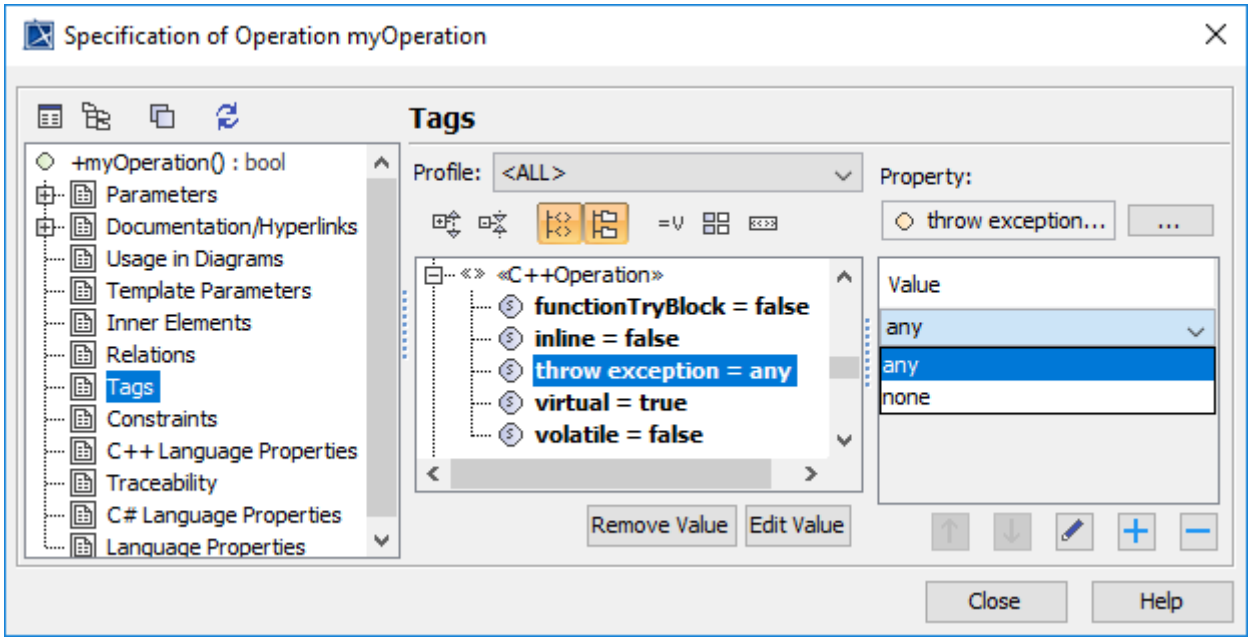
CORBA IDL Mapping To UML(see page 2680)
- WSDL(see page 2683)

- Conversion from an Earlier Project Version(see page 2511)
  - Translation Activity Diagram(see page 2512)
  - Language Properties(see page 2517)
  - Conversion with Array Type Modifiers(see page 2528)
  - Stereotypes from Old Project Version(see page 2531)
  - Thrown exception tag value translation(see page 2550)
  - Constructor and Destructor Name From Old Project Versions(see page 2551)
  - Data Type From Old Project Versions(see page 2553)

Thrown exception tag value translation

The stereotype «C++Operation» can be applied to an Operation(see page 2534).

In the modeling tool stereotype, «C++Operation» must apply all operations; the default throw exception tag value is *any*.



C++ThrownExceptions tag value

Old value	Translation
Operation does not apply the «C++Operation» stereotype in an old profile (C++ Profile).	Apply the «C++Operation» stereotype in a new profile (c++ ANSI profile) and set the throw exception tag value to any.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

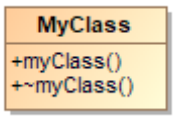
Old value	Translation
Operation applies the «C++Operation» stereotype in an old profile (C++ Profile) and set the C++ThrownExceptions tag value to any.	Apply the «C++Operation» stereotype in a new profile (c++ ANSI profile) and set the throw exception tag value to any. Remove the «C++Operation» stereotype (old profile) and the C++ThrownExceptions tag value.
Operation applies the «C++Operation» stereotype in old profile (C++ Profile) and set the C++ThrownExceptions tag value to none.	Apply the «C++Operation» stereotype in a new profile (c++ ANSI profile) and set the throw exception tag value to none. Remove the «C++Operation» stereotype (old profile) and the C++ThrownExceptions tag value.

Related Pages:

- Conversion from an Earlier Project Version(see page 2511)
  - Translation Activity Diagram(see page 2512)
  - Language Properties(see page 2517)
  - Conversion with Array Type Modifiers(see page 2528)
  - Stereotypes from Old Project Version(see page 2531)
  - Thrown exception tag value translation(see page 2534)
  - Constructor and Destructor Name From Old Project Versions(see page 2535)
  - Data Type From Old Project Versions(see page 2535)
- Code Engineering(see page 2398)
  - Code Engineering Sets(see page 2399)
  - Generating Code(see page 2403)
  - Reverse Options(see page 2406)
  - Global options for Code Engineering(see page 2415)
  - Files of Properties(see page 2426)
  - Java Code Engineering(see page 2427)
  - C++ Code Engineering(see page 2454)
  - C# Code Engineering(see page 2622)
  - CORBA IDL Mapping To UML(see page 2680)
  - WSDL(see page 2683)

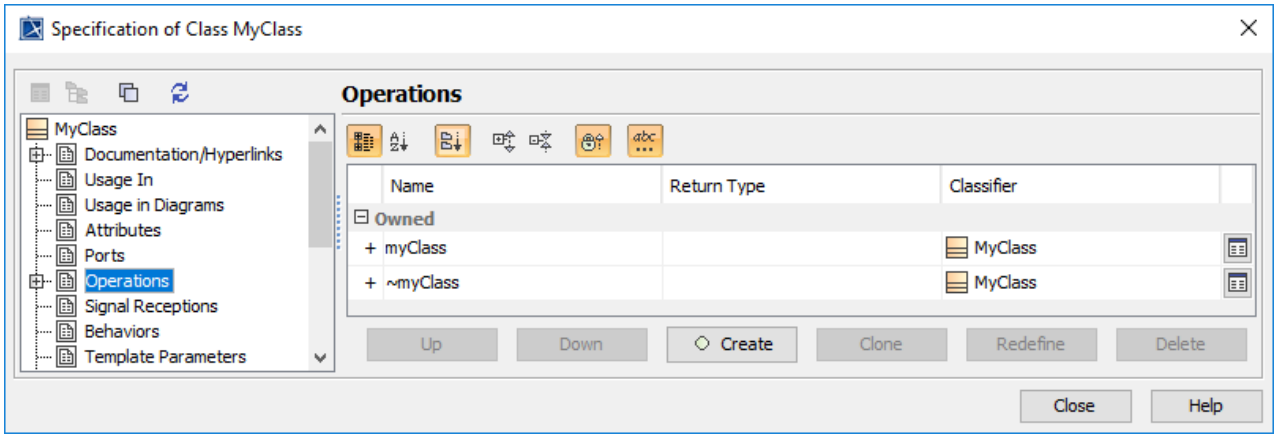
Constructor and Destructor Name From Old Project Versions

This example shows the Class *myClass* that has Operation *myClass()*, which is the constructor, and Operation ~myClass(), which is the destructor.



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example Constructor and Destructor in the Class diagram

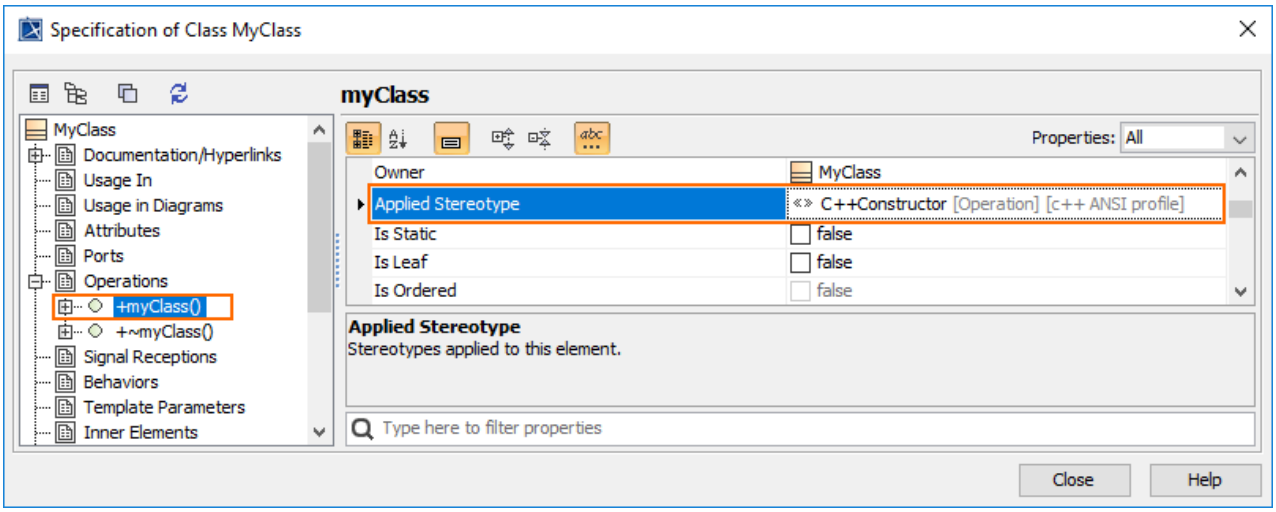


The Constructor and the Destructor operations.

To create a Constructor

---

1. Create a Class with an Operation.
2. Go to the Class Specification window.
3. Select Operation
4. Apply the stereotype «C++Constructor» for Operation.

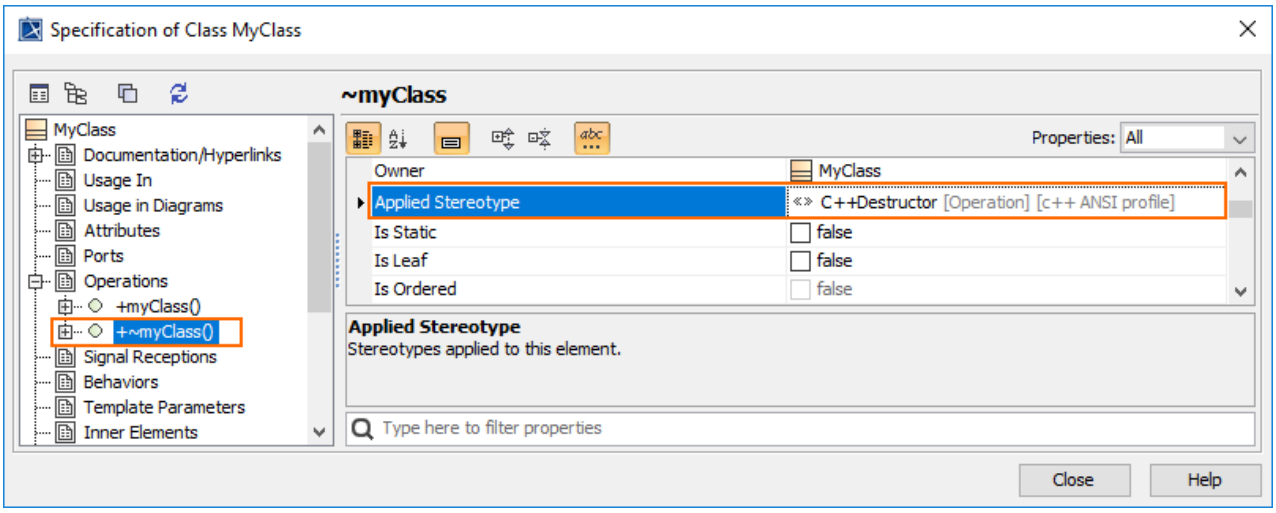


To create a Destructor

---

1. Create a Class with an Operation.
2. Go to the Class Specification window.

3. Select Operation
4. Apply the stereotype «C++Destructor» for Operation.



Related Pages:

- Conversion from an Earlier Project Version(see page 2511)
  - Translation Activity Diagram(see page 2512)
  - Language Properties(see page 2517)
  - Conversion with Array Type Modifiers(see page 2528)
  - Stereotypes from Old Project Version(see page 2531)
  - Thrown exception tag value translation(see page 2534)
  - Constructor and Destructor Name From Old Project Versions(see page 2535)
  - Data Type From Old Project Versions(see page 2553)
- Code Engineering(see page 2398)
  - Code Engineering Sets(see page 2399)
  - Generating Code(see page 2403)
  - Reverse Options(see page 2406)
  - Global options for Code Engineering(see page 2415)
  - Files of Properties(see page 2426)
  - Java Code Engineering(see page 2427)
  - C++ Code Engineering(see page 2454)
  - C# Code Engineering(see page 2622)
  - CORBA IDL Mapping To UML(see page 2680)
  - WSDL(see page 2683)

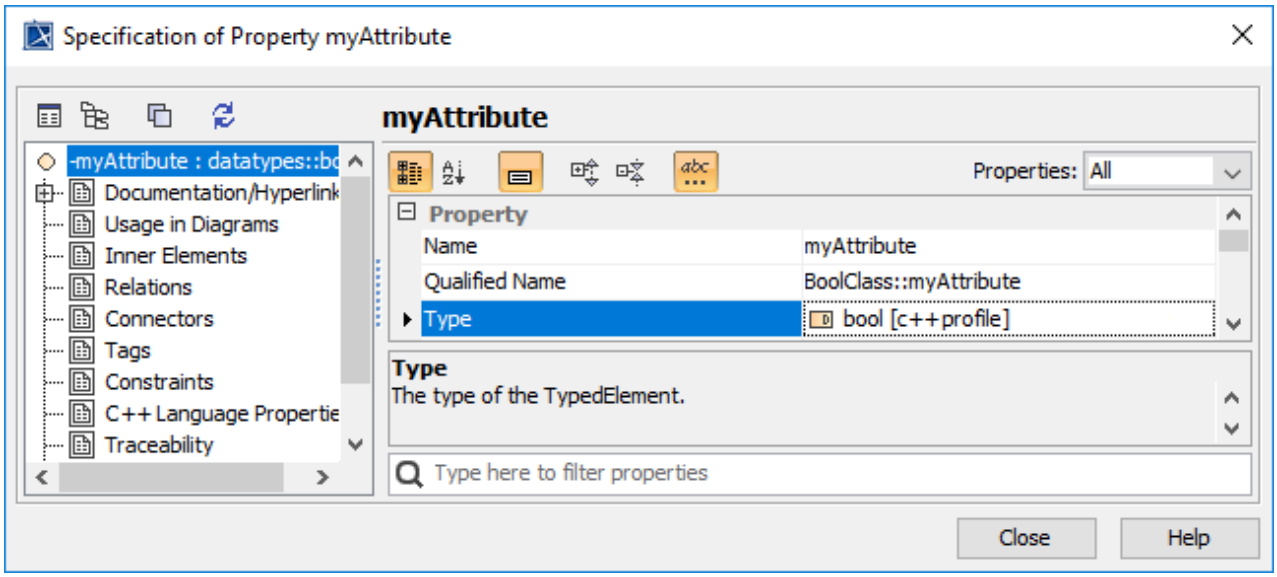
Data Type From Old Project Versions

bool

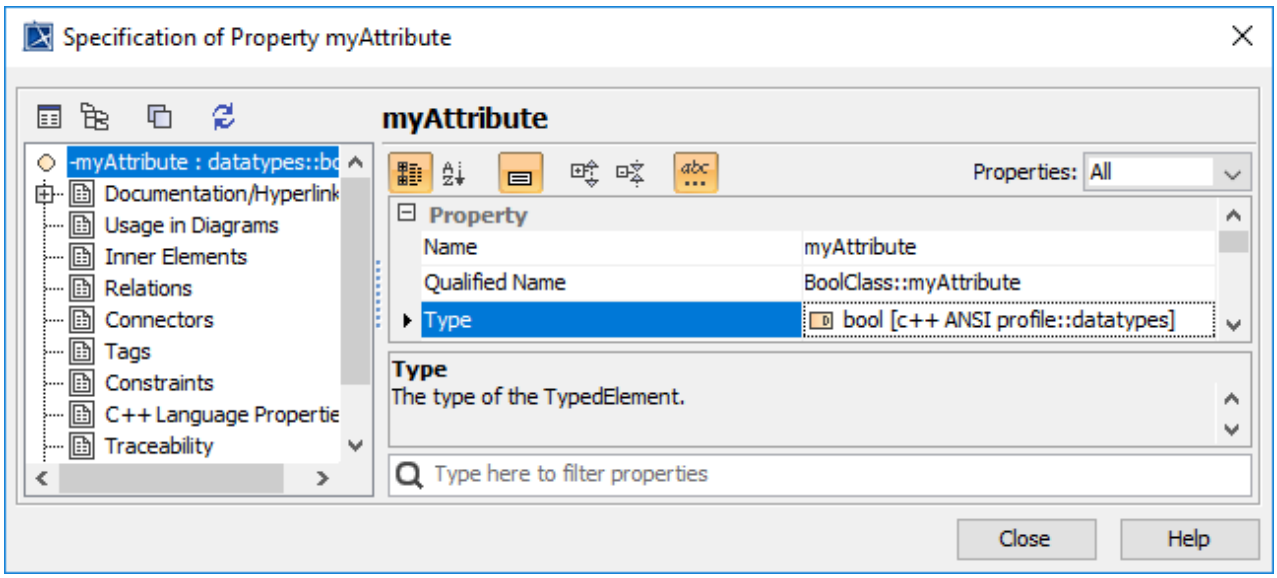
There are five cases of using the bool data type: Attribute type, Parameter type, Return type, Actual in Template Parameter Substitution and Default in Classifier Template Parameter. Our modeling tool will change the old bool data type to new bool data type in a new profile.

Attribute type

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



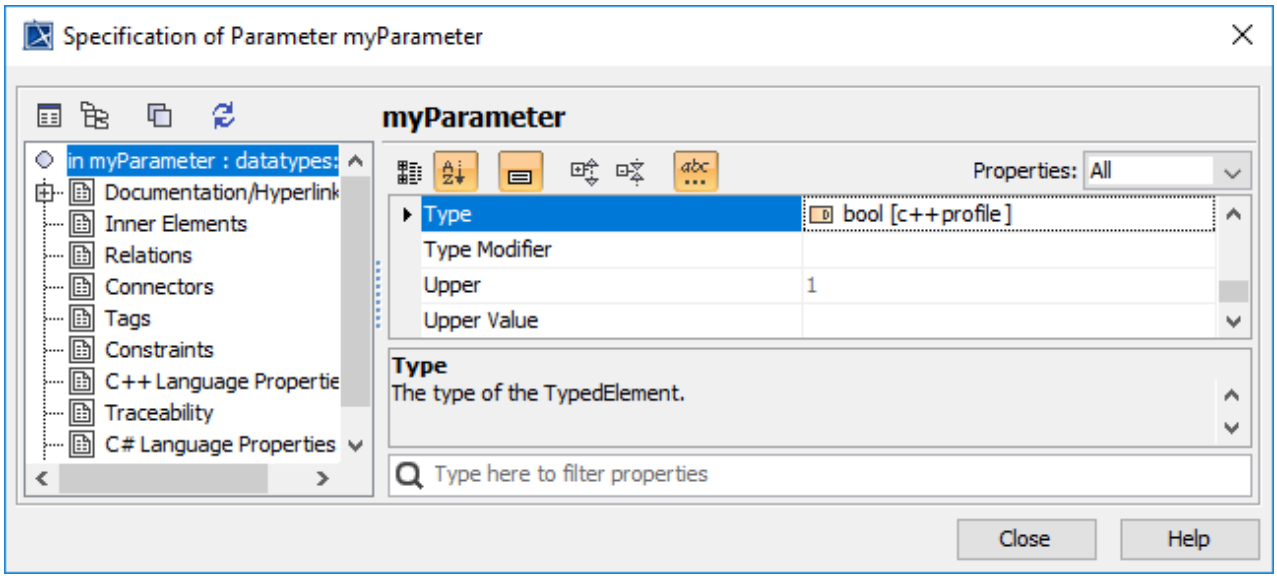
Old bool data type in old profile (C++ Profile).



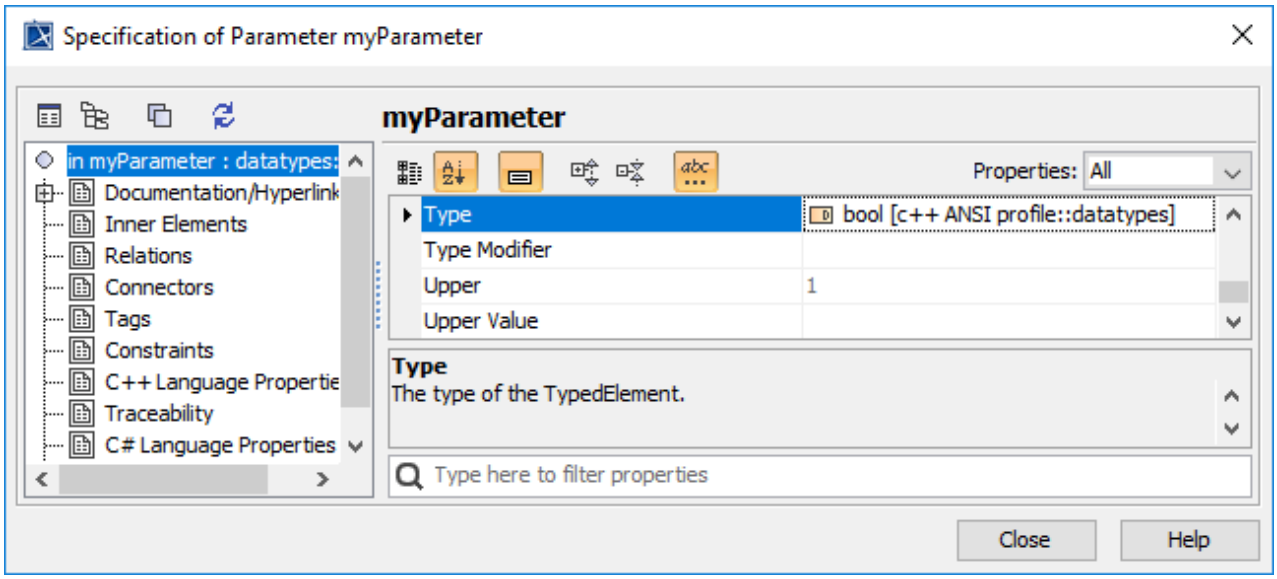
Change to the new bool data type in new profile (c++ ANSI profile).

Parameter type

The old and new bool data types in the Parameter type are shown in the figures below.



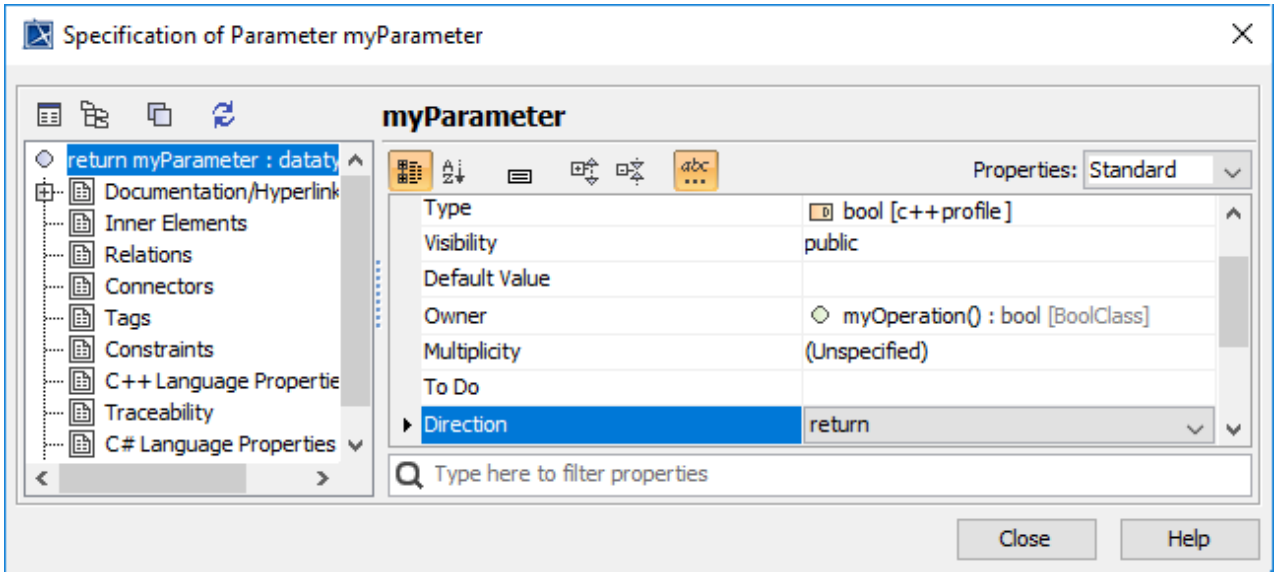
Old bool data type in old profile (C++ Profile)



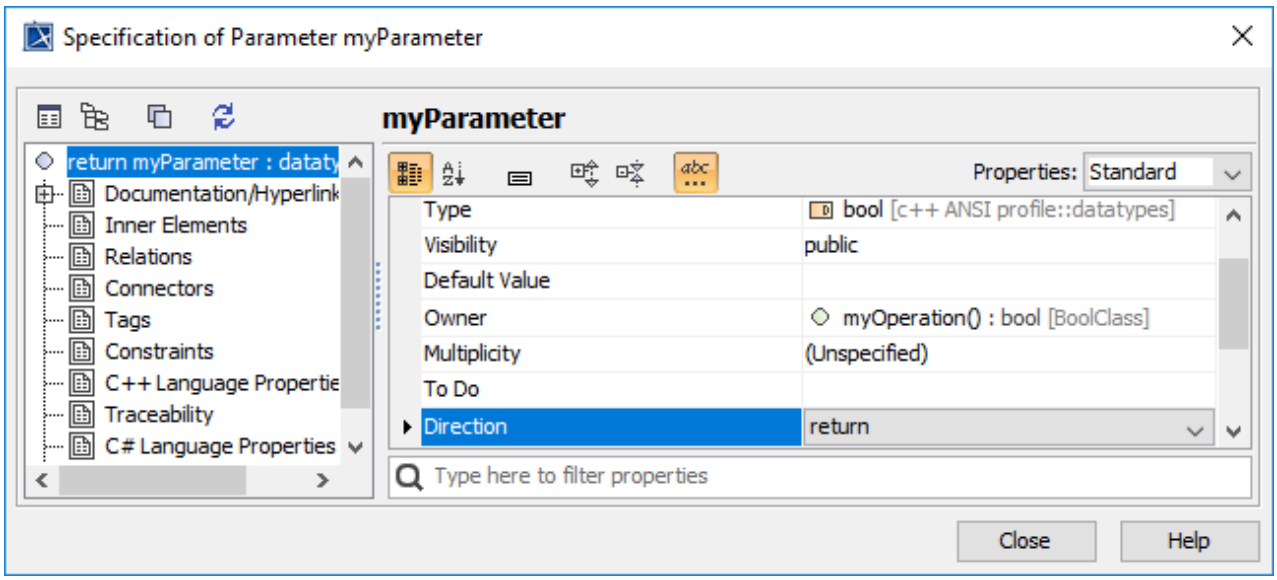
Change to the new bool data type in new profile (c++ ANSI profile).

Return type

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



Old bool data type in old profile (C++ Profile).



Change to the new bool data type in new profile (c++ ANSI profile).

Actual in Template Parameter Substitution

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

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).

Default in Classifier Template Parameter

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

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).

Related Pages:

- Conversion from an Earlier Project Version(see page 2511)
  - Translation Activity Diagram(see page 2512)
  - Language Properties(see page 2517)
  - Conversion with Array Type Modifiers(see page 2528)
  - Stereotypes from Old Project Version(see page 2531)
  - Thrown exception tag value translation(see page 2534)
  - Constructor and Destructor Name From Old Project Versions(see page 2535)
  - Data Type From Old Project Versions(see page 2537)
- Code Engineering(see page 2398)
  - Code Engineering Sets(see page 2399)
  - Generating Code(see page 2403)
  - Reverse Options(see page 2406)
  - Global options for Code Engineering(see page 2415)
  - Files of Properties(see page 2426)
  - Java Code Engineering(see page 2427)
  - C++ Code Engineering(see page 2454)
  - C# Code Engineering(see page 2622)
  - CORBA IDL Mapping To UML(see page 2680)
  - WSDL(see page 2683)

DSL Customization

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

Related Pages:

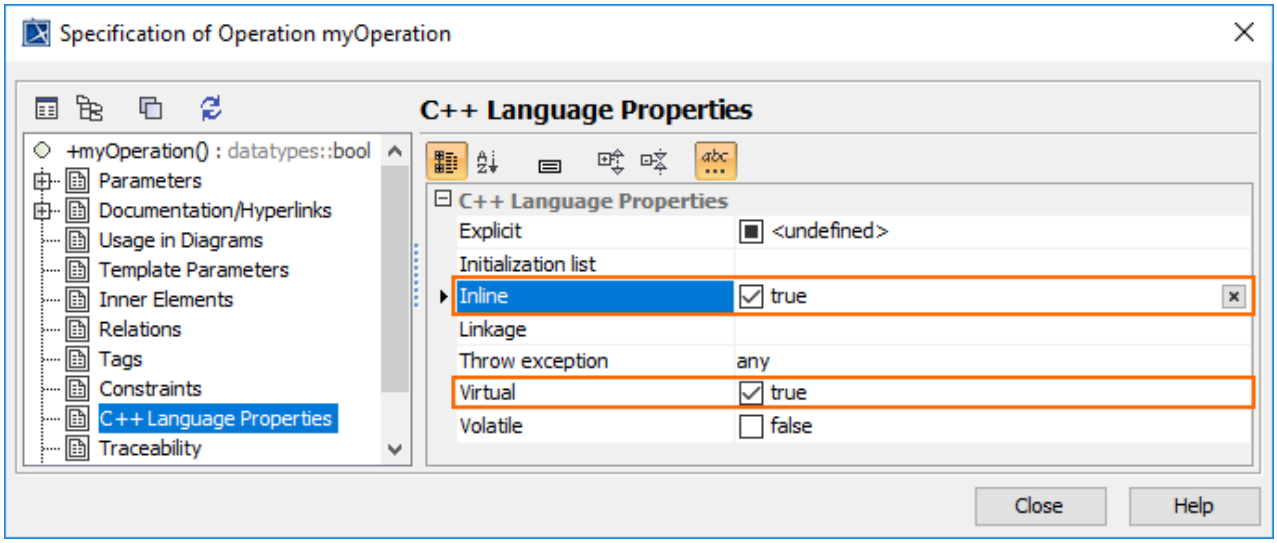
- DSL Customization(see page 2541)
  - Operation and Constructor(see page 2558)
  - Attribute in DSL Customization(see page 2560)
  - Generalization in DSL Customization(see page 2561)
  - Enumeration Literal in DSL Customization(see page 2561)
  - Namespace in DSL Customization(see page 2562)
  -

Template Parameter in DSL Customization(see page 2541)

- Code Engineering(see page 2398)
  - Code Engineering Sets(see page 2399)
  - Generating Code(see page 2403)
  - Reverse Options(see page 2406)
  - Global options for Code Engineering(see page 2415)
  - Files of Properties(see page 2426)
  - Java Code Engineering(see page 2427)
  - C++ Code Engineering(see page 2454)
  - C# Code Engineering(see page 2622)
  - CORBA IDL Mapping To UML(see page 2680)
  - WSDL(see page 2683)

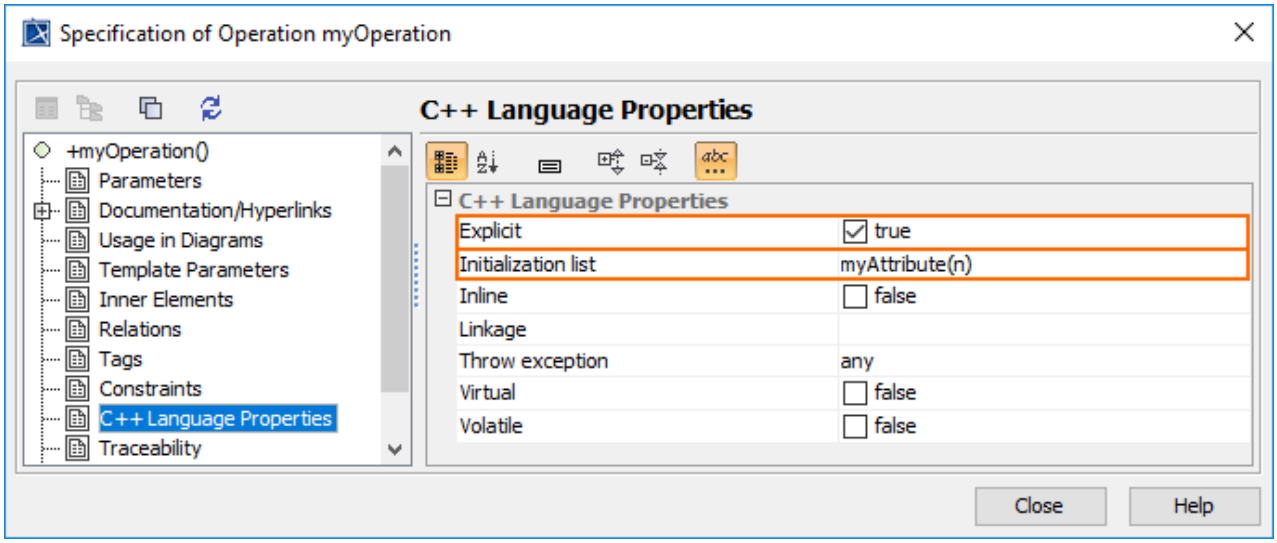
Operation and Constructor

The **Inline** and **Virtual** C++Language Properties translate to Operation tag values.



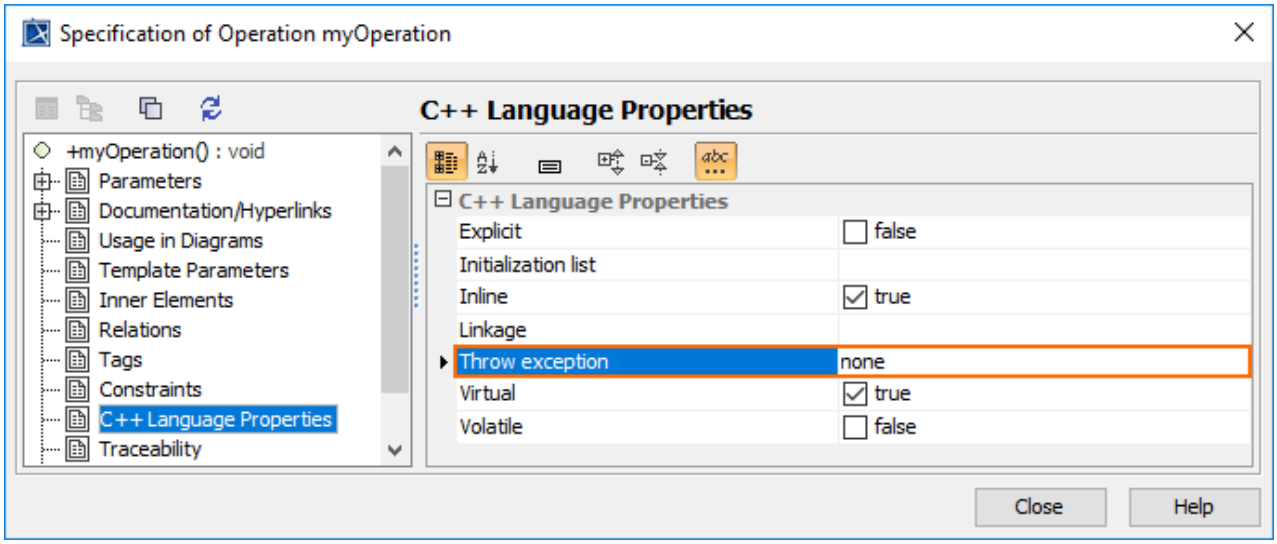
Tag values in Operation

The **Explicit** and **Initialization list** in C++Language Properties translate to Constructor tag values.



Tag values in Constructor

The C++ThrownExceptions tag value translates to Operation tag value.



Throw exception tag value in Operation

Related Pages:

- Code Engineering(see page 2398)
  - Code Engineering Sets(see page 2399)
  - Generating Code(see page 2403)
  - Reverse Options(see page 2406)
  - Global options for Code Engineering(see page 2415)
  - Files of Properties(see page 2426)
  - Java Code Engineering(see page 2427)
  - C++ Code Engineering(see page 2454)
  -

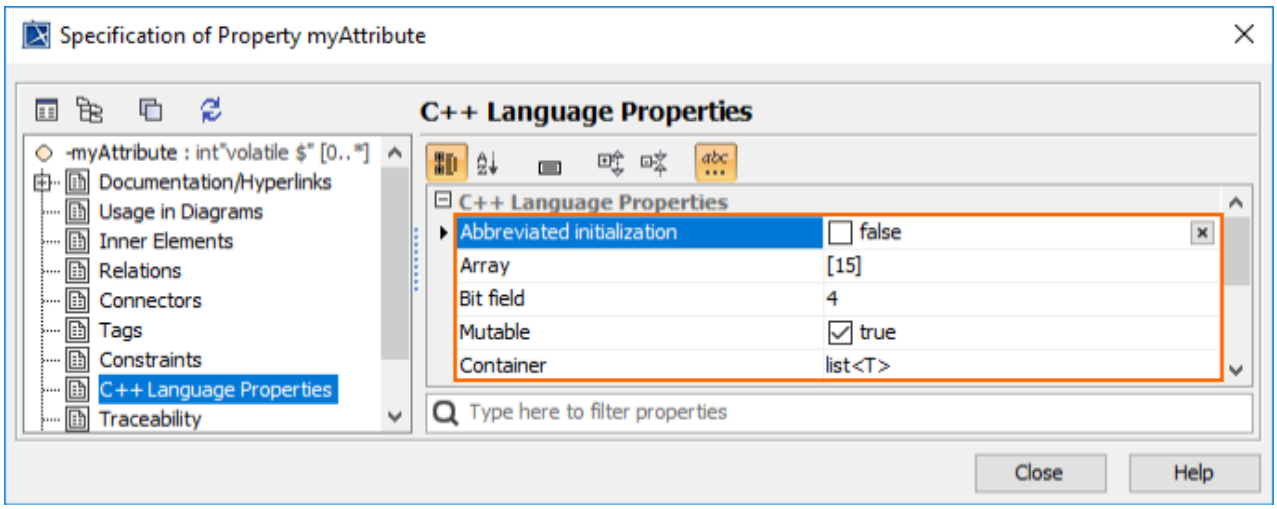
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

C# Code Engineering(see page 2622)
- CORBA IDL Mapping To UML(see page 2680)
- WSDL(see page 2683)

- DSL Customization(see page 2541)
  - Operation and Constructor(see page 2542)
  - Attribute in DSL Customization(see page 2560)
  - Generalization in DSL Customization(see page 2561)
  - Enumeration Literal in DSL Customization(see page 2561)
  - Namespace in DSL Customization(see page 2562)
  - Template Parameter in DSL Customization(see page 2562)

Attribute in DSL Customization

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



Tag values in Attribute

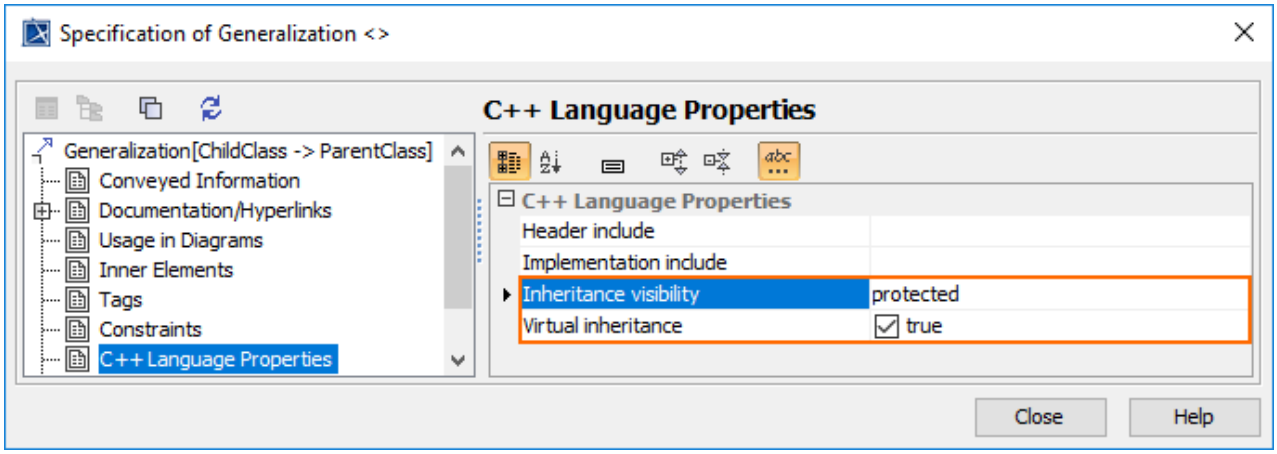
Related Pages:

- Code Engineering(see page 2398)
  - Code Engineering Sets(see page 2399)
  - Generating Code(see page 2403)
  - Reverse Options(see page 2406)
  - Global options for Code Engineering(see page 2415)
  - Files of Properties(see page 2426)
  - Java Code Engineering(see page 2427)
  - C++ Code Engineering(see page 2454)
  - C# Code Engineering(see page 2622)
  - CORBA IDL Mapping To UML(see page 2680)
  - WSDL(see page 2683)
- DSL Customization(see page 2541)
  - Operation and Constructor(see page 2542)
  - Attribute in DSL Customization(see page 2544)
  -

Generalization in DSL Customization(see page 2561)
- Enumeration Literal in DSL Customization(see page 2561)
- Namespace in DSL Customization(see page 2562)
- Template Parameter in DSL Customization(see page 2562)

Generalization in DSL Customization

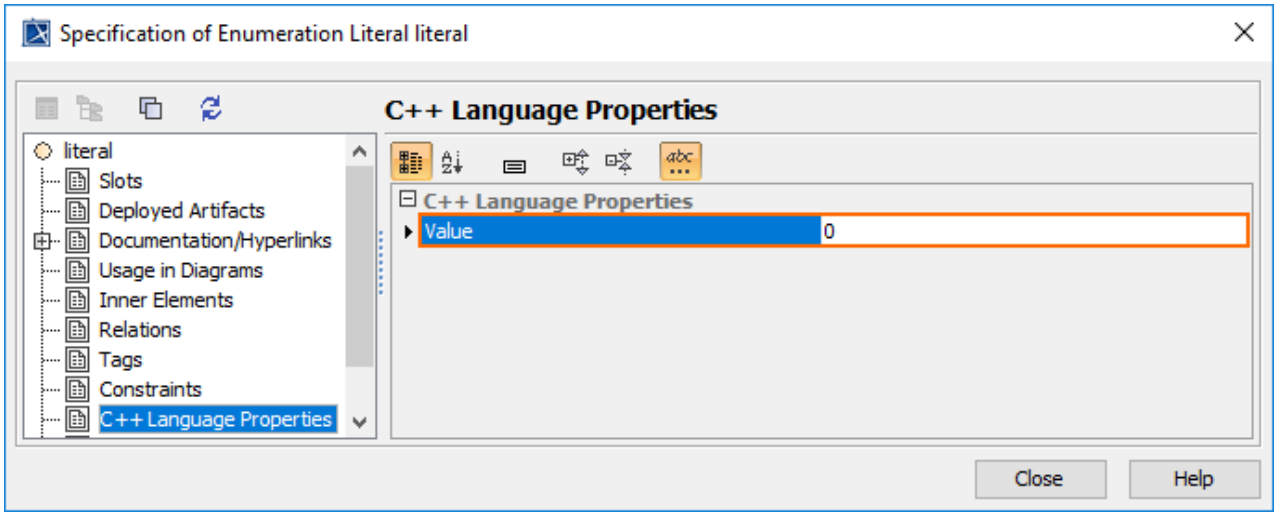
The Virtual inheritance and Inheritance Visibility in Generalization Language Properties translate to Generalization tag values.



Tag values in Generalization

Enumeration Literal in DSL Customization

Thefrom «C++EnumerationLiteral» stereotype translates to the Enumeration literal tag value.

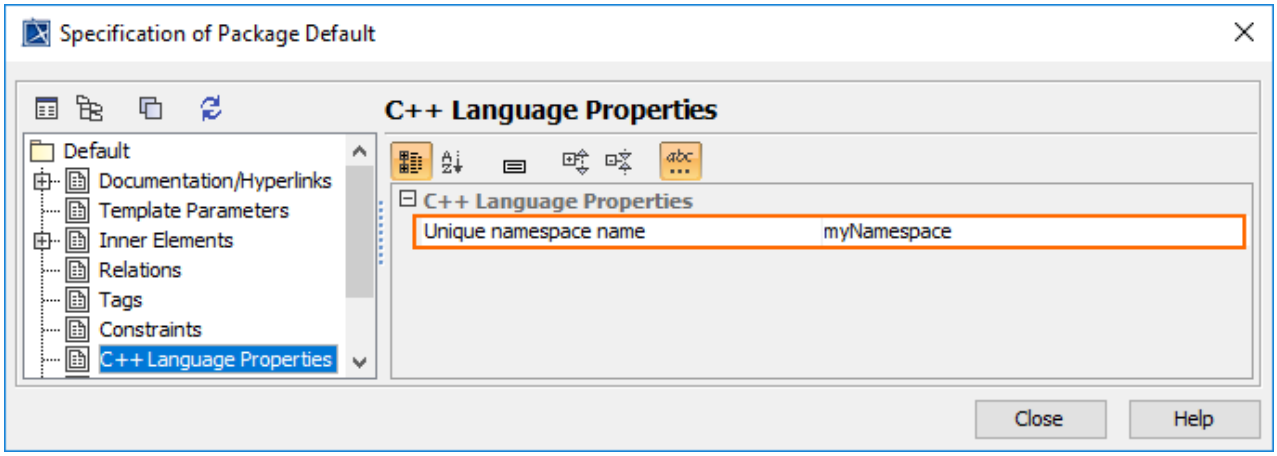


Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Tag value in Enumeration literal

Namespace in DSL Customization

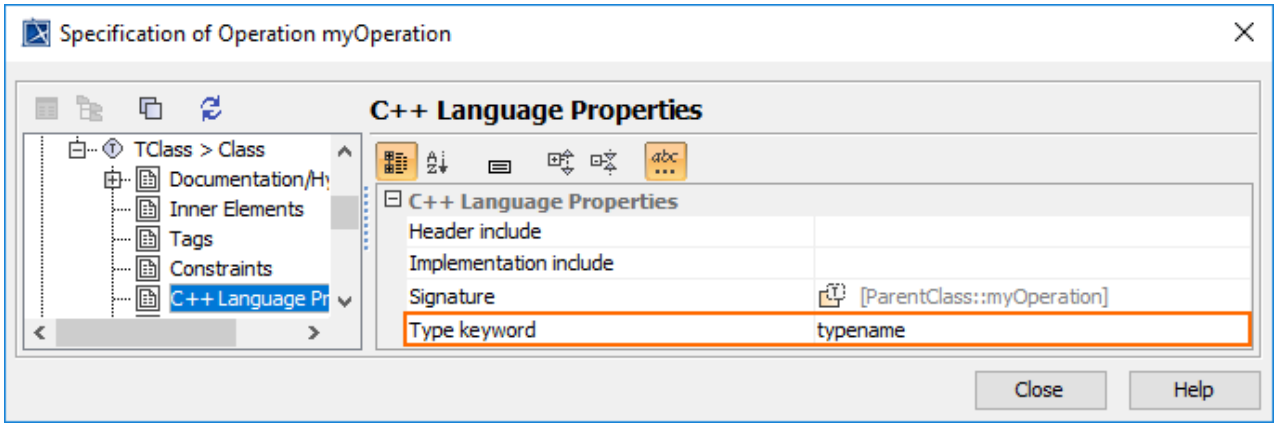
The value from «C++Namespace» stereotype translates to the Namespace tag value.



Tag value in Namespace

Template Parameter in DSL Customization

The value from the «C++Typename» stereotype translates to the Template parameter tag value.



Tag value in Template parameter

ANSI Profile Constraints

This chapter describes all constraints added by the ANSI profile.

Operation

- isQuery set to true only applies to a member operation (non global).
- virtual tag set to true only applies to a non static member operation.

Constructor

- operation with «C++Constructor» stereotype.
- Name should be the same as the owner's name.

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.
- There can be only one global class by namespace.
- Owner of a global class can only be a 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 a Classifier or Operation.
- «C++Typedef» class cannot contain an 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.

Project Constraints

This chapter describes how to set include paths for an include system file.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Include system file example

If you include any system file, you must set the system file path as well.

System file paths

To set the include path

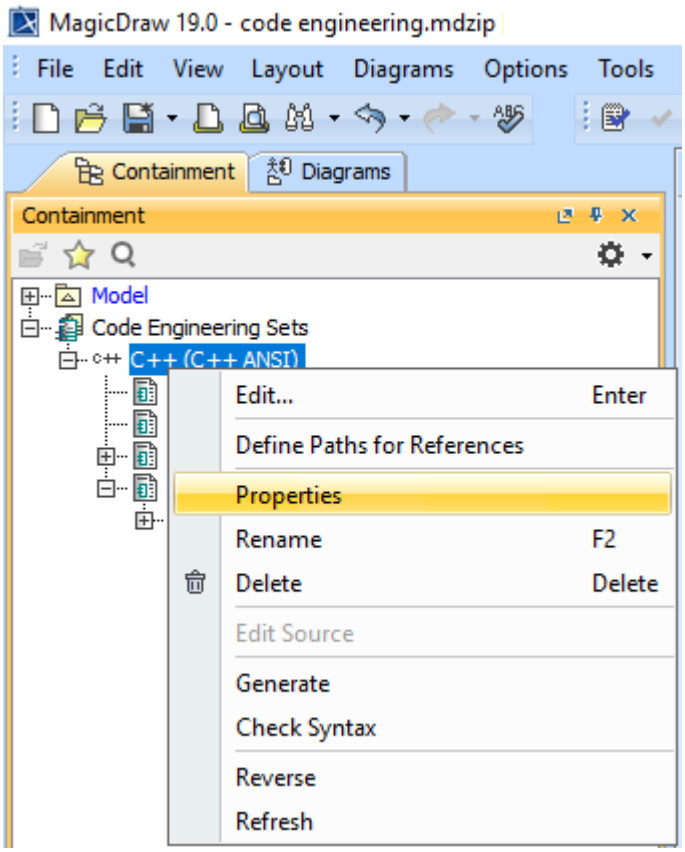
1. From the **Options** menu choose **Project**.
2. Expand **Code Engineering**, and choose **C++ Language Options**. Select the Include path check-box. Click the **Use include paths** button (...) appearing on the right side of the **Project** options dialog box.
3. The Set Include Path dialog appears.

Set Include Path dialog box

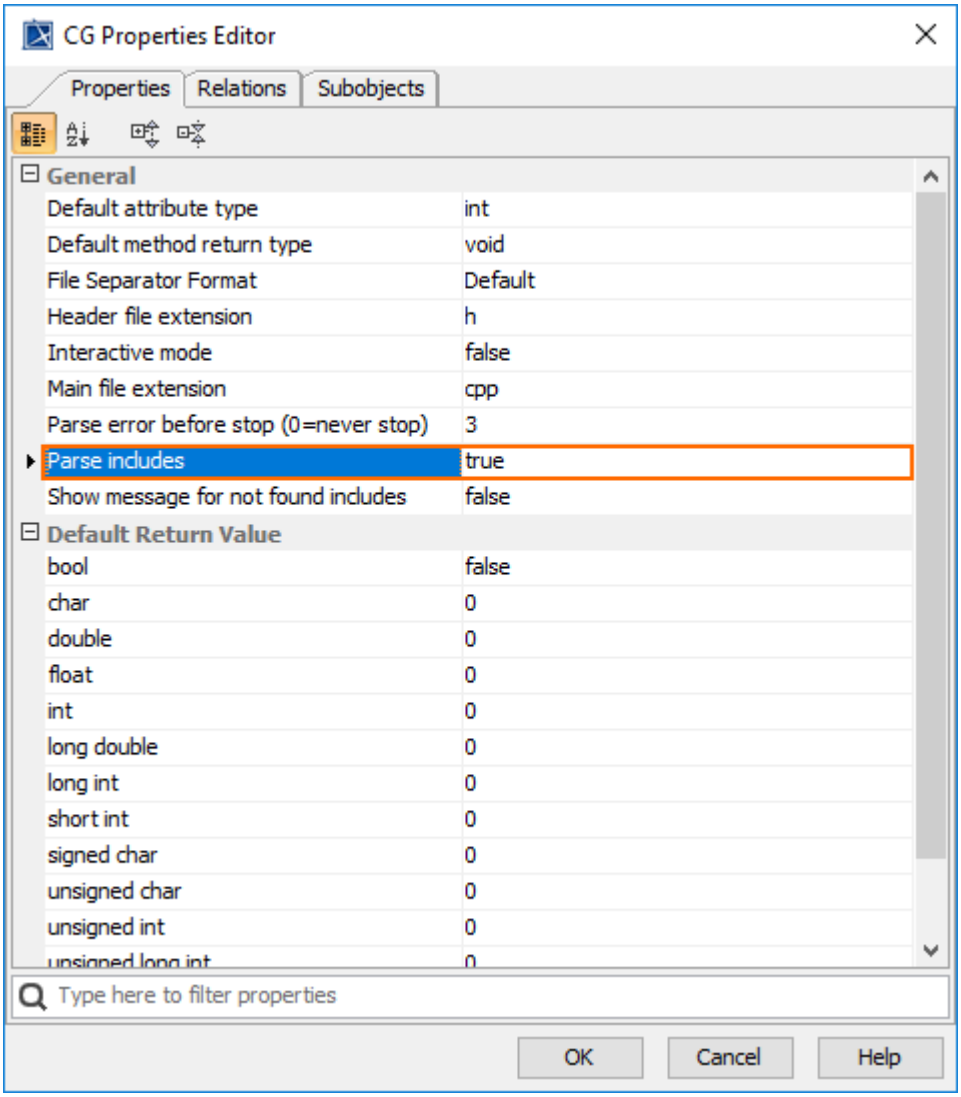
Button	Action
Add	The **Open** dialog box appears for the select directory path.
Remove	Remove the selected Include path.
Up	Move the selected Include path upward.
Down	Move the selected Include path downward.

Select the code-engineering set and then select **Properties**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Finally, set the “Parse includes” property to “true”.

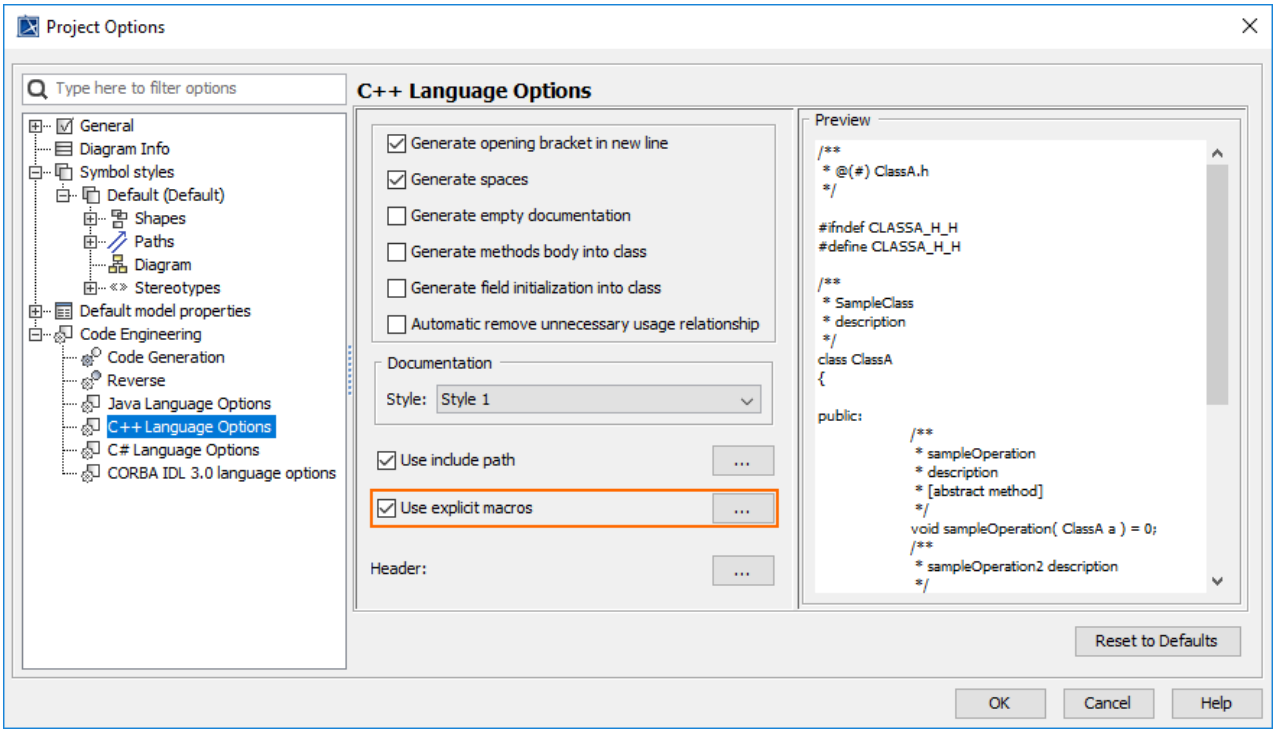


Parse Includes option
If the user sets the “Parse Includes” property to “false”, during reverse engineering of the C++ code, MagicDraw will not parse inside the header file; all unknown types will be created in the “Default” package of the model. The user can then solve keyword-specific problems in the library, speed up the reverse process, and remove unnecessary models inside the header file.

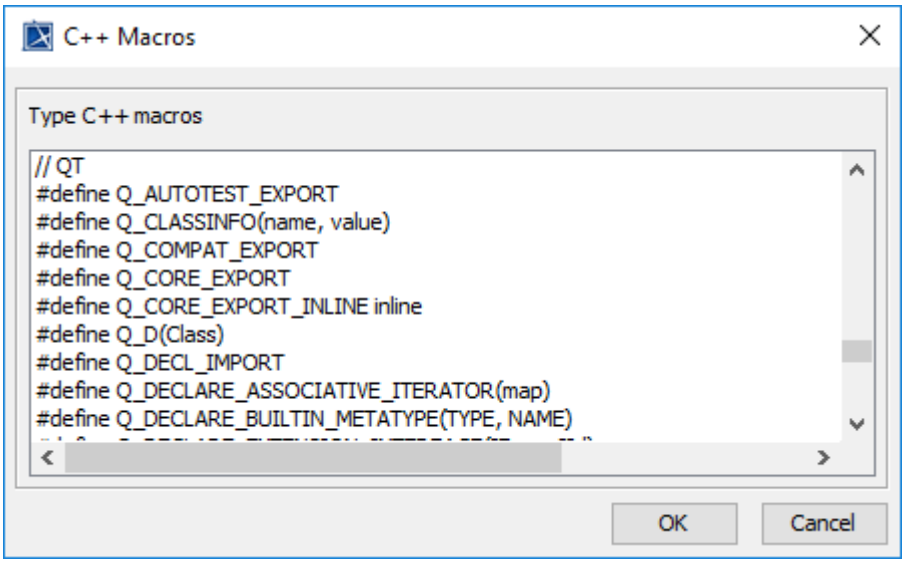
Working with QT

To reverse engineer QT code, we recommend you set the **Parse Includes** option to "false". You must also set the MagicDraw preprocessor to skip some QT macros ("**Options**"-> "**Project**"-> "**C++ Language Options**"-> "**Use explicit macros**").

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



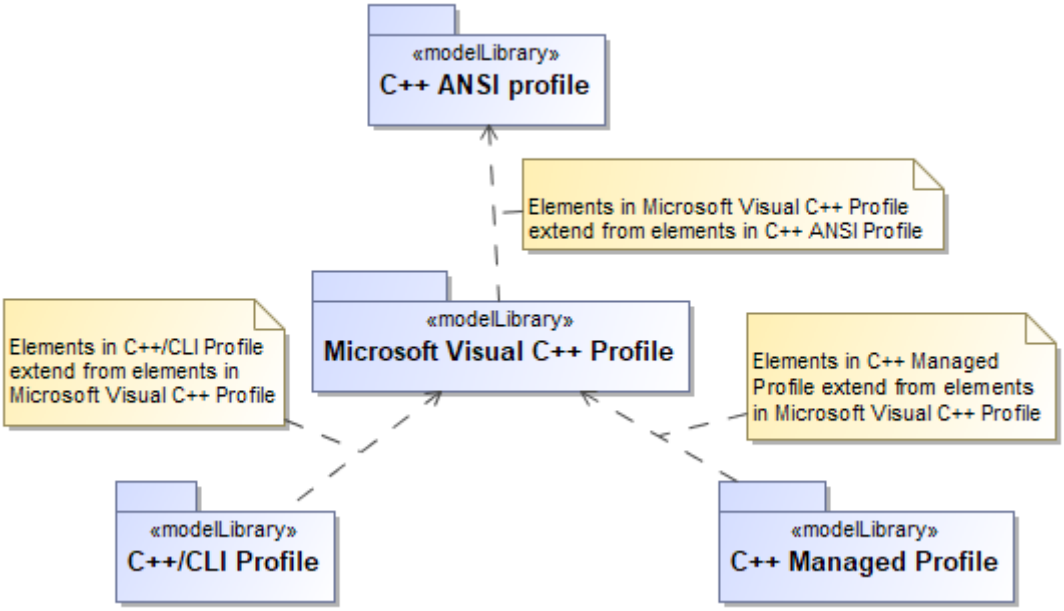
MagicDraw has already defined some default skipped macros for QT.



Microsoft C++ Profiles

Our modeling tool provides new profiles Microsoft Visual C++ Profile(see page 2551), C++/CLI Profile(see page 2551), and C++ managed profile(see page 2551) to support Microsoft-specific structure and keywords.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Profile Dependency

The *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.

Microsoft Visual C++ has been called *Managed C++*, Visual Studio 2003. 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++**

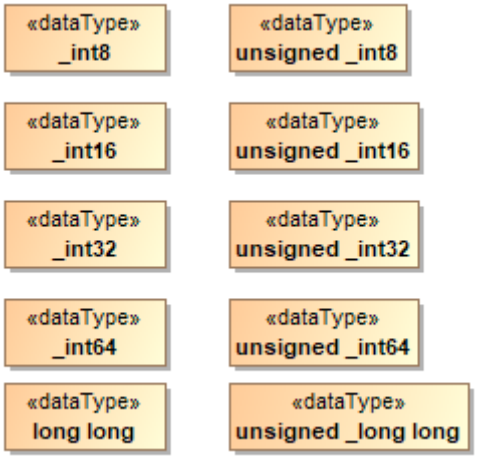
Profile Module Name: **C++_MS_Profile.xml**

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 __intn type specifier, where n is 8, 16, 32, or 64.

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.

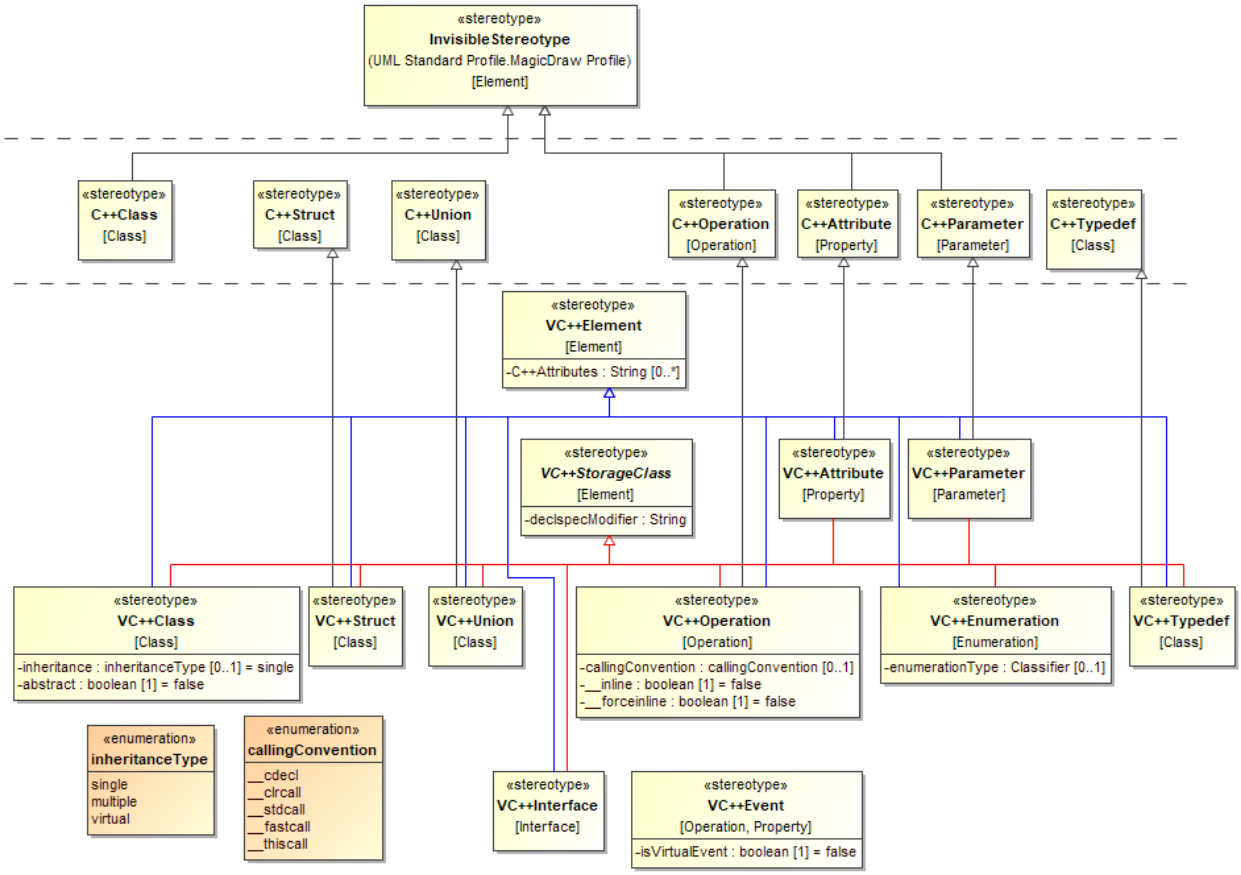
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Microsoft Visual C++ Data Type

Stereotype

Profile Table
The profile table and description in this section do not include the tagged value inherited from C++ ANSI profile.



VC++Class

«VC++Class» inherits from the «C++Class» and «VC++StorageClass»

Name	Meta Class	Constraints
VC++Class	Class	
Tag	**Type**	**Description**
inheritance	inheritanceType[0..1] = single (Enumeration) See *inheritanceType*	Represents the utilization of VC++ keywords, **__single_inheritance,** **__multiple_inheritance, and** **__virtual_inheritance.**
abstract	boolean[1] = false	Represents the utilization of keyword __abstract or abstract, depending on C++ dialects.
Inherited tag	**Type**	**Description**
declspecModifier «VC++StorageClass»	String	For keeping an extended declaration modifier of __declspec See Extended storage-class attributes with __declspec
C++Attributes «VC++Element»	String	For keeping C++ Attributes

«VC++Struct» inherits from «C++Struct» and «VC++StorageClass»

Name	Meta Class	Constraints
VC++Struct	Class	
Inherited Tag	**Type**	**Description**

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

declspecModifier «VC++StorageClass»	String	For keeping an extended declaration modifier of __declspec See Extended storage-class attributes with __declspec
C++Attributes «VC++Element»	String	For keeping C++ Attributes

VC++Union

Name	Meta Class	Constraints
VC++Union	Class	

Inherited Tag	Type	Description
declspecModifier «VC++StorageClass»	String	For keeping an 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]	Represents 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
VC++Attribute		
Name	**Meta Class**	**Constraints**
VC++Attribute	Proeprty	
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	Represents the utilization of keywords. • __cdecl • __clrcall • __stdcall • __fastcall • __thiscall
__inline	boolean[1] = false	Represents the utilization of the __inline keyword.
__forceinline	boolean[1] = false	Represents the utilization of the __forceinline keyword.
Inherited Tag	**Type**	**Description**

declspecModifier «VC++StorageClass»	String	For keeping an extended declaration modifier of __declspec See Extended storage-class attributes with __declspec
C++Attributes «VC++Element»	String	For keeping C++ Attributes

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**
declspeModifier	String	For keeping an 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

VC++Interface

Name	Meta Class	Description
VC++Interface	Interface	Represents interface declaration with __interface keyword.
VC++Event		
Name	**Meta Class**	**Description**
VC++Event	Operation, Property	Represents interface, method or member declaration with __event keyword.
Tag	**Type**	**Description**
isVirtualEvent	boolean[1] = false	Supports virtual event as shown in the example below. // data member as event virtual __event ClickEventHandler*OnClick;

Enumeration

inheritanceType

---

inheritanceType will be used as the value of the 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 the value of the tag named "**calling convention**" under **«VC++Operation»**

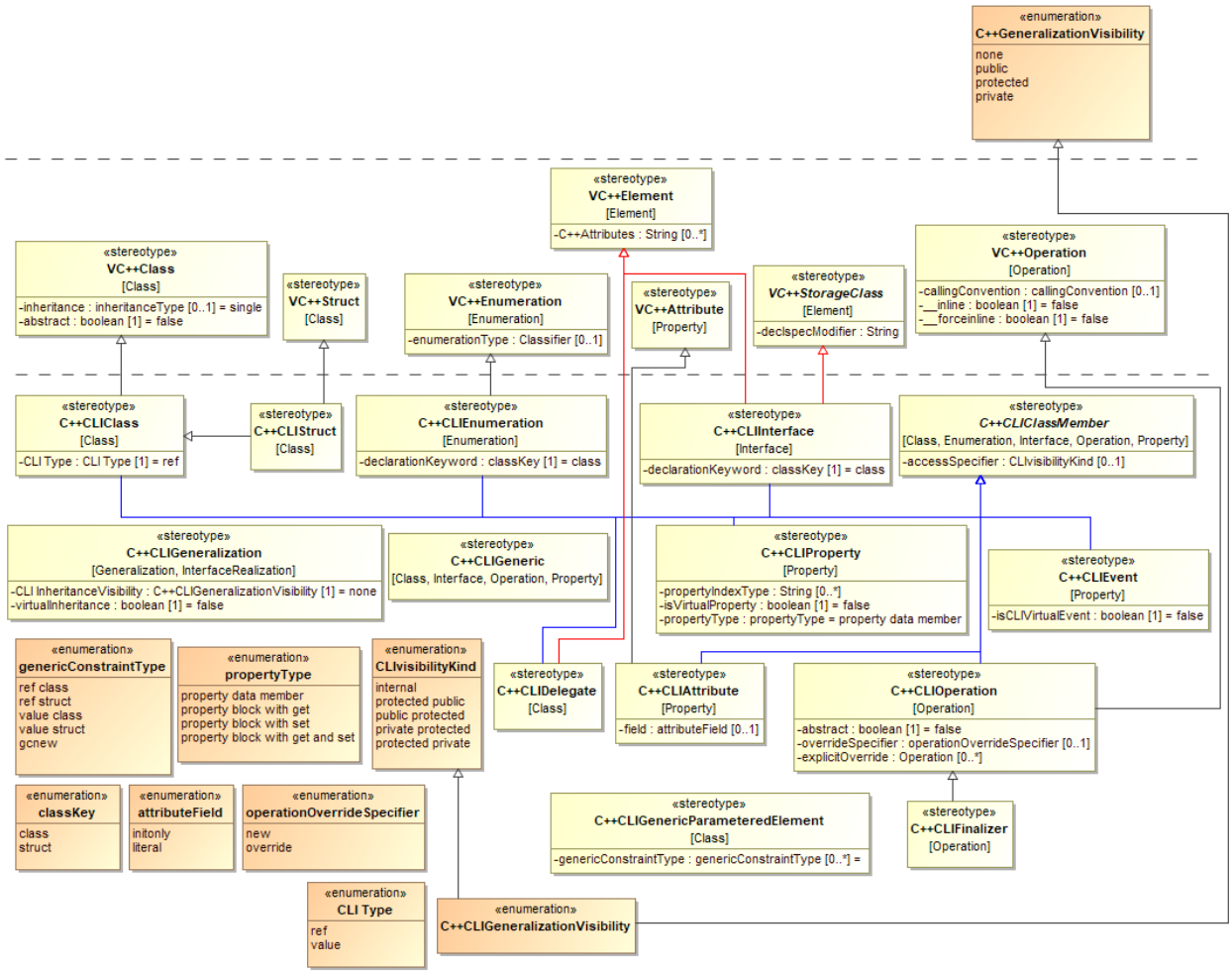
Literal	Description
__cdecl	Represents the utilization of __cdecl modifier.
__clrcall	Represents the utilization of __clrcall modifier.
__stdcall	Represents the utilization of __stdcall modifier.
__fastcall	Represents the utilization of __fastcall modifier.
__thiscall	Represents the utilization of __thiscall modifier.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

C++/CLI Profile

> **C++/CLI Profile**
> The profile table and description in this section do not include the tagged value inherited from C++ ANSI profile.

Stereotype



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(see page 0)	Represents the utilization of **ref class** or **value class** keywords for defining a CLR class.
Inherited Tag	**Type**	**Description**
inheritance «VC++Class»	inheritanceType[0..1] = single (Enumeration) See inheritanceType	Represents the utilization of VC++ keywords, **__single inheritance**, **__multiple_inheritance**, and **__virtual_inheritance**.
declspecModifier «VC++StorageClass»	String	For keeping an 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	Represents the utilization of an **abstract** keyword.
accessSpecifier «C++CLIClassMember»	CLIvisibilityKind[0..1] (Enumeration) See CLIvisibilityKind(see page 2559)	Represents 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(see page 0)	Represents the utilization of a **ref class** or **value class** keywords for defining a CLR class.
abstract «C++CLIClass»	boolean[1] = false	Represents the utilization of an **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 an extended declaration modifier of **__declspec** See *Extended storage-class attributes with __declspec*
C++Attributes «VC++Element»	String	For keeping C++ Attributes

C++CLIEnumeration

Name	Meta Class	Constraints
C++CLIEnumeration	Enumeration	
Tag	**Type**	**Descriptions**
declarationKeyword	classKey[1] = class (Enumeration) See classKey(see page 0)	Represents the utilization of **enum class** or **enum struct** keywords. See enum class, enum struct
Inherited Tag	**Type**	**Description**

type «VC++Enumeration»	Classifier[0..1]	Represents the type of enumeration literal

C++CLIInterface

Name	Meta Class	Constraints
C++CLIInterface	Interface	
Tag	**Type**	**Description**
declarationKeyword	classKey[1] = class (Enumeration) See classKey(see page 2562)	Represents 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 an extended declaration modifier of **__declspec** See *Extended storage-class attributes with __declspec*
C++Attributes «VC++Element»	String	For keeping C++ Attributes

C++CLIClassMember

Name	Meta Class	Constraints
C++CLIClassMember	Operation, Property	This stereotype is an abstract stereotype for keeping access - Specifier tag definition.
Tag	**Type**	**Descriptions**

accessSpecifier	CLIvisibilityKind[0..1] (Enumeration) See CLIvisibilityKind(see page 0)	Represents the usage of visibility kind introduced in C++/CLI including: • internal • protected public • public protected • private protected • protected private

C++CLIAttribute

Name	Meta Class	Constraints
C++CLIAttribute	Property	
Tag	**Type**	**Descriptions**
field	attributeField[0..1 (Enumeration) See attributeField(see page 2563)	Represents the usage of **initonly** or **literal** keywords.
Inherited Tag	**Type**	**Description**
accessSpecifier «C++CLIClassMember»	CLIvisibilityKind[0..1] (Enumeration) See CLIvisibilityKind(see page 2563)	Represents the usage of visibility kind introduced 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

C++CLIOperation

Name	Meta Class	Constraints
C++CLIOperation	Operation	
Tag	**Type**	**Descriptions**
abstract	boolean[1] = false	Represents the utilization of **abstract** keyword.
overrideSpecifier	operationOverrideSpecification[0..1] (Enumeration) See Override Specifiers(see page 2564)	Represents the utilization of C++/CLI operation override specification, **new** and **override.**
explicitOverride	Operation[0..*]	Represents the usage of explicit override feature.
Inherited Tag	**Type**	**Description**
accessSpecifier «C++CLIClassMember»	CLIvisibilityKind[0..1] (Enumeration) See CLIvisibilityKind(see page 2564)	Represents the usage of visibility kind introduced in C++/CLI including: • internal • protected public • public protected • private protected • protected private
callingConvention «VC++Operation»	callingConvention[0..1] See callingConvention	Represents the utilization of keywords. • __cdecl • __clrcall • __stdcall • __fastcall • __thiscall
C++Attributes «VC++Element»	String	For keeping C++ Attributes

C++CLIProperty

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

«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**	**Descriptions**
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(see page 2585)	Represents the usage of visibility kind introduced in C++/CLI including: • internal • protected public • public protected • private protected • protected private

C++CLIDelegate

Name	Meta Class	Constraints

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

C++CLIDelegate	Class	Declaration of delegate can be only in a managed type. The declaration of delegate cannot have attribute and operation. Represents the utilization of delegate keyword.
Tag	**Type**	**Descriptions**
C++CLIEvent	boolean[1] = false	Specify whether the event is virtual or not.
Inherited Tag	**Type**	**Description**
accessSpecifier «C++CLIClassMember»	CLIvisibilityKind[0..1] (Enumeration) See CLIvisibilityKind(see page 2565)	Represents the usage of visibility kind introduced in C++/CLI including: • internal • protected public • public protected • private protected • protected private

C++CLIGeneric

Name	**Meta Class**	**Constraints**
C++CLIGeneric	Class, Interface, Operation, Property	Represents the usage of a **generic** keyword.
C++CLIGenericParameteredElement	Class	
Tag	**Type**	**Descriptions**
genericConstraintType	genericConstraintType[0..*] (Enumeration) See genericConstraintType(see page 2566)	Specifies generic constraint type, including • ref class • ref struct • value class • value struct • gcnew

C++CLIGeneralization

Name	Meta Class	Constraints
C++CLIGeneralizationParameteredElement	Class	
Tag	**Type**	**Descriptions**
genericConstraintType	genericConstraintType[0..*] (Enumeration) See genericConstraintType(see page 2585)	Specifies generic constraint type, including • ref class • ref struct • value class • value struct • gcnew

C++CLIGeneralization

Name	Meta Class	Constraints
C++CLIGeneralization	Generalization	
Tag	**Type**	**Descriptions**
CLIInheritanceVisibility	C++CLIGeneralizationVisibility[1] = none (Enumeration) See C++CLIGeneralizationVisibility (see page 2586)	
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

operation OverrideSpecifier

---

operationOverrideSpecifier will be used as value of tag named "**overrideSpecifier**" under **«VC++Operation»**. The possible values are **new** and **override** which are keywords that can be used to qualify override behavior for derivation.

Literal	Description
new	Indicates the use of a **new** keyword to qualify override behavior for derivation. In VC++, **new** is a keyword indicating that a virtual member will get a new slot in the table; the function does not override a base class method.
override	Indicates the use of the **override** key to qualify override behavior for derivation. In VC++, **override** is a keyword indicating 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 the value of the tag named "field" under **«C++CLIAttribute»**. The possible values are initonly and literal, keywords used to qualify the field type of attribute.

Literal	Description
initonly	Represents 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	Represents 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

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Literal	Description
class	Represents the keyword that has the word, **class.**
struct	Represents the keyword that has the word, **struct.**

Remark:

- **enum class** and **enum struct** are equivalent declarations.
- **interface class** and **interface struct** are equivalent declarations.

CLIvisibilityKind

Literal	Description
internal	Represents the **internal** visibility.
protected public	Represents the **protected public** visibility.
public protected	Represents the **public protected** visibility.
private protected	Represents the **private protected** visibility.
protected private	Represents the **protected private** visibility.

genericConstraintType

Literal	Description
ref class	Represents the usage of the **ref class** keyword in a generic constraint clause.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Literal	Description
ref struct	Represents the usage of the **ref struct** keyword in a generic constraint clause.
value class	Represents the usage of the **value class** keyword in a generic constraint clause.
value struct	Represents the usage of the **value strict** keyword in a generic constraint clause.
gcnew	Represents the usage of the **gcnew** keyword in a generic constraint clause.

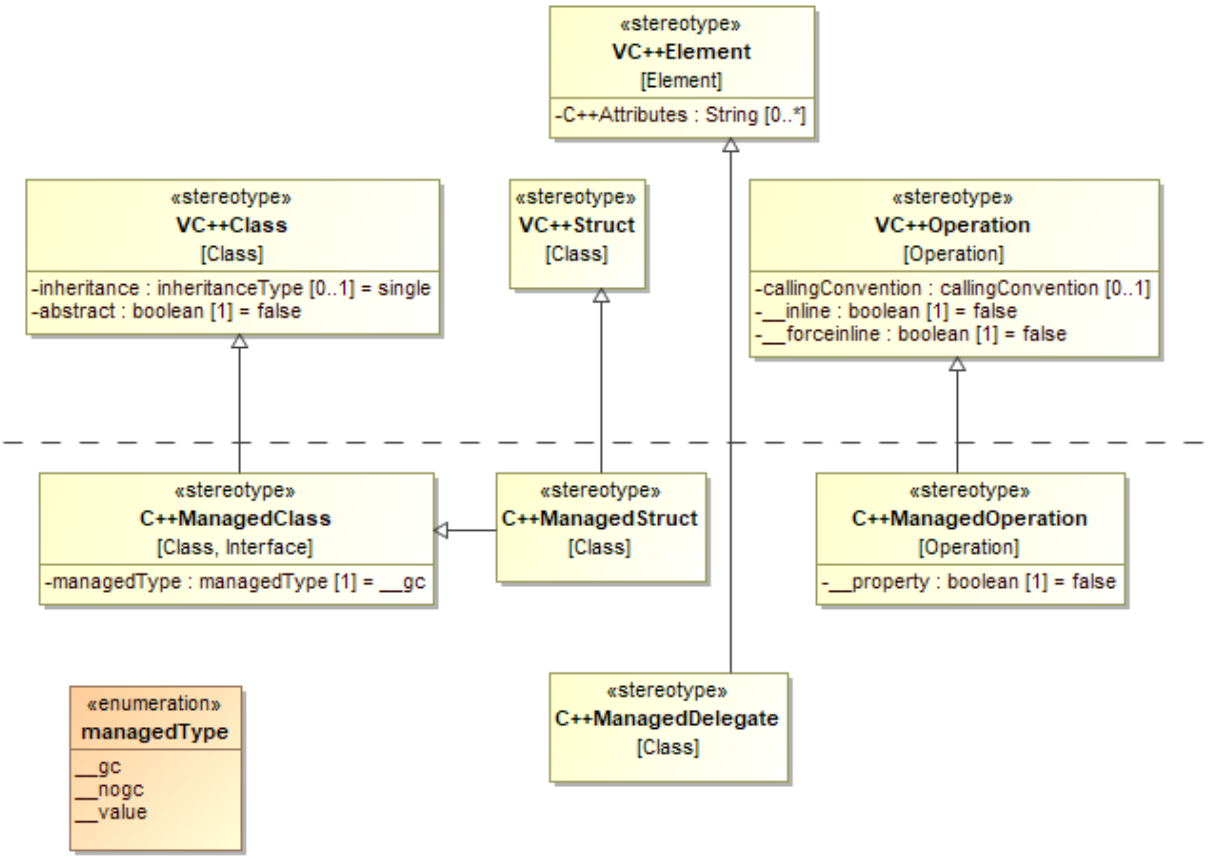
C++CLIGeneralizationVisibility

C++CLIGeneralizationVisibility is an enumeration inheriting from C++GeneralizationVisibility and CLIvisibility- Kind. Its literals include:

- **none,**
- **public,**
- **protected,**
- **private,**
- **internal,**
- **protected public,**
- **public protected,**
- **private protected,** and
- **protected private**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

C++ managed profile



C++ managed profile stereotypes

> **Note**
> The profile table and description in this section does not include the tagged value inherited from other profiles.

Stereotype

C++ManagedClass

Name	Meta Class	Constraints

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

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(see page 2571)	

C++ManagedStruct

Name	**Meta Class**	**Constraints**
C++ManagedStruct	Class	Represent the class declaration with version 1 of Managed Extension for C++.

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

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

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.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Modeling with Microsoft Specific Profiles

CLR Data Type Keyword

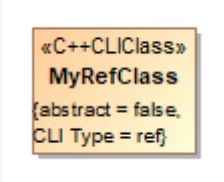
ref class, ref struct, value class, value struct, interface class, interface struct

- class_access ref class name modifier : inherit_access base_type {};

Code:

```
ref class MyRefClass
{
};
```

UML model with applied Stereotype *«C++CLIClass»* and Tag value **CLI Type** = ref

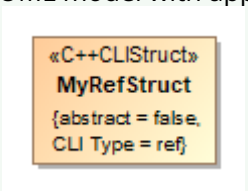


- class_access ref struct name modifier : inherit_access base_type {};

Code:

```
ref struct MyRefStruct
{
};
```

UML model with applied Stereotype *«C++CLIStruct»* and Tag value **CLI Type** = ref



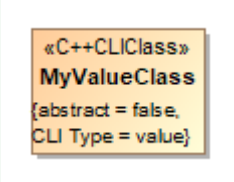
- class_access value class name modifier : inherit_access base_type {};

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Code:

```
value class MyValueClass
{
};
```

UML model with applied Stereotype *«C++CLIClass»* and Tag value **CLI Type** = value

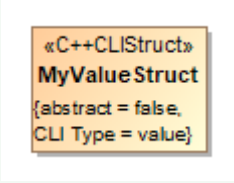


- class_access value struct name modifier : inherit_access base_type {};

Code:

```
value struct MyValueStruct
{
};
```

UML model with applied Stereotype *«C++CLIStruct»* and Tag value **CLI Type** = value

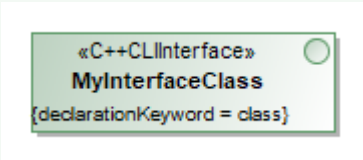


- interface_access interface class name : inherit_access base_interface {};

Code:

```
interface class MyInterfaceClass
{
};
```

UML model with applied Stereotype *«C++CLIInterface»* and Tag value **declarationKeyword** = class



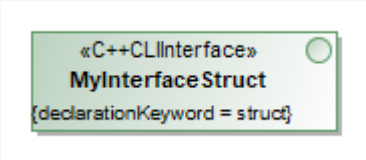
- interface_access interface struct name : inherit_access base_interface {};

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Code:

```
interface struct MyInterfaceStruct
{
};
```

UML model with applied Stereotype *«C++CLIInterface»* and Tag value **declarationKeyword** = struct



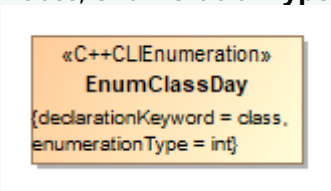
enum class, enum struct

- access enum class name [: type] { enumerator-list } var;

Code:

```
enum class EnumClassDay:int
{
    sun,
    mon
};
```

UML model with applied Stereotype *«C++CLIEnumeration»* and Tag value **declarationKeyword** = class, **enumerationType** = int



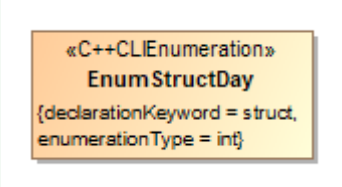
- access enum struct name [:type] { enumerator-list } var;

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Code:

```
enum class EnumClassDay:int
{
    sun,
    mon
};
```

UML model with applied Stereotype *«C++CLIEnumeration»* and Tag value **declarationKeyword** = struct, **enumerationType** = int

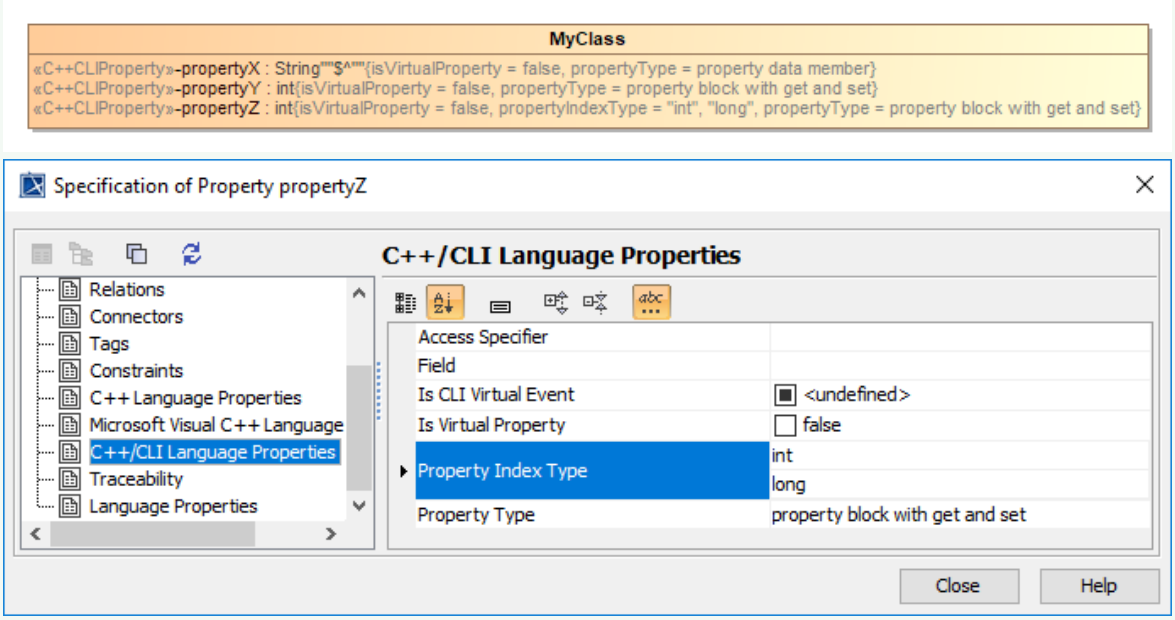


Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

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);
    }
};
```

UML model:



Example of C++/CLI Language Properties for describing Property *zropertyZ*

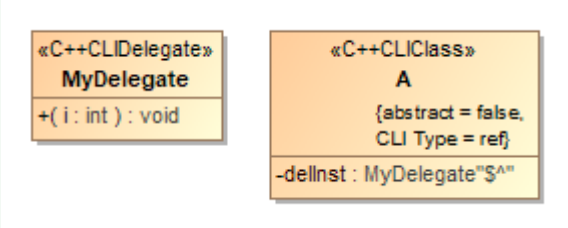
delegate

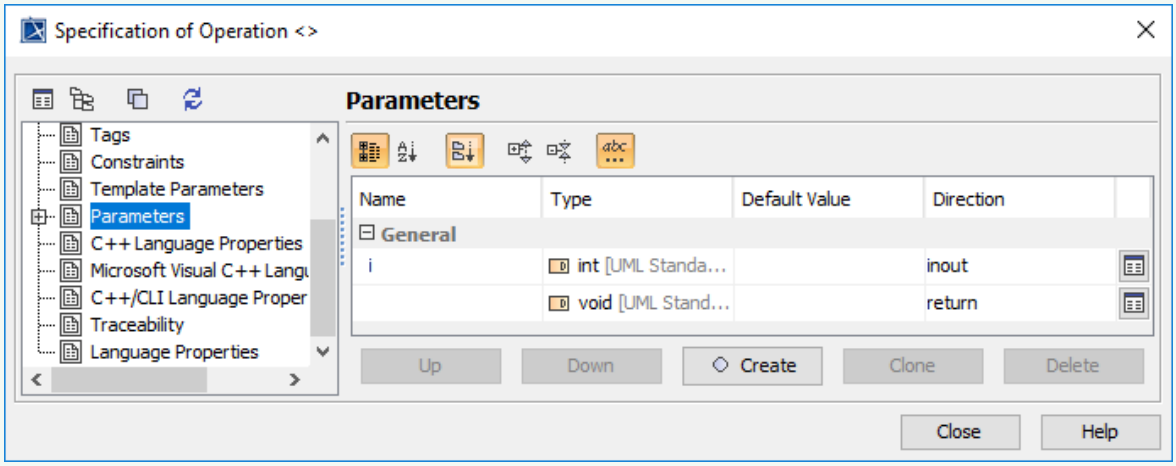
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Code:

```
public delegate void MyDelegate(int i);
ref class A
{
   MyDelegate^ delInst;
};
```

UML model:



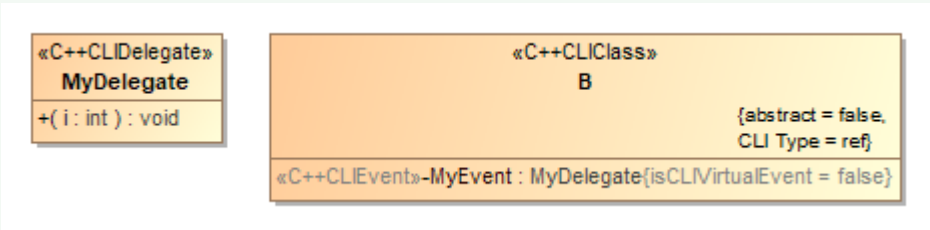


event

Code:

```
public delegate void
MyDelegate(int);
ref class B {
     event MyDelegate^ MyEvent;
};
```

UML model:



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

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.

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.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

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) {}`
`};`	

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

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[, ...]

Code	MD-UML

`interface class MyInterface {};`
`ref class MyBase{};`
`generic <class T1, class T2>`
`where T1 : MyInterface, MyBase`
`where T2 : MyBase`
`ref class MyClass {};`	

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)

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>`
`{`
`};`	

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Generic Delegates

[attributes]

generic < [class | typename] type-parameter-identifiers >

[type-parameter-constraints-clauses]

[accessibility-modifiers] delegate result-type identifier

([formal-parameters]);

Parameters

attributes (Optional)

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);`	

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;`
`  }`
`};`
	MyStruct with «C++CLIStruct»
	staticConst1 and staticConst2 have «C++CLIAttribute» applied with tagged value field = initonly and Is Static = true

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
`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

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

The name of the class being declared.

Code	MD-UML
`class __single_inheritance S;`	Class with «VC++Class»

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 {};`	

__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;`
`};`
	__event __interface IEvents;
	
	is mapped to attribute without name

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

`__interface IEvents {`
`  };`
`class CSource {`
`public:`
`  __event __interface IEvents;`
`};`
	__event __interface IEvents;
	
	is mapped to attribute without name

public __delegate void D();

public __gc class X { public:

__event D* E;

__event void noE(); };

NOTE __delegate is in C++ Managed Profile whereas __event is in Microsoft Visual C++ Profile.

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;`	

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();`
`};`
	__event __interface IEvents;
	
	is mapped to attribute without name

__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

`__interface IEvents {`
`  };`
`class CSource {`
`public:`
`  __event __interface IEvents;`
`};`
	__event __interface IEvents;
	
	is mapped to attribute without name

Code

void __clrcall Test1( ) {} void (__clrcall *fpTest1)( ) = &Test1;

MD-UML

217

Copyright © 1998-2015 No Magic, Inc.

C++ CODE ENGINEERING Microsoft C++ Profiles

__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 com- piler makes vararg functions __cdecl. Functions that use this calling convention require a function prototype.

Code	MD-UML

`__interface IEvents {`
`  };`
`class CSource {`
`public:`
`  __event __interface IEvents;`
`};`
	__event __interface IEvents;
	
	is mapped to attribute without name

Code MD-UML

class CmyClass2 {
void __stdcall mymethod();

};

__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();

};

Code	MD-UML

`__interface IEvents {`
`  };`
`class CSource {`
`public:`
`  __event __interface IEvents;`
`};`
	__event __interface IEvents;
	
	is mapped to attribute without name

Code MD-UML

class CmyClass3 {
void __fastcall mymethod();

};

__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

`__interface IEvents {`
`  };`
`class CSource {`
`public:`
`  __event __interface IEvents;`
`};`
	__event __interface IEvents;
	
	is mapped to attribute without name

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Code

class CmyClass4 {
void __thiscall mymethod();

};

MD-UML

220

Copyright © 1998-2015 No Magic, Inc.

C++ CODE ENGINEERING Microsoft C++ Profiles

__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 }

Code	MD-UML

`__interface IEvents {`
`  };`
`class CSource {`
`public:`
`  __event __interface IEvents;`
`};`
	__event __interface IEvents;
	
	is mapped to attribute without name

Code

__except(1) { puts("exception");

}}

int __unaligned *p1;

221

Copyright © 1998-2015 No Magic, Inc.

C++ CODE ENGINEERING Microsoft C++ Profiles

__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 }
```

222

Copyright © 1998-2015 No Magic, Inc.

C++ CODE ENGINEERING Microsoft C++ Profiles

Code	MD-UML

`__interface IEvents {`
`  };`
`class CSource {`
`public:`
`  __event __interface IEvents;`
`};`
	__event __interface IEvents;
	
	is mapped to attribute without name

Code

int __w64 i;

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

MD-UML

223 Copyright © 1998-2015 No Magic, Inc.

C++ CODE ENGINEERING Microsoft C++ Profiles

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, appdo- main, 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 dll- export 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 declara- tion.

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

`__interface IEvents {`
`  };`
`class CSource {`
`public:`
`  __event __interface IEvents;`
`};`
	__event __interface IEvents;
	
	is mapped to attribute without name

Code MD-UML

class __declspec(dllimport) X {};

Copyright © 1998-2015 No Magic, Inc.

Code	MD-UML

`__interface IEvents {`
`  };`
`class CSource {`
`public:`
`  __event __interface IEvents;`
`};`
	__event __interface IEvents;
	
	is mapped to attribute without name

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 declara- tions and definitions.

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

`__interface IEvents {`
`  };`
`class CSource {`
`public:`
`  __event __interface IEvents;`
`};`
	__event __interface IEvents;
	
	is mapped to attribute without name

Code

MD-UML

int * __restrict x;

226

Copyright © 1998-2015 No Magic, Inc.

C++ CODE ENGINEERING Microsoft C++ Profiles

__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 pag- ing 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 __forcein- line 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

`__interface IEvents {`
`  };`
`class CSource {`
`public:`
`  __event __interface IEvents;`
`};`
	__event __interface IEvents;
	
	is mapped to attribute without name

Code

__inline int max(int a , int b) { if(a > b)

```
 return a; return b;
```

}

MD-UML

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 {};`	__event __interface IEvents; is mapped to attribute without name

Override Specifiers

abstract

Code	MD-UML
`ref class MyAbstractClass abstract { };`	with «C++CLIClass» and tagged value **abstract**=true

Code	MD-UML
`ref class MyAbstractClass2 { public: virtual void func() abstract; };`	func() with «C++CLIOperation» and tagged value **abstract** = true and **virtual** = true

new

Code	MD-UML

`ref class MyClassWithNewFunction`
`{`
`public:`
`  virtual void func() new {}`
`};`
	func() with «C++CLIOperation» and tagged value **overrideSpcifier** = new

override

Code	MD-UML

`ref class BaseClass {`
`  public:`
`virtual void f();`
`};`
`ref class SubClass : public`
`BaseClass {`
`  public:`
`virtual void f() override {}`
`};`	

sealed

Code	MD-UML

`interface class MyInterface`
`{`
`  virtual void f();`
`  virtual void g();`
`};`
`ref class MyClass sealed: public`
`MyInterface`
`{`
`public:`
`  virtual void f(){};`
`  virtual void g(){};`
`};`	

`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 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

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
```

@see on documentation

This feature only works in the import-code-only mode.

Documentation with doxygen @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

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.

Doxygen-after-member documentation

> **Functionality**
> This feature only works in import-code-only mode.

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

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

CG Properties Editor

File Separator Format

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

Default Return Value

You can set the default return value to 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 a .cpp file.

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

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Property Name	Description

Interactive mode	If enabled, you 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 a class or namespace.
	
	Figure 122 -- Question dialog for interactive mode
	
	Default Mode
	If you are using the default mode, rather than the interactive mode, the reverse module will automatically give the best guess according to the information in the code.

Parse error before stop	Allows you to specify the number of errors to ignore before the reverse process will be terminated. You 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. **Parse Includes** You must set the property Parse Includes to true in order to use the property Show message for not found includes.

Support C++ dialects

C# Code Engineering

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.

Related pages:

- C# 2.0 features(see page 2606)
- C# 3.0 features(see page 2606)
- Migration from old C# project version(see page 2650)
- C# mapping to UML rules(see page 2655)
- C# Project Constraints(see page 2676)
- Code Engineering(see page 2398)
  - Code Engineering Sets(see page 2399)
  - Generating Code(see page 2403)
  - Reverse Options(see page 2406)
  - Global options for Code Engineering(see page 2415)
  - Files of Properties(see page 2426)
  - Java Code Engineering(see page 2427)
  - C++ Code Engineering(see page 2454)
  - C# Code Engineering(see page 2606)
  - CORBA IDL Mapping To UML(see page 2680)
  - WSDL(see page 2683)

C# 2.0 features

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.

The MagicDraw supports these C# 2.0 language extensions:

- Generics(see page 2606)
- Anonymous method(see page 2606)
- Partial types(see page 2606)
- Nullable types(see page 2606)
- Accessor declarations(see page 2606)
- Static Class(see page 2606)
- Extern alias directive(see page 2606)
- Fix size buffer(see page 2606)

This chapter gives an introduction how these features are described in MD tool.

Related pages

- Code Engineering(see page 2398)
  - Code Engineering Sets(see page 2399)
  - Generating Code(see page 2403)
  -

Reverse Options(see page 2406)
- Global options for Code Engineering(see page 2415)
- Files of Properties(see page 2426)
- Java Code Engineering(see page 2427)
- C++ Code Engineering(see page 2454)
- C# Code Engineering(see page 2606)
- CORBA IDL Mapping To UML(see page 2680)
- WSDL(see page 2683)

Generics

Generics permit classes, structs, interfaces, delegates, and methods to be parameterized by the data types they store and manipulate. Generic class declaration should be mapped to the UML classifier (class or interface) with a Template parameter. Additionally, Generics still affect other parts of the program structures such as Attribute, Operation, Parameter, Parent Class, and Overloading Operators. In this chapter you will find how C# code structures are mapped to the UML model.

Here are the Generics presented throughout the page:

- Generic Class(see page 2607)
- Generic Interface(see page 2607)
- Generic Delegate(see page 2607)
- Generic Attribute(see page 2607)
- Generic Operation(see page 2607)
- Generic parent class(see page 2607)
- Generic using alias(see page 2607)
- Generic constraints(see page 2607)

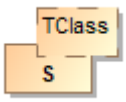
Generic Class

Class S has one template parameter named T. The default type of T is Class.

Example
Code:

```
public class S<T>
{
}
```

Reversed UML model:



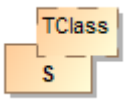
Generic Struct

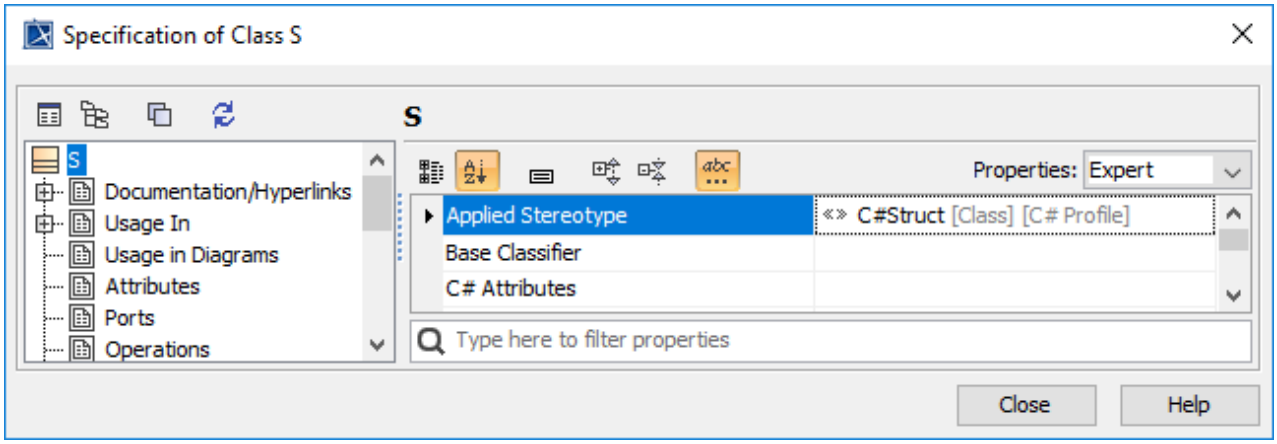
The type parameter of generic struct is created the same as generic class, but we apply «C#Struct» stereotype to the model.

Example
Code:

```
public class S<T>
{
}
```

Reversed UML model:





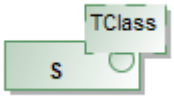
Sample: Class S with applied stereotype C#Struct

Generic Interface

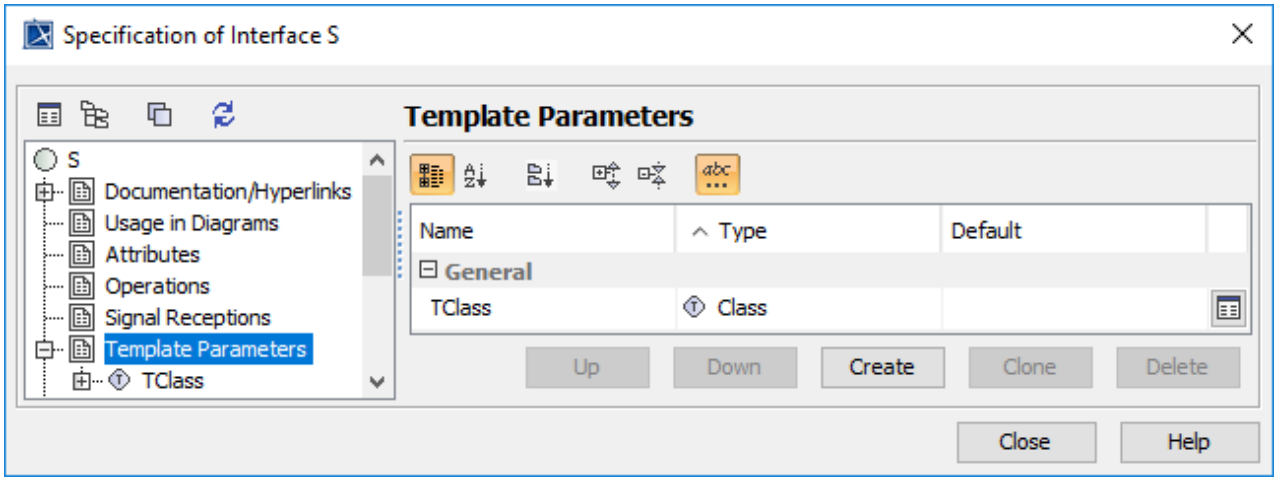
Example
Code:

```
interface S<T>
{
}
```

Reversed UML model:



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Sample: Interface S template parameters

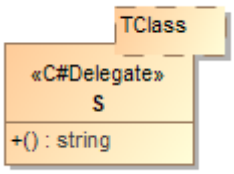
Generic Delegate

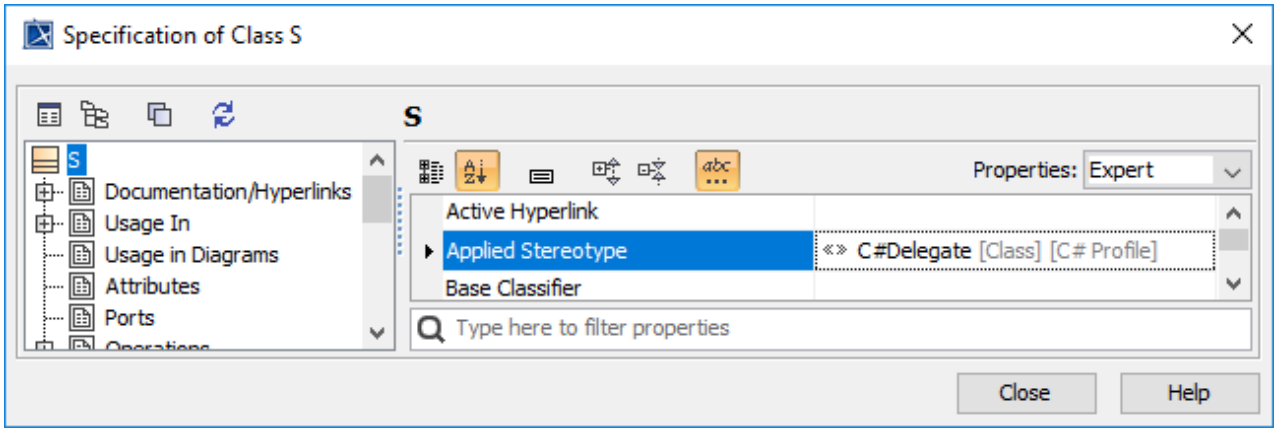
To create a generic delegate, we create a class model and apply the «C#Delegate» to the model. We, then, create an empty named method with delegate return type, and add template parameter like a normal generic class.

Example
Code:

```
delegate string D<T>();
```

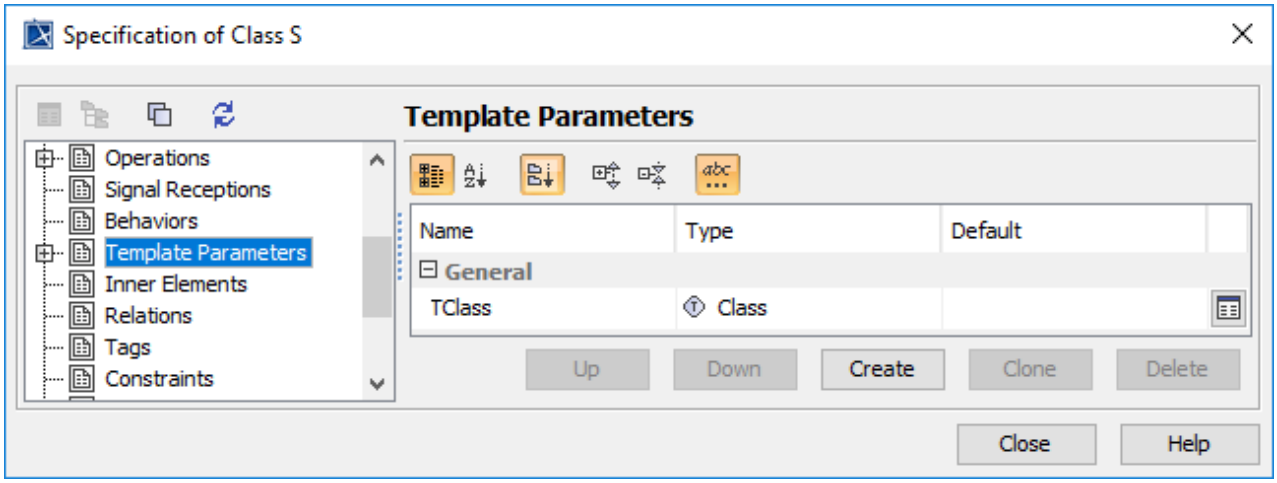
Reversed UML model:





Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Sample: Applied Stereotype is C#Delegate



Sample: Template parameters

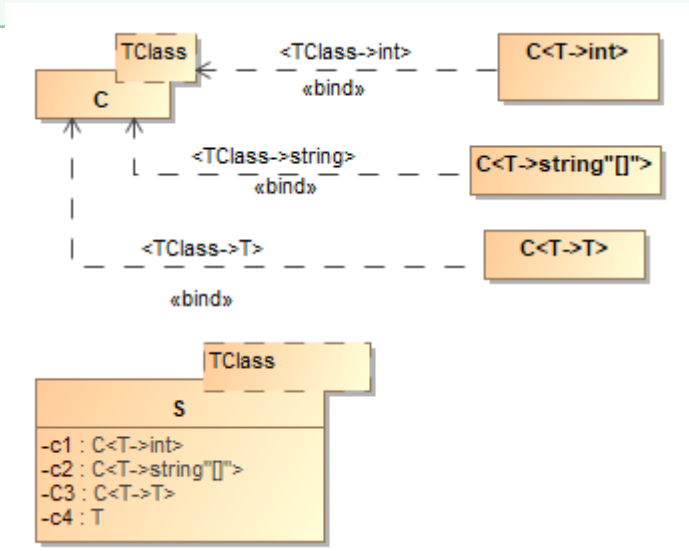
Generic Attribute

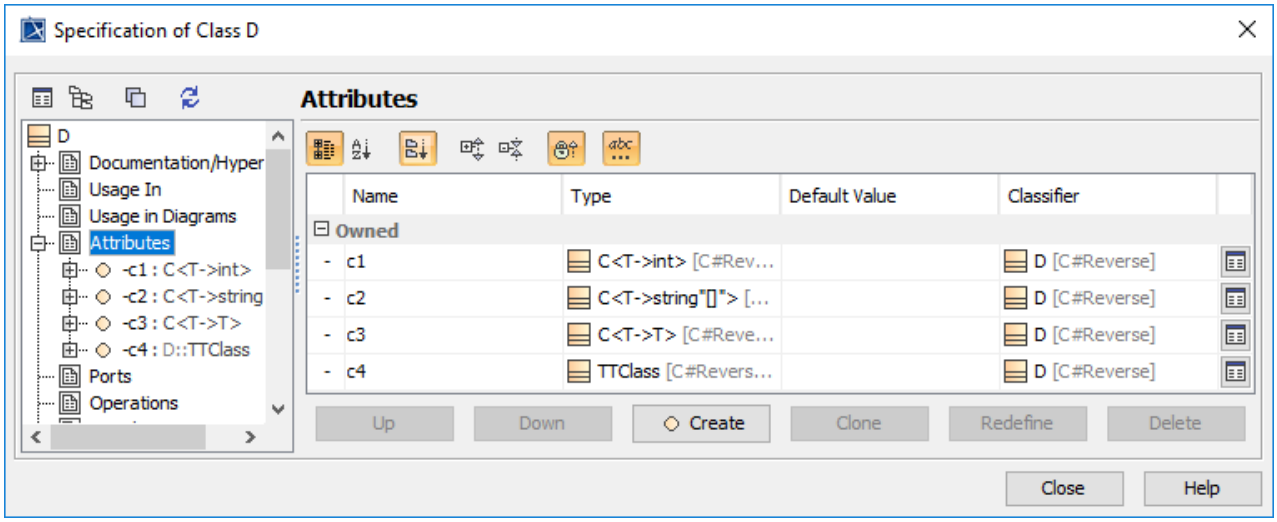
The type of attributes in the class can be generic binding (template binding) or template parameter of owner class. The example code shows attributes that use template binding as its type

Example
Code:

```
class C<T>
{
}
class D<T>
{
        private C<int> c1;
        private C<string[]> c2;
        private C<T> c3;
        private T c4;
}
```

Reversed UML model:

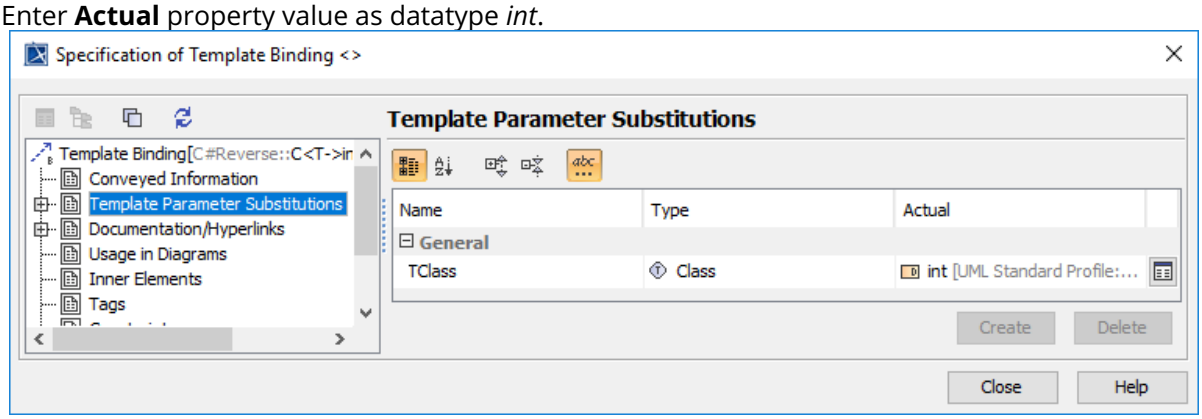




Class *D* Attributes with generic type

The following shows how Template binding for *C<int>* is created:

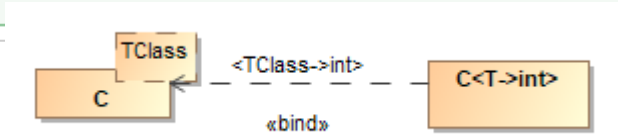
1. From the Template binding link shortcut menu, select **Specification**.
2. From Specification window property group list, select **Template parameter Substitutions** property group.
3. Enter **Actual** property value as datatype *int*.



Template binding for C<int>
Code:

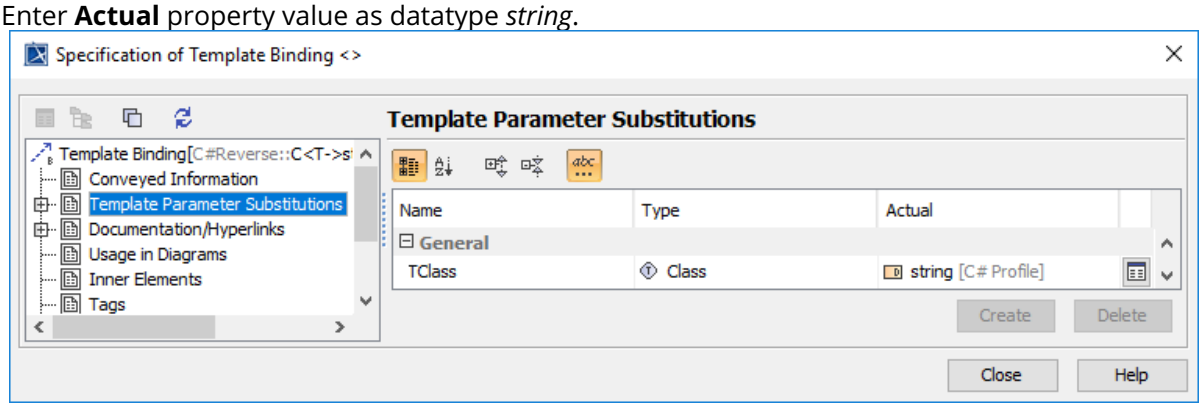
```
..
Private C<int> c1;
```

Reversed UML model:



Creating Template binding for *C<string[]>*:

. From the Template binding link shortcut menu, select **Specification**.
2. From Specification window property group list, select **Template parameter Substitutions** property group.
3. Enter **Actual** property value as datatype *string*.

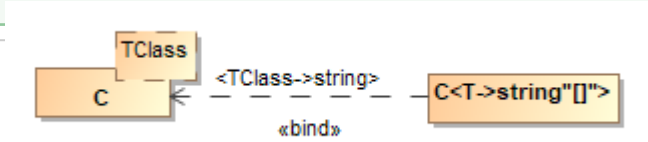


Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Template binding for C<string[]>
Code:

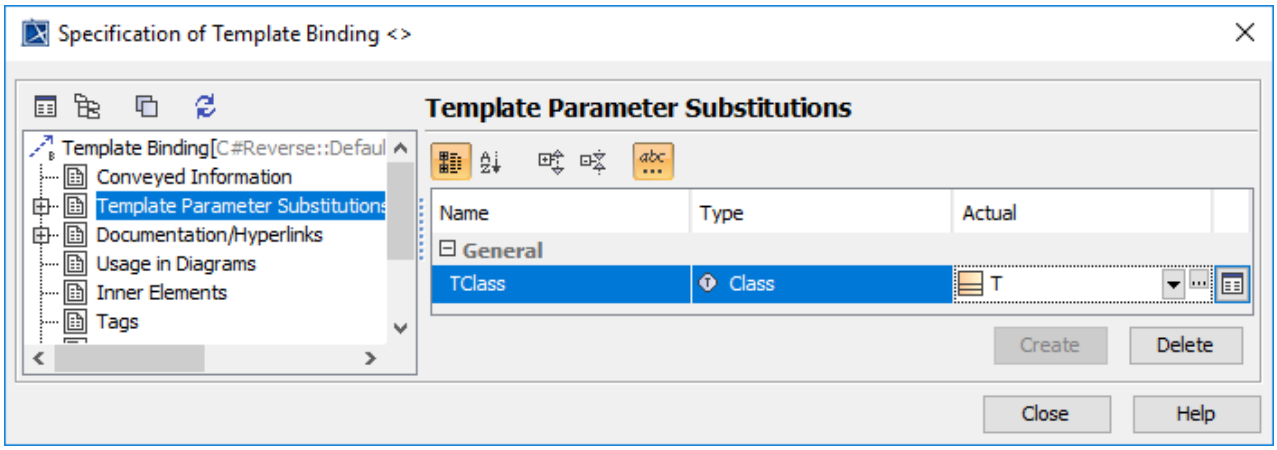
```
..
Private C<string[]> c2;
```

Reversed UML model:



Creating Template binding for *C<T>*:

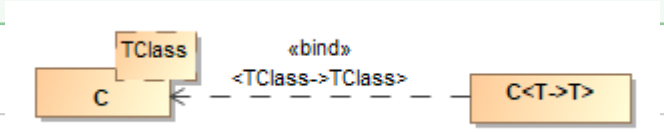
. From the Template binding link shortcut menu, select **Specification**.
2. From Specification window property group list, select **Template parameter Substitutions** property group.
3. Enter **Actual** property value as Binding type *T*. *T* is the template parameter of its Owner Class *S<T>*.



Template binding for C<T>
Code:

```
..
Private C<T> c3;
```

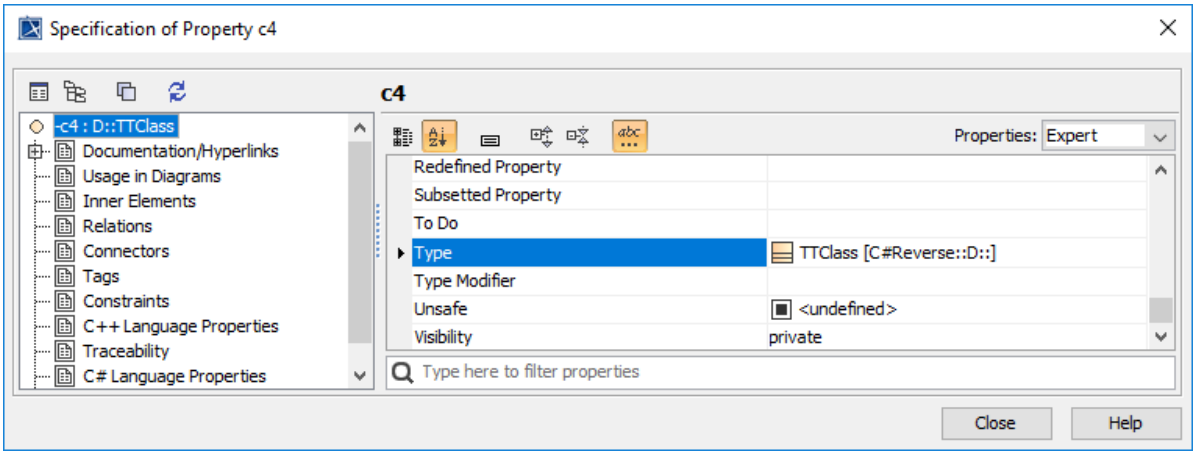
Reversed UML model:



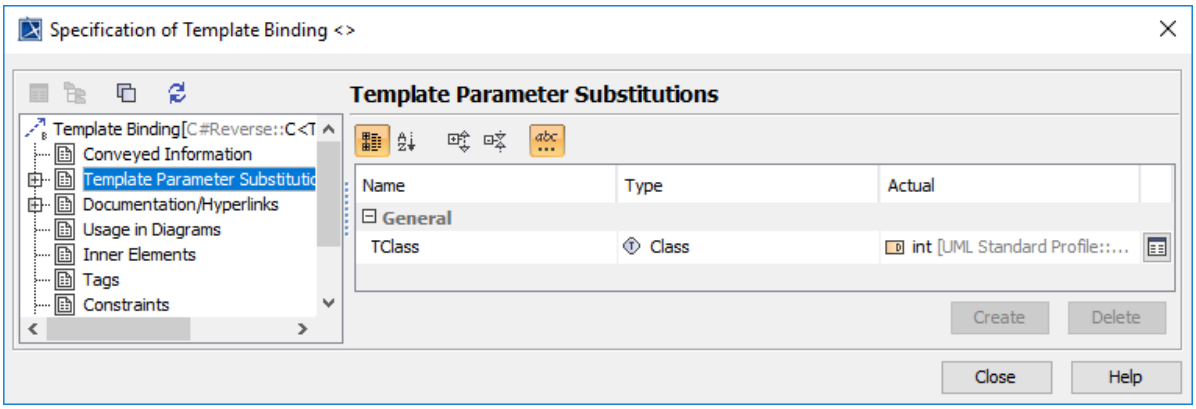
Creating attribute with the type template parameter of the owner class S<T>:

1. Create Attribute *c4* (c4 is not a template binding class).
2. From the Attribute link shortcut menu, select **Specification**.
3. Specify **Type** property, select Template parameter of the Owner Class *S<T>*.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



4. From Specification window property group list, select **Template parameter Substitutions** property group.
5. Enter **Actual** property value as datatype *int.*



Template binding for Attribute c4
Code:

```
..
Private T c4;
```

Generic Operation

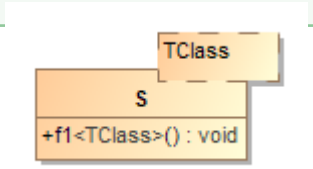
Language extention Generic can be applied to **Operation**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

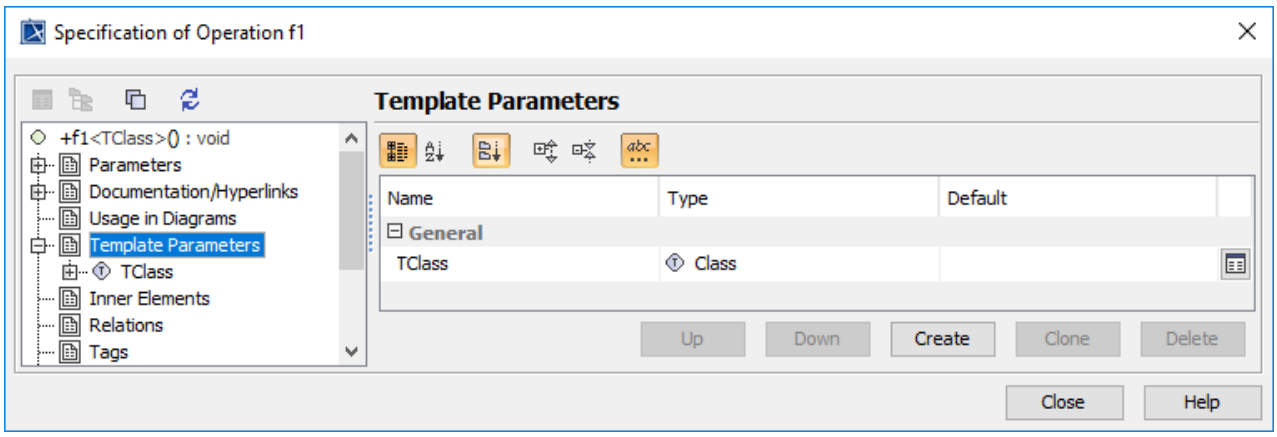
Example
Code:

```
public class S<T>
{
        public void f1<T>() {}
}
```

Reversed UML model:



To create **Generic Operation**, open the operation Specification window(see page 913), and from **Property group** create Template(see page 866) Parameters(see page 866).



To create Operations with parameters and return type, create them like a **Generic Attribute**.

Binding Types
We have to create or select the correct binding types.

In the example code, the type of parameter *t* is public void *f1<T> (T t)*, it must be the **Template Parameter** of the owner method, but type *f1<T>* is not the template parameter of the owner Class *S<T>*.

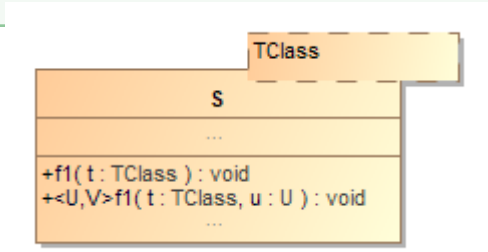
In the second method, public void *f1<U, V> (T t, U u)*, parameter *t* type must be the **Template Parameter** of the owner Class, *S<T>*.

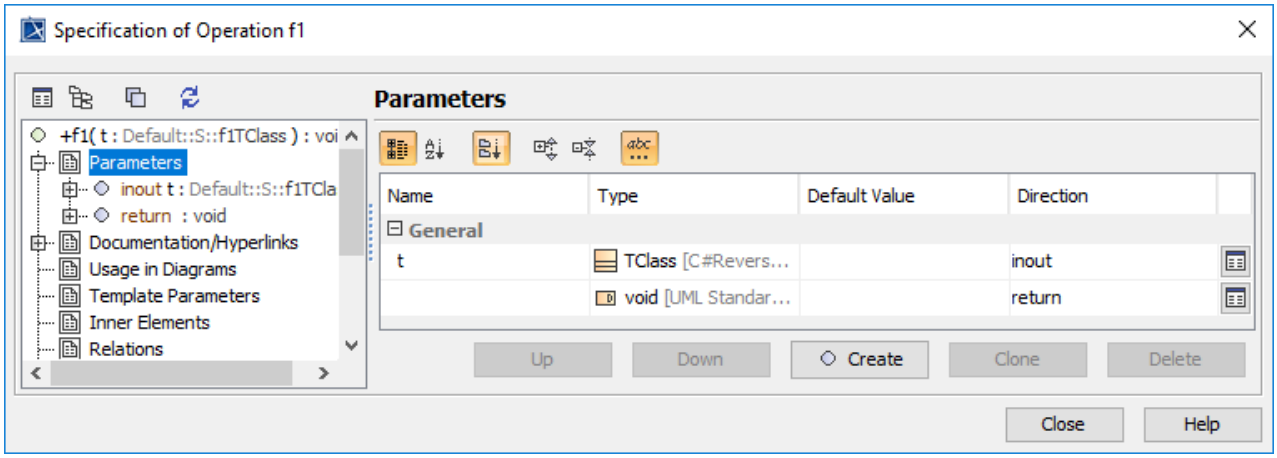
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example
Code:

```
public class S<T>
{
public void f1<T> (T t)
{
}
public void f1<U, V> (T t, U u)
{
}
}
```

Reversed UML model:

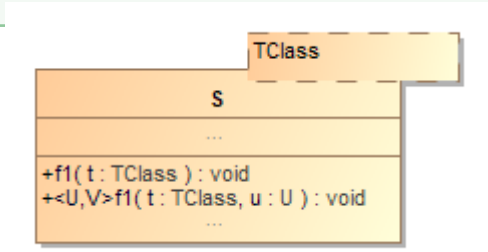


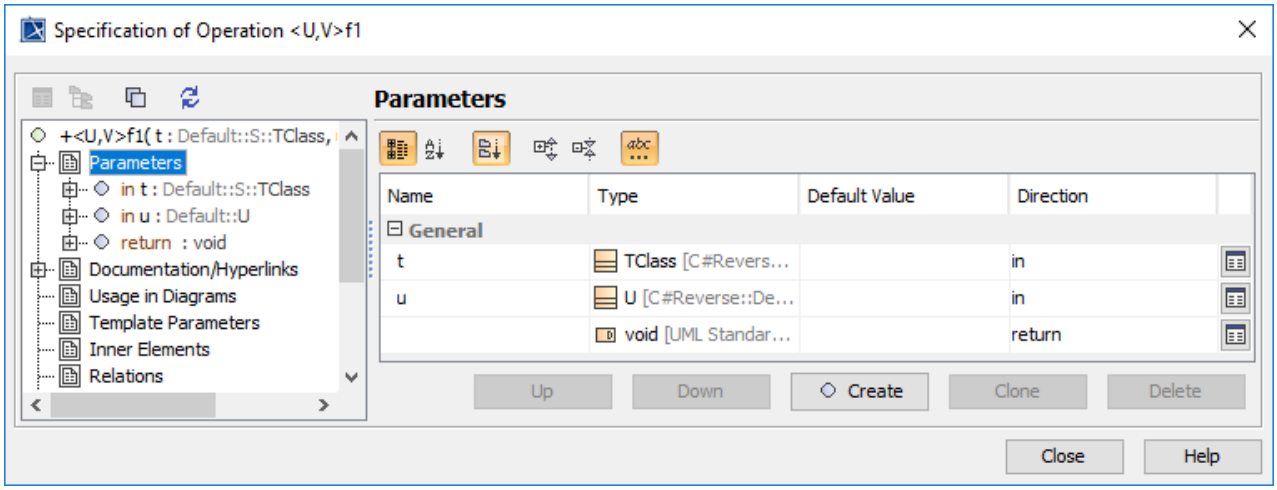


Example
Code:

```
class b
{
 public T f1<T, U>(T t, U u)
       whereU:T{returnt; }
}
```

Reversed UML model:





Generic overloading

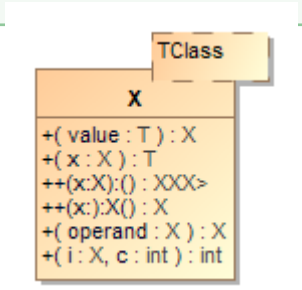
Methods, constructors, indexers, and operators within a generic class declaration can be overloaded. While signatures as declared must be unique, it is possible that substitution of type arguments results in identical signatures.

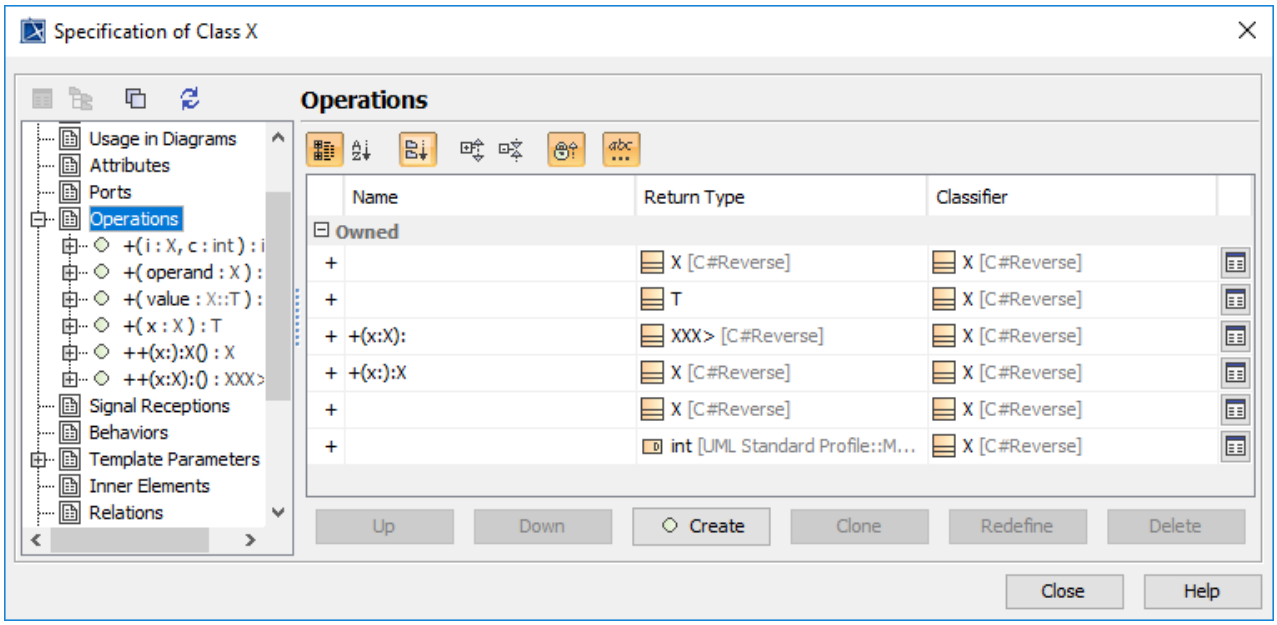
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example
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; }
}
```

Reversed UML model:





Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Generic parent class

Example
Code:

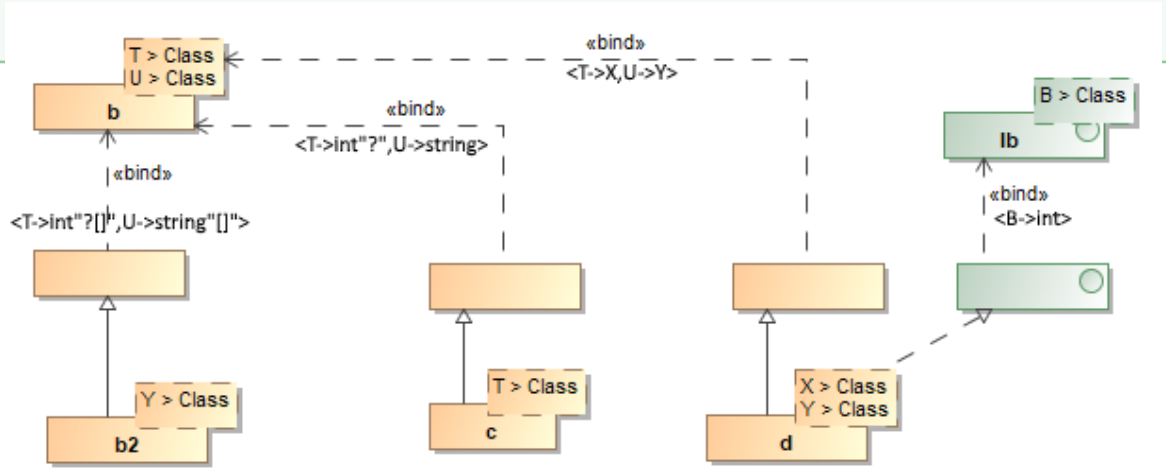
```
class b<T, U>
{}
class b1<X> : b<X[], X[,]>
{}
```

Reversed UML model:

Example
Code:

```
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>
{}
```

Reversed UML model:



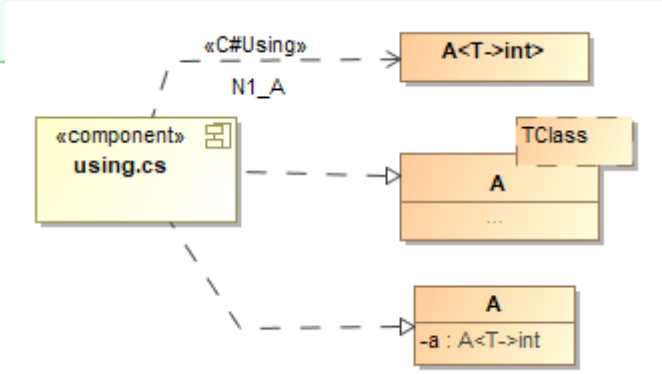
Generic usi

For example, using N1_A = N1.A<int>, we create Template binding for *A<int>* in Namespace *N1*, and then we create the dependency *Usage* from the parent component (in this case it is file component) to the class with template binding.

Example
Code:

```
using N1_A = N1.A<int>; namespace N1
{
public class A<T> {}
}
class A
{
N1_A a;
}
```

Reversed UML model:



Generic constraints

Generic type and method declarations can optionally specify type parameter constraints by including *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*

The list of constraints given in a where clause can include any of the following components, in this order: a single primary constraint, one or more secondary constraints, and the constructor constraint, new().

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

A *primary constraint* can be a class type or the reference type constraint class or the value type constraint struct.

class-type

`class`

`struct`

A *secondary constraint* can be a type-parameter or interface-type:

interface-type

type-parameter

secondary-constraints , *interface-type*

secondary-constraints , *type-parameter*

constructor-constraint:

`new ( )`

Each *type-parameter-constraints-clause* consists of the token *where*, followed by the name of a type parameter, followed by a colon and the list of constraints for that type parameter. There can be at most one where clause for each type parameter, and the where clauses can be listed in any order.

The given list of constraints in a *where* clause can include any of the following components, in this order: a single primary constraint, one or more secondary constraints, and the constructor constraint, new().

Anonymous method

Anonymous method allow code blocks to be written *in-line* where delegate values are expected. Anonymous methods are similar to lambda functions in the Lisp programming language. C# 2.0 supports the creation of *closures* where anonymous methods access surrounding local variables and parameters.

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

Del d = delegate(int k) { /* ... */ };
```

By using anonymous method, you reduce the coding overhead in instantiating delegates by eliminating the need to create a separate method.

Now, there is no direct mapping for Anonymous method, but it uses the mapping for delegate method feature.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Partial types

Partial types allow classes, structs, and interfaces to be broken into multiple pieces stored in different source files for easier development and maintenance.

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.

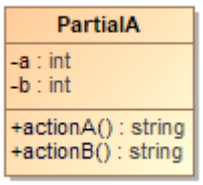
Each part of a partial type declaration must include a Type modifier *partial* and must be declared in the same namespace as the other parts.

All child elements will be merged into one Class element.

Example 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()
    {
    }
}
```

Reversed UML model:



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.

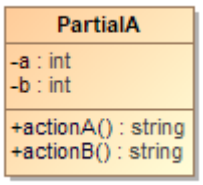
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

All child elements will be merged into one Class element, the same as both classes are written into one Class file.

Example 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()
    {
    }
}
```

Reversed UML model:



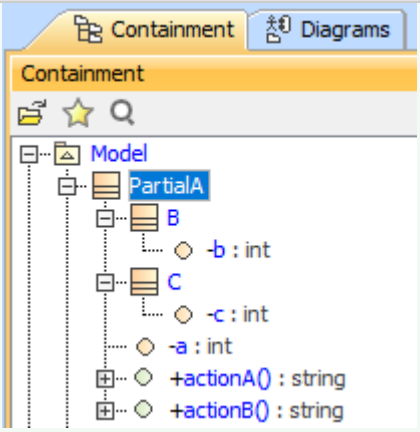
Nested types can be declared in multiple parts by using the Type modifier *partial*. Typically, the containing type is declared using partial as well, and each part of the nested type is declared in a different part of the containing type.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

All inner classes have to be located as inner class of the parent class (partial class).

Example 3

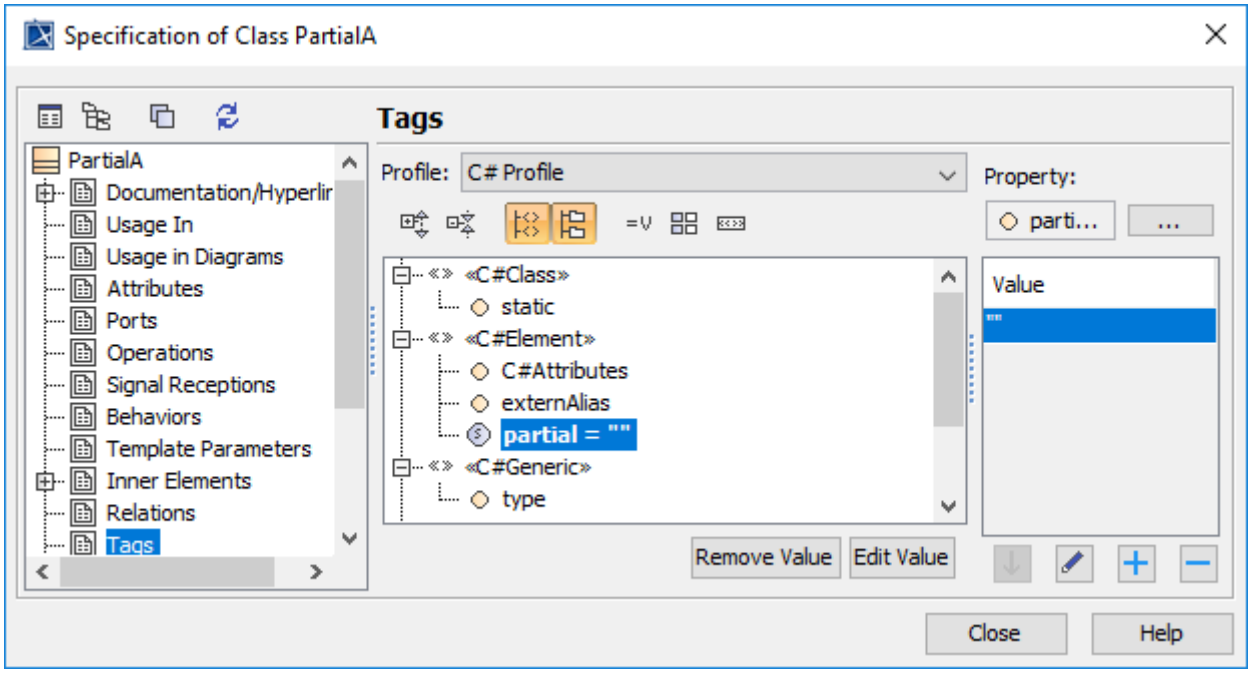
```
//The partial class with inner class
public partial class PartialA{
    public class B
    {
        int b;
    }
}
public partial class PartialA{
    int a;
    string actionB()
    {
    }
    public class C
    {
        int c;
    }
}
```



The type modifier *partial* is not permitted on delegate or enum declarations.

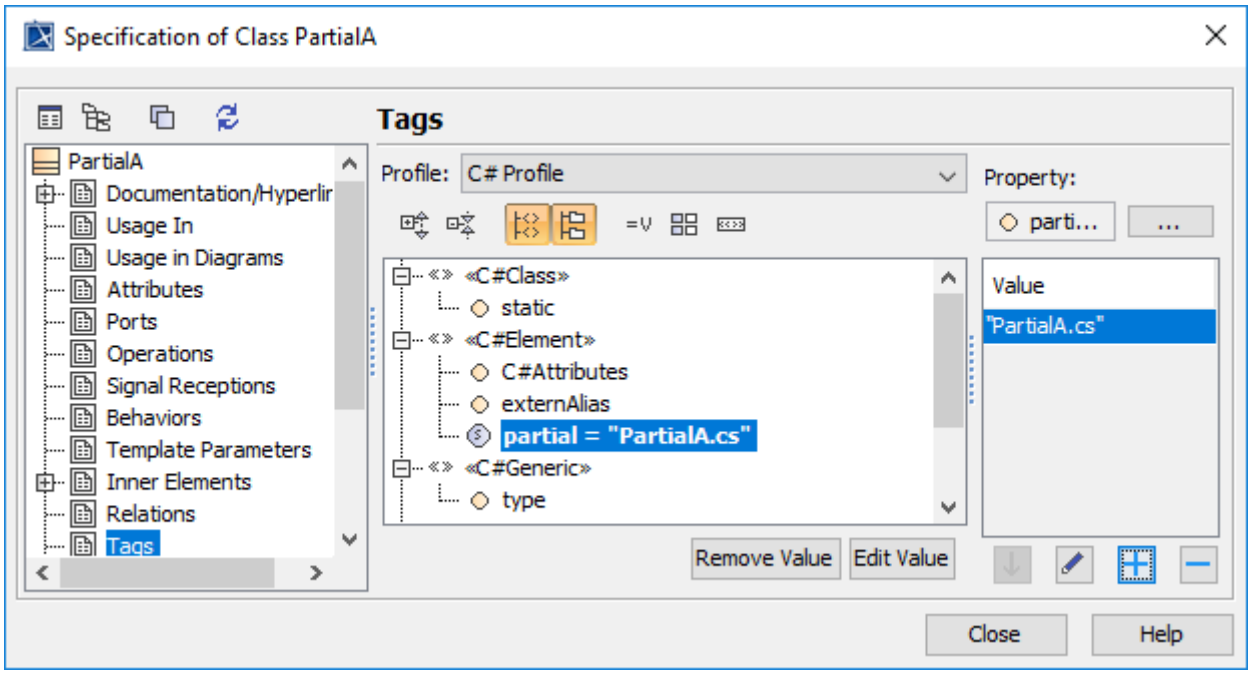
The Tag *partial* will have a blank tagged value for Partial Class Element:

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Partial Class Element with blank Tag partial value

The Tag *partial* will have a value of the file name that the class belongs to for each child element in Partial Class Element:



Partial Class Element with Tag partial "PartialA.cs" value

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Nullable types

Nullable types are constructed using the ? type modifier. For example, int? is the nullable form of the predefined type int. A nullable type's underlying type must be a non-nullable value 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 st r uct constraint). The underlying type of a nullable type cannot be a nullable type or a reference type.

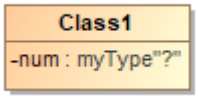
A nullable type can represent all values of its underlying type plus an additional null value.

The syntax *T?* is shorthand for System.Nullable<T>, and the two forms can be used interchangeably.

Add Class *Nullable* type to C# profile.

```
class class1 {
}
int? a = null;
System.Nullable a = null;
}
```

Reversed UML model:



Accessor declarations

The use of *accessor-modifiers* is governed by the following restrictions:

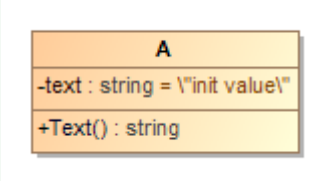
- An *accessor-modifier* may not be used in an interface or in an explicit interface member implementation.
- For a property or indexer that has no override modifier, an *accessor-modifier* is permitted only if the property or indexer has both a get and set accessor, and then is permitted only on one 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 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

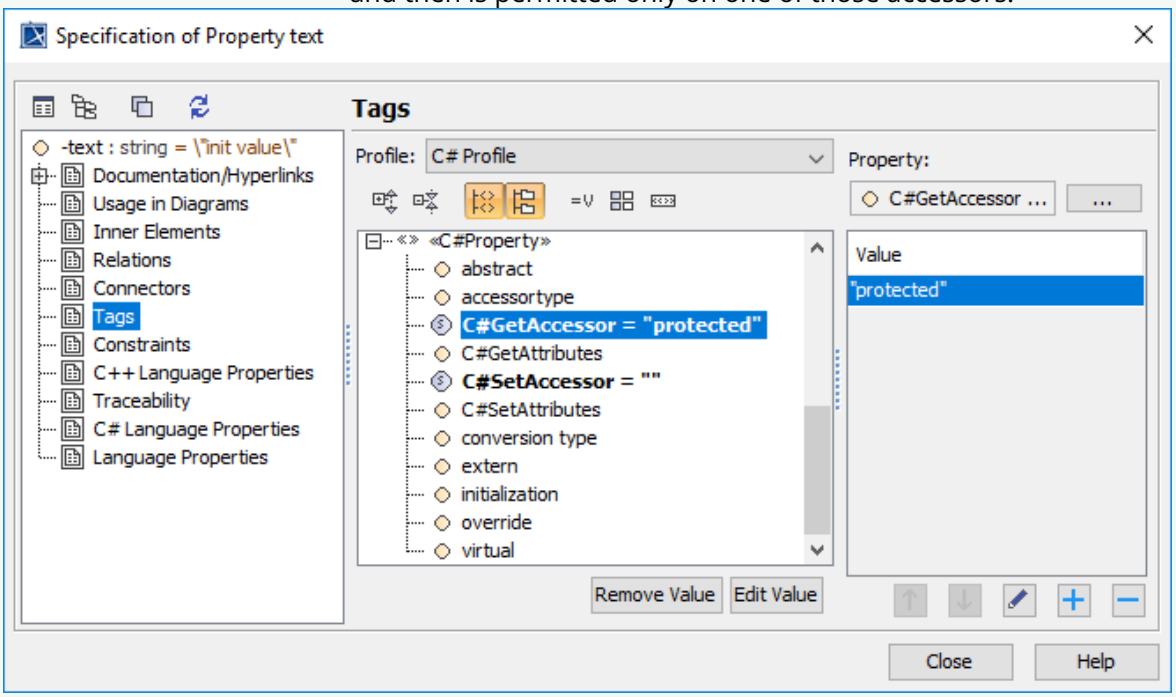
Example

```
Class A
{
private string text = “init value”;
public String Text
{
        protected get{ return text;}
        set{ text = value;}
}
}
```

Reversed UML model:



Example of Tags required when 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:



Property *text* has both a get and set accessor.

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.

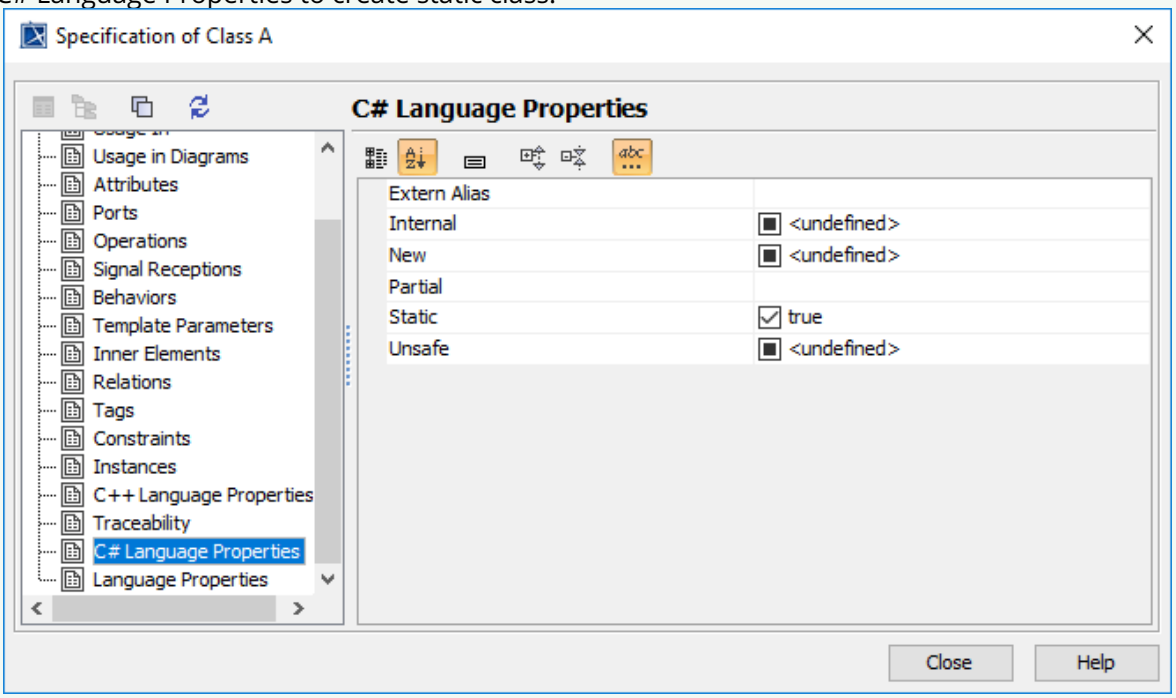
When a class declaration includes a static modifier, the class being declared is said to be a static class.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example
Code:

```
static class A
{
    ...
}
```

C# Language Properties to create static class:



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.

Consider the following two assemblies:

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

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:

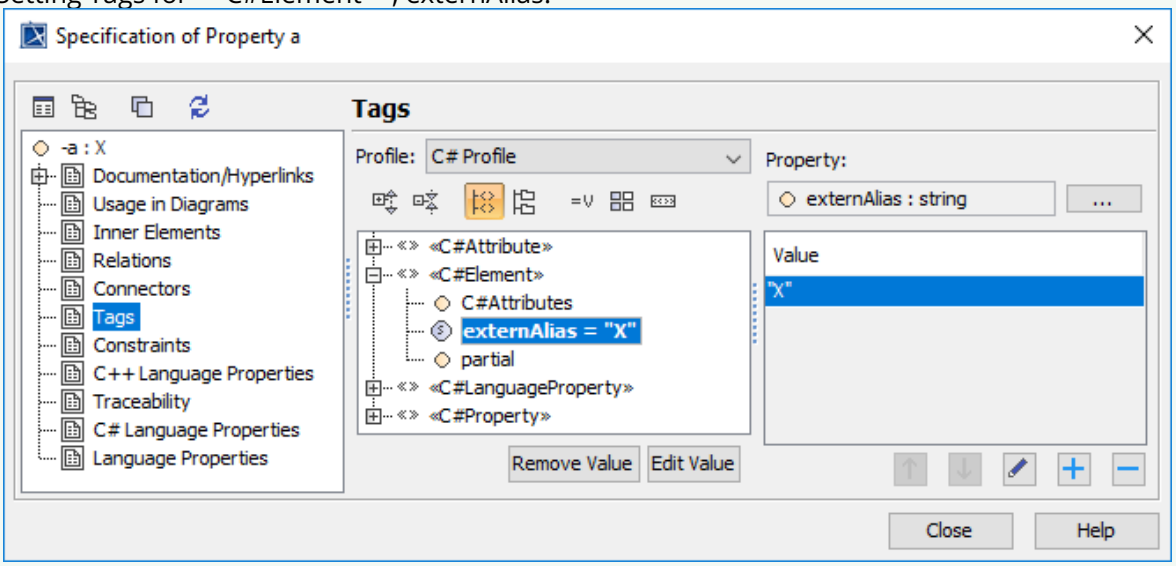
```
class Test

{

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.

Code:

```
extern alias X;
extern alias Y;
class Test
{
    X::N.A a;
    X::N.B b1;
    Y::N.B b2;
    Y::N.C c;
}
```

Setting Tags for <<C#Element>>, externAlias:



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.

A fixed size buffer declaration that declares multiple fixed size buffers is equivalent to multiple declarations of a single fixed sized buffer declaration with the same attributes, and element types.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

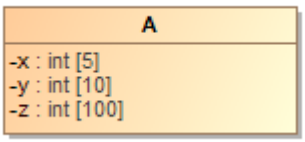
This code:

```
unsafe struct A
{
 public fixed int x[5], y[10], z[100];
}
```

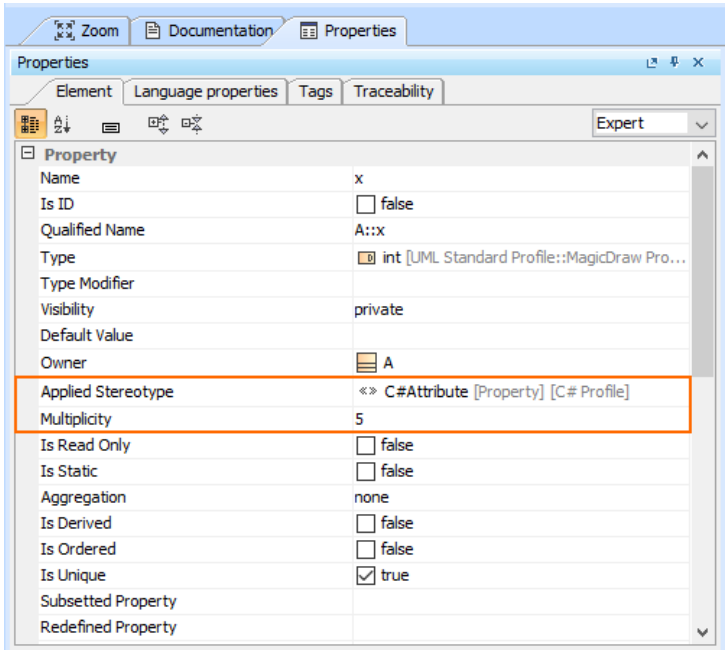
is equivalent to

```
unsafe struct A

{

   public fixed int x[5];

   public fixed int y[10];

   public fixed int z[100];

}
```

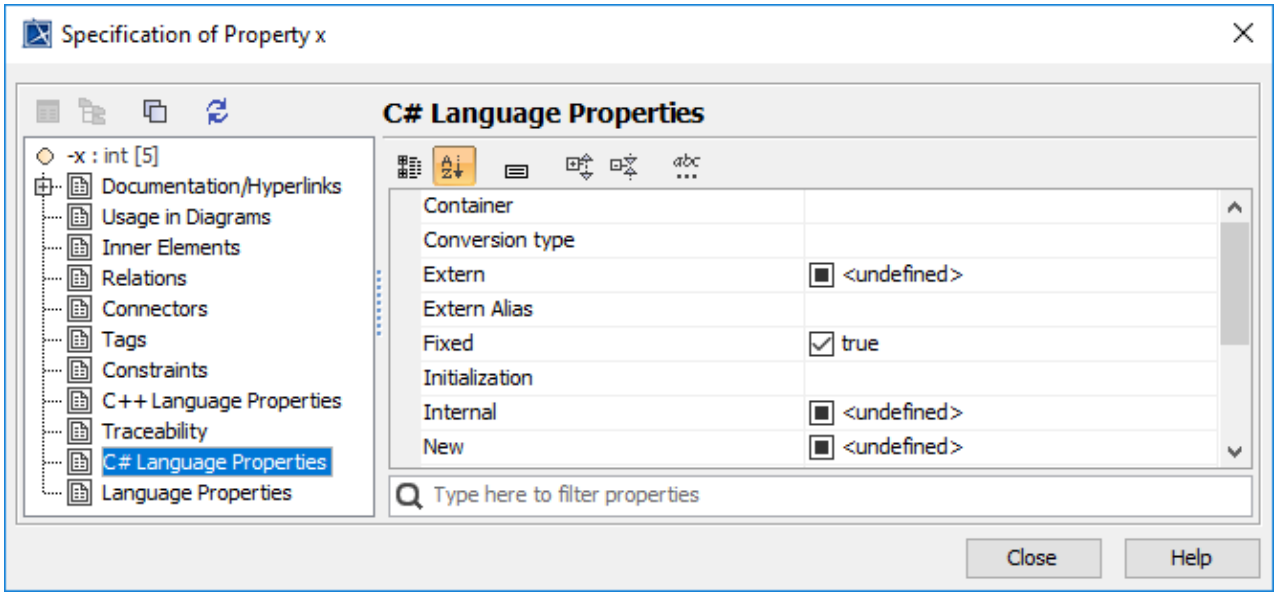
Reversed UML model:



Attribute *x* Element Properties:



To declare fix size buffer, you need in **C# Language Properties** set Property **Fixed** value to *true*:



C# 3.0 features

C# 3.0 introduces several language extensions that build on C# 2.0 to support the creation and use of higher order, functional style class libraries. The extensions enable construction of compositional APIs that have equal expressive power of query languages in domains such as relational databases and XML. In this chapter, these extensions will be described:

- Extension methods(see page 2632)
- Lambda expression conversion(see page 2632)

Related pages

- Code Engineering(see page 2398)
  - Code Engineering Sets(see page 2399)
  - Generating Code(see page 2403)
  - Reverse Options(see page 2406)
  - Global options for Code Engineering(see page 2415)
  - Files of Properties(see page 2426)
  - Java Code Engineering(see page 2427)
  - C++ Code Engineering(see page 2454)
  - C# Code Engineering(see page 2606)
  - CORBA IDL Mapping To UML(see page 2680)
  - WSDL(see page 2683)

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.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

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 non-generic, non-nested static classes.

The following is an example of a static class that declares two extension methods:

```
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 =newT[count];
        Array.Copy(source, index, result, 0, count);
        return result;
    }
}
```

It becomes possible to invoke the extension methods in the static class **Extensions** using instance method syntax:

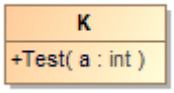
Instance method syntax

```
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)
```

Example
Code:

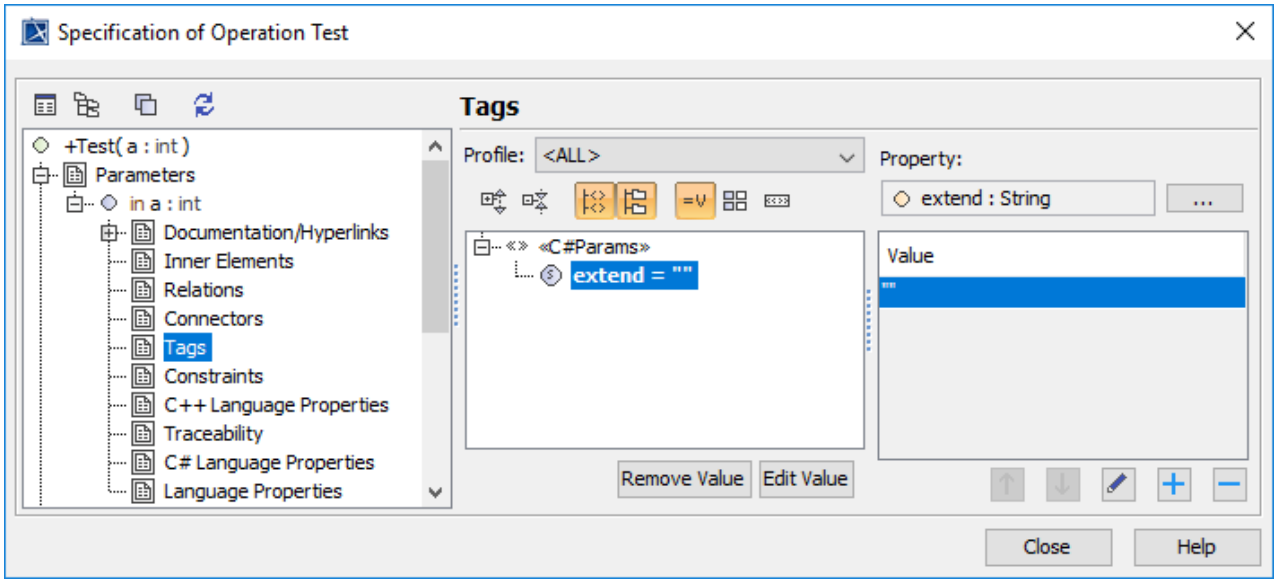
```
class K
{
        Test (this int a)
        {}
}
```

Reversed UML model:



The value is created to Tag *extend* in «C#Params»:

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Lambda expression conversion

A lambda expression is written as a parameter list, followed by an expression or a statement block, the basic form:

```
(input-parameters) => expression
```

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.

Examples of lambda expressions

```
x => x + 1                              // Implicitly typed, expression body
x => { return x + 1; }              // Implicitly typed, statement body
(int x) => x + 1                        // Explicitly types, 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.

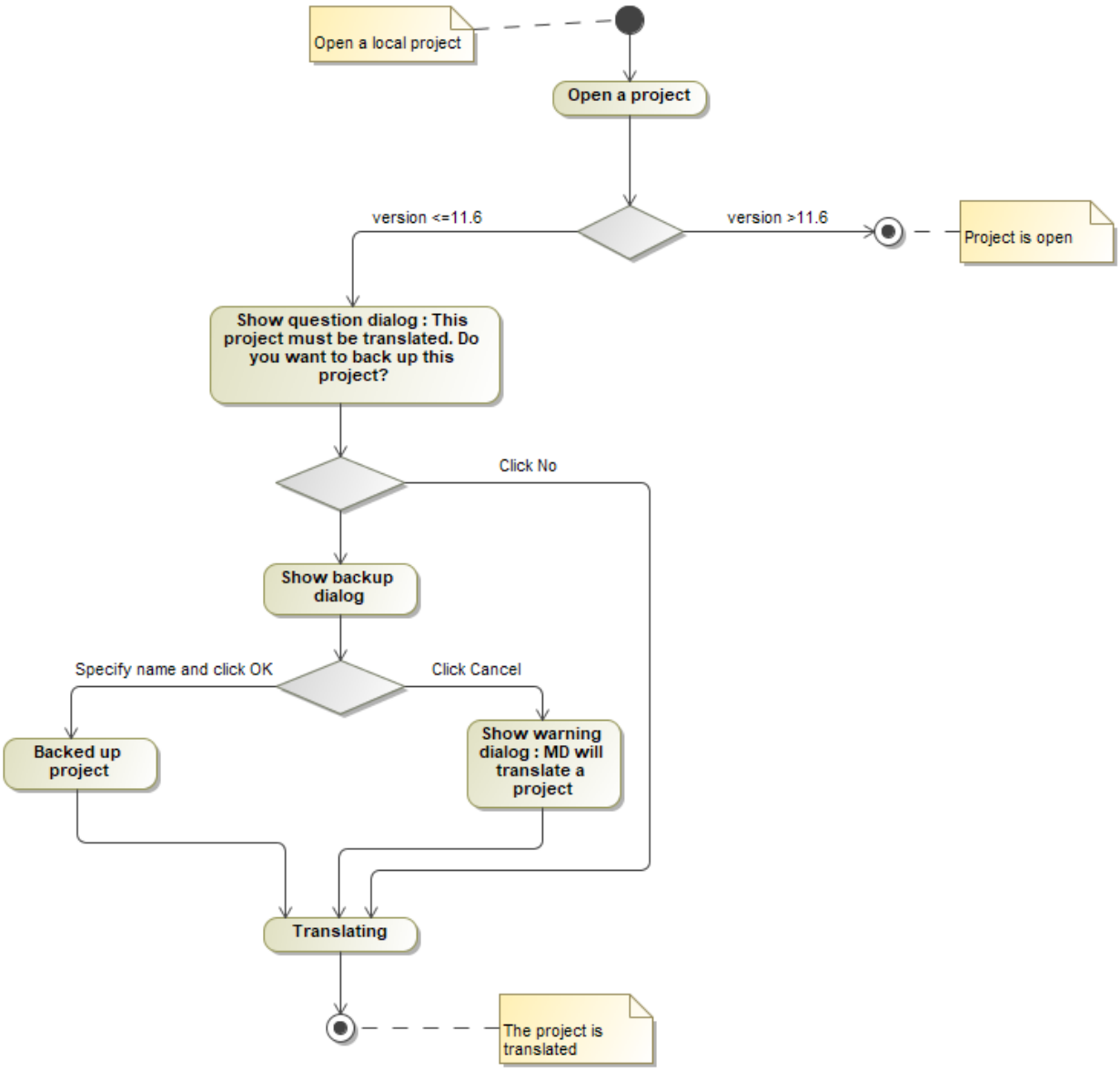
Migration from old C# project version

Projects created with MagicDraw version 11.6 or earlier and using C# programming language properties, profiles, or having type modifiers must be converted with a later version of MagicDraw.

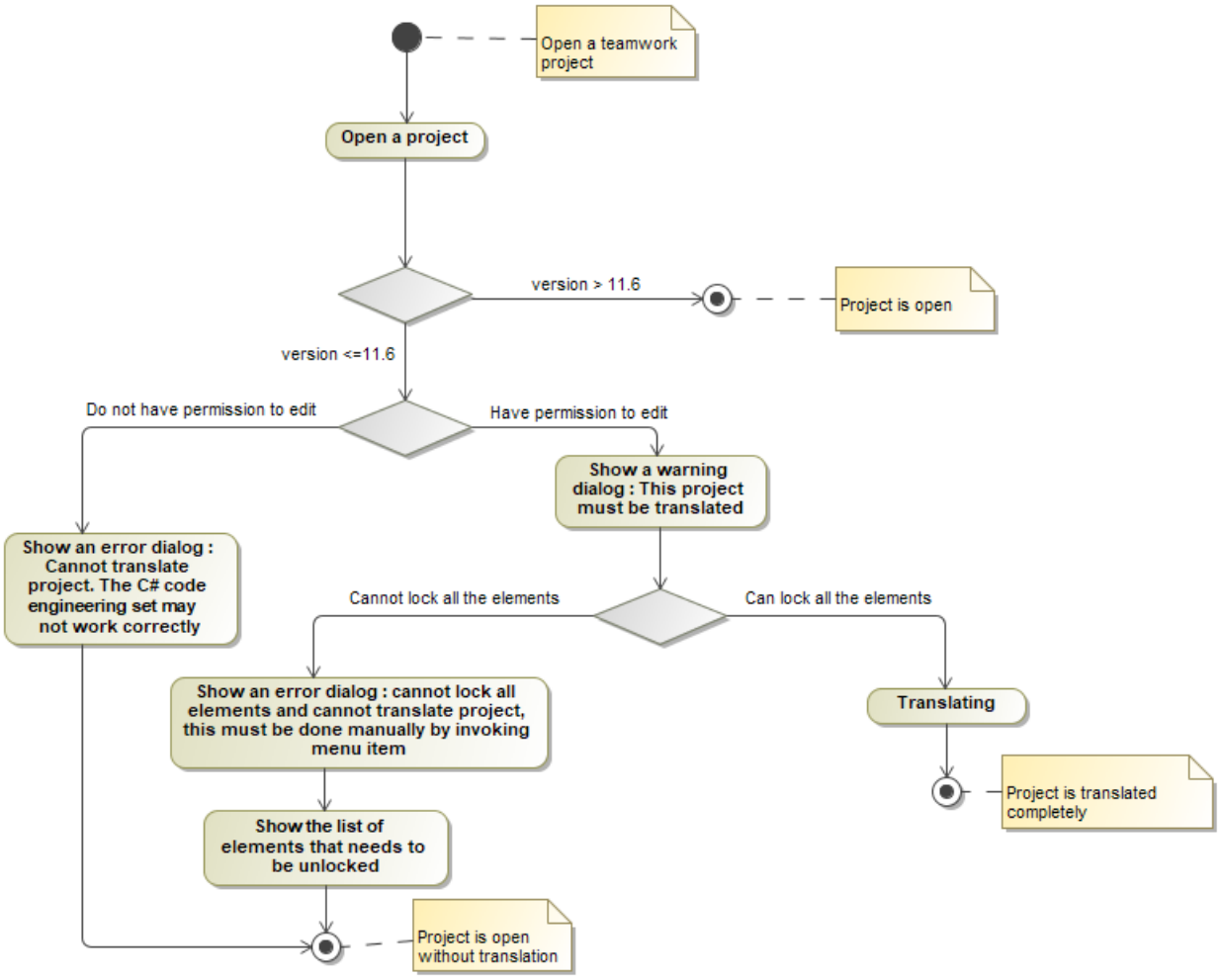
You can see the conversion process in the following diagrams.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Migrate local project

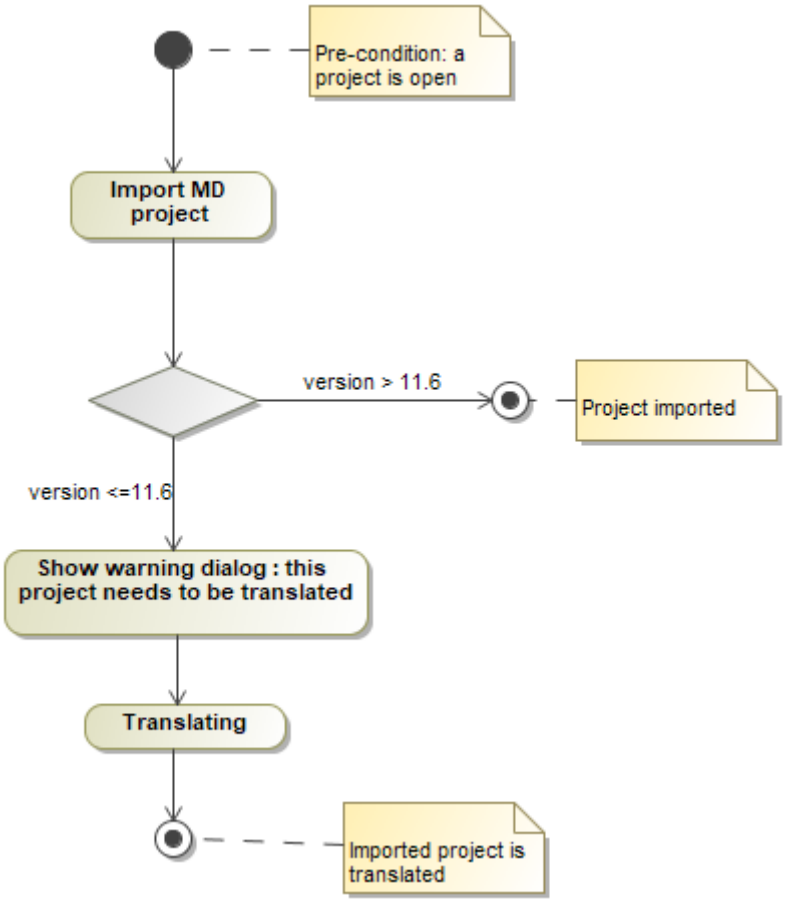


Migrate Teamwork project



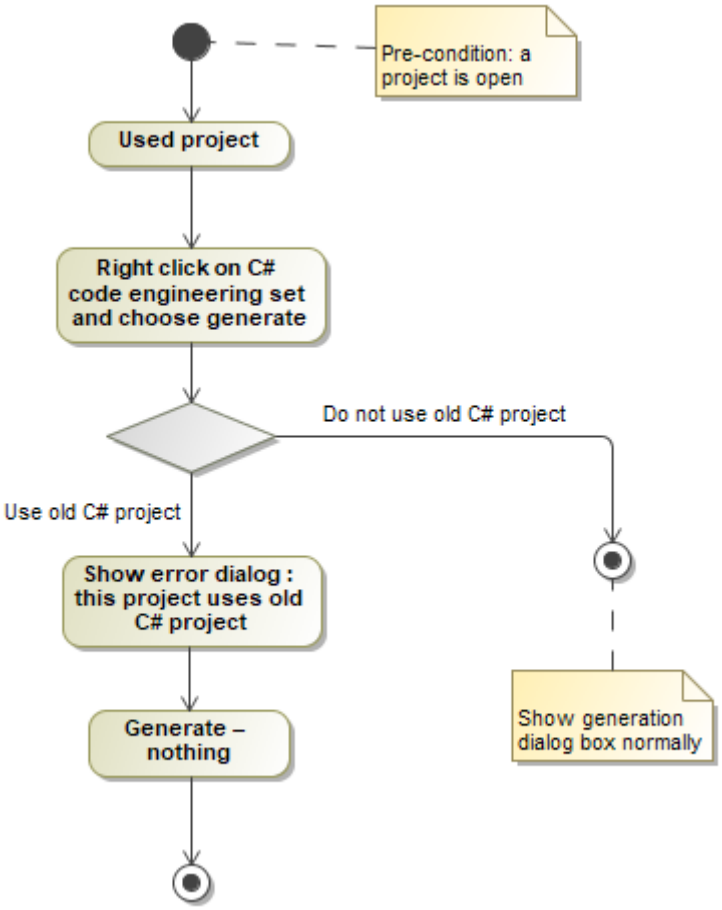
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Import and migrate MagicDraw Project

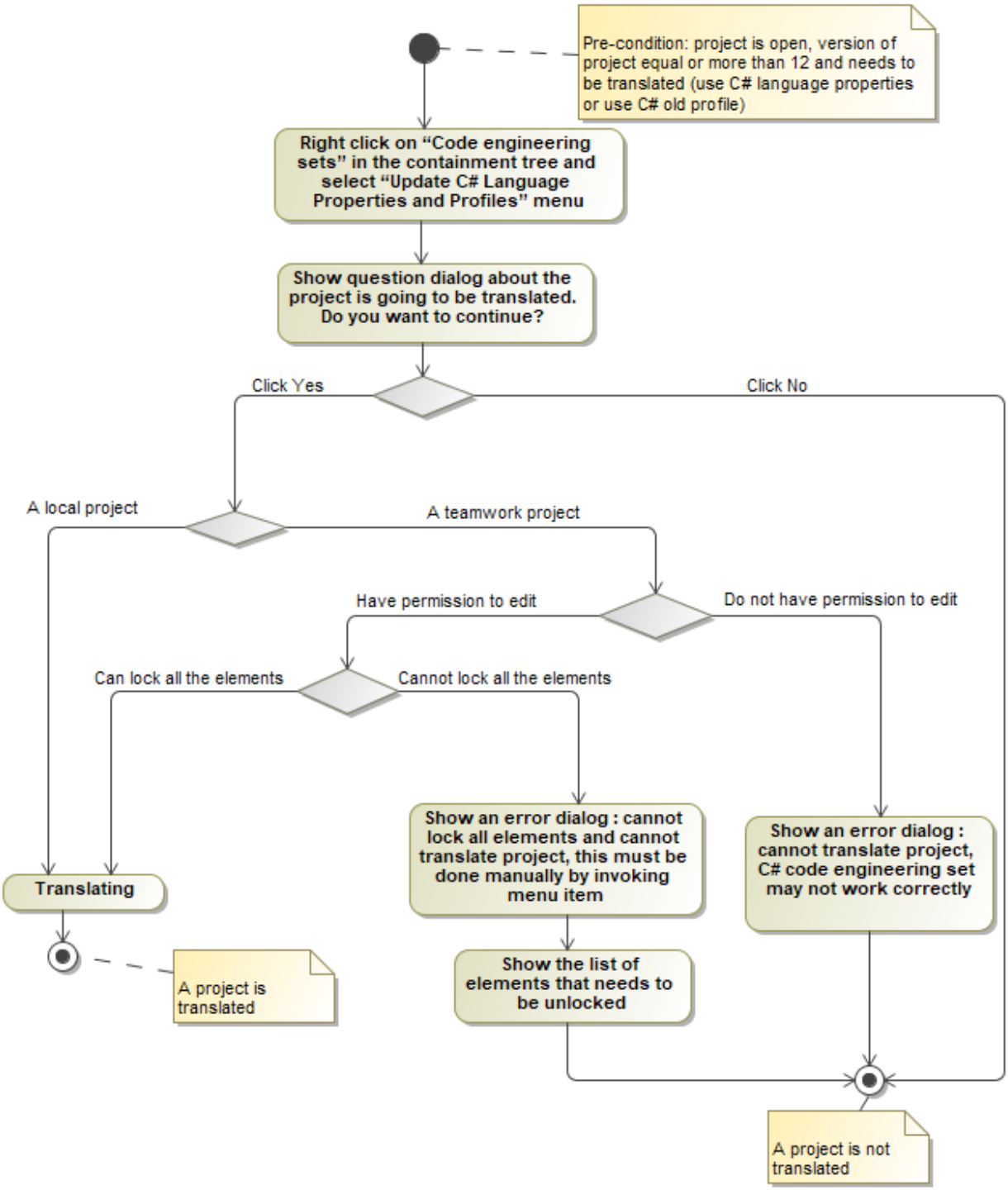


Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Used Project



Update C# Language Properties and Profiles



C# mapping to UML rules

In class diagrams, UML is used to represent C# elements. Class diagrams represent not only C# classes, but other C# elements such as attributes, operations.

A stereotype is an extension mechanism that broadens the vocabulary of the UML and gives more specific meaning to a C# class and other elements.

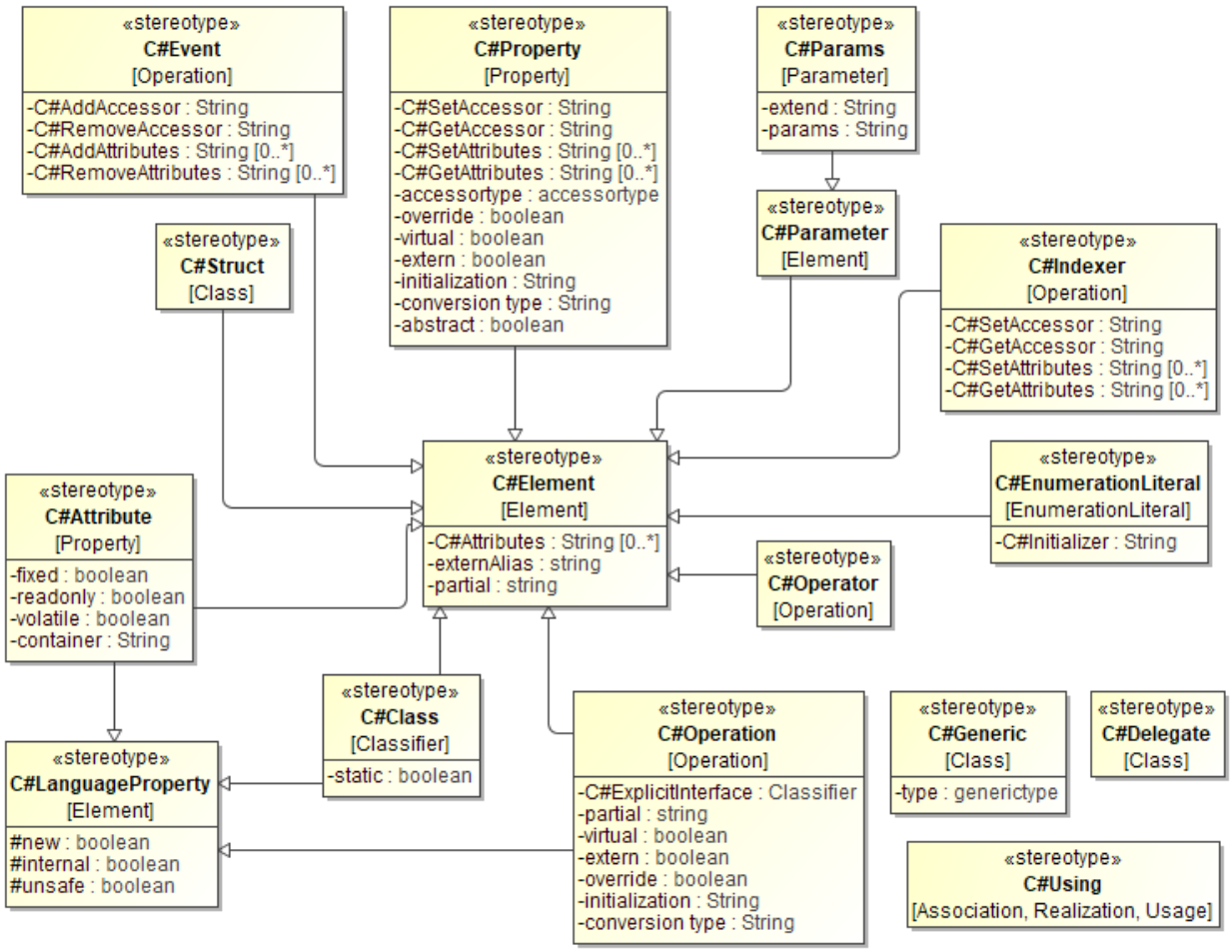
In this chapter summarizes the mapping between C# elements and UML elements in class diagrams.

Related pages

- C# stereotypes(see page 2656)
- C# data type(see page 2665)
- C# Language Properties(see page 2666)
- C# Properties Customization(see page 2669)
- Using Directive Mapping(see page 2671)
- Code Engineering(see page 2398)
  - Code Engineering Sets(see page 2399)
  - Generating Code(see page 2403)
  - Reverse Options(see page 2406)
  - Global options for Code Engineering(see page 2415)
  - Files of Properties(see page 2426)
  - Java Code Engineering(see page 2427)
  - C++ Code Engineering(see page 2454)
  - C# Code Engineering(see page 2606)
  - CORBA IDL Mapping To UML(see page 2680)
  - WSDL(see page 2683)

C# stereotypes

The diagram below shows stereotypes of the C# profile.



C# Stereotypes with Tag values

Now our modeling tool`s C# Profile language properties is moved to tagged value in a specific stereotypes. Language properties were moved:

- Class is extended with Stereotype «C#Class»
- Attribute is extended with Stereotype «C#Attribute»
- Operation is extended with Stereotype «C#Operation»
- The common information of the old language properties (Class, Attribute and Operation) is extended with Stereotype C#LanguageProperty

Here you will find description about:

- C#Attribute(see page 2641)
- C#Class(see page 2641)
- C#Delegate(see page 2641)
- C#Element(see page 2641)
- C#EnumerationLiteral(see page 2641)
- C#Event(see page 2641)
- C#Generic(see page 2641)
- C#Indexer(see page 2641)
- C#LanguageProperty(see page 2641)

- C#Operation(see page 2663)
- C#Operator(see page 2663)
- C#Parameter(see page 2664)
- C#Params(see page 2664)
- C#Property(see page 2664)
- C#Struct(see page 2665)
- C#Using(see page 2665)

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
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;`
		`}`

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

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
C#Class	Class

Tag	Type	Description

Static	boolean	Represents the usage of static class
		
		`static Class A`
		`{`
		`public int x;`
		`}`

C#Delegate

«C#Delegate» is a stereotype that indicates that model class represents C# delegate.

Name	Meta class
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.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

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
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.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Name	Meta class
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.

C#Generic

«C#Generic» is a stereotype that is used to define generic properties.

Name	Meta class
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.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Name	Meta class
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
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
C#Operation	Operation

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
C#Operator	Operation

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

C#Parameter

«C#Parameter» is a stereotype that is used to indicate that the element represents C#Parameter.

Name	Meta class
C#Parameter	Element

C#Params

«C#Params» is a stereotype that is used to indicate parameter arrays.

Name	Meta class
C#Params	Parameter

Tag	Type
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
C#Property	Operation

Tag	Type	Description
C#GetAccessor	String	Adds get accessor for indexer.

Tag	Type	Description
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#Struct

«C#Struct» is a stereotype that is used to indicate that the model class represents C# structure.

Name	Meta class
C#Struct	Class

C#Using

«C#Using» is a stereotype that is used to indicate that the usage dependency is C# Using directive.

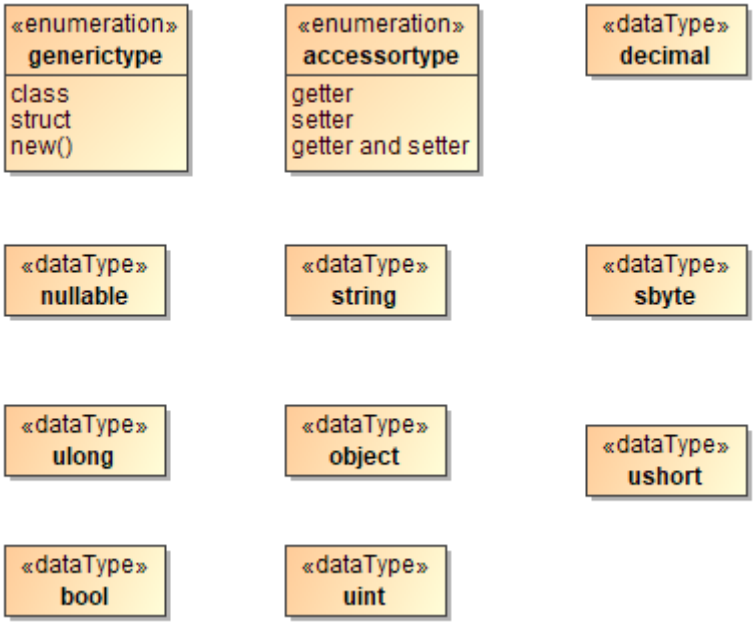
Name	Meta class	Constraints
C#Using	Association, Realization, Usage	The clients of dependency are Component, and Namespace.

Related pages:

- C# stereotypes(see page 2640)
- C# data type(see page 2665)
- C# Language Properties(see page 2666)
- C# Properties Customization(see page 2669)
- Using Directive Mapping(see page 2671)

C# data type

In addition to stereotype and tagged value, C# datatype should be created in the profile. The following figure shows the data types of C# profile.



Related pages

- C# stereotypes(see page 2640)
- C# data type(see page 2649)
- C# Language Properties(see page 2650)
- C# Properties Customization(see page 2650)
- Using Directive Mapping(see page 2650)

C# Language Properties

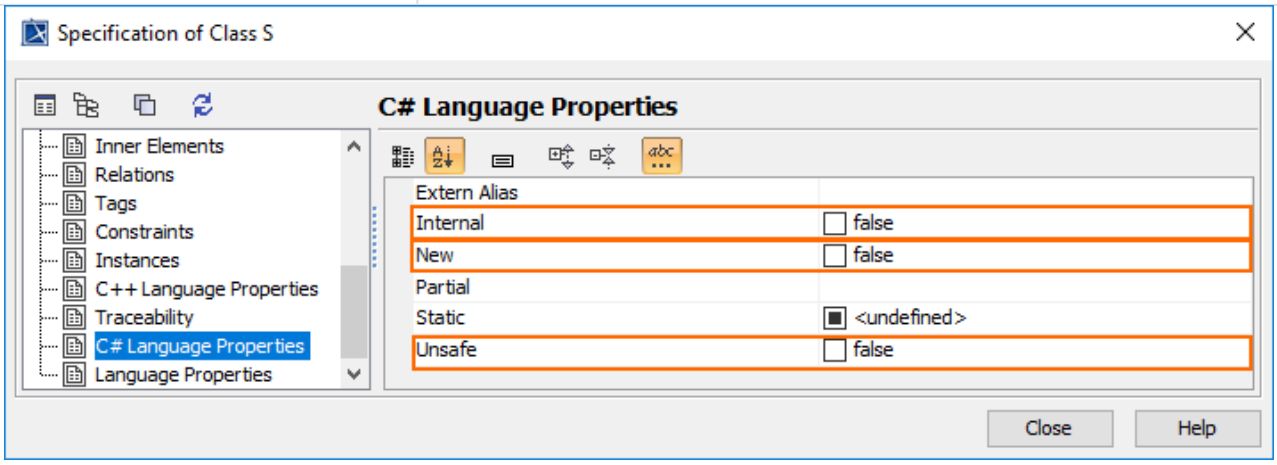
Until MD version 11.6, language properties are stored in a specific format, since MD version 12 language properties are moved to stereotype tag value and using DSL to customize to C# Language Properties. (The language properties will move to C# Language Properties)

Class

Old Value	Translation
New	Mapped to the tag “New” of «C#LanguageProperty» (The usage for this is to apply «C# Class» and set value to tag “new”)
Internal	Mapped to the tag “Internal” of «C#LanguageProperty» (The usage for this is to apply «C# Class» and set value to tag “internal”)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Old Value	Translation
Unsafe	Mapped to the tag “Unsafe” of «C#LanguageProperty» (The usage for this is to apply «C# Class» and set value to tag “unsafe”)



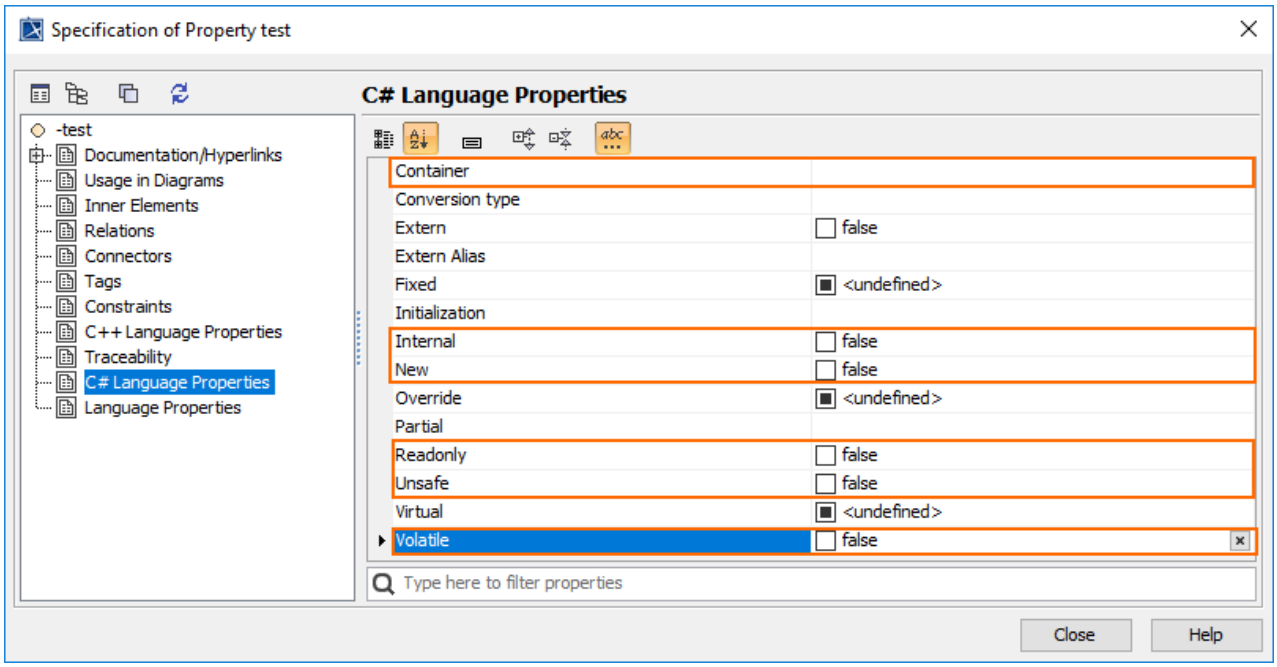
C# Language properties for Class

Attribute

Old Value	Translation
New	Mapped to the tag “New” of «C#LanguageProperty» (The usage for this is to apply «C#Attribute» and set value to tag “new”)
Internal	Mapped to the tag “Internal” of «C#LanguageProperty» (The usage for this is to apply «C#Attribute» and set value to tag “internal”)
Unsafe	Mapped to the tag “Unsafe” of «C#LanguageProperty» (The usage for this is to apply «C#Attribute» and set value to tag “unsafe”)
Readonly	Mapped to the “Readonly” of «C#Attribute»
Volatile	Mapped to the “Volatile” of «C#Attribute»

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Old Value	Translation
Container	Mapped to the “Container” of «C#Attribute»



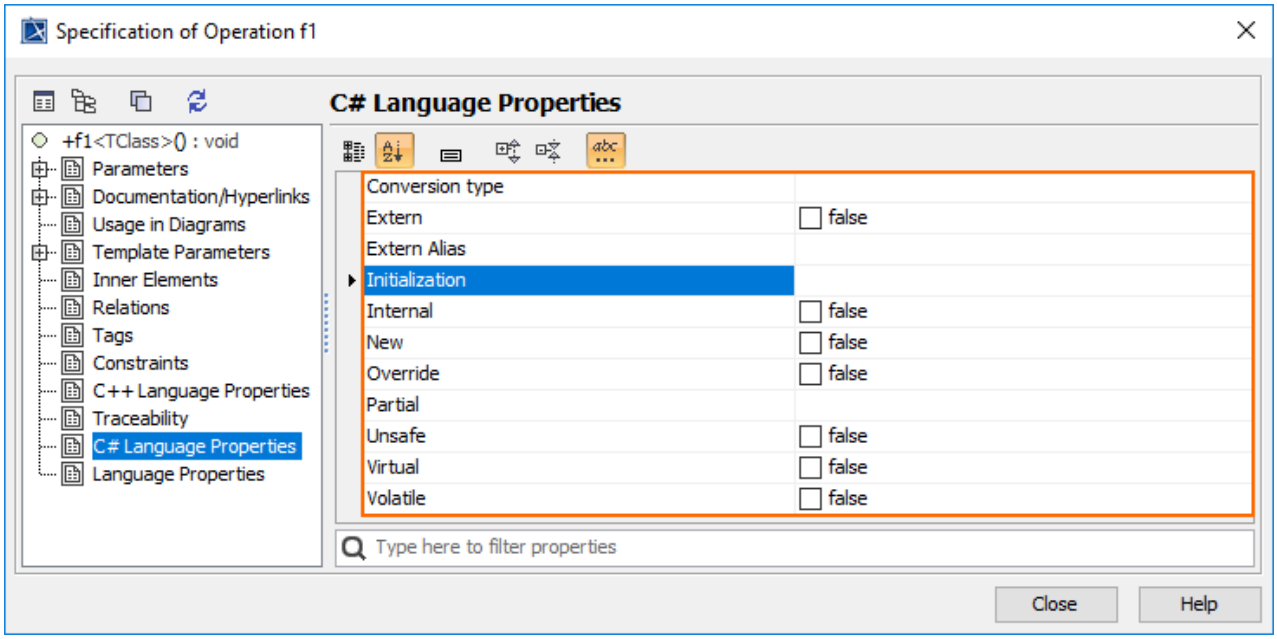
C# Language properties for Attribute

Operation

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»

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Old Value	Translation
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»



C# Language properties for Operation

Related pages

- C# stereotypes(see page 2640)
- C# data type(see page 2649)
- C# Language Properties(see page 2650)
- C# Properties Customization(see page 2653)
- Using Directive Mapping(see page 2653)

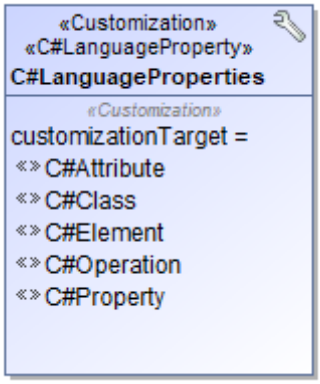
C# Properties Customization

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.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

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.

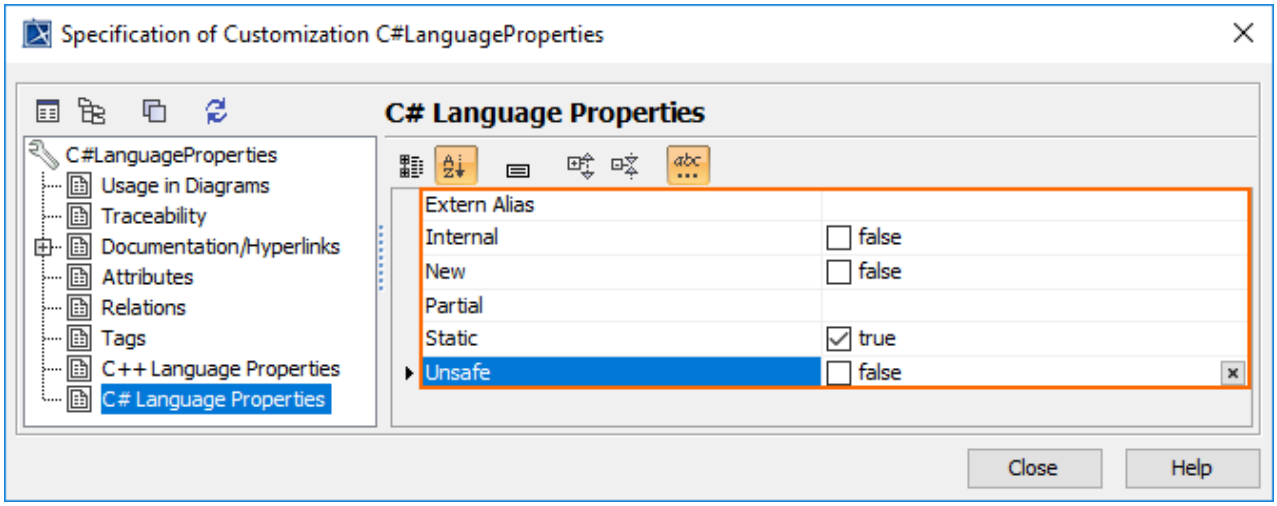
For an old project containing old C# language properties, after performing open or import, they will be translated to the appropriate stereotype and tagged value.



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 customization in MD GUI and behavior:



Related pages

- C# stereotypes(see page 2640)
- C# data type(see page 2649)
- C# Language Properties(see page 2650)
- C# Properties Customization(see page 2653)
- Using Directive Mapping(see page 2654)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Using Directive Mapping

In MagicDraw version 12.1, the C# Using directive is mapped to the model as usage dependency with «C#Using» stereotype.

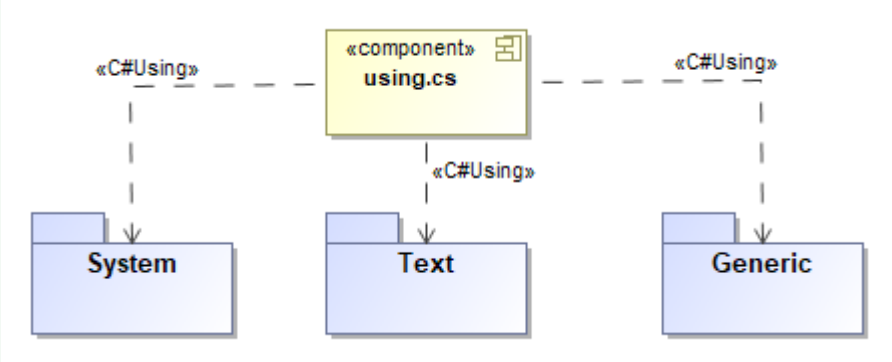
C# Using Namespace

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.

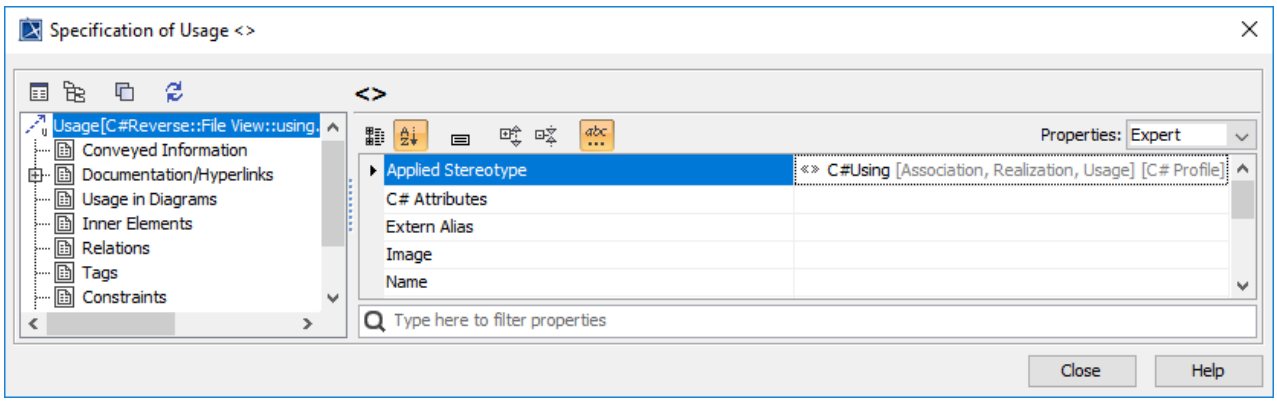
Example
Code:

```
using System;
using System.Collections.Generic;
using System.Text;
namespace n0
{
class using1 {}
}
```

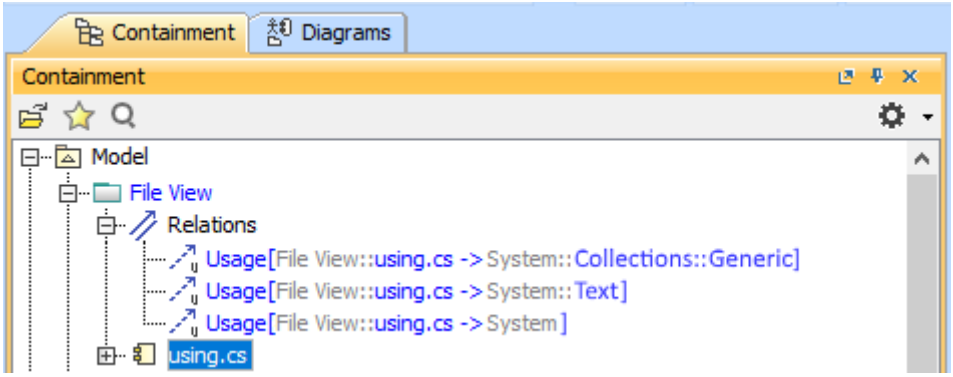
Reversed UML model:



Open usage dependency specification apply «C#Using» stereotype. For using namespace, leave the name empty:



Applying «C#Using» stereotype



Relations in the containment tree

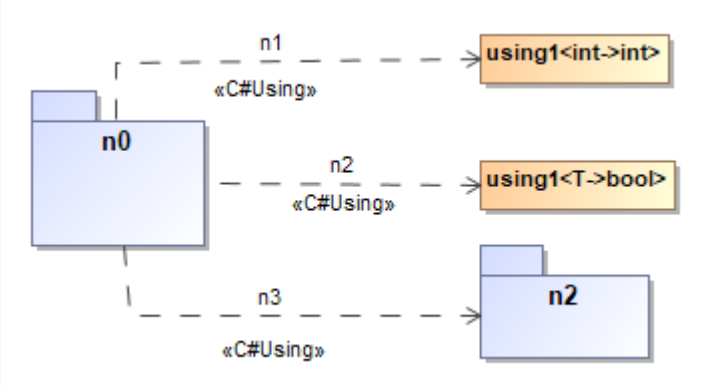
C# Using alias

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.

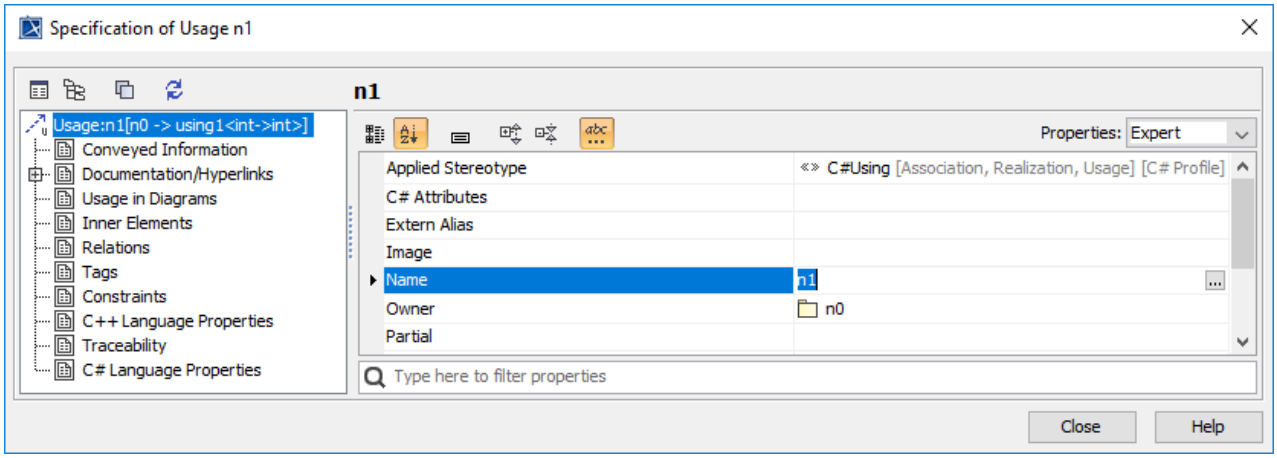
Example
Code:

```
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> { }}
```

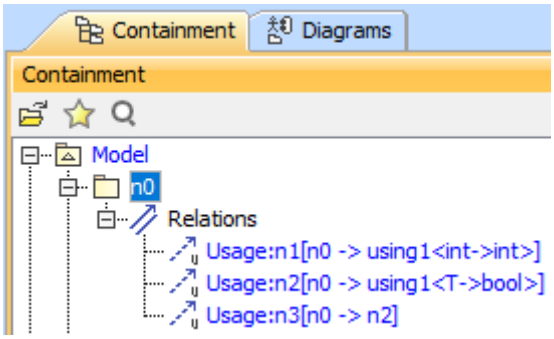
Reversed UML model:



Open usage dependency specification apply «C#Using» stereotype. For using alias, enter the alias name:



Applying «C#Using» stereotype



Relations in the containment tree

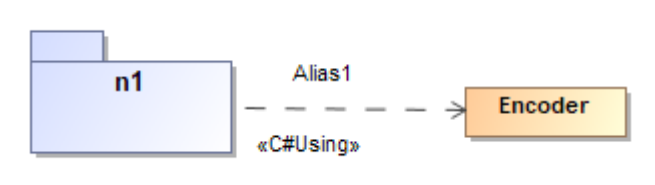
C# Partial feature

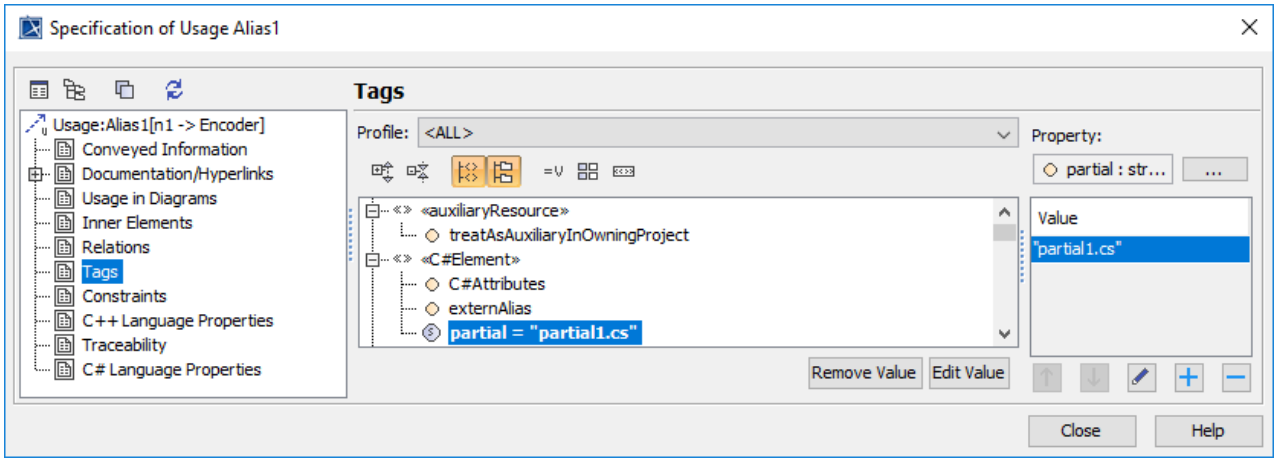
For mapping C# Using directive in the C# Partial feature, add file component name into **Partial Tag** value of usage link specification.

Example
Code for creating value partial1.cs

```
//partial1.cs
namespace n1 {
using LL =
System.Text.Encoder;
}
partial class using1
{
LL a;
}
```

Reversed UML model:





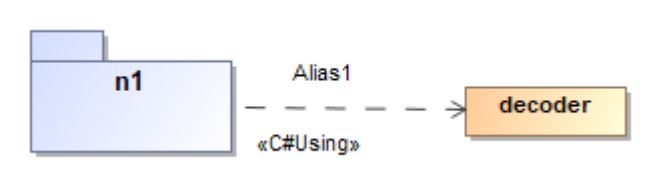
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

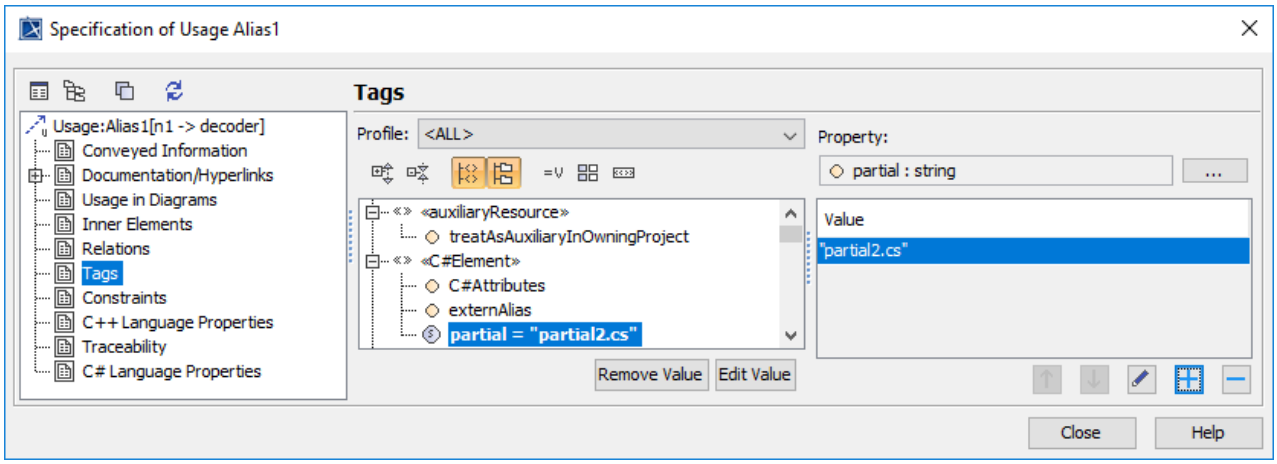
Partial Tag with value partial1.cs

Example
Code for creating value partial2.cs

```
//partial2.cs
namespace n1 {
using LL =
System.Text.Decoder;
}
partial class using1
{
LL b;
}
```

Reversed UML model:





Partial Tag with value partial1.cs

Related pages

- C# stereotypes(see page 2640)
- C# data type(see page 2649)
- C# Language Properties(see page 2650)
- C# Properties Customization(see page 2653)
- Using Directive Mapping(see page 2655)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

C# Project Constraints

This section describes constraints which applies C# projects in modeling process.

Related pages:

- UML constraints(see page 2660)
- Translation Constraints(see page 2660)

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.

Code 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.

When creating manually Class *A* in the model, it would show the error message when trying to create Class *A*.

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.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Case #1

Code used for reverse engineer

```
//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()
       {
       }
}
```

Code after generation

```
public partial class PartialA{
int a;
int b;
string actionA( )
{return ;}
string actionB( )
{return ;}
}
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Case #2

Code used for reverse engineer

```
//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()
       {
       }
}
```

Code after generation

```
public partial class PartialA{
int a;
int b;
string actionA( )
{ return ;}
string actionB( )
{return ;}
}
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Case #3

Code used for reverse engineer

```
//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;
       }
}
```

Code after generation

```
public partial class PartialA{
int a;
string actionB( )
{return ;}
public class B
{int b;}
public class C
{int c;}
}
```

Translation Constraints

Property Translation

From version 12.1 MagicDraw C# translates C#Property from operation to attribute. After translation, the message window will show the following message:

Information message
`Orphaned proxy found in module "C#_Profile.xml". Use search functionality with option "Orphaned proxies only" to locate them.`

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

This message occurrs, 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 appear again.

CORBA IDL Mapping To UML

Reference:

http://www.omg.org/technology/documents/profile_catalog.htm#UML_for_CORBA.

CORBA IDL mapping to UML is based on the UML Profile for CORBA Specification, Version 1.0, April 2002.

> **Availability**
> This functionality is available in the Enterprise edition only.

The differences between the UML Profile for CORBA specification and mapping in MagicDraw are listed below.

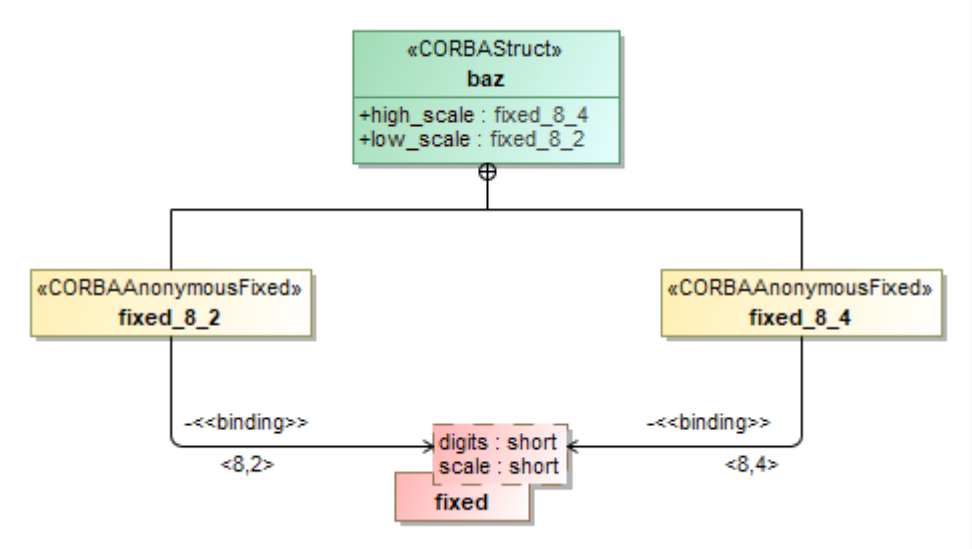
- Constraints defined in the UML Profile for CORBA specification are not checked in MagicDraw.
- The stereotype CORBAAnonymousFixed is introduced. It is used to represent anonymous fixed types of a stereotype. Fixed types without names are mapped to inner classes with the stereotype CORBAAnonymousFixed. These classes are bound to CORBA::fixed classes.IDL Code:

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example
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 that simplifies the creation of standard CORBA IDL elements. The following elements are available in the CORBA IDL diagram:

Button	Shortcut Key	Model Element
	M	CORBA IDL Module
	I	CORBA IDL Interface
	V	CORBA IDL Value

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Button	Shortcut Key	Model Element
	Shift + P	Class by Pattern
	G	Generalization
		Truncatable Generalization
		Value Support Generalization
		CORBA IDL Association
	I	Interface

CORBA interface implementation

You can select the UML Interface or the UML Class as a base element for the CORBA Interface. The default element as a base element for CORBA Interfaces is:

- **UML Class**: a class as a base element for CORBA Interfaces appears in projects created in MagicDraw version 17.0.1. The CORBA Interfaces are modeled as classes in these projects.
- **UML Interface**: an interface as a base element for CORBA Interfaces appears in projects created in 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. Thus, you can use the generalization relationship between two CORBA Intarfaces based on a class or between two CORBA Interfaces based on an interface, but not between a CORBA Interface based on a class and a CORBA Interface based on an interface.

To set a base element for CORBA Interfaces

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

1. On the **Options** menu, click **Project**. The **Project Options** dialog opens.

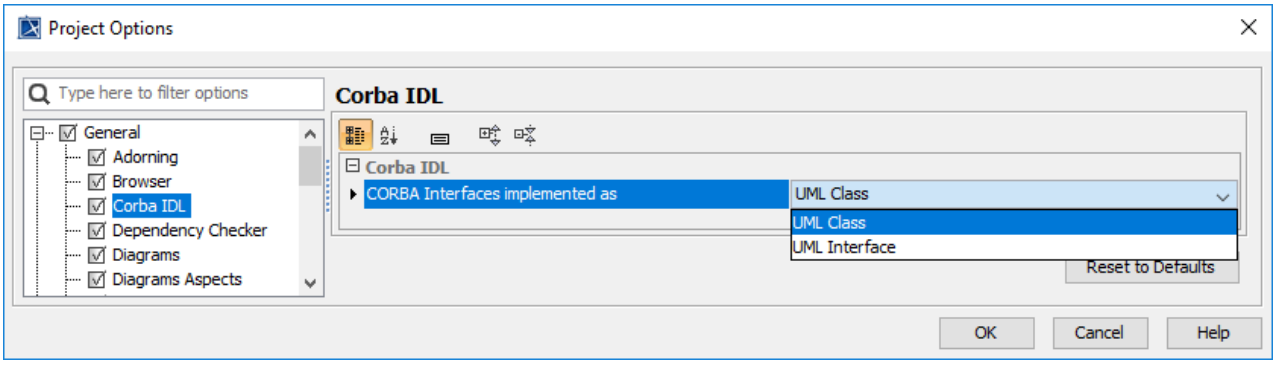
2. In the Tab tree, select **General**.

3. In the **General** list, click the **CORBA IDL** and then **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 of CORBA Interfaces

1. Select the CORBA Interface element in the Model Browser or its symbol on the diagram pane.

2. From the shortcut menu of the selected CORBA Interface, point to **Refactor** > **Convert To**, and then do one of 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.

> **Important!**
> Convert the 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 to the element the CORBA Interface is based on.

For more information about refactoring, see Refactoring[1262].

WSDL

1262 https://docs.nomagic.com/display/MD190/Refactoring

Reference:

http://www.w3.org/TR/2001/NOTE-wsdl-20010315

> This functionality is available in the Enterprise edition only.

WSDL (Web Services Description Language) is an XML format for describing network services as a set of communication endpoints capable of exchanging messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly. They are 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. A WSDL diagram is dedicated for WSDL modeling.

WSDL code engineering contains the WSDL Profile and XML Schema Profile.

WSDL uses the following elements in the definition of network services:

- **types**, providing data type definitions used to describe the messages exchanged.
- **message**, representing an abstract definition of the data being transmitted. A message consists of logical parts, each of which is associated with a definition within a type system.
- **portType**, a set of abstract operations. Each operation refers to an input message and output messages.
- **binding**, specifying concrete protocol and data format specifications for the operations and messages defined by a particular portType.
- **port**, specifying an address for a binding, thus defining a single communication endpoint.
- **service,** used to aggregate a set of related ports.

Related pages:

- WSDL Mapping to UML Elements(see page 2668)
  - Defined stereotypes(see page 2668)
  - Definitions(see page 2668)
  - Import, namespace(see page 2668)
  - Messages(see page 2668)
  - Types(see page 2668)
  - Port types(see page 2668)
  - Bindings(see page 2668)
  - Services(see page 2668)
  - Ports(see page 2668)
- Code Engineering(see page 2398)
  - Code Engineering Sets(see page 2399)
  - Generating Code(see page 2403)
  - Reverse Options(see page 2406)
  - Global options for Code Engineering(see page 2415)
  - Files of Properties(see page 2426)
  - Java Code Engineering(see page 2427)
  - C++ Code Engineering(see page 2454)
  - C# Code Engineering(see page 2606)
  - CORBA IDL Mapping To UML(see page 2664)
  - WSDL(see page 2667)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

WSDL Mapping to UML Elements

This section describes how the WSDL elements map to UML elements in the visual representation of WSDL elements in modeling class diagrams.

Related pages:

- WSDL Mapping to UML Elements(see page 2669)
  - Defined stereotypes(see page 2685)
  - Definitions(see page 2686)
  - Import, namespace(see page 2687)
  - Messages(see page 2688)
  - Types(see page 2690)
  - Port types(see page 2691)
  - Bindings(see page 2694)
  - Services(see page 2696)
  - Ports(see page 2698)

Defined stereotypes

This chapter describes how our modeling tool maps WSDL elements to UML and helps to choose which stereotypes and Tag definitions to use.

WSDL Element	UML element	Stereotype Name	Tag definitions	Details
Definition	Component	WSDLdefinitions	extension - string name - string targetNamespace - string	
Message	Class	WSDLmessage		
Port Type	Interface	WSDLportype		
Binding	Class	WSDLbinding	extension - string	
Port	Instance Specification	WSDLport	extension - string	

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Service	Component	WSDLservice	extension - string	
Type	Component	WSDLtypes	extension - string	
	ElementImport PackageImport	WSDLimport		
	ElementImport PackageImport	xmlns		From the XML Schema Profile
	Package	XSDnamespace		From the XML Schema Profile
	Parameter	WSDLresponse	extension - string	
	Operation	WSDLoperation	extension - string	
	Property	WSDLpart	typing Attribute - string	
	Parameter	WSDLfault	extension - string	
	Parameter	WSDLrequest	extension - string	

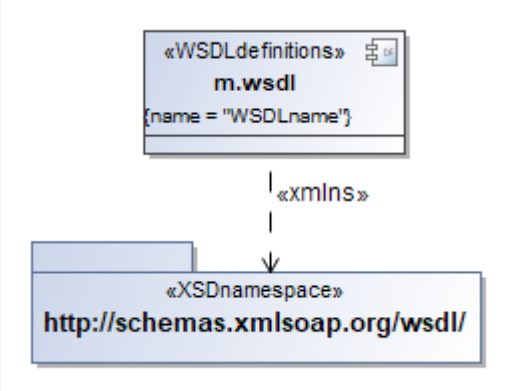
Definitions

A WSDL document is simply a set of definitions, with a definitions element at the root, and definitions inside.

Example
Code:

```
<definitions name="WSDLname" xmlns="http://schemas.xmlsoap.org/wsdl/"/>
```

Reversed UML model

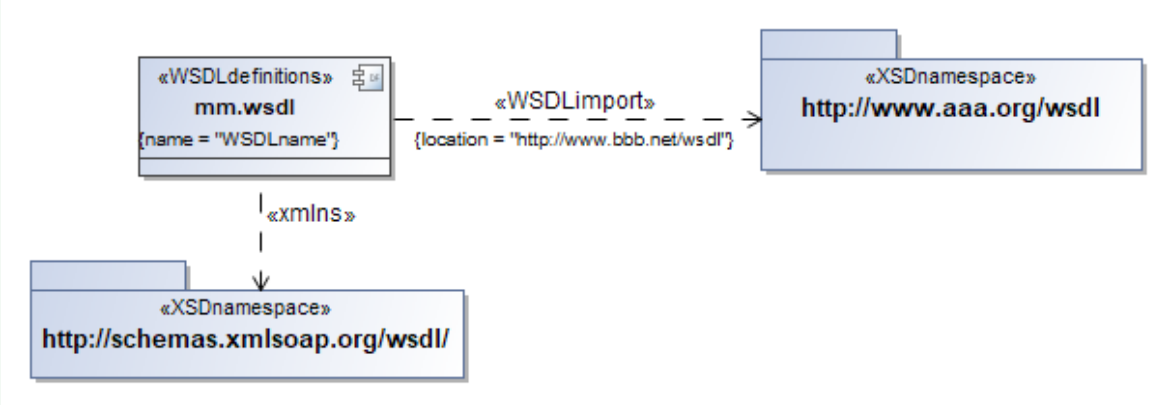


Import, namespace

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:



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Messages

Messages consist of one or more logical parts. Each part is associated with a type from a 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:** an XSD element using a QName.
- **type:** an XSD simpleType or complexType using a QName.

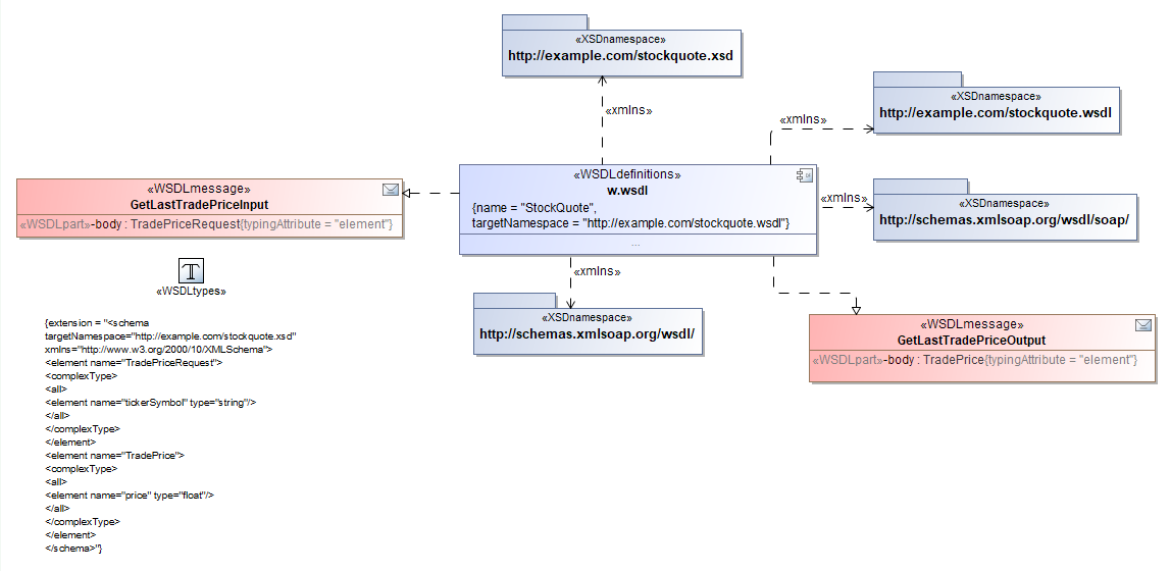
Other message-typing attributes may be defined as long as they use a namespace different from that of WSDL. Binding extensibility elements may also use message-typing attributes.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example
Code

```
<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>
</definitions>
```

Reversed UML model:



Types

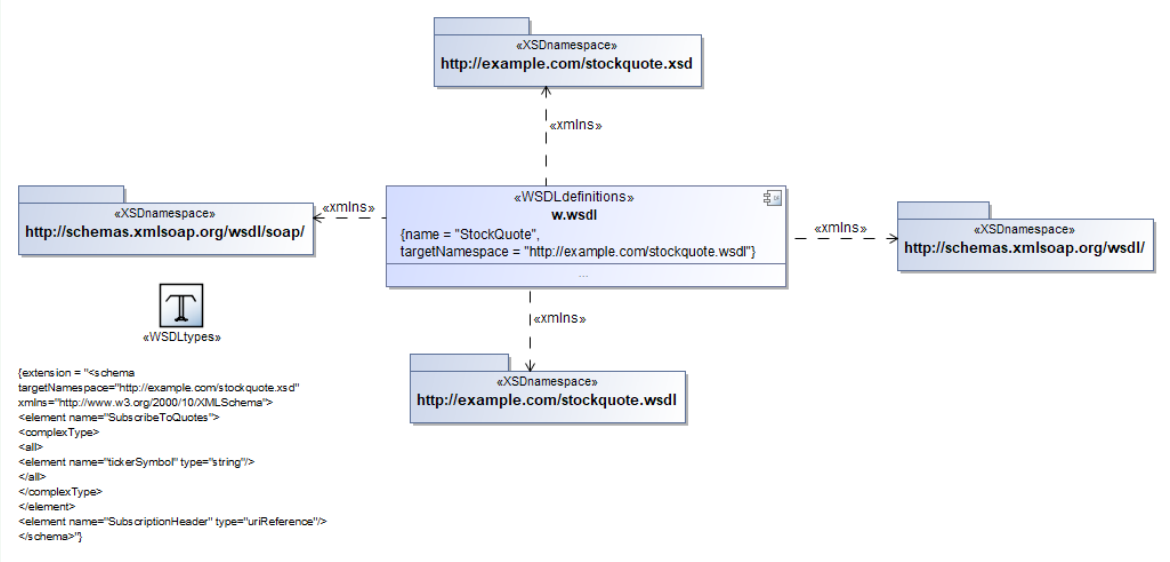
The **types** element encloses data type definitions that are relevant for the exchanged messages. For maximum interoperability and platform neutrality, WSDL prefers the use of XSD as the canonical type system, and treats it as the intrinsic type system.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example
Code:

```
<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:



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 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.See the example below for more information.

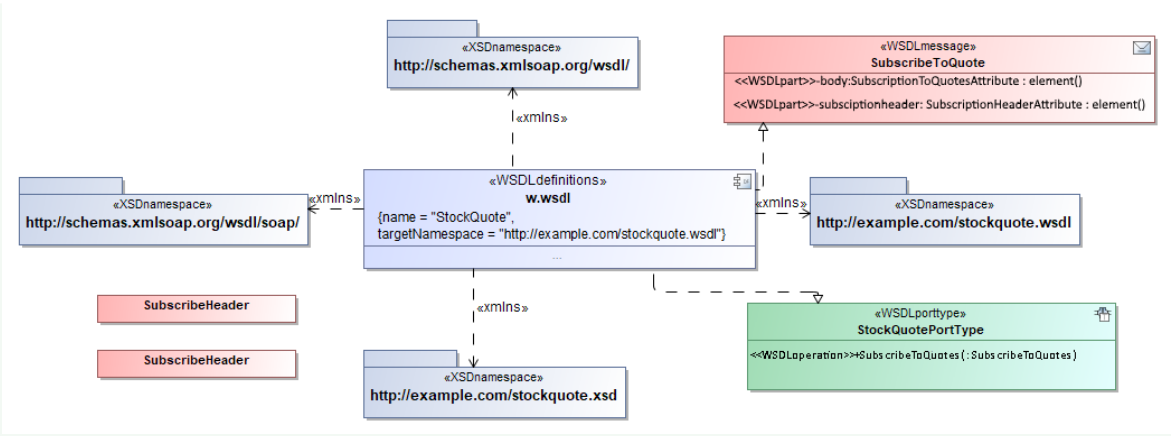
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Example:
Code:

```
<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>
                <all>
                    <element name="tickerSymbol" type="string"/>
                </all>
            </complexType>
        </element>
        <element name="SubscriptionHeader" type="uriReference"/> </schema>
</types>
</definitions>
```

Reversed UML model:

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Bindings

A **binding** defines the message format and protocol details for operations and messages defined by a particular portType. There can be any number of bindings for a given portType.

The **name** attribute provides a unique name among all bindings defined in the enclosing WSDL document.

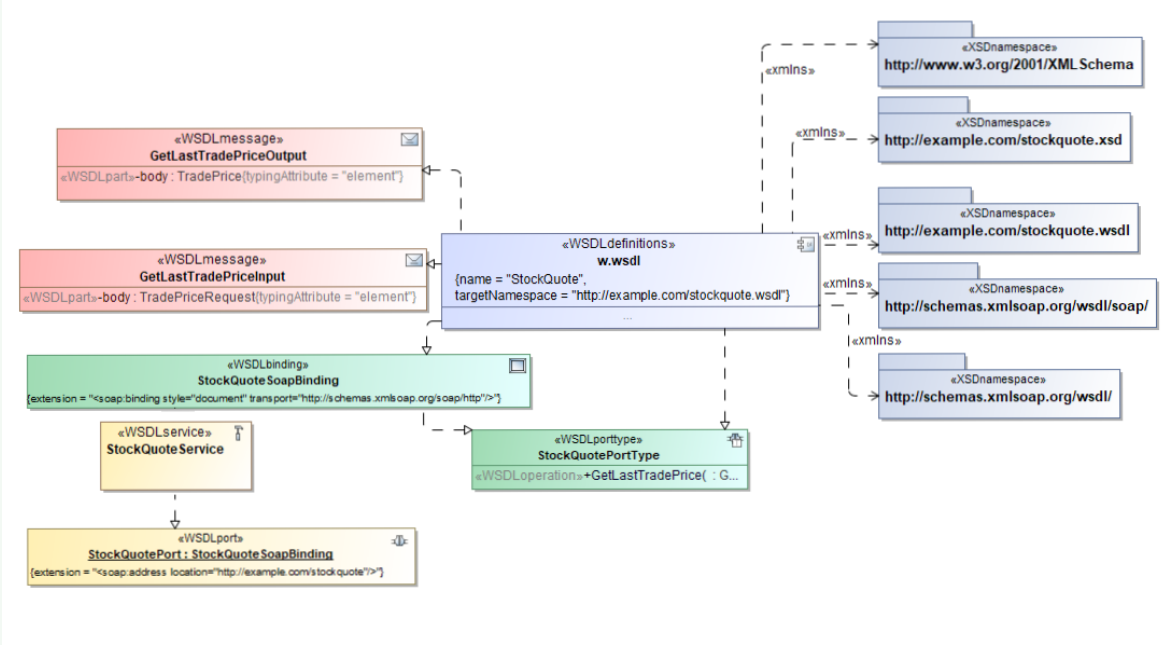
A binding references the portType it binds using the **type** attribute. Binding extensibility elements are used to specify the concrete grammar for the input, output, and fault messages. Per-operation binding information as well as per-binding information can also be specified. See the example below for more information.

Example:
Code:

```
<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"/>
                <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:

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Services

A service groups a set of related ports together.
The name attribute provides a unique name among all services defined in the enclosing WSDL document.

Ports within a service have the following relationships:

- 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 its ports, we can determine a service's port types. This allows a consumer of a WSDL document to determine if it wishes to communicate with a particular service based on whether or not it supports several port types.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

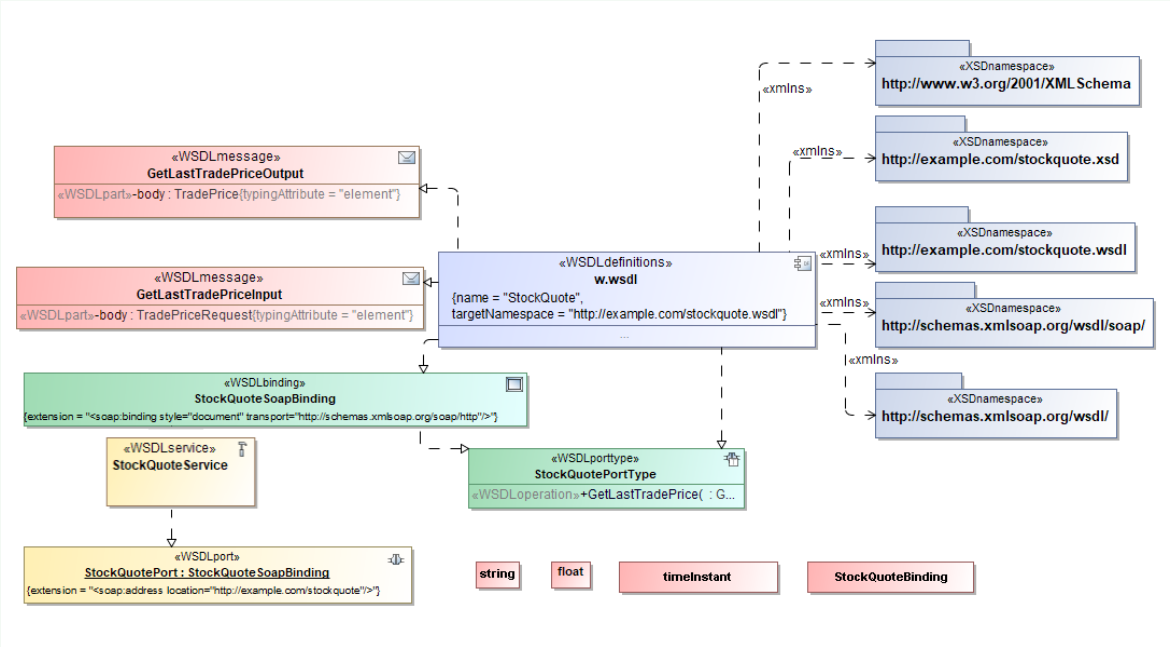
Example
Code:

```
<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">
    <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>
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

```
    </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:



Ports

The following rules describe the Ports element:

- 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 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.

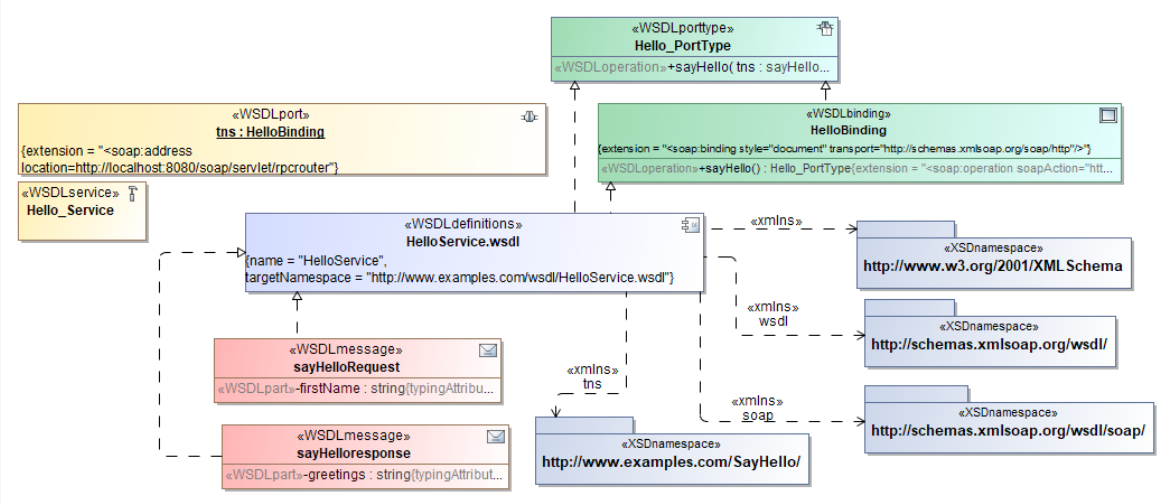
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Code:

```
<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 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Reversed UML model:



Macro Engine Documentation

The following pages present detailed instructions on working with macros using Macro Engine. Macro Engine enables you to easily create, add, and edit your macros to customize them to your needs.

Introducing Macro Engine

Macro Engine (previously called Script Engine) in MagicDraw allows you to create your own macro (script) by using BeanShell, Groovy, JRuby, JavaScript (Nashorn and Rhino), or Jython. With Macro Engine, you can control everything that is allowed in Open API, for example, transforming and manipulating models. Learn more about open API of MagicDraw or other Cameo Suite modeling tool (for example, Cameo Systems Modeler) in Developer Guide(see page 2155) and find sample macros in *<MagicDraw installation directory>/samples/product features/macros*.

Macro Engine comes with Professional, Architect, and Enterprise Editions starting from MagicDraw version 16.6 and later.

MagicDraw 18.1 Macro Engine supports two types of Javascript: (i) Javascript Nashorn and (ii) Javascript Rhino. This support is intended to prevent some incompatibility issues in the language syntax with macros that were created using Java 7 or earlier (Javascript Rhino), because Oracle changed the built-in Javascript engine from Javascript Rhino to Javascript Nashorn since Java 8 was released.

When you refer to Javascript, Macro Engine will refer to the default Javascript that comes with the JRE, which is Javascript Nashorn. For example, if you use Java 8, the default Javascript will be Javascript Nashorn. If you find any incompatibility issues, Macro Engine provides the migration capability to downgrade the Javascript engine for all of the existing macros to use Javascript Rhino.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

To migrate to Javascript Rhino

- Click **Tools** > **Macros** > **Migrate to Javascript Rhino**.

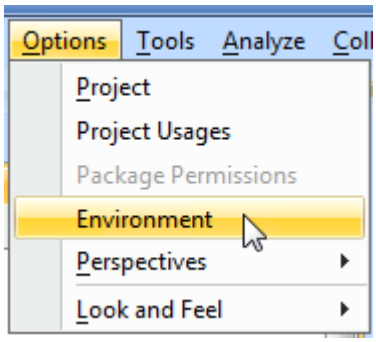
If you are using MagicDraw 18.1 or later with Java 8 and created a macro with the new language syntax in Nashorn, and then downgraded your Java to Java 7, this will cause an execution problem for your macro.

Selecting a default macro language

Use the **Environment Options** dialog to select a default macro language.

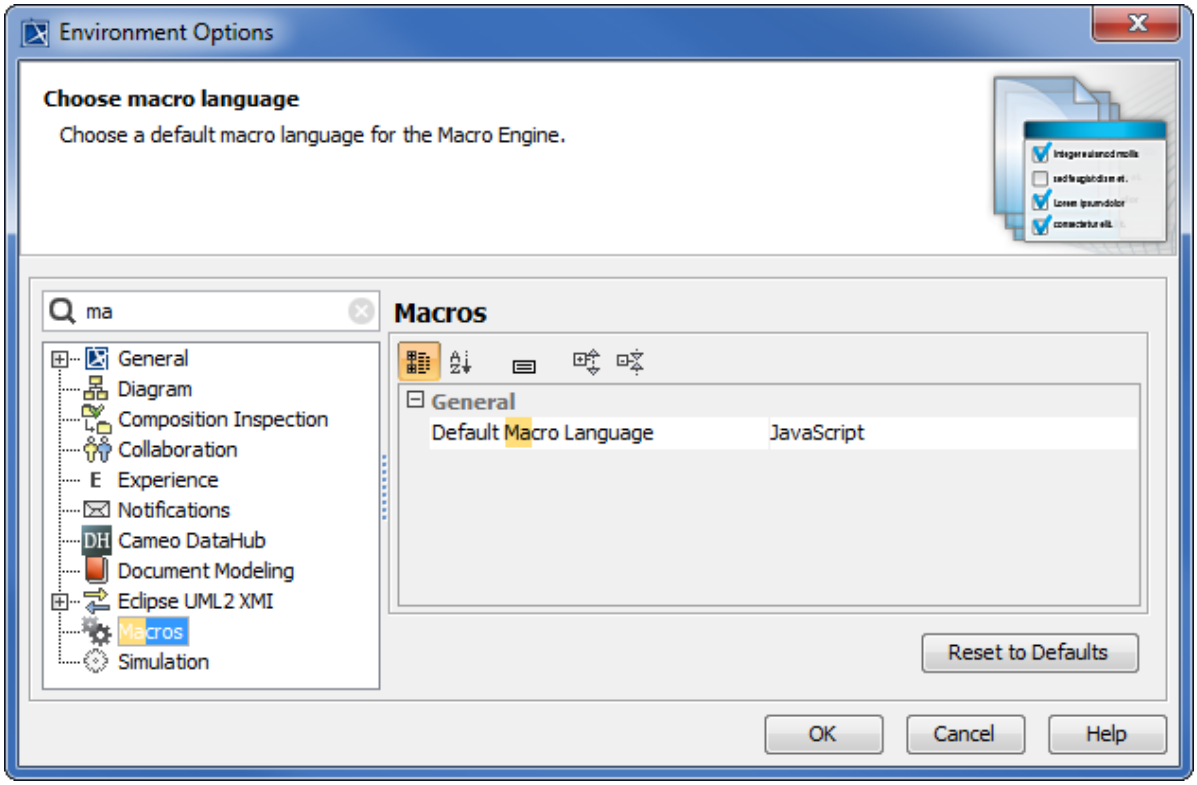
To select a default macro language

1. Click **Options** > **Environment** on the main menu.

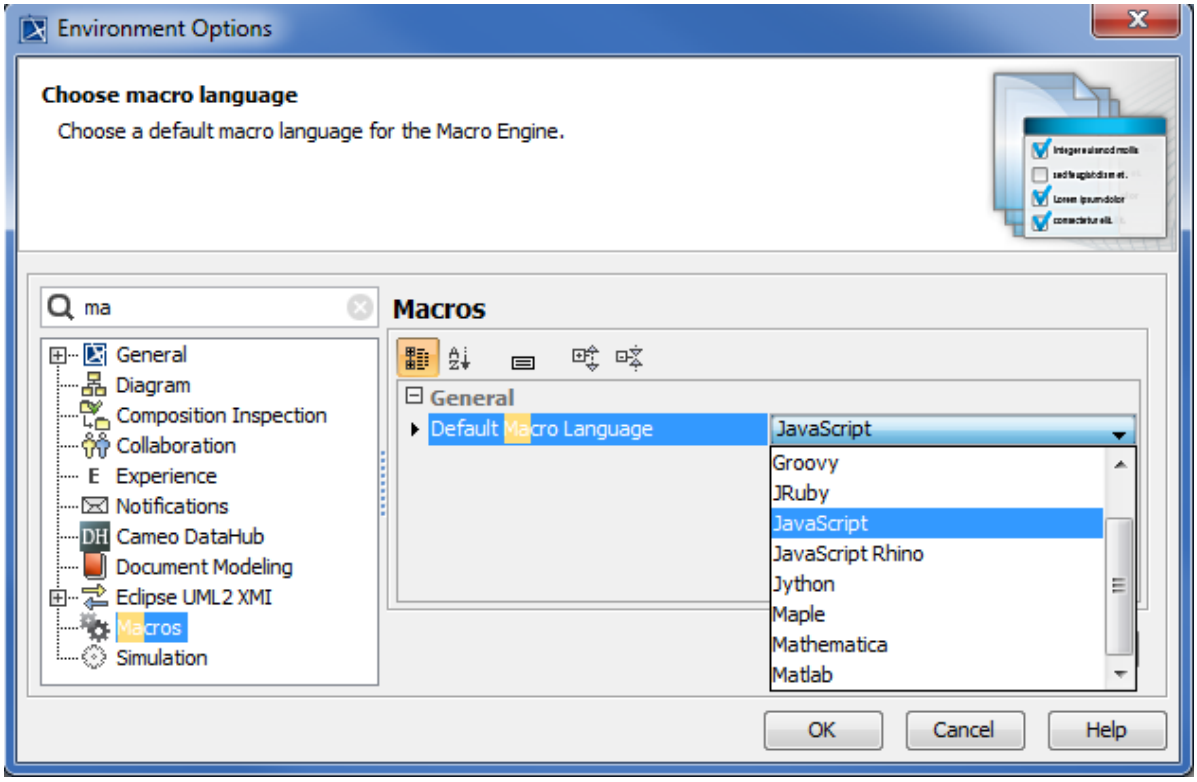


The **Environment Options** dialog opens.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



2. Select the **Macros** node.
3. Click the box next to the **Default Macro Language** box to see a list of supported programming languages, see figure below.



4. Select **Jython**, **BeanShell**, **Groovy**, **JRuby**, or **JavaScript**.

5. Click **OK** to save the selected language as the default macro language.

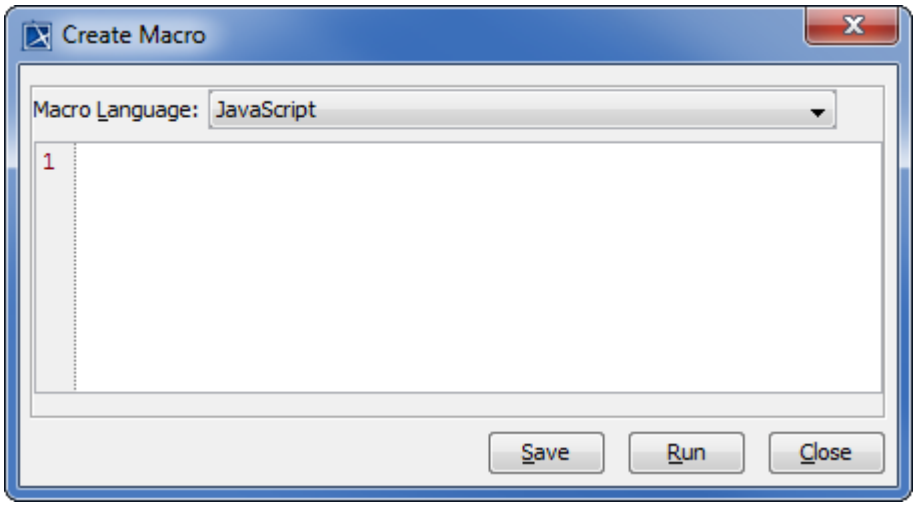
> - JavaScript is the default macro language.
> - Macro Engine currently supports BeanShell, Groovy, JRuby, JavaScript (Nashorn and Rhino), and Jython only.

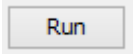
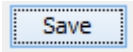
Creating a Macro

Once you have selected a default macro language (see Selecting a default macro language(see page 2685)), you can create a new macro by using the **Create Macro** dialog, see figure below. The dialog allows you to specify a macro language, enter source code, and save it.

To create a new macro

1. Click **Tools** > **Macros** > **Create Macro** on the **MagicDraw** main menu. The **Create Macro** dialog will open, see figure below.



2. Select a macro language in the **Macro Language** box.
3. Enter source code in the text box.
4. Click  to test and make sure that the source code works properly.
5. Click  . The **Macro Information** dialog will open. Follow the steps described in Adding a Macro and Its Information(see page 2703).
6. After clicking **OK** in the **Macro Information** dialog, the new macro will be saved in the location you have specified in the **File** text box.

Adding a Macro and editing Macro information

You can add a new macro and enter all necessary information about it in MagicDraw by following the steps described in Opening Macro Information Dialog and Adding a Macro and Its Information below.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

On this page

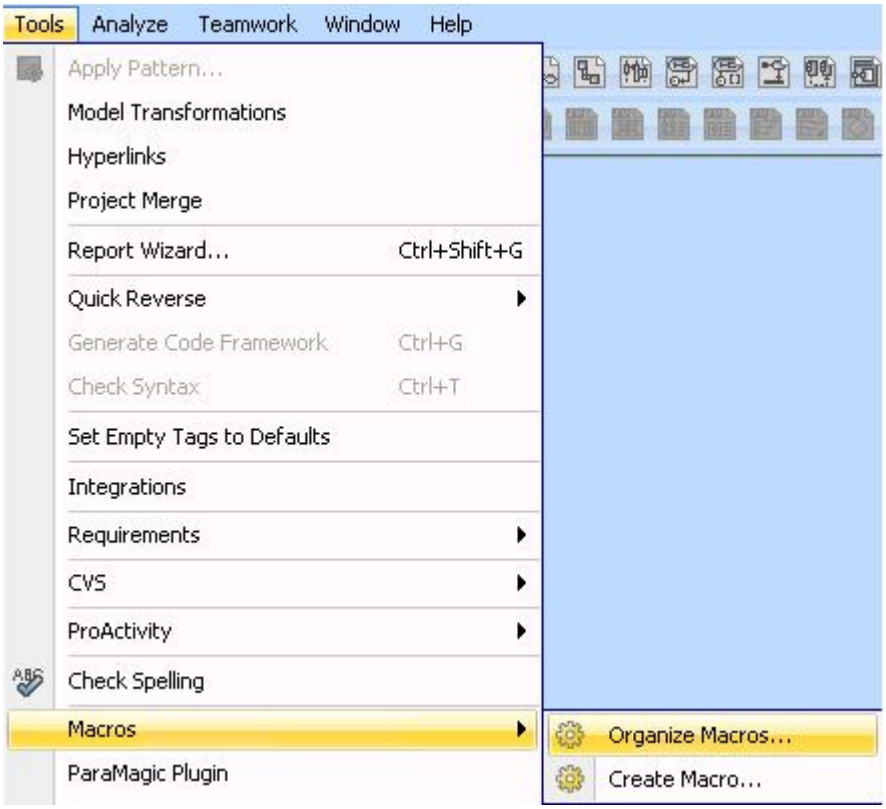
- Opening Macro Information Dialog(see page 2688)
- Adding a Macro and Its Information(see page 2688)
- Editing Macro Information(see page 2688)
- Macro Information Dialog Mnemonic Keys(see page 2688)

Opening Macro Information Dialog

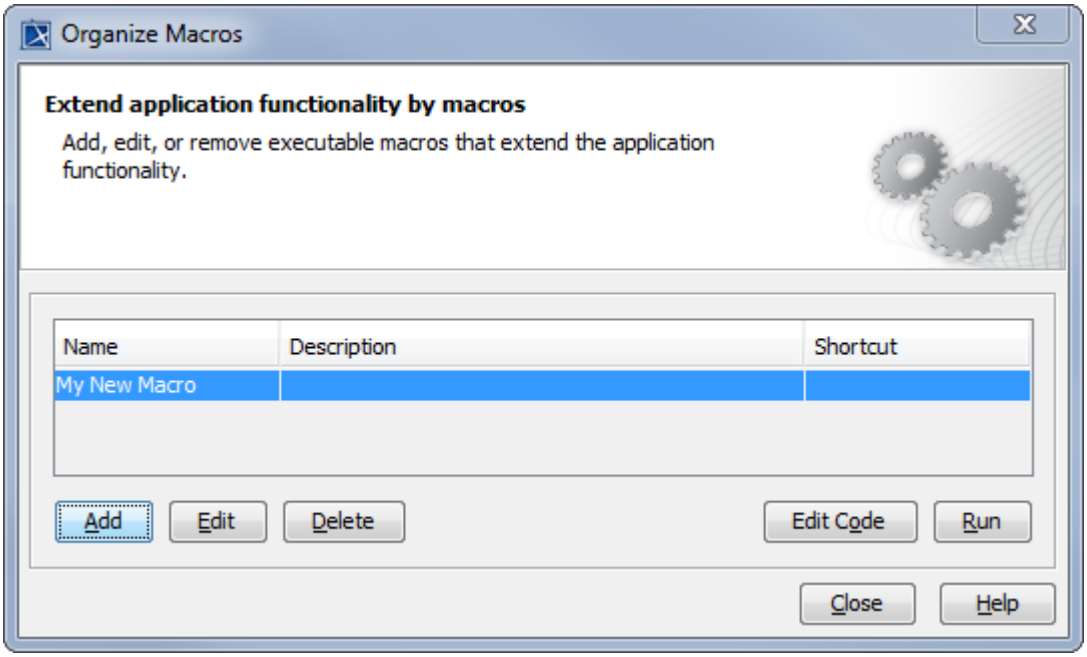
You can add or modify macro information such as the macro name and description, in the **Macro Information** dialog. To open the **Macro Information** dialog, you need to open the **Organize Macros** dialog first.

To open the **Organize Macros** dialog

- Click **Tools** > **Macros** > **Organize Macros** on the MagicDraw main menu. The **Organize Macros** dialog will open, see figures below.



Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



To open the **Macro Information** dialog

1. Click **Tools** > **Macros** > **Organize Macros** on the MagicDraw main menu to open the **Organize Macros** dialog, see above figure.
2. Click Add. The **Macro Information** dialog will open.

Adding a Macro and Its Information

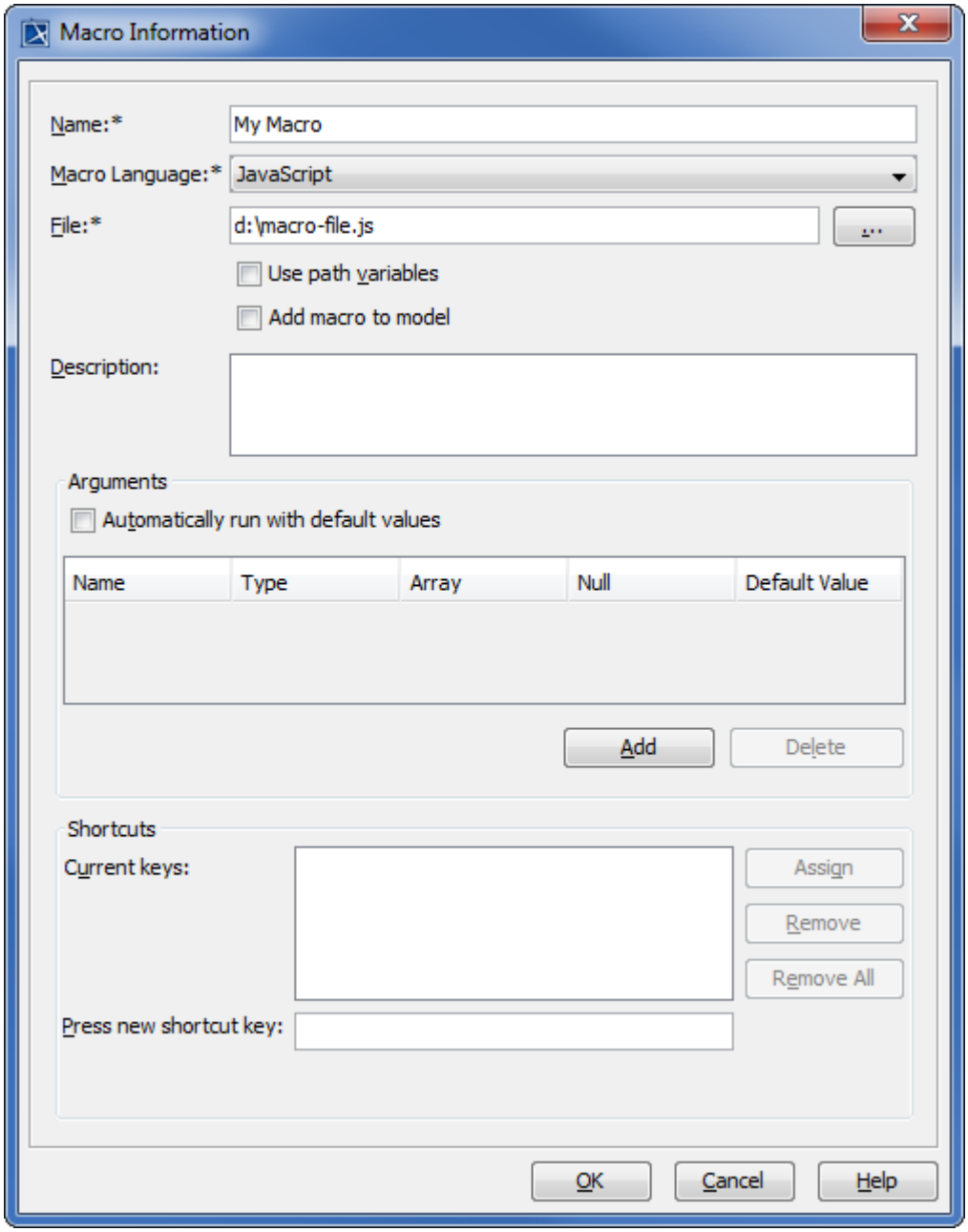
Use the Add or Edit button in the **Organize Macros** dialog to add or edit a macro and its information in the **Macro Information** dialog. You can also press the mnemonic keys to add or edit a macro, see Macro Information Dialog Mnemonic Keys.

To add a macro and enter macro information in the **Macro Information** dialog

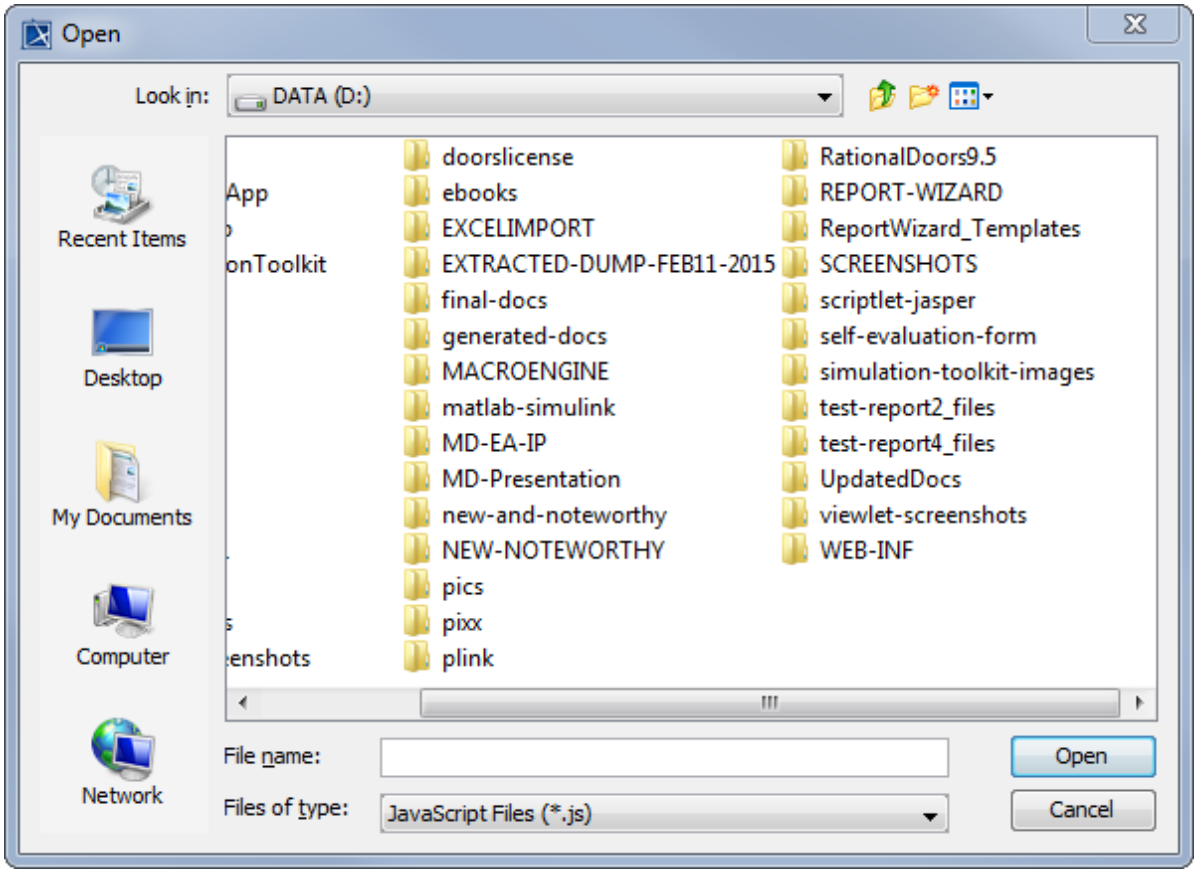
1. Open the **Organize Macros** dialog (click **Tools** > **Macros** > **Organize Macros** on the MagicDraw main menu).
2. Click Add. The **Macro Information** dialog will open, see figure below.

> The Edit, Delete, Edit Code, and Run buttons in the **Organize Macros** dialog will be disabled if there is no macro in the macro table or if you do not select any macro from the table.

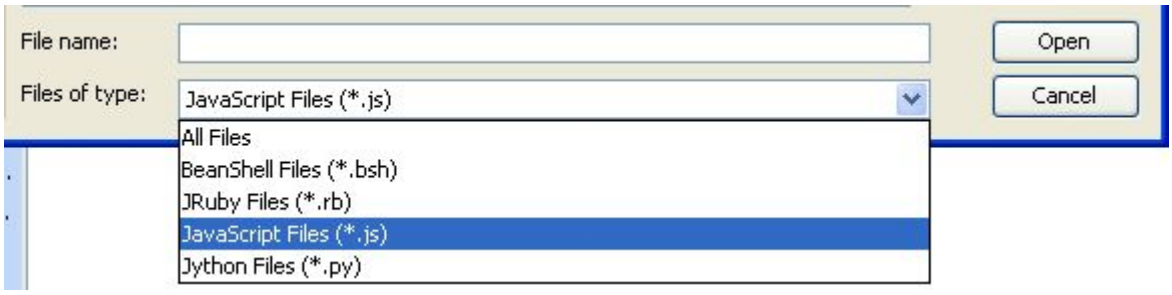
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

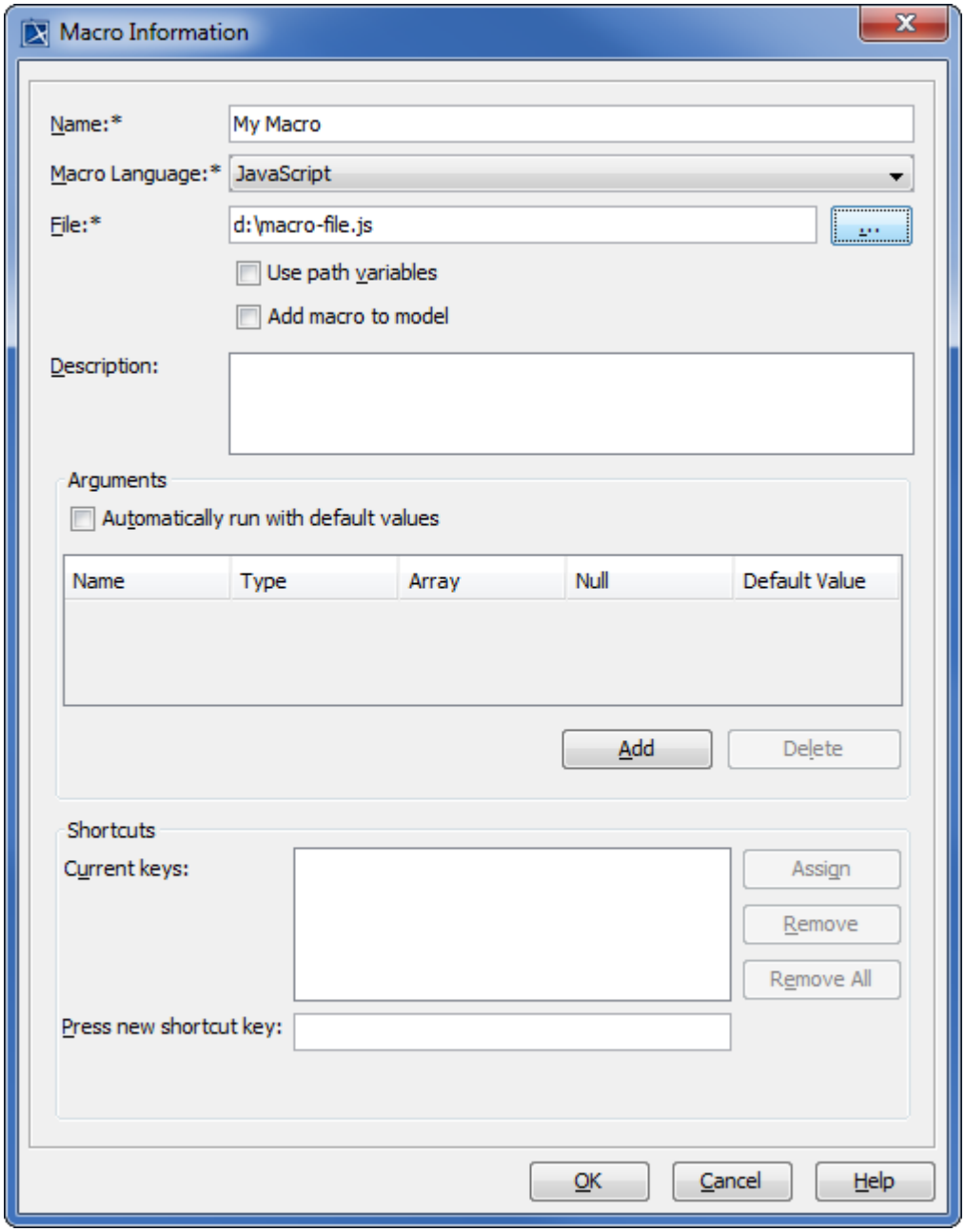


3. Type the macro name in the **Name** box.
4. The default macro language you have previously selected (see Selecting a Default Macro Language(see page 2685)) will appear in the **Macro Language** box.
5. Click the ... button, to locate a macro file. The **Open file** dialog will open, see figure below.
6. Select the file and its type (there are 5 types of file filter: **.bsh*, **.groovy*, **.rb*, **.js*, or **.py*), see figure below.



7. Click **Open,** see figure above. The selected pathname will appear in the **File** box in the **Macro Information** dialog, see figure below.





8. Select either (i) the **Use path variables** check box or (ii) the **Add macro to model** check box.

- If you have specified the file or network path in the **Environment Options** dialog by clicking **Options** > **Environment** > **Path Variables** and selected the [Use path variables] check box in **Macro Information** dialog, the <Path Variable name> will show in front of the file pathname. This field is the [Required] field, for example, *<mypath>/<macro_name>.js.*
- If you select the [Use path variables] check box, the full pathname will not be saved.
- If you select the [Add macro to model] check box, your source code will be imported from the file to the model. The location to keep the model is Data::MacroEngine.
- You can open the **Macro Information** by using Ctrl+Alt+m as a shortcut key.

9. Type the macro description in the **Description** box, see figure above.

A macro name must be unique and cannot be duplicated.

10. The function of the [Automatically run with default values] check box is to allow you to run the default values automatically. If you select the check box, the system will not open a dialog to prompt you to input the value.

If you select the [Automatically run with default values] check box, you need to enter the valid default value of each parameter.

11. Click **Add** to specify the arguments of the macro. The arguments specified in the **Arguments** tabl e will be the global variables of a specific macro.
    - **Name** column contains the name of a parameter
    - **Type** column contains the type of a parameter
    - **Array** check box is to identify if an array is the parameter
    - **Null** check box is to identify if null is the parameter value
    - **Default Value** column contains an initial value to run the macro

- A parameter type can be a String, Integer, Long, Double, Date, or ElementPath.
- If the **Null** check box is selected, you cannot enter the default value of that particular parameter.
- An empty value in the **Default Value** column does not necessarily mean a null value, for example an empty string value is an empty string.
- If you input an invalid default value or you do not enter the argument name, the system will display the following error message when you run the script: **The following argument(s) are invalid: <List of the invalid argument>**.

12. Type a keyboard shortcut that will be used to run the macro in the [Press new shortcut key:] box and click [Assign]. The newly assigned keyboard shortcut will appear in the Current keys box.
13. Click **OK**. The **Organize Macros** dialog will open, showing the newly added macro name, description, and keyboard shortcut.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Macro names, filenames, and languages are required.
- Macro description and keyboard shortcuts are optional.
- If any of the required fields are not entered or duplicate macro name is entered, the following message will open: **The following field(s) are invalid: <List of the problems>**.
- If the ☑ Automatically run with default values check box is selected and at least one value is empty or not valid, then the following error message will open: **The following value(s) are invalid. <List of the invalid value>**
- If the ☑ Automatically run with default values check box is selected, all variables must have valid values or are set to null.

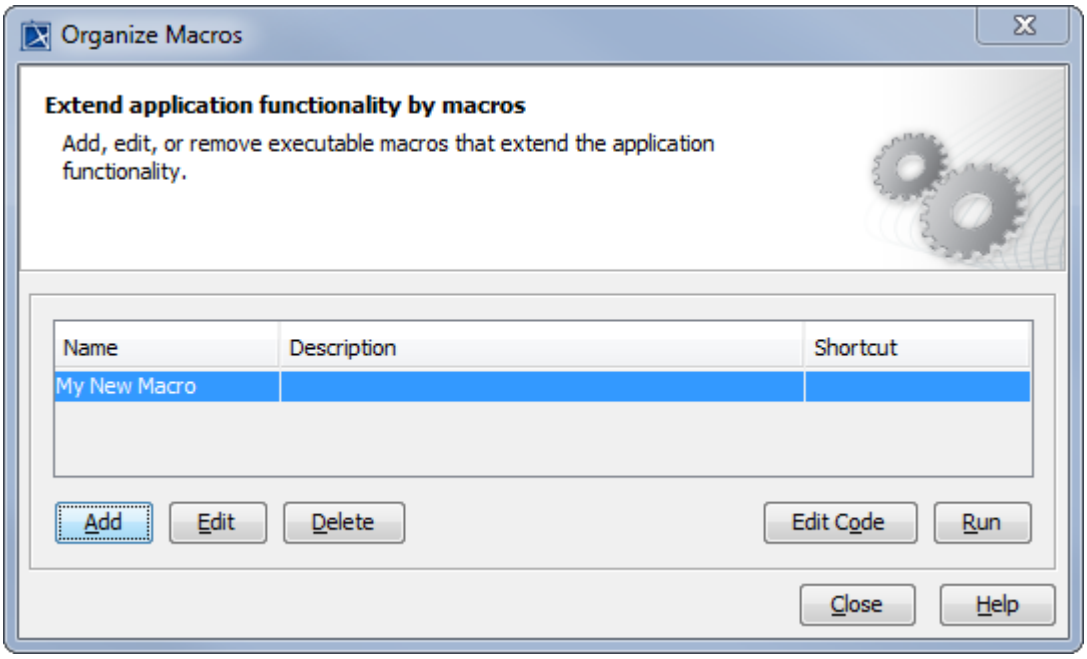
14. Click Close to close the **Organize Macros** dialog.

Editing Macro Information

You can see macro information such as names and descriptions, as well as the macro keyboard shortcuts in the **Organize Macros** dialog.

To edit macro information

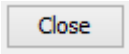
1. Click **Tools** > **Macros** > **Organize Macros**. The Organize Macros dialog will open, see figure below.



2. Select a macro from the table and either click **Edit** or press **Alt + E**. The **Macro Information** dialog will open.

3. Edit the macro information.

4. Click **OK** to save it. You will see the altered information in the **Organize Macros** dialog.

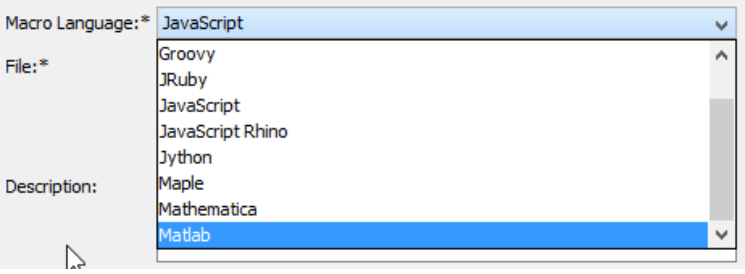
5. Click 

.

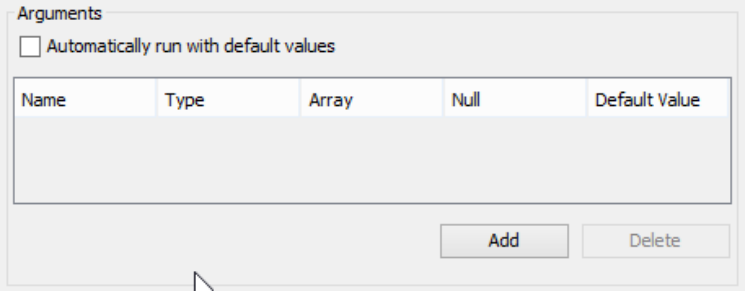
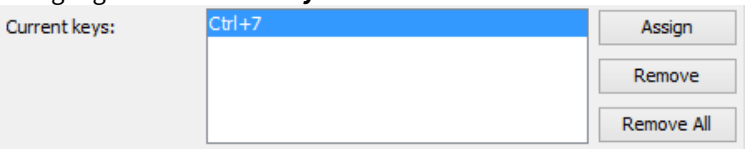
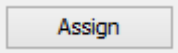
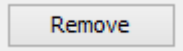
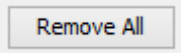
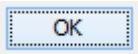
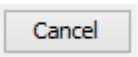
Macro Information Dialog Mnemonic Keys

Macro Engine provides mnemonic keys for you to perform some operations, for example, highlight a textbox and click a specific button in the **Macro Information** dialog. The table below lists the **Macro Information** dialog mnemonic keys and their function.

Table listing of Macro Information Dialog Mnemonic Keys

Mnemonic keys	Function
Alt + N	To place the pointer in the box.
Alt + M	To highlight the  box.
Alt + F	To place the pointer in the box.
Alt + .	To open the **File** dialog.
Alt + D	To place the pointer in the box.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Mnemonic keys	Function
Alt + A	To add an argument in the **Arguments**  area.
Alt + l	To delete an argument in the **Arguments** area.
Alt + U	To highlight the **Current keys**  box.
Alt + G	To click the  button.
Alt + R	To click the  button.
Alt + E	To click the  button.
Alt + P	To place the pointer in the box.
Alt + O	To click the  button.
Alt + C	To click the  button.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Mnemonic keys	Function
Alt + H	To click the Help button.

Deleting and executing Macros

You can click the Delete or Run button in the **Organize Macros** dialog to delete or execute a selected macro. You can also press the pre-designed mnemonic keys to delete or run a macro as described in Organizing Macros Dialog Mnemonic Keys below.

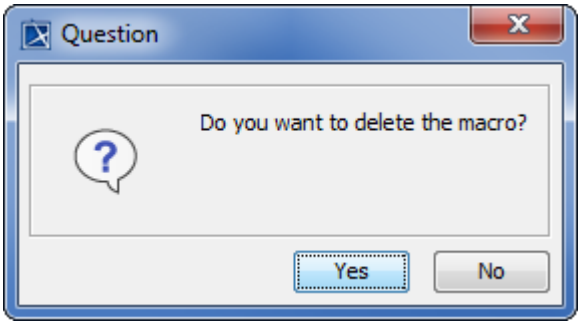
On this page

- Deleting a Macro(see page 2697)
- Executing a Macro(see page 2697)
- Organizing Macros Dialog Mnemonic Keys(see page 2697)

Deleting a Macro

To delete a macro from the **Organize Macros** dialog

1. Click **Tools** > **Macros** > **Organize Macros** to open the **Organize Macros** dialog.
2. Select a macro from the table and either click **Delete** or press **Alt + D**. A dialog will open, asking whether you want to delete the macro, see figure below.



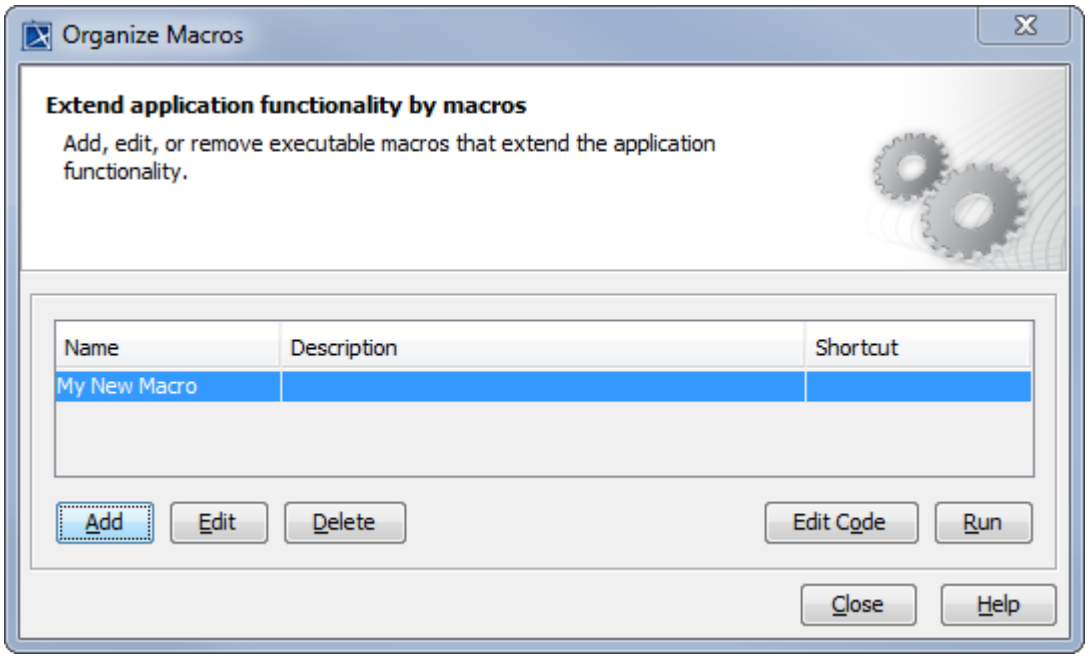
3. Click **Yes** and the macro will be deleted from the **Organize Macros** dialog.

Executing a Macro

To execute a macro from the **Organize Macros** dialog

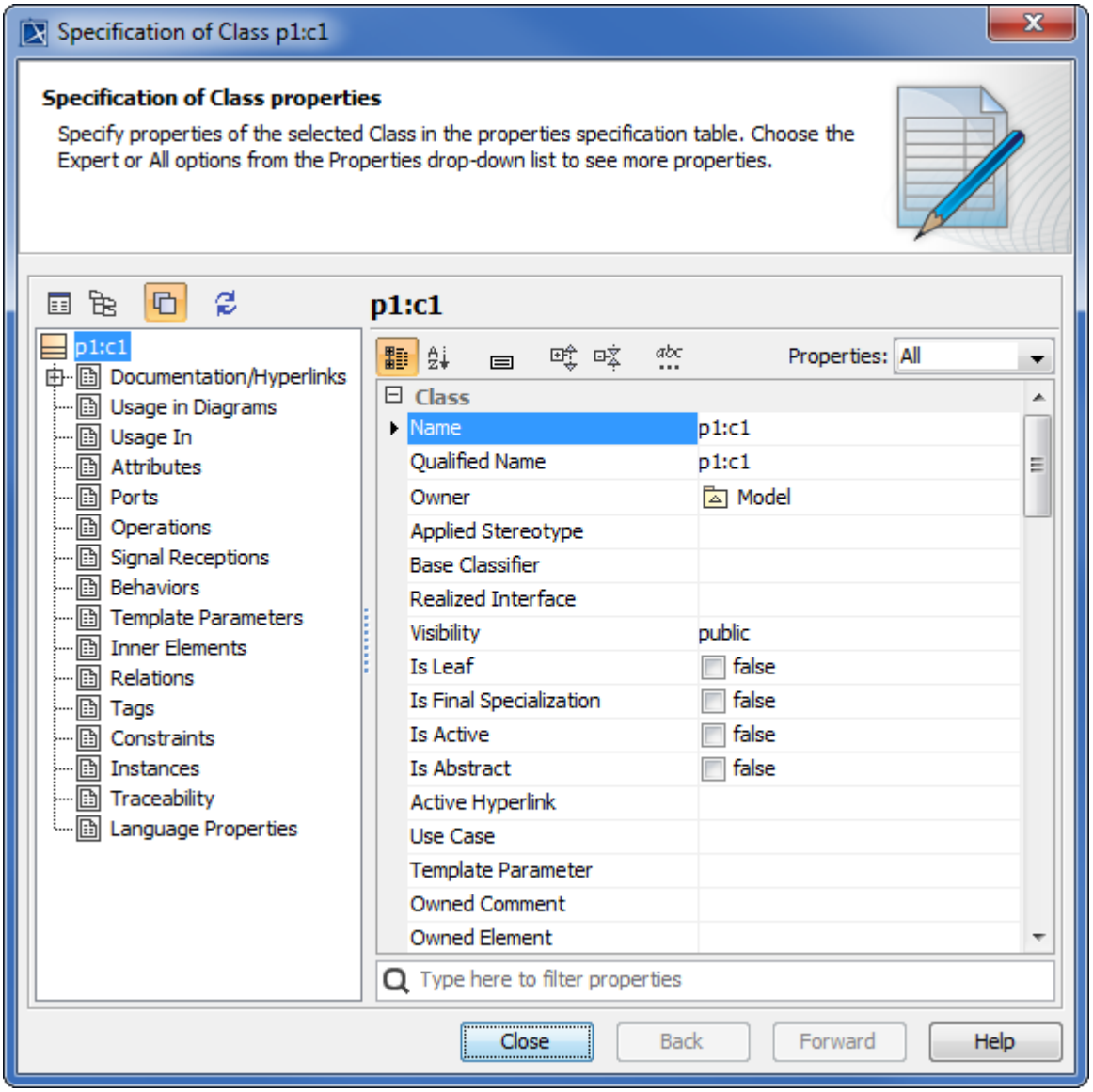
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

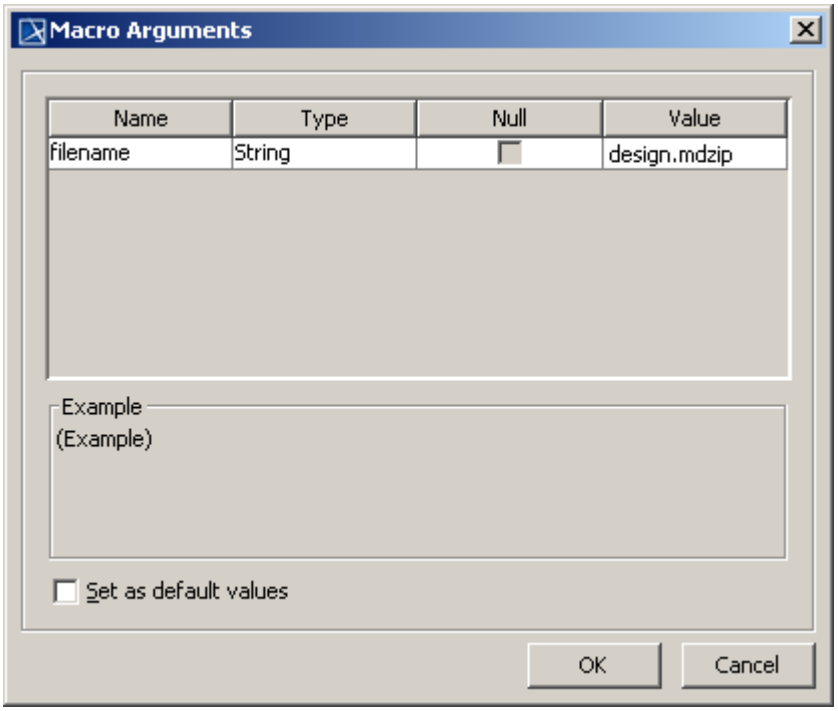
1. Click **Tools** > **Macros** > **Organize Macros** The **Organize Macros** dialog will open, see figure below.



2. Select a macro from the table and click Run. After the macro has been executed, a message will open: **The macro <macro name> has been executed**.
3. If you have the parameters in the **Macro Information** dialog, you need to specify the value in the **Macro Arguments** dialog, see figure below, before running the macro.

- For an array datatype, you need to click the ... button in the **Value** column in the **Macro Arguments** dialog and enter each value into each line. The value of the first line will be the value in array index 0.
- An ElementPath is a Qualified Name. You can find this information in the specification dialog of each element, see figure below.
- The **Macro Arguments** dialog, see the figure below, will be displayed if the Automatically run with default values check box is not selected.
- If you want to save argument values in the **Macro Arguments** dialog, you need to select the **Set as default values** check box before you click **OK**.


Specification of Class p1:c1
Specification of Class properties
Specify properties of the selected Class in the properties specification table. Choose the Expert or All options from the Properties drop-down list to see more properties.
p1:c1
p1:c1
Documentation/Hyperlinks
Usage in Diagrams
Usage In
Attributes
Ports
Operations
Signal Receptions
Behaviors
Template Parameters
Inner Elements
Relations
Tags
Constraints
Instances
Traceability
Language Properties
Properties: All
Class
Name p1:c1
Qualified Name p1:c1
Owner Model
Applied Stereotype
Base Classifier
Realized Interface
Visibility public
Is Leaf false
Is Final Specialization false
Is Active false
Is Abstract false
Active Hyperlink
Use Case
Template Parameter
Owned Comment
Owned Element
Type here to filter properties
Close
Back
Forward
Help



- You can also execute a macro from the main browser in MagicDraw by pressing the shortcut keys that you have defined in the **Organize Macros** dialog.
- You can only run macro one at a time.
- If there is an error while running a macro, for example, syntax error, the following message will open: **MagicDraw cannot execute the <macro language> macro, please make sure that <path, filename, extension> is correct. <error description>**.
- If MagicDraw cannot find a macro file in the location that you have specified in the **Open** dialog, the following message will open: **MagicDraw cannot find the macro: <path, filename, extension>**.

Organizing Macros Dialog Mnemonic Keys

Macro Engine also provides mnemonic keys to add, edit, delete, and run a macro from the **Organize Macros** dialog. The table below shows the **Organize Macros** dialog mnemonic keys and their function.

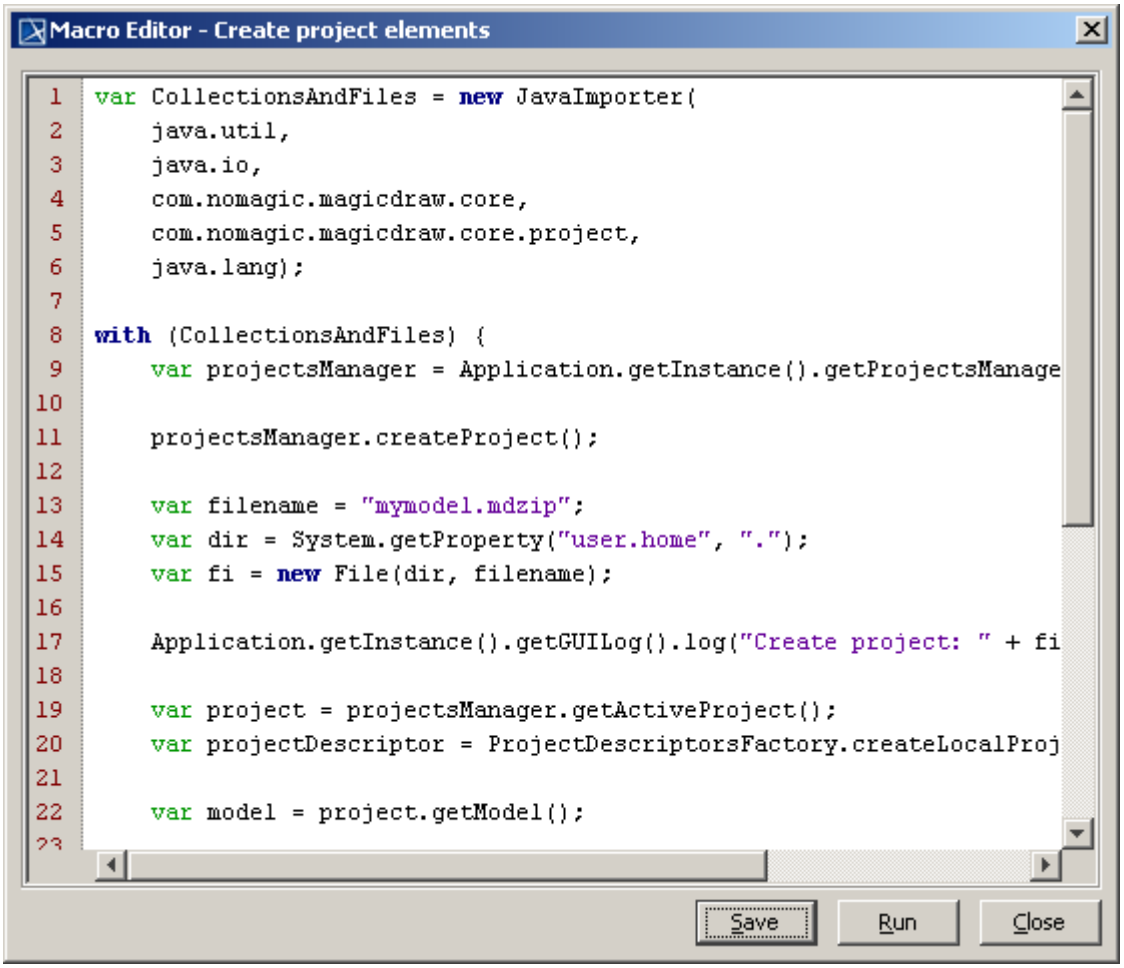
Table listing of Organize Macros Dialog Mnemonic Keys

Mnemonic keys	Button	Function
Alt + A	Add	To add a macro in the **Macro Information** dialog.
Alt + E	Edit	To edit a macro in the **Macro Information** dialog.
Alt + D	Delete	To delete a macro from the **Organize Macros** dialog.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Mnemonic keys	Button	Function
Alt + O	Edit Code	Code To edit source code in **Macro Editor**.
Alt + R	Run	To run a macro from the **Organize Macros** dialog.
Alt + C	Close	To click the **Close** button.
Alt + H	Help	To click the **Help** button.

- You can click the Edit Code button in the **Organize Macros** dialog, see figure above, to edit and save source code in the **Macro Editor** dialog, see figure below.
- You can click Save to save the source code or click Run to run the macro in the **Macro Editor** dialog, see figure below.



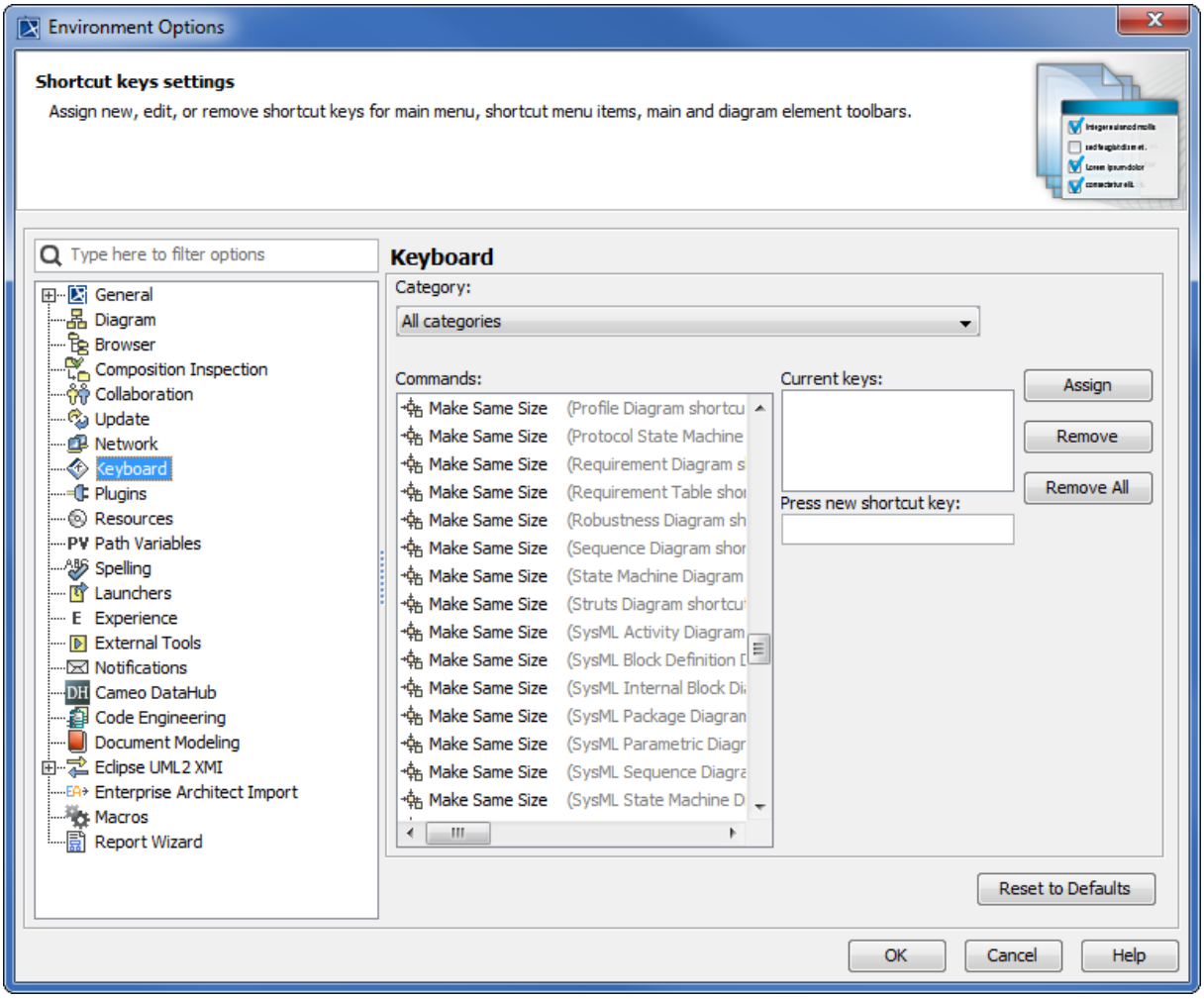
Macro keyboard shortcuts

Macro Engine allows you to assign keyboard shortcuts to the macros that you have created using either the **Environment Options** or **Macro Information** dialog, see figures below. You can later press the keyboard shortcuts to execute or run the macros in MagicDraw.

Related pages

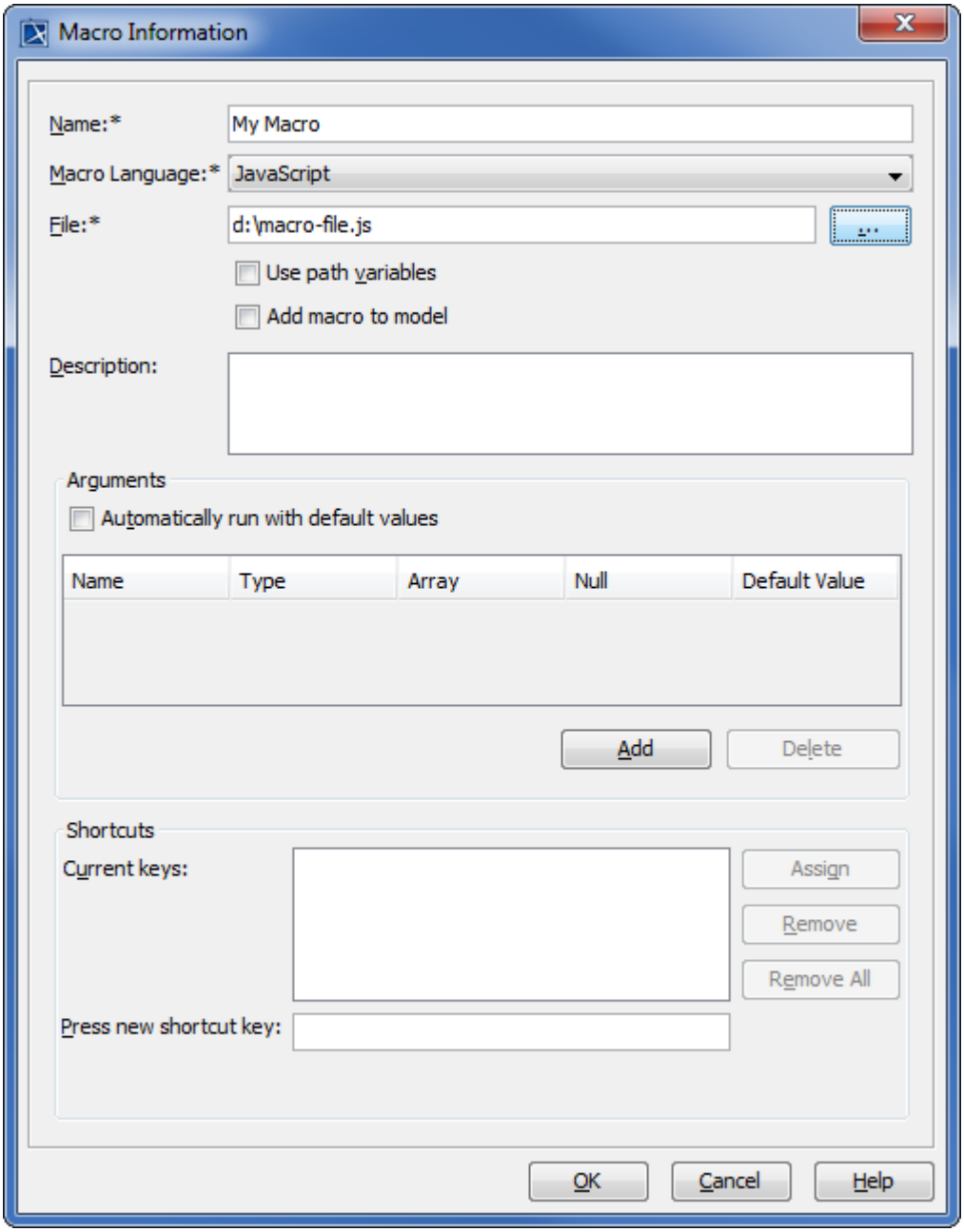
To open the keyboard shortcuts pane in the **Environment Options** dialog

1. Click **Options** > **Environment** on the MagicDraw main menu. The **Environment Options** dialog will open.
2. Click **Keyboard,** see figure below.



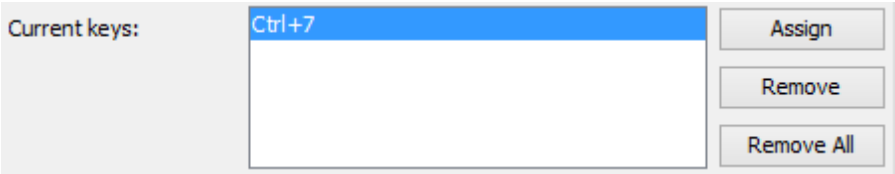
Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

The macro information and keyboard shortcuts that are saved either in the **Environment Options** or **Macro Information** dialog will be stored as a MagicDraw environment. They will not be kept in a specific project file [*.mdzip*].



- The Assign button in the **Macro Information** or **Environment Options** dialog will be disabled until you type a keyboard shortcut in the Press new shortcut key: box.

The table below lists the Keyboard Shortcut Buttons

Button	Function
Assign	To assign a new keyboard shortcut to a macro.
Remove	To remove the keyboard shortcuts from the selected item in  box.
Remove All	To remove all keyboard shortcuts from a macro.

The table below lists the Keyboard Shortcut Text Boxes

Field	Function
Current keys	To store a list of keyboard shortcuts currently assigned to a macro.
Press new shortcut key:	To type a new keyboard shortcut to be assigned to a macro.

Assigning a keyboard shortcut to a Macro

To add a new keyboard shortcut to a macro

1. Open either the (i) **Environment Options** or (ii) **Macro Information** dialog.
2. Type a keyboard shortcut in the Press new shortcut key: box.
3. Click Assign to assign the keyboard shortcut to a macro.
4. Click **OK**.

> ⓘ If a keyboard shortcut key entered in the Press new shortcut key: box has already been assigned to another macro, the following message will appear under the **Press new shortcut key** box: **Currently assigned to <the command name>**.

Removing keyboard shortcuts from Macro

To remove a keyboard shortcut

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

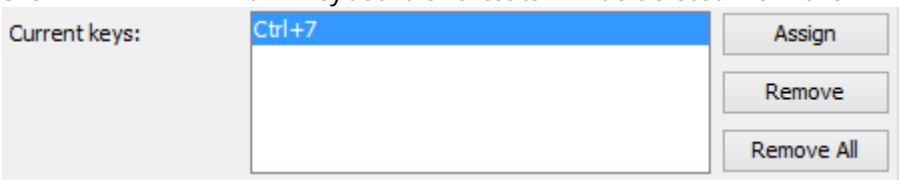
1. Open either the (i) **Environment Options** or (ii) **Macro Information** dialog.
2. Select a keyboard shortcut from the **Current keys** box.
3. Click Remove. The selected keyboard shortcut will be removed from the **Current keys** box.

> You can also remove a keyboard shortcut through the **Environment Options** dialog by clicking **Options** > **Environment** > **Keyboard** on MagicDraw main menu.

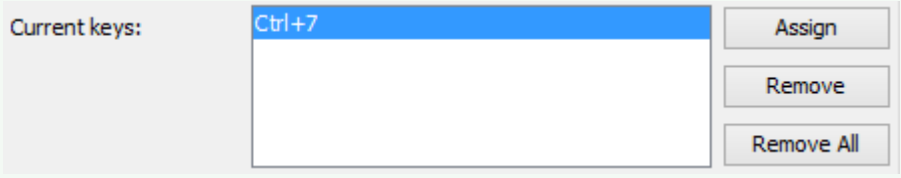
To remove all keyboard shortcuts

---

1. Open either the (i) **Environment Options** or (ii) **Macro Information** dialog.
2. Click Remove All. All keyboard shortcuts will be deleted from the



box.

> - The Assign, Remove, and Remove All buttons in the **Macro Information** or **Environment Options** dialog will be disabled if there is no keyboard shortcut either in the Press new shortcut key: or
>
> 
>
>
> box.
>
> - The Assign button will be enabled if the new keyboard shortcut entered has not been assigned to any other macro. You can assign more than one keyboard shortcut to a macro.

Opaque objects

Macro Engine creates opaque objects to represent the elements in MagicDraw. Through these opaque objects, you can access the elements, retrieve, or assign values to them instead of using MagicDraw OpenAPI to do it.

> All examples given in this section is written in Javascript.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

On this page

- Getting an Opaque Object(see page 2706)
- Getting Element Property Values(see page 2706)
  - Getting Element Property Value Examples(see page 2706)
- Setting Element Property Values(see page 2706)
  - Setting Element Property Value Examples(see page 2706)
- Getting the Child of an Element(see page 2706)
- Getting the Owner of an Element(see page 2706)
- Creating a New Element(see page 2706)
- Creating a Relationship Between Elements(see page 2706)
- Removing an Element(see page 2706)
- Adding a Stereotype to an Element(see page 2706)
- Removing a Stereotype from an Element(see page 2706)
- Printing Element Details(see page 2706)

Getting an Opaque Object

You can get an opaque object of an existing MagicDraw element by using either:

```
(i) AutomatonMacroAPI.getOpaqueObjectByPath(String path)
(ii) AutomatonMacroAPI.getOpaqueObject(Element element)
```

If the above methods cannot find the element, they will return null.

(i) getOpaqueObjectByPath(String path)

To use getOpaqueObjectByPath(String path), for example, type:

```
AutomatonMacroAPI.getOpaqueObjectByPath ("PackageA::Element2");
```

(ii) getOpaqueObject(Element element)

To use getOpaqueObject(Element element), for example, type:

```
var element = ModelHelper.findElementWithPath("PackageB::ClassB");
var a = AutomatonMacroAPI.getOpaqueObject(element);
```

```
You can also use two other methods to get an opaque object as follows:
(iii) AutomatonMacroAPI.getModelData()
(iv) AutomatonMacroAPI.getSelectedElementFromContainmentTree()
```

(iii) getModelData()

This method will obtain an opaque object of the model Data in the Containment tree.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

(iv) getSelectedElementFromContainmentTree()
This method will obtain an opaque object of the selected element in the Containment tree.
Macro Engine uses methods (iii) getModelData() and (iv) getSelectedElementFromContainmentTree() to retrieve opaque objects in order to identify the defined scope in its recording mechanism.

Getting Element Property Values

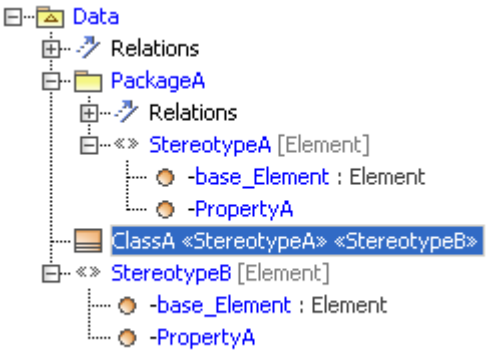
Once you have obtained an opaque object, you can get the property value of the element by using any of the following methods:

```
(i) <variableName>.get<PropertyName>()
(ii) <variableName>.<PropertyName>
(iii) <variableName>._automatonGetValue(String realPropertyName,
String stereotypePath)
```

The <variableName> is the name of a macro variable that stores the opaque object. The <PropertyName> is the name of the property that appears in the Specification dialog.

The <variableName> is the name of a macro variable that stores the opaque object. The <PropertyName> is the name of the property that appears in the Specification dialog.

You need to capitalize the first letter of <PropertyName> and replace the whitespace with an underscore. If a duplicate property name occurs, you can refer to the right property name by using the following additional information: <SterotypeName>_<PropertyName><RunningNumber>.



If there are two stereotypes applied to the same element, see above figure, you can differentiate one from the other, for example, by specifying <PropertyName> as StereotypeA_PropertyA1 and StereotypeA_PropertyA2 in the macro.

You can also use the method _automatonGetValue to get a property value. If you want to get the value of a PropertyA from SterotypeA in PackageA., for example, you can use _automatonGetValue("PropertyA", "PackageA::StereotypeA").

A realPropertyName is the real property name that is used in MagicDraw openAPI. A stereotypePath is the path of a stereotype that contains the property. This property will not be needed if it is in the Element itself.

If you refer to a property that does not exist, Macro Engine may or may not throw an error, depending on which language library you use. For example, if you use Javascript to call the property that does not

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

exist, Macro Engine will not throw an error. But if you use JRuby, it will throw an exception to report the error condition: org.jruby.exceptions.RaiseException.

If the value of a property is an element, Macro Engine will convert it to an opaque object and you can call it, for example, by typing classA.Owner.Name.

If a property has more than one value, Macro Engine will convert the values to a list of opaque objects. If you need to get a value from the list, you can call the method of the class java.util.List, for example, by typing:

```
(i) classA.appliedStereotype.get(<index>).Name
or
(ii) classA.appliedStereotype.add(anotherOpaque)
If a property is read-only, an exception will be thrown.
```

Getting Element Property Value Examples

The following are some examples of how to get an element property value using the methods given in the section above:
to use get<PropertyName>, for example, type:

```
var classA = AutomatonMacroAPI.getOpaqueObjectByPath ("MyClass");
Application.getInstance().getGUILog().log(classA.getName());
```

to use <PropertyName>, for example, type:

```
var classA = AutomatonMacroAPI.getOpaqueObjectByPath("MyClass");
Application.getInstance().getGUILog().log(classA.Name);
```

```
to use _automatonGetValue, for example, type:
```

```
var reqA = AutomatonMacroAPI.getOpaqueObjectByPath("MyRequirements");
Application.getInstance().getGUILog().log(reqA._automatonGetValue("PropertyA",
"PackageA::StereotypeA");
```

to use a SysML Element, for example, type:

```
var reqA = AutomatonMacroAPI.getOpaqueObjectByPath("MyRequirements");
Application.getInstance().getGUILog().log(reqA.getID());
```

Setting Element Property Values

You can assign values to a MagicDraw element by using any of the following methods:

```
(i) <variableName>.set<PropertyName>(Object value)
(ii) <variableName>.<PropertyName> = value; and then call persist()
```

```
    (iii) <variableName>._automatonSetValue(String realPropertyName, String
stereotypePath, Object value)
```

The value of an element can be a primitive data type, an opaque object, or an element. If you use a setter to set the value, for example, _automatonSetValue() or change the value on a list, it will be saved in the MagicDraw element automatically.

If you use <variableName>.<PropertyName> = value to set the value, you must call persist() to persist the change to the MagicDraw element. You need to first set the data, call persist(), and finally call a getter method in order to set the data and retrieve them. This process will force an opaque object to retrieve the current value from a MagicDraw model and overwrite the value that you have just specified in the opaque object.

> ⓘ If you use JRuby, do not capitalize the first letter of <PropertyName> in <variableName>.<PropertyName>.

Setting Element Property Value Examples

The following are some examples of how to set an element property value by using the methods given in section above.
to use set<PropertyName>(value), for example, type:

```
var classB = AutomatonMacroAPI.getOpaqueObjectByPath("Element2");
classB.setName("NewElementName")
```

to use <PropertyName> = value), for example, type:

```
var classB = AutomatonMacroAPI.getOpaqueObjectByPath("Element2");
classB.Name = "NewElementName";
classB.Is_Abstract = true;
classB.persist();
```

to use _automatonSetValue, for example, type:

```
var classB = AutomatonMacroAPI.getOpaqueObjectByPath("Element2");
classB._automatonSetValue("PropertyA", "PackageA::StereotypeA", "Demo String value");
```

to set an opaque object to another opaque object, for example, type:

```
var ele1 = AutomatonMacroAPI.getOpaqueObjectByPath("Element1");
var ele2 = AutomatonMacroAPI.getOpaqueObjectByPath("Element2");
ele1.setPackaged_Element(ele2);
```

The table below lists the supported element properties in Macro Engine.

Property	Type	Support Operation
Active Hyperlink	String	Read
All General Classifiers	List<AutomatonOpaqueObject>	Read
All Realizing Elements	List<AutomatonOpaqueObject>	Read
All Specific Classifiers	List<AutomatonOpaqueObject>	Read
All Specifying Elements	List<AutomatonOpaqueObject>	Read
Applied Stereotype	List<AutomatonOpaqueObject>	Read
Applied_Stereotype_Instance	N/A	Read
Attribute	List<AutomatonOpaqueObject>	Read
Base Classifier	List<AutomatonOpaqueObject>	Read
Class	AutomatonOpaqueObject	Read
Classifier Behavior	AutomatonOpaqueObject	Read
Client Dependency	List<AutomatonOpaqueObject>	Read
Collaboration Use	List<AutomatonOpaqueObject>	Read
Element	ID	N/A -
Extension	List<AutomatonOpaqueObject>	Read
Feature	List<AutomatonOpaqueObject>	Read
Generalization	List<AutomatonOpaqueObject>	Read
Image	N/A	-
Imported Member	List<AutomatonOpaqueObject>	Read
Inherited Member	List<AutomatonOpaqueObject>	Read
Interface Realization	List<AutomatonOpaqueObject>	Read
Is Leaf	Boolean	Read/Write
Is Final Specialization	Boolean	Read/Write
Is Active	Boolean	Read/Write
Is Abstract	Boolean	Read/Write

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Property	Type	Support Operation
Member	List<AutomatonOpaqueObject>	Read
Name	String	Read/Write
Name Expression	AutomatonOpaqueObject	Read
Namespace	AutomatonOpaqueObject	Read
Nested Classifier	List<AutomatonOpaqueObject>	Read
Owned Attribute	List<AutomatonOpaqueObject>	Read
Owned Connector	List<AutomatonOpaqueObject>	Read
Owned Comment	List<AutomatonOpaqueObject>	Read
Owned Diagram	List<AutomatonOpaqueObject>	Read
Owned Element	List<AutomatonOpaqueObject>	Read
Owned Member	List<AutomatonOpaqueObject>	Read
Owned Operation	List<AutomatonOpaqueObject>	Read
Owned Port	List<AutomatonOpaqueObject>	Read
Owned Reception	List<AutomatonOpaqueObject>	Read
Owned Rule	List<AutomatonOpaqueObject>	Read
Owned Template Signature	AutomatonOpaqueObject	Read
Owned Trigger	List<AutomatonOpaqueObject>	Read
Owned Use Case	List<AutomatonOpaqueObject>	Read
Owning Package	List<AutomatonOpaqueObject>	Read
Owning Template Parameter	N/A	-
Owner	AutomatonOpaqueObject	Read/Write
Package	AutomatonOpaqueObject	Read
Package Import	List<AutomatonOpaqueObject>	Read
Part	List<AutomatonOpaqueObject>	Read
Participates In Activity	List<AutomatonOpaqueObject>	Read
Participates In Interaction	List<AutomatonOpaqueObject>	Read

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Property	Type	Support Operation
Powertype Extent	List<AutomatonOpaqueObject>	Read
Qualified Name	List<AutomatonOpaqueObject>	Read
Realized Interface	N/A	-
Realizing Component	List<AutomatonOpaqueObject>	Read
Realizing Element	List<AutomatonOpaqueObject>	Read
Redefined Classifier	List<AutomatonOpaqueObject>	Read
Redefined Element	List<AutomatonOpaqueObject>	Read
Redefinition Context	List<AutomatonOpaqueObject>	Read
Representation	N/A	-
Role	List<AutomatonOpaqueObject>	Read
Specific Classifier	List<AutomatonOpaqueObject>	Read
Specifying Component	List<AutomatonOpaqueObject>	Read
Specifying Element	List<AutomatonOpaqueObject>	Read
Supplier Dependency	List<AutomatonOpaqueObject>	Read
Template Binding	List<AutomatonOpaqueObject>	Read
Template Parameter	N/A	-
To Do	String	Read/Write
Use Case	List<AutomatonOpaqueObject	Read/Write
Visibility	String	Read/Write

Getting the Child of an Element

You can get the child of an Element by typing the following:

- `<variableName>._getChild(String childElementName)`

If the above method cannot find the childElementName, it will throw an error.

To get the child of an element named ClassB from ClassA, for example, type:

```
var classA = AutomatonMacroAPI.getOpaqueObjectByPath("ClassA");
var classB = classA._getChild("ClassB");
```

Getting the Owner of an Element

You can get the owner of an element by typing the following:

- <variableName>._getOwner()

If the above method cannot find the owner, for example Model Data, it will throw an error.
To get the owner of an element named ClassA, for example, type:

```
var classA = AutomatonMacroAPI.getOpaqueObjectByPath("ClassA");
var classB = classA._getOwner();
```

Creating a New Element

You can create a new element by using its Meta-class name, such as a Class in Standard UML or a Requirement in SysML by using the following method:

- AutomatonMacroAPI.createElement(String metaClassName)

This method will return an opaque object of the created element. If the method cannot find the Meta-class, it will throw an exception.

To create a new Class & Requirement element (Javascript), for example, type:

```
AutomatonMacroAPI.createElement("Class");
AutomatonMacroAPI.createElement("Requirement");
```

Creating a Relationship Between Elements

You can create a relationship between elements by typing:

- AutomatonMacroAPI.createRelationship(AutomatonOpaqueObject opque1, AutomatonOpaqueObject opque2, String relationName)

To create a Copy & Abstraction relationship between Element1 and Element2 (Javascript), for example, type:

```
var a = AutomatonMacroAPI.getOpaqueObjectByPath("Element1");
var b = AutomatonMacroAPI.getOpaqueObjectByPath("Element2");
AutomatonMacroAPI.createRelationship(a, b, "Copy");
AutomatonMacroAPI.createRelationship(a, b, "Abstraction");
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Removing an Element

You can remove an Element from MagicDraw by typing the following:

- AutomatonMacroAPI.removeElement(AutomatonOpaqueObject opaqueObj)

To remove an Element1 (Javascript), for example, type:

```
var a = AutomatonMacroAPI.getOpaqueObjectByPath("Element1");
AutomatonMacroAPI.removeElement(a);
```

Adding a Stereotype to an Element

You can apply a stereotype to an element by typing either:

(i) AutomatonMacroAPI.addStereotype(AutomatonOpaqueObject opaque, AutomatonOpaqueObject opaqueStereotype)
(ii)AutomatonMacroAPI.addStereotypeByStereotypeName(AutomatonOpaqueObject opaque, String stereotypeName)

To add a StererotypeA to ClassA (Javascript), for example, type:

```
var classA = AutomatonMacroAPI.getOpaqueObjectByPath("ClassA");
var stereotypeA = AutomatonMacroAPI.getOpaqueObjectByPath("StereotypeA");
AutomatonMacroAPI.addStereotype(classA, stereotypeA);
```

Removing a Stereotype from an Element

You can remove a Stereotype from an element by typing either:

(i) AutomatonMacroAPI.removeStereotype(AutomatonOpaqueObject opaque, AutomatonOpaqueObject opaqueStereotype)
(ii) AutomatonMacroAPI.removeStereotypeByStereotypeName(AutomatonOpaqueObject opaque, String stereotypeName)

To remove a StereotypeA from ClassA (Javascript), for example, type:

```
var classA = AutomatonMacroAPI.getOpaqueObjectByPath("ClassA");
var stereotypeA = AutomatonMacroAPI.getOpaqueObjectByPath("StereotypeA");
AutomatonMacroAPI.removeStereotype(classA, stereotypeA);
```

Printing Element Details

If you want to know the method that is used in the opaque object of an element, you can print the element details by typing the following:

- AutomatonMacroAPI.printElementDetail(AutomatonOpaqueObject opaque);

This method will print the field, constructor, and method that belong to the opaque object in the MagicDraw Message dialog.

To print the details of ClassA, for example, type:

```
var classA = AutomatonMacroAPI.getOpaqueObjectByPath("ClassA");
AutomatonMacroAPI.printElementDetail(classA);
```

Recording Macros

Macro Engine has the capability to record changes in a model. It uses opaque objects to generate macros and record them. This capability is especially useful when you want to redo some of your repetitive tasks.

The following is a list of actions that you can record:

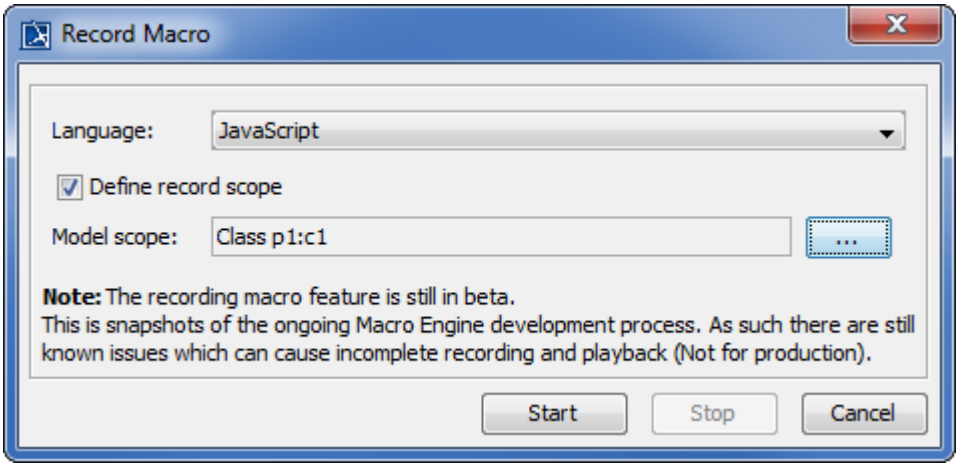
- Creating UML, Stereotype, and DSL elements
- Creating relationships between UML, Stereotype, or DSL elements

> You cannot move the element defined as a record scope during recording.

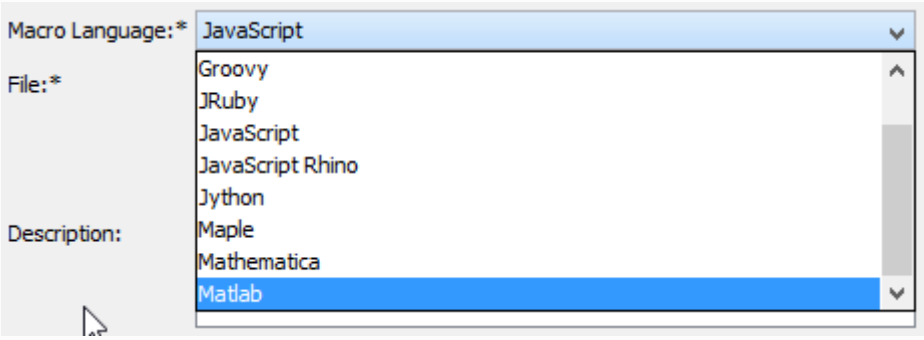
To record a macro

1. Click **Tools** > **Macros** > **Record Macro** on the MagicDraw main menu. The **Record Macro** dialog will open, see figure below.
2. Select a macro language in the **Language** box (you will see the default macro language that you have previously selected in that particular box (see (see page 2685)Selecting a Default Macro Language(see page 2685)).[1263]
3. Select the Define record scope check box and click the **Model Scope** button to locate a scope in the Containment tree. The generated macros will later record the change in the element by using a relative path that refers to the defined scope.
4. Click Start to start recording.
5. Work with the model in the scope you have defined.
6. Click Stop to stop recording. The **Record Macro** dialog will close and the **Create Macro** dialog will open, see figure below, showing the recorded macros.
7. Click either Save or Run.

[1263] http://docs.nomagic.com/display/MacroEngineUserGuide/Working+with+Macro+Engine#WorkingwithMacroEngine-selecting-a-default-macro-language

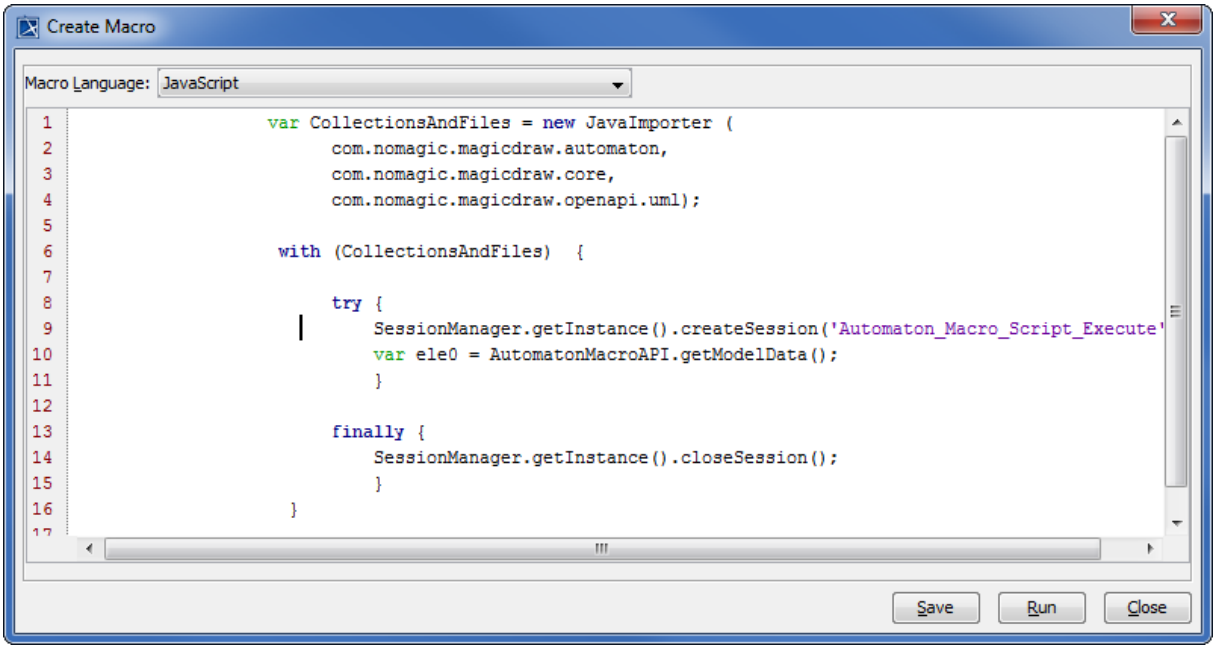


- If you do not open a project, the menu **Tools** > **Macros** > **Record Macro** will be disabled.
- You can select a record scope only before you start recording
- You cannot change a record scope during recording.
- If you do not define a record scope, the model **Data** will become the record scope.
- If you click the Cancel button, the **Record Macro** dialog will be closed.
- If you click the Start button, it will be changed to Pause and the Stop button will enabled.
- You cannot alter the **Language**, Define record scope, or **Model Scope** options after you click the Start button.
- If you click the Pause button, the recording will pause and the button will be changed to Resume.
- If you click the Resume button, the following things will happen:
  - the recording will continue
  - the button will be changed to Pause
- The recording mechanism of Macro Engine can generate code for Beanshell, Groovy, Javascript (Nashorn and Rhino), JRuby, and Jython. Unsupported languages will be filtered out of the



box.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



Supported scripting languages

Macro Engine supports five scripting languages: BeanShell, JavaScript (Nashorn and Rhino), JRuby, Jython, and Groovy.

BeanShell
BeanShell is a lightweight scripting language for Java. The shipped version is 2.1.7. The advantage of using BeanShell is that its syntax is compatible with Java; therefore, you can use the code assistant feature in most Java IDE. The BeanShell syntax documentation is available at http://www.beanshell.org/docs.html.

JavaScript
JavaScript is a scripting language based on Java syntax. The documentation is available at: http://docs.oracle.com/javase/8/docs/technotes/guides/scripting/nashorn/index.html and https://developer.mozilla.org/en/Rhino_documentation.

JRuby
JRuby is a 100% pure-Java implementation of the Ruby programming language. Macro Engine uses JRuby 1.7.11 The documentation is available at http://jruby.org/documentation.

Jython
Jython, a successor of JPython, is a Python programming language implemented in Java. The documentation is available at http://wiki.python.org/jython. Besides automating MagicDraw by using Jython scripting, the plugin development in the Jython programming language is also supported in MagicDraw. For more information on Jython, see JPython Scripting in MagicDraw OpenAPI User Guide.

Groovy
Groovy is an agile and dynamic language for the Java Virtual Machine. Macro Engine uses Groovy 2.0.1. The advantage of using Groovy is that its Java-like syntax seamlessly integrates with the existing Java source code and libraries. It supports many IDEs that provide code completion and debugging. BeanShell scripts can be easily moved to Groovy, with some modifications. For more information on Groovy, go to http://groovy-lang.org/.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related pages

Using code completion to develop BeanShell scripts

You can use any text editor to develop a scripting language. However, a standard text editor lacks of code assistant features. Most scripting languages are loose type. For example, to define a variable in JavaScript, you need to type:

```
var a;
```

It is difficult to determine what type of "a" later in the source code. With BeanShell, you can use a variable without declaring it, for example:

```
a = new File("file.txt");
```

Or you can declare it first:

```
File a = new File("file.txt");
```

Java IDE does not officially support code completion for scripting languages. However, there is a workaround if you use BeanShell. First you need a Java IDE. If you do not have one, you can select NetBeans because it has the smallest file size. You can download the latest version of NetBeans at http://www.netbeans.org/downloads/index.html. The NetBeans Java SE package is enough. Second you need to set up a MagicDraw classpath.

To set up a classpath point in the MagicDraw library in NetBeans

1. Click **Tools** > **Libraries** on the main menu. The **Library Manager** dialog will open.
2. Click the **New Library** button. The **New Library** dialog will open.
3. Specify a library name, for example, MD16.6. The Library type must be **Class Libraries**.
4. Click **OK** to close the **New Library** dialog.
5. Select your new library in the Libraries tree.
6. Click the **Add JAR/Folder** button and add all the JAR files in **<MagicDraw>/lib**.
7. Click **OK** to close the **Library Manager** dialog.

To develop a BeanShell script in NetBeans

1. Click **File** > **New Project** on the main menu to create a Java application project. The **New Project** dialog will open.
2. Select **Java** in the **Categories** box and **Java Application** in the **Projects** box, and then follow the instructions.
3. Expand your project node in the **Project** window. The Libraries node will appear.
4. Right-click the Libraries node and select **Add Library** from the shortcut menu.
5. Select the MagicDraw library that you have previously created (see "To set up a classpath point in the MagicDraw library in NetBeans:") and click **Add Library**.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

6. Click **File** > **New File** on the main menu to add a new Java file.
7. Select **Java** in the **Categories** box and **Empty Java File** in the **File Types** box, and then follow the instructions until finish.

You need to create a public static method in a Java file, for example, main() method, to follow the standard Java programming language. At the end of the file, insert a statement to call the static method. See the example in *create_project_elements.bsh* in the *<MagicDraw>/samples/product features/macros* directory.

> The official filename extension for BeanShell is *.bsh*. However, you can add a *.java* file to the BeanShell scripting language.

Using NetBeans IDE to develop Groovy Scripts

You will need NetBeans IDE 6.7.1 or 6.8 with Groovy Plugin to develop Groovy scripts. The Groovy support comes with NetBeans "Java" and "All" package. If your NetBeans does not have Groovy, download the plugin through NetBeans Plugin Manager. For more information about adding a new plugin, click **IDE Basics** > **Plugins** > **Updating the IDE** on the NetBeans **Help** menu.

Use the following three simple steps to develop and run a Groovy script in NetBeans:

1. Set up a classpath.
2. Develop a Groovy script.

To set up a classpath

---

1. Click **Tools** > **Libraries** on the main menu. The **Library Manager** dialog will open.
2. Click the **New Library** button. The **New Library** dialog will open.
3. Specify a library name, for example, MD16.8. The Library Type must be Class Libraries.
4. Click **OK** to close the **New Library** dialog.
5. Select your new library in the Libraries tree.
6. Click the **Add JAR/Folder** button and add all the JAR files in <MagicDraw>/lib.
7. Repeat steps 2 to 6 to add the Groovy library that is in *<MagicDraw>/plugins/com.nomagic.magicdraw.automaton/engines/groovy-2.0.1/embeddable/groovy-all-2.0.1.jar.*
8. Click **OK** to close the **Library Manager** dialog.

To develop a Groovy script

---

1. Click **File** > **New Project** on the main menu to create a Java application project. The **New Project** dialog will open.
2. Select **Java** in the **Categories** box and **Java Application** in the **Projects** box, and then click **Next**.
3. Choose a project location. Be sure that you do not select **Create Main Class**.
4. Click **Finish**.
5. Expand your project node in the **Project** window. The Libraries node will appear.
6. Right-click the Libraries node and select **Add Library** from the shortcut menu.
7. Click **File** > **New File** on the main menu to add a new Groovy file.
8. Select Groovy in the **Categories** box and Groovy Script in the **File Types** box.
9. Follow the instructions until finish.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Using Eclipse to develop Groovy scripts

Use the following three simple steps to develop and run a Groovy script in NetBeans:

1. Install Groovy-Eclipse Plugin.
2. Develop a Groovy script.

To install Groovy-Eclipse Plugin

---

1. Go to https://github.com/groovy/groovy-eclipse.
2. Follow the instructions for installing Groovy-Eclipse Plugin.

To develop a Groovy Script

---

1. Create a Groovy project.
2. Right-click the project in Package Explorer and click **Properties**.
3. Click Java Build Path in the tree on the left-hand side. Click the **Libraries** tab on the right pane.
4. Click **Add External JARs** to add all the JAR files in *<MagicDraw>/lib* and in *<MagicDraw>/plugins/com.nomagic.magicdraw.automaton/engines/groovy-2.0.1/embeddable/groovy-all-2.0.1.jar*.
5. Click **OK** to close the dialog.
6. Create a new Groovy class in the project. Remove a class declaration, and then put the script there.

Installing Gems for JRuby

To install a gem for Ruby engine inside MagicDraw

---

1. Open command line.
2. Go to folder *<MagicDraw folder>\plugins\com.nomagic.magicdraw.automaton\engines*.
3. Enter the following command to install a gem.

```
java -jar jruby-complete-<version>.jar -S gem install [--user-install] <gem name1>
<gem name2> ...
```

- <version> is the version of the JRuby, for example, 1.5.1 or 1.6.7.2.
- <gem name> is the name of a gem.
- The three dots (...) mean that you can type a list of gem names there.

Once the gem has been installed, you can use it in MagicDraw Macro Engine, for example

```
require 'java'
require 'rubygems'
require 'uuid'
Application = com.nomagic.magicdraw.core.Application
uuid = UUID.new Application.getInstance().getGUILog().log(uuid.generate);
```

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

The above example shows how to create a macro that can generate a unique ID by using a gem.

To install a gem for the existing Ruby environment on your machine and use it in Macro Engine

1. Add the following properties to specify the home and library of JRuby in *<MagicDraw>/bin/mduml.properties* under JAVA_ARGS section (assuming you have JRuby installed in *C:/jruby-1.5.3* on your machine).

```
-Djruby.home\="C:/jruby-1.5.3" -Djruby.lib\="C:/jruby-1.5.3/lib"
```

For example:

```
..JAVA_ARGS=-Xmx800M -Djruby.home\="C:/jruby-1.5.3" -Djruby.lib\="C:/jruby-1.5.3/lib"
```

2. Change the JRuby library path to navigate to your *jruby.jar* in *<MDdir>/plugins/com.nomagic.magicdraw.automaton/plugin.xml*.

```
... <library name="c:/jruby-1.5.3/lib/jruby.jar"/>
```

1. The command in this section should run in *<MDdir>/plugins/com.nomagic.magicdraw.automaton/engine*.
2. If you have a whitespace in the property file path, you need to type a double quote to wrap both the property and its value, for each property, for example:
3. *"-Djruby.home=C:/my white space/jruby-1.5.3" "-Djruby.lib=C:/my white space/jruby-1.5.3/lib"*

Adding a scripting language to MagicDraw

You can add a new scripting language that supports JSR-223 to MagicDraw by editing the Macro Engine's *plugin.xml* file.

To add a new scripting language

1. Open the *plugin.xml* file in *<MagicDraw>/plugins/com.nomagic.magicdraw.automaton/plugin.xml*.
2. Add the following jar file tags to the runtime section and save the file.

```
... <library name="<path to jar file>"/>
```

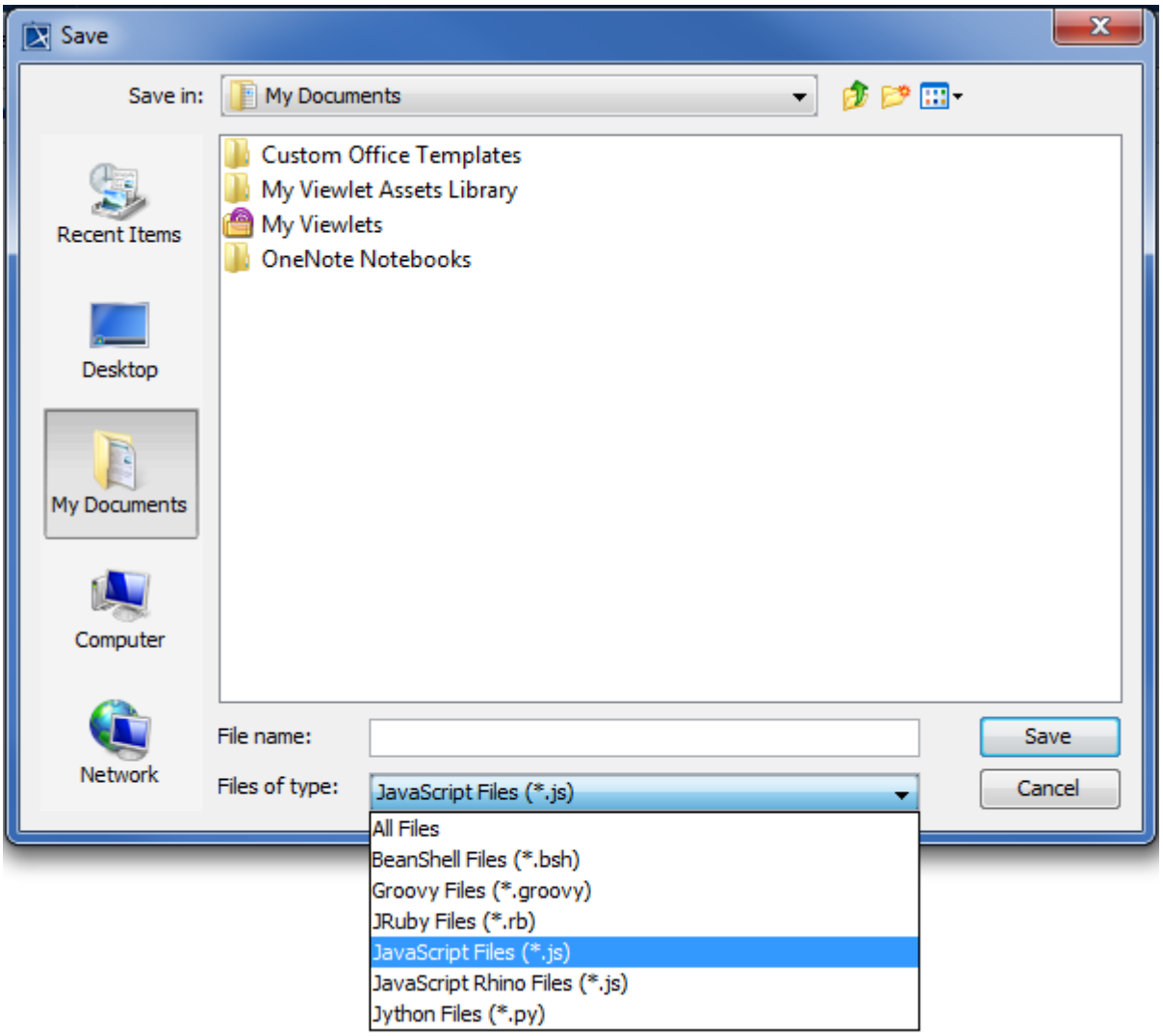
The path to the jar file is relative to *<MagicDraw>/plugins/com.nomagic.magicdraw.automaton* or the absolute pathname can be used, for example, if you want to add Sleep programming language, add the following tag to the runtime section:

```
... <library name="c:/sleep_2.1.jar"/>
```

A slash (/) is used as a directory separator for all platforms, including Windows. The new language will be displayed in most of the Macro Engine dialogs, for example, in the Macros section in the **Environment Options** dialog, the **Create Macro** dialog, and in the **Macro Information** dialog. The language name is automatically configured from the information inside the jar file. This version of Macro Engine does not allow you to modify the language name.

Script Filename Extension Filter

When you open an **Open** file dialog to browse for a script file in Macro Engine, you can find the filename extension for the script file in the dialog file filter. If you have added a new scripting language to MagicDraw, for example, Sleep programming language, Macro Engine will add the filename extension (**.sl*)" in the **Open** file dialog, see figure below.



The filename extensions are derived from jar files. There can be more than one filename extension for each engine. However, the current Macro Engine version only shows the first filename extension that is queried from the engine jar file.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

You can remove a scripting language from Macro Engine by removing the associated library tag in the *plugin.xml* file. Once the tag has been removed, the language will be removed from all of the dialogs in Macro Engine. However, the script configured to be used with the language will not be removed from MagicDraw. The language of the script will be reset to the default language, which has been previously configured in the MagicDraw **Environment Options** dialog.

UML 2.5.1 Meta Model

1264

UML 2.5.1 support

Modeling tools developed by No Magic, Inc. always support the latest versions of UML standards. Version 19.0 supports changes in the UML specification from version 2.5 to 2.5.1.

1264 https://docs.nomagic.com/display/MP/UML+Meta+Model

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

UML 2.5.1 provides several minor enhancements:

- Vertexes are redefinable in State Machines. A vertex is either a State, or Pseudo State, or Connection Point Reference. In earlier standard versions, only the State was redefinable.
- The semantic of Context and Constrained Element properties of the Constraint are clarified in the standard specification.
- The Behavior of an Opaque Expression is now allowed to have input parameters.

The latter two items do not affect the modeling tool functionality.

Migration to UML 2 user guide

With MagicDraw 10, we moved from the UML 1.4 metamodel to UML 2. It is very important to become familiar with these changes before migrating your projects to UML 2, as this is a significant change.

We've put all our best efforts into implementing the UML 1.4 model migration to version UML 2. However, the new standard is not backward compatible. Situations may occur where the migration facilities may not import 100% of the model and data loss may occur.

Related pages

Basic concepts and instructions

Unlike previous MagicDraw versions (which were based on UML 1.4), MagicDraw is based on the UML 2 metamodel. One of the biggest issues with UML 2 is that it is not backward compatible with UML 1.4. UML 2 also defines new model element types and introduces new properties. Some elements and properties were removed.

Projects based on UML 1.4 must converted to UML 2 to work with MagicDraw. The project load mechanism in MagicDraw will automatically detect UML 1.4 projects andconvert it to UML 2.

We strongly recommend backing up all projects before conversion.

We also do not recommend migrating to UML 2 in the middle of an intensive project.

You will not be able to open the converted project in previous MagicDraw versions. Only native MagicDraw files with UML 1.4 models can be loaded correctly into MagicDraw. We recommend that you load third-party XMI files with UML 1.4 into a previous MagicDraw version (up to 9.5) first, then save into a native MagicDraw XML file and try to open it (migrate to UML 2) using MagicDraw.

As of version 18.1, a module is referred as a used project. The keywords "used project" and "project usage" replace the keyword "module" everywhere in MagicDraw UI.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Related pages

- Migration of projects using MagicDraw Project Converter(see page 2725)
- Migration of used projects and profiles(see page 2725)
- Migration of Teamwork Projects(see page 2725)
- Tracing migration problems(see page 2725)
- Autoupdate issues(see page 2725)

Migration of projects using MagicDraw Project Converter

Since version 17.0.1, MagicDraw supports UML 2.x notation only. MagicDraw 17.0 is the last version that supports the UML 1.x notation. To open a project based on UML 1.x, such as a project created with MagicDraw 9.x or earlier versions, you must convert your project to a MagicDraw 17.0 project.

To make the conversion process easier, you can use a free standalone MagicDraw Project Converter. This converter is specifically designed to convert MagicDraw 9.x or earlier version projects to MagicDraw 17.0 projects.

To start the MagicDraw Project Converter

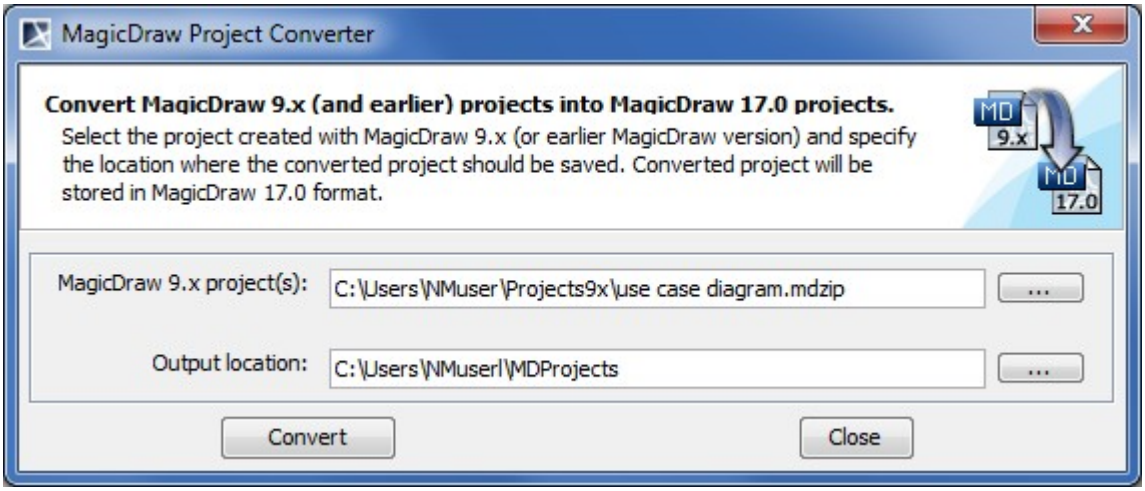
1. Download MagicDraw Project Converter at https://www.magicdraw.com/files/plugins/MagicDraw_Project_Converter.zip or please open the No Cost Plugin page at https://www.magicdraw.com/download_no_cost_plugins#free[1265] and select the converter from the plugins list.
2. Extract the file.
3. Go to the bin directory and run *converter.exe* to start MagicDraw Project Converter.

To convert a project

1. Open MagicDraw Project Converter.
2. In the **MagicDraw 9.x project(s)** box, select the project you want to convert.
3. In the **Output location** box, specify the location wherein the converted project will be saved.
4. Click **Convert**.

1265 http://www.magicdraw.com/download_no_cost_plugins#free

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.



If your project contains modules, you must save them as separate projects at first. Start to save from modules that do not have two-way interconnections.

Related pages

- Basic concepts and instructions(see page 2724)
- Migration of used projects and profiles(see page 2726)
- Migration of Teamwork Projects(see page 2726)
- Tracing migration problems(see page 2726)
- Autoupdate issues(see page 2726)

Migration of used projects and profiles

On this page

- Converting Used Projects(see page 2726)
- Solving Cyclic Usages(see page 2726)
- Converting Stereotypes/Profiles(see page 2726)

MagicDraw projects may use multiple used projects or profiles.

A project based on UML 1.4 cannot be loaded into MagicDraw if it uses UML 1.4 used projects. You must convert all used projects to UML 2 before converting your main project to UML 2.

Converting Used Projects

You should start from independent used projects that are not using other used projects.

1. Back up the used project.

2. Open the used project as a normal project. It will be converted into UML 2.
3. Save it into the same file.

> It is not possible to load a converted used project with previous MagicDraw versions or to use them in projects based on UML 1.4.

After all used projects are converted into UML 2, you will be able to load projects that are using them.

Solving Cyclic Usages

Cyclic usage occurs when used project A is using project B and used project B is using project A. In this case, neither used project A nor used project B can be loaded with MagicDraw. The solution to this problem is:

1. Open one of the used projects with previous MagicDraw version.
2. Try to solve cyclic usage by importing (merging) one used project into another or remove cyclic usage.

Converting Stereotypes/Profiles

Stereotypes in a UML 2 metamodel must be contained by a Profile model element, and all tags must be defined as properties of a stereotype. By contrast, in the UML 1.4 metamodel you could store stereotypes and tags anywhere in the model with no restrictions.

To fix this problem during the migration process, MagicDraw creates a Profile named "**Profile_for_<project_name>**" and moves all stereotypes from various packages into it.

For tags not owned by a stereotype, a new owning stereotypes is created. It will have the same name as the tag.

> **Example**
> If you had a tag definition called "duration", after project conversion to UML 2 you will have a stereotype «duration», which will have a tag definition "duration". The stereotype will be placed in a Profile called "**Profile_for_<project_name>**". All elements that had this tag in UML 1.4 will also have the stereotype assigned in UML 2.

The UML Standard Profile was redesigned as well. Base classes for some stereotypes have been changed; some stereotypes were removed. Stereotypes with modified base classes will still be assigned to elements, even if the elements cannot have this stereotype assigned. These elements will be marked as problematic.

> For more details, see Tracing migration problems.

All stereotypes used in old projects but removed from UML Standard Profile will be added into the new Profile "**Profile_for_<project_name>**".

Related pages

- Basic concepts and instructions(see page 2724)

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

- Migration of projects using MagicDraw Project Converter(see page 2725)
- Migration of Teamwork Projects(see page 2744)
- Tracing migration problems(see page 2750)
- Autoupdate issues(see page 2750)

Migration of Teamwork Projects

We strongly recommend that you install the MagicDraw Teamwork Server into a new location and prepare old Teamwork Server projects for migration.

Preparing old Teamwork projects

1. Ensure that all changes are saved into the old Teamwork Server. Notify all teamwork users to commit all changes and unlock all data.
2. Please review the project to avoid possible migration problems (see Problems and solutions of data mapping(see page 2728)) and make appropriate modifications.
3. Install the MagicDraw Teamwork Server to a new location.
4. During the first launch of Teamwork Server, select the **Import all projects** check box in the opened **Import Configuration** dialog box.

The following chapters contain more details about migrating Teamwork projects:

Related pages

- Basic concepts and instructions(see page 2724)
- Migration of projects using MagicDraw Project Converter(see page 2725)
- Migration of used projects and profiles(see page 2726)
- Migration of Teamwork Projects(see page 2728)
- Tracing migration problems(see page 2728)
- Autoupdate issues(see page 2728)

Converting Teamwork Projects

All teamwork projects must be converted to UML 2 data following the same instructions for regular MagicDraw projects. See Migration of used projects and profiles(see page 2726).

We recommend that you convert all teamwork used projects before converting projects. All standard profiles, bundled with MagicDraw installations, will be automatically updated on the first Teamwork Server launch.

Steps for project or used project conversion:

1. Open the latest project version from the Teamwork Server.
2. Commit it to the Server.

Only the latest project version will be converted into UML 2. All old versions (history) will remain based on UML 1.4. Any old versions can be opened only in read-only mode.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Special Cases Of Teamwork Used Projects

When a fixed version of a teamwork used project is used

If an old project uses a fixed version of a teamwork used project (fixed version), this option will be automatically turned off after conversion. Your project will use the latest version of the teamwork used project.

When you load a project that uses a fixed version of a teamwork project, you will be warned that MagicDraw will automatically switch to the latest version of a teamwork used project. If it is very important to use a particular version of a teamwork used project, you should not commit the loaded project to the teamwork server.

Follow these steps to preserve usage of a particular version of a teamwork used project:

1. Login into Teamwork Server with MagicDraw. From the **Teamwork** menu, choose **Projects**.
2. Select your project, expand the used projects tree and check which version the teamwork project uses.
3. Find the used project in the list of projects (select the **Show Used Projects** check box). Click the **Versions** button to open the **Versions** dialog box.
4. Select the version used by your project and set it as the latest one (click **Set As Latest**).
5. Click the **Open** button to open this version. During this step you will convert the latest version of the teamwork used project to UML 2.
6. Commit the converted used project into the Teamwork Server and close it.
7. Open a project that uses a fixed version of this used project. The last used project version will be loaded.
8. Commit the converted project into the Teamwork Server and close it.
9. From the **Teamwork** menu, choose **Projects** again.
10. Select your project in the list of projects and expand the tree of used projects. Change the used project version from **Latest** to a latest version number.
11. Find the used project in the projects list (select the **Show Used Projects** check box). Click the **Versions** button to open the **Versions** dialog box.
12. Select the latest version existing before these actions. Set this version as the latest one (click **Set As Latest**).
13. Open the used project and commit it to the Teamwork Server. During this step you will convert the latest version of the teamwork used project to UML 2.

After these steps, your project will use a fixed version of a teamwork used project that is converted to UML 2, but other projects will be able to use the latest version of this used project.

If multiple projects use various versions of the same used project, repeat these steps for each project.

When a used project/profile is used by a custom diagram

Before converting your projects, we recommend that you convert your teamwork used projects used in custom diagrams. All standard profiles, bundled with MagicDraw installations, will be converted automatically on the first Teamwork Server launch

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Converting Local Versions of Teamwork Projects

You can commit locally saved teamwork projects based on UML 2 into Teamwork Server only when there are no newer versions on the server and the user is connected to the Teamwork Server.

> ⚠ After a Teamwork project is converted to UML 2, it will not be possible to commit local versions based on UML 1.4.

Converting Teamwork Projects on IDE Integration

1. Ensure that all changes are saved into the old Teamwork Server. Notify all teamwork users to commit all changes and unlock all data.
2. Launch the standalone MagicDraw client and login into Teamwork Server.
3. Convert teamwork projects to UML 2.

   > ⓘ For more details, see Converting Teamwork Projects(see page 2728).

4. Save the integration project locally by overriding the old project file.

Problems and solutions of data maping

To ensure that your migration process is as fluent as possible, we recommend reading the following suggestions to modify your projects based on UML 1.4 before conversion to UML 2 to avoid mapping problems.

> ⚠ All modifications based on these recommendations should be applied using MagicDraw 9.5 or older.

Class Diagram

	Issue	Solution
Dependencies	Dependency can be connected only between NamedElements in UML 2. Dependencies between other elements will be deleted after load.	Try to avoid dependencies between generalizations, merge, import and other relationships that are not NamedElements in UML 2. If a dependency contains important information, try to connect it to another semantically close element supported in UML 2.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Subsystem	Subsystem is no longer subtype of package.	Subsystem will be automatically converted to a component with stereotype «subsystem» during conversion. If the subsystem had inner elements, they will be moved to a package that now owns the component.
Stereotypes and tags	Stereotypes in UML 2 metamodel must be contained by Profile model element, and all tags must be defined as properties of stereotype. The UML 1.4 metamodel allowed you to store stereotypes and tags anywhere in the model with no restrictions.	As described in Converting Stereotypes/ Profiles(see page 0), MagicDraw will create new profile and owner stereotypes for all tags. However, you may want to do some changes manually. We recommend creating stereotypes for “loose” tags. Grouping multiple tags into the same stereotype may be a good option as well. If you have a UML 1.4 based project with all stereotypes placed in a package, assign a «profile» stereotype to this package. In this case the package will be converted to a Profile model element. If you had many packages with stereotypes, you can move them into a new package with the stereotype «profile» assigned.

Sequence Diagram

	Issue	Solution
Concurrent lifelines and message branching	Concurrent lifelines and message branching are no longer supported in a sequence diagram. They will be loaded into MagicDraw, but the user will not be allowed to draw these elements.	UML 2 provides new notation for concurrency and branching using combined fragments. We recommend redrawing sequence diagrams before or after migration.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

State Diagram

	Issue	Solution
StubState and SynchState	StubState and SynchState elements are no longer supported in UML 2. StubStates and SynchStates will be removed together with connected transitions.	Please try to remove StubStates and SynchStates from state diagrams before migration. We suggest redesigning your model in alternative ways, trying to connect important transitions to other states.
Parameters of event	Event can't have parameters in UML 2. All parameters will be lost during the migration.	Use the parameters of Operation when using CallOperationEvent. Add additional parameters to an operation if you are using more CallOperationEvent parameters than the Operation has. Assign stereotypes and tags to Operation parameters instead of CallEvent parameters.

The same rules should be applied to the parameters of the SendSignalEvent and the attributes of the Signal.

Activity Diagram

	Issue	Solution
States	UML 2 Activity diagram is no longer based on StateMachine, so State model elements (SimpleState, CompositeState, and ConcurrentState) can no longer be used in the activity diagram.	Redraw your activity diagrams without using ConcurrentStates, move internal elements from ConcurrentState to the diagram and show concurrent flows by using synchronization bars or other notation. CompositeState in activity diagrams should be modified in a similar manner by moving internal elements into the diagram. SimpleState will be mapped to ObjectNode with this state assigned to the InState property.

Implementation Diagram

	Issue	Solution

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Deployment	Components cannot be deployed in Nodes in UML 2. Artifacts related to these components should be deployed instead.	During migration, MagicDraw will automatically create artifacts (named as components) deployed in Nodes and will manifest Components (Manifestation relationship between artifacts and components will be created). The diagram view will not change - Components will still be located over Nodes; however, these components are not added into Nodes. You should redraw your diagrams, showing artifacts and relations between components, artifacts and nodes. Drag artifacts created automatically from the browser tree directly to diagrams, use “Display paths” to display relationships.

On this page

- Class Diagram(see page 2730)
- Sequence Diagram(see page 2731)
- State Diagram(see page 2732)
- Activity Diagram(see page 2732)
- Implementation Diagram(see page 2732)

Tracing migration problems

You can select automatic marking of problematic elements during the conversion process.

All recognized data mapping or other migration problems will be marked by adding a ToDo tagged value with the problem description into the problematic element.

To trace marked elements

First, try to avoid the cases described previously.

If data loss cannot be avoided, all problems, conflicts, and data loss occurrences are registered to the model element's **ToDo** tags on project load:

1. Load your project. The warning message appears.
2. Click **Yes**. Warnings are displayed in **ToDo** tags.
3. Double-click the element or, from the shortcut menu, choose **Specification**. Information about changes is added to the **ToDo** field in the **General** tab.

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Use the **Find ToDo** feature to search for elements containing **ToDo** information. From the **Tools** main menu, choose **Find ToDo**.

Related pages

- Basic concepts and instructions(see page 2724)
- Migration of projects using MagicDraw Project Converter(see page 2725)
- Migration of used projects and profiles(see page 2726)
- Migration of Teamwork Projects(see page 2728)
- Autoupdate issues(see page 2734)

Autoupdate issues

Be cautious with MagicDraw autoupdate.

We do not recommend using autoupdate, as you may need a previous MagicDraw version installation for migrating your projects to UML 2.

MagicDraw is able to convert UML 1.4 to UML 2, but some mapping problems may appear. Previous MagicDraw version should be used for solving these problems before conversion. For more details about mapping solutions, see Problems and solutions of data mapping(see page 2730).

We recommend downloading the MagicDraw installation manually and installing it into a new location (not the location suggested by default).

If you have read this document carefully and you think you will not have problems in all your projects with elements migration, you may use the autoupdate function.

Autoupdate for the MagicDraw Teamwork Server is disabled.

Related pages

- Basic concepts and instructions(see page 2724)
- Migration of projects using MagicDraw Project Converter(see page 2725)
- Migration of used projects and profiles(see page 2726)
- Migration of Teamwork Projects(see page 2728)
- Tracing migration problems(see page 2733)

Legal Notices

This page specifies the patents, trademarks, copyrights, and restricted rights for MagicDraw 19.0:

Trademarks

NO MAGIC, MAGICDRAW, MAGICGRID, MAGICENTERPRISE, CAMEO, CAMEO EWORK, CAMEOSUITE, TEAMWORK CLOUD, 3DEXPERIENCE, the Compass icon, the 3DS logo, CATIA, SOLIDWORKS, ENOVIA, DELMIA, SIMULIA, GEOVIA, EXALEAD, 3D VIA, BIOVIA, NETVIBES, IFWE and 3DEXCITE are commercial trademarks or registered trademarks of Dassault Systèmes, a French "société européenne" (Versailles Commercial Register # B 322 306 440), or its subsidiaries in the United States and/or other countries. **All other trademarks are owned by their respective owners.** Use of any Dassault Systèmes or its subsidiaries trademarks is subject to their express written approval.

DS Offerings and services names may be trademarks or service marks of Dassault Systèmes or its subsidiaries.

Third-Party Copyrights Notices

Certain portions of MagicDraw 19.0 contain elements subject to copyright owned by the following entities:

© 2007 Python Software Foundation; All Rights Reserved

© 1998, 2019, Oracle and/or its affiliates. All rights reserved.

© Siemens PLM

© The MathWorks Inc.

© 2019 yWorks

© 2001-2019 Sferyx Srl. All rights reserved.

© 2002-2018, JIDE Software, Inc. All Rights Reserved.

© Pure Varients

© Maplesoft

MagicDraw 19.0 may include open source software components. Source code for these components is available upon request. The original licensors of said open source software components provide them on an "as is" basis and without any liability whatsoever to customer (or licensee).

IP Asset Name	IP Asset Version	Copyright notice
Under Apache 1.1[1266]		

1266 https://www.apache.org/licenses/LICENSE-1.1

IP Asset Name	IP Asset Version	Copyright notice
oro	2.0.8	Copyright (c) 2000-2002 The Apache Software Foundation
Under Apache 2.0[1267]		
apache.log4j	1.2.14	Copyright © 1999-2018 The Apache Software Foundation
apache.log4j	1.2.15	Copyright © 1999-2018 The Apache Software Foundation
apache.servicemix.bundles.jsr305	2.0.1.1	Copyright © 1999-2018 The Apache Software Foundation
apache-lucene	5.5.2	Copyright © 1999-2018 The Apache Software Foundation
apiguardian	1.0.0	© API Guardian Team
avalon.logkit	2.1	Copyright © 2005-2019 The Apache Software Foundation
avalon-framework	4.1.5	Copyright © 1999-2018 The Apache Software Foundation
avalon-framework	4.2.0	Copyright © 1999-2018 The Apache Software Foundation
axis.config		Copyright © 2005-2019 The Apache Software Foundation
axis	1.4	Copyright © 1999-2018 The Apache Software Foundation
axis-jaxrpc	1.4	Copyright © 1999-2018 The Apache Software Foundation

1267 http://www.apache.org/licenses/LICENSE-2.0

IP Asset Name	IP Asset Version	Copyright notice
axis-saaj	1.4	Copyright © 1999-2018 The Apache Software Foundation
axis-wsdl4j	1.5.1	Copyright © 1999-2018 The Apache Software Foundation
batik		Copyright 1999-2018 The Apache Software Foundation
bsf	2.4.0	Copyright © 1999-2018 The Apache Software Foundation
bsh	2.1.7	Copyright ©1997-2012 Patrick Niemeyer
caffeine	2.1.0	Copyright © 1999-2018 The Apache Software Foundation
caffeine	2.6.2	Copyright © 1999-2018 The Apache Software Foundation
cassandra-drive-core	3.1.4	Copyright 2017, DataStax
classmate	1.3.3	Copyright ©2009-2011 FasterXML
classworlds	1.1.2	Copyright 2002 © The Codehaus. All Rights Reserved.
com.datastax.driver.core	3.1.2	Copyright 2017, DataStax
com.fasterxml.jackson.core.jackson-annotations	2.6.0	Copyright © 1999-2018 The Apache Software Foundation
com.fasterxml.jackson.core	2.6.3	Copyright © 1999-2018 The Apache Software Foundation

IP Asset Name	IP Asset Version	Copyright notice
com.google.gson	2.7.0	Copyright 2008-2011 Google Inc.
com.google.guava	18.0.0	Copyright 2008, Google Inc
com.typesafe.akka	2.5.10	Copyright © 2011-2016 Typesafe Inc.
commons-beanutils	1.9.3	Copyright © 1999-2018 The Apache Software Foundation
com.typesafe.config	1.3.0	Copyright © 2011-2016 Typesafe Inc.
com.typesafe.sslconfig	0.2.2	Copyright © 2011-2016 Typesafe Inc.
commons-beanutils-core	1.8.0	Copyright © 1999-2018 The Apache Software Foundation
commons-cli	1	Copyright © 1999-2018 The Apache Software Foundation
commons-cli	1.2	Copyright © 1999-2018 The Apache Software Foundation
commons-codec	1.1	Copyright © 1999-2018 The Apache Software Foundation
commons-codec	1.9	Copyright © 1999-2018 The Apache Software Foundation
commons-collections	3.2.1	Copyright © 1999-2018 The Apache Software Foundation
commons-collections	3.2.2	Copyright © 1999-2018 The Apache Software Foundation

IP Asset Name	IP Asset Version	Copyright notice
commons-configuration	1.6	Copyright © 1999-2018 The Apache Software Foundation
commons-digester	2	Copyright © 1999-2018 The Apache Software Foundation
commons-digester	2.1	Copyright © 1999-2018 The Apache Software Foundation
commons-discovery	0.2	Copyright © 1999-2018 The Apache Software Foundation
commons-discovery	0.4	Copyright © 1999-2018 The Apache Software Foundation
commons-httpclient	3.1	Copyright © 1999-2018 The Apache Software Foundation
commons-io	1.3.1	Copyright © 1999-2018 The Apache Software Foundation
commons-io	1.4	Copyright © 1999-2018 The Apache Software Foundation
commons-io	2.4	Copyright © 1999-2018 The Apache Software Foundation
commons-jexl	1.1	Copyright © 1999-2018 The Apache Software Foundation
commons-lang	2.4	Copyright © 1999-2018 The Apache Software Foundation
commons-lang	2.6	Copyright © 1999-2018 The Apache Software Foundation
commons-lang3	3.4	Copyright © 1999-2018 The Apache Software Foundation

IP Asset Name	IP Asset Version	Copyright notice
commons-logging	1.1.1	Copyright © 1999-2018 The Apache Software Foundation
commons-logging	1.2	Copyright © 1999-2018 The Apache Software Foundation
commons-rdf-api	0.1.0	Copyright © 1999-2018 The Apache Software Foundation
commons-scxml	0.9	Copyright © 1999-2018 The Apache Software Foundation
config	1.2.1	© Lightbend
derby	10.6.1.0	Copyright © 1999-2018 The Apache Software Foundation
derbytools		Copyright © 1999-2018 The Apache Software Foundation
drools	5.3.1	© Copyright 2006-2019, Red Hat, Inc.
drools	5.3.1	© Copyright 2006-2019, Red Hat, Inc.
dropwizard.metrics	3.1.2	© Copyright 2010-2013 Coda Hale and Yammer, Inc., 2014-2017 Dropwizard Team
ehcache	2.10.1	© Terracotta, Inc.
fluent-hc	4.4	Copyright © 1999-2018 The Apache Software Foundation

IP Asset Name	IP Asset Version	Copyright notice
fop		Copyright © 1999-2018 The Apache Software Foundation
geronimo-annotation	1.1.1	Copyright © 1999-2018 The Apache Software Foundation
geronimo-jaspic	1.1	Copyright © 1999-2018 The Apache Software Foundation
geronimo-jta	1.1.1	Copyright © 1999-2018 The Apache Software Foundation
gpars	1.2.1	Copyright © The GPars Team, 2016.
groovy	2.4.15	Apache Groovy Copyright © 2003-2018
groovy	2.4.11	Copyright © 1999-2018 The Apache Software Foundation
gson	2.2.4	Copyright 2008 Google Inc.
gson	2.2.2	Copyright 2008 Google Inc.
guava	19	Copyright 2008 Google Inc.
guice	4	Copyright © Google
hibernate-validator	5.2.4	Copyright © 2009 - 2013 Red Hat, Inc. & Gunnar Morling

IP Asset Name	IP Asset Version	Copyright notice
hibernate-validator	5.3.5	Copyright © 2009 - 2013 Red Hat, Inc. & Gunnar Morling
httpclient	4.4	Copyright © 2005-2019 The Apache Software Foundation
httpclient	4.5.1	Copyright © 2005-2019 The Apache Software Foundation
httpclient	4.5.2	Copyright © 2005-2019 The Apache Software Foundation
httpclient	4.5.3	Copyright © 2005-2019 The Apache Software Foundation
httpclient	4.5.5	Copyright © 2005-2019 The Apache Software Foundation
httpclient-cache	4.4	Copyright © 2005-2019 The Apache Software Foundation
httpclient-cache	4.5.1	Copyright © 2005-2019 The Apache Software Foundation
httpclient-win	4.4	Copyright © 2005-2019 The Apache Software Foundation
httpcore	4.4.3	Copyright © 2005-2019 The Apache Software Foundation
httpcore	4.4.4	Copyright © 2005-2019 The Apache Software Foundation
httpcore	4.4.5	Copyright © 2005-2019 The Apache Software Foundation
httpcore	4.4.6	Copyright © 2005-2019 The Apache Software Foundation

IP Asset Name	IP Asset Version	Copyright notice
httpcore	4.4.9	Copyright © 2005-2019 The Apache Software Foundation
httpcore	4.4	Copyright © 2005-2019 The Apache Software Foundation
httpmime	4.4	Copyright © 1999-2018 The Apache Software Foundation
io.aeron	1.7.0	Copyright 2014-2019 Real Logic Limited
io.netty	4.1.21	Copyright © 2019 The Netty project
ivy	2.4.0	Copyright © 2005-2019 The Apache Software Foundation
jackson-annotations	2.6.0	Copyright © 2005-2019 The Apache Software Foundation
jackson-annotations	2.8.0	Copyright © 2005-2019 The Apache Software Foundation
jackson-annotations	2.9.5	Copyright © 2005-2019 The Apache Software Foundation
jackson-core	2.6.3	Copyright © 2005-2019 The Apache Software Foundation
jackson-core	2.8.1	Copyright © 2005-2019 The Apache Software Foundation

IP Asset Name	IP Asset Version	Copyright notice
jackson-core	2.8.8	Copyright © 2005-2019 The Apache Software Foundation
jackson-core	2.9.5	Copyright © 2005-2019 The Apache Software Foundation
jackson-core-asl	1.9.2	Copyright © 2005-2019 The Apache Software Foundation
jackson-databind	2.6.3	Copyright © 2005-2019 The Apache Software Foundation
jackson-databind	2.8.1	Copyright © 2005-2019 The Apache Software Foundation
jackson-databind	2.8.8	Copyright © 2005-2019 The Apache Software Foundation
jackson-databind	2.9.5	Copyright © 2005-2019 The Apache Software Foundation
jackson-dataformat	2.8.1	Copyright © 2005-2019 The Apache Software Foundation
jackson-mapper-asl	1.9.2	Copyright © 2005-2019 The Apache Software Foundation
jansi	1.11	Copyright © 2009, Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.
javassist	3.21.0	Copyright © 1999-2019 by Shigeru Chiba, All rights reserved.
jboss.netty	3.10.6	Copyright © 2005-2019 The Apache Software Foundation
jboss-logging	3.3.1	Copyright © 2005-2019 The Apache Software Foundation

IP Asset Name	IP Asset Version	Copyright notice
jbpm	5.2.0	© Copyright 2006-2019, Red Hat, Inc.
jcommander	1.47	Copyright © 2010 the original author or authors {Cedric Beust}
jettison	1.3.3	Copyright © 2006, Envoi Solutions LLC
jettison	1.3.7	Copyright © 2006, Envoi Solutions LLC
jetty	8.0.4	Copyright © Eclipse Foundation, Inc.
jffi	1.2.10	Copyright © Wayne Meissner
jna	3.5.1	Copyright © 2017 Matthias Bläsing, All Rights Reserved
jna	4.1.0	Copyright © 2017 Matthias Bläsing, All Rights Reserved
jnr-constants	0.9.0	Wayne Meissner
jnr-ffi	2.0.7	Wayne Meissner
jnr-posix	3.0.27	Wayne Meissner
jnr-x86asm	1.0.2	Wayne Meissner

IP Asset Name	IP Asset Version	Copyright notice
joda.time	2.9.1	Copyright ©2002-2016 Joda.org[1268]. All Rights Reserved.
joda-time-2.9.9	2.9.9	Copyright ©2002-2016 Joda.org[1269]. All Rights Reserved.
json_simple	1.1	Copyright 2009 Google LLC.
json-smart	1.3.1	Copyright 2014 Google LLC.
json-smart	2.2.1	Copyright © Chemouni Uriel
jsr305	2.0.1	Copyright © 2005-2019 The Apache Software Foundation
jsr305	3.0.2	Copyright © 2005-2019 The Apache Software Foundation
junit.platform.commons	1.0.0	Copyright © 2004-2018 jUnit Developers
JUnitParams	1.0.1	Copyright © 2004-2018 jUnit Developers
log4j	1.2.15	Copyright © 2005-2019 The Apache Software Foundation
log4j	1.2.16	Copyright © 2005-2019 The Apache Software Foundation
log4j	1.2.17	Copyright © 2005-2019 The Apache Software Foundation

1268 http://Joda.org
1269 http://Joda.org

IP Asset Name	IP Asset Version	Copyright notice
log4j-api	2.11.0	Copyright © 2005-2019 The Apache Software Foundation
log4j-api	2.8.2	Copyright © 2005-2019 The Apache Software Foundation
log4j-core	2.11.0	Copyright © 2005-2019 The Apache Software Foundation
log4j-core	2.8.2	Copyright © 2005-2019 The Apache Software Foundation
log4j-over-slf4j	1.7.2	Copyright © 2005-2019 The Apache Software Foundation
log4j-over-slf4j	1.7.25	Copyright © 2005-2019 The Apache Software Foundation
logkit	1.2	Copyright © 2011-2017, James Zhan
logstash-logback-encoder	5	Copyright © 2012–2016 Elasticsearch
lucene	4.10.3	Copyright © 2005-2019 The Apache Software Foundation
lz4-java	1.3.0	
maven-artifact	3.0.5	Copyright © 2005-2019 The Apache Software Foundation
maven-model	3.0.5	Copyright © 2005-2019 The Apache Software Foundation

IP Asset Name	IP Asset Version	Copyright notice
maven-plugin-api	3.0.5	Copyright © 2005-2019 The Apache Software Foundation
maven-plugin-registry	2.2.1	Copyright © 2005-2019 The Apache Software Foundation
maven	2.2.1	Copyright © 2005-2019 The Apache Software Foundation
metrics-core	3.1.4	Copyright 2010-2013 Coda Hale and Yammer, Inc., 2014-2017 Dropwizard Team
netty	4.1.21	Copyright © 2018 The Netty Project
netty-buffer	4.0.37	The Netty Project Contributors
netty-codec	4.0.37	The Netty Project Contributors
netty-common	4.0.37	The Netty Project Contributors
netty-handler	4.0.37	The Netty Project Contributors
netty-transport	4.0.37	The Netty Project Contributors
nimbus-jose-jwt	4	Copyright 2012 - 2018, Connect2id Ltd.

IP Asset Name	IP Asset Version	Copyright notice
not-yet-commons-ssl	0.3.9	Copyright © 2005-2019 The Apache Software Foundation
opencsv-2.3.jar	2.3	Copyright © 2011 Bytecode Pty Ltd.
openws-1.5.1.jar	1.5.1	Copyright 2006 University Corporation for Advanced Internet Development, Inc.
opentest4j	1.0.0	Copyright © 2019 The JUnit Team
org.agrona	0.9.12	Copyright 2014-2019 Real Logic Limited
org.apache.avro	1.4.0	Copyright © 2005-2019 The Apache Software Foundation
org.apache.cassandra	1.2.0	Copyright © 2005-2019 The Apache Software Foundation
org.apache.commons.beanutils	1.8.3	Copyright © 2005-2019 The Apache Software Foundation
org.apache.commons.cli	1.4.0	Copyright © 2005-2019 The Apache Software Foundation
org.apache.commons.codec	1.9.0	Copyright © 2005-2019 The Apache Software Foundation
org.apache.commons.collections	3.2.1	Copyright © 2005-2019 The Apache Software Foundation
org.apache.commons.fileupload	1.3.0	Copyright © 2005-2019 The Apache Software Foundation

IP Asset Name	IP Asset Version	Copyright notice
org.apache.commons.httpclient	3.1.0	Copyright © 2005-2019 The Apache Software Foundation
org.apache.commons.io[1270]	2.4.0	Copyright © 2005-2019 The Apache Software Foundation
org.apache.commons.lang	2.6.0	Copyright © 2005-2019 The Apache Software Foundation
org.apache.commons.logging	1.1.1	Copyright © 2005-2019 The Apache Software Foundation
org.apache.directory	1.0.0	Copyright © 2005-2019 The Apache Software Foundation
org.apache.felix.gogo	0.10.0	Copyright © 2005-2019 The Apache Software Foundation
org.apache.felix.scr	2.0.10	Copyright © 2005-2019 The Apache Software Foundation
org.apache.httpcomponents.httpclient	4.5.3	Copyright © 2005-2019 The Apache Software Foundation
org.apache.httpcomponents.httpcore	4.4.6	Copyright © 2005-2019 The Apache Software Foundation
org.apache.log4j	19.0.0	Copyright © 2005-2019 The Apache Software Foundation

1270 http://org.apache.commons.io

IP Asset Name	IP Asset Version	Copyright notice
org.apache.mina.core	2.0.16	Copyright © 2005-2019 The Apache Software Foundation
org.apache.servicemix.bundles.commons-	1.0.0	Copyright © 2005-2019 The Apache Software Foundation
org.apache.shiro.core	1.2.4	Copyright © 2005-2019 The Apache Software Foundation
org.apache.sshd.core	0.7.0	Copyright © 2005-2019 The Apache Software Foundation
org.apache.thrift	0.7.0	Copyright © 2005-2019 The Apache Software Foundation
org.bitbucket.b_c.jose4j	0.5.2	Copyright 2012-2015 Brian Campbell
org.codehaus.jettison.jettison	1.2.0	Copyright © 2006, Envoi Solutions LLC
org.freemarker.freemarker	2.3.20	Copyright © 2003 The Visigoth Software Society. All rights reserved.
org.jboss.netty	3.10.6	Copyright © 2019 The Netty project
org.restlet	2.2.3	Copyright © 2018 Restlet S.A.S.
org.uncommons.maths	1.2.2	© 2006 - 2012 Daniel W. Dyer
owlapi-distributio	5.0.2	Ignazio Palmisano

IP Asset Name	IP Asset Version	Copyright notice
plexus-build	0.0.7	Copyright © 2005-2019 The Apache Software Foundation
plexus-classworlds	2.4	Copyright © 2005-2019 The Apache Software Foundation
plexus-compiler	2.2	Copyright © 2005-2019 The Apache Software Foundation
plexus-component-annotations	1.5.5	Copyright © 2005-2019 The Apache Software Foundation
plexus-container-default	1.0.9.1	Copyright © 2005-2019 The Apache Software Foundation
plexus-interpolation	1.11	Copyright © 2005-2019 The Apache Software Foundation
plexus-utils	1.5.15	Copyright © 2005-2019 The Apache Software Foundation
poi	3.14	Copyright © 2005-2019 The Apache Software Foundation
poi	3.12	Copyright © 2005-2019 The Apache Software Foundation
qdox	1.12.1	Copyright 2002-2009 Joe Walnes and QDox Project Team
quartz.all	2.1.5	© Terracotta, Inc., a wholly-owned subsidiary of Software AG USA, Inc. All rights reserved.

IP Asset Name	IP Asset Version	Copyright notice
resolver	No version	Copyright © 2005-2019 The Apache Software Foundation
rhino	1.7R4	Copyright © 2005-2019 The Apache Software Foundation
rsclient	No version	Copyright © 2005-2019 The Apache Software Foundation
saaj	No version	Copyright © 2005-2019 The Apache Software Foundation
semargl	0.6.1	Copyright © 2005-2019 The Apache Software Foundation
serializer	2.7.1	Copyright © 2005-2019 The Apache Software Foundation
simple-xml	2.7.1	Copyright © 2005-2019 The Apache Software Foundation
soap	No version	Copyright © 1999-2003 The Apache Software Foundation. All rights reserved.
snakeyaml	1.17	Copyright ©2008, http://www.snakeyaml.org
spring	4.3.3	Copyright © 2005-2019 The Apache Software Foundation
spring	4.3.9	Copyright © 2005-2019 The Apache Software Foundation

IP Asset Name	IP Asset Version	Copyright notice
spring	5.0.5	Copyright © 2005-2019 The Apache Software Foundation
spring-boot	1.5.4	Copyright © 2005-2019 The Apache Software Foundation
spring-cql	1.5.4	Copyright © 2005-2019 The Apache Software Foundation
spring-data-cassandra	1.5.4	Copyright © 2005-2019 The Apache Software Foundation
spring-data-commons	1.13.4	Copyright © 2005-2019 The Apache Software Foundation
spring-jcl	5.0.5	Copyright © 2005-2019 The Apache Software Foundation
spring-security	4.1.3	Copyright © 2005-2019 The Apache Software Foundation
spring-security	4.2.3	Copyright © 2005-2019 The Apache Software Foundation
spring-security-jwt	1.0.5	Copyright © 2005-2019 The Apache Software Foundation
spring-security-jwt	1.0.9	Copyright © 2005-2019 The Apache Software Foundation
spring-security-saml2-core-	1.0.2	Copyright © 2005-2019 The Apache Software Foundation
spring-session	1.3.1	Copyright © 2005-2019 The Apache Software Foundation
spring-tx	4.3.9	Copyright © 2005-2019 The Apache Software Foundation

IP Asset Name	IP Asset Version	Copyright notice
stax-api	1.0.1	Copyright © 2006 Aleksander Slominski, Chris Fry. All Rights Reserved.
testng	6.8.13	Copyright © 2013 Cedric Beust, org.sonatype.oss
thymeleaf	2.1.5	Copyright © The Thymeleaf Team
thymeleaf-layout-dialect-	1.4.0	Copyright © The Thymeleaf Team
tomcat-embed	8.5.15	Copyright © 2005-2019 The Apache Software Foundation
unbescape	1.1.0	Copyright © The UNBESCAPE Team.
uncommons.maths	1.2.2	© 2006 - 2012 Daniel W. Dyer
validation-api	1.1.0	Copyright © Red Hat, Inc.
velocity	1.7	Copyright © 2016 The Apache Software Foundation
velocity.tools	1.4	Copyright © 2007 The Apache Software Foundation
velocity-dep	1.7	Copyright © 2016 The Apache Software Foundation
wagon-provider	1.0.6	Copyright 2009 The Apache Software Foundation. All rights reserved.
woodstox	4.1.4	Copyright © 2004-2018, Woodstox Project

IP Asset Name	IP Asset Version	Copyright notice
xalan	2.7.1	Copyright © 2005-2019 The Apache Software Foundation
xercesImpl	2.10.0	Copyright © 2005-2019 The Apache Software Foundation
xercesImpl	2.4.0	Copyright © 2005-2019 The Apache Software Foundation
xml-apis	1.4.01	Copyright © 2005-2019 The Apache Software Foundation
xmlbeans	2.6.0	Copyright © 2004-2013 The Apache Software Foundation
xmlgraphics-commons	1.5	Copyright © 2005-2019 The Apache Software Foundation
xml-resolver	1.2	Copyright © 1999-2018 The Apache Software Foundation
xmlsec	1.5.6	Copyright © 1999-2018 The Apache Software Foundation
xmltooling	1.4.1	Copyright © 1999-2018 The Apache Software Foundation
javax.inject	1.0	Copyright © 2005-2018 Oracle and/or its affiliates. All rights reserved.
javax.el	2.2.0	Copyright © 2005-2018 Oracle and/or its affiliates. All rights reserved.
Under BSD 3-Clause		
antlr	2.7.7	© Copyright ANTLR / Terence Parr 2014

IP Asset Name	IP Asset Version	Copyright notice
antlr	2.7.5	© Copyright ANTLR / Terence Parr 2014
antlr	3.1.3	© Copyright ANTLR / Terence Parr 2014
antlr4	4-4.3	© Copyright ANTLR / Terence Parr 2014
antlr-runtime	3.3	© Copyright ANTLR / Terence Parr 2014
antlr-runtime	3.5.2	© Copyright ANTLR / Terence Parr 2014
antlr-runtime	3.5	©Copyright ANTLR / Terence Parr 2014
asm	3.3	Copyright © 2000-2011 INRIA, France Telecom
asm	3.1	Copyright © 2000-2011 INRIA, France Telecom
asm	5.0.3	Copyright © 2000-2011 INRIA, France Telecom
bridj	0.7.0	Copyright © 2010-2015, Olivier Chafik (http://ochafik.com/)
com.esotericsoftware.kryo	2.22.0	Copyright © 2008-2018, Nathan Sweet All rights reserved.
jsch	0.1.50	

IP Asset Name	IP Asset Version	Copyright notice
jsonld-java	0.8.0	Copyright © 2012, Deutsche Forschungszentrum für Künstliche Intelligenz GmbH Copyright © 2012-2017, JSONLD-Java contributors All rights reserved.
org.abego.treelayout.core -	1.0.1	Copyright © 2011, abego Software GmbH, Germany
org.scala-lang.modules.scala-java8-compat	0.8.0	Copyright © 2002-2015 EPFLCopyright © 2011-2015 Typesafe, Inc. All rights reserved.
org.scala-lang.scala-library	2.12.4	Copyright © 2002-2015 EPFLCopyright © 2011-2015 Typesafe, Inc. All rights reserved.
prefuse		Copyright © 2004-2011 Regents of the University of California. All rights reserved.
scala.lang.modules.scala.parser.combinators	1.0.4	Copyright © 2002-2015 EPFLCopyright © 2011-2015 Typesafe, Inc. All rights reserved.
sesame	4.0.2	Copyright © 2001-2013 Aduna
D3.js	5	Copyright © 2010-2017 Mike Bostock All rights reserved.

IP Asset Name	IP Asset Version	Copyright notice
xstream	1.4.10	Copyright © 2003-2006, Joe Walnes Copyright © 2006-2017 XStream Committers
xstream	1.3.1	Copyright © 2003-2006, Joe Walnes Copyright © 2006-2008 XStream Committers
Under BSD 2-Clause[1271]		
esapi	2.0.1	The Open Web Application Security Project (OWASP)
hamcrest.core	1.3.0	Copyright © 2012 Hamcrest
jing		Copyright © 2001, 2002, 2003, 2008 Thai Open Source Software Center Ltd
jline	2.12	Copyright © 2002-2016, the original author or authors, {Jason Dillon, Marc Prud'hommeaux, Guillaume Nodet}. All rights reserved.
stax2	3.1.1	Copyright © 2005- 2018, Tatu Saloranta, tatu.saloranta@iki.fi[1272]
urlrewritefilter	4.0.3	© Copyright 1995-2012 Paul Tuckey
Under MIT License[1273]		
angular.1.6.9js	1.6.9	© 2010-2018 Google, Inc

1271 https://opensource.org/licenses/BSD-2-Clause
1272 mailto:tatu.saloranta@iki.fi
1273 https://opensource.org/licenses/MIT

IP Asset Name	IP Asset Version	Copyright notice
angular-1.5.9.js	1.5.9	© 2010-2018 Google, Inc
angular-material-1.1.1.js	1.1.1	© 2010-2018 Google, Inc
base64.js		Copyright © 2014 Jameson Little
docbook.jar		Copyright © 2000 Norman Walsh
fr.opensagres.xdocreport.converter	1.0.4	Copyright © 2011-2015 The XDocReport Team
jcl-over-slf4j	1.7.13	Copyright © 2004-2019 QOS.ch[1274]
jcl-over-slf4j	1.7.21	Copyright © 2004-2019 QOS.ch[1275]
jcl-over-slf4j	1.7.25	Copyright © 2004-2019 QOS.ch[1276]
jcl-over-slf4j	1.7.5	Copyright © 2004-2019 QOS.ch[1277]
jquery	2.1.1	Copyright 2014 jQuery Foundation and other contributors
jquery	3.1.0	Copyright 2014 jQuery Foundation and other contributors

1274 http://QOS.ch
1275 http://QOS.ch
1276 http://QOS.ch
1277 http://QOS.ch

IP Asset Name	IP Asset Version	Copyright notice
jsoup	1.6.1	© 2009 - 2018 Jonathan Hedley
jul.to[1278].slf4j	1.7.7	Copyright © 2004-2019 QOS.ch[1279]
jul-to-slf4j	1.7.25	Copyright © 2004-2019 QOS.ch[1280]
karma.js		Copyright © 2011-2019 Google, Inc.
kxml2	2.3.0	Copyright © 2002,2003, Stefan Haustein, Oberhausen, Rhld., Germany
moment	2.19.4	Copyright © JS Foundation and other contributors
protractor.js		Copyright © 2010-2017 Google, Inc.
slf4j	1.7.7	Copyright © 2004-2013 QOS.ch[1281]
slf4j	1.6.1	Copyright © 2004-2013 QOS.ch[1282]
slf4j	1.7.22	Copyright © 2004-2013 QOS.ch[1283]

1278 http://jul.to
1279 http://QOS.ch
1280 http://QOS.ch
1281 http://QOS.ch
1282 http://QOS.ch
1283 http://QOS.ch

IP Asset Name	IP Asset Version	Copyright notice
slf4j	1.7.25	Copyright © 2004-2013 QOS.ch[1284]
xdocreport	1.0.4	Copyright © 2011-2015 The XDocReport Team
Under Common Public License 1.0[1285]		
wsdl4j	1.5.1	Copyright © 2011 Jonathan Gallimore
Under LBC License[1286]		
bcpkix-jdk15on	1.56	Copyright (c) 2000 - 2019 The Legion of the Bouncy Castle Inc.
bcprov-jdk15	1.46	Copyright (c) 2000 - 2019 The Legion of the Bouncy Castle Inc.

The following components are distributed and licensed under the terms of their original licenses:

IP Asset Name	IP Asset Version	Copyright notice
Under Eclipse Public License 1.0[1287]		
aspectjweaver	1.8.10	Copyright © Eclipse Foundation, Inc.
aspectjweaver	1.9.0	Copyright © Eclipse Foundation, Inc.

1284 http://QOS.ch
1285 https://opensource.org/licenses/cpl1.0.php
1286 https://www.bouncycastle.org/license.html
1287 https://opensource.org/licenses/EPL-1.0

IP Asset Name	IP Asset Version	Copyright notice
ch.qos.logback.classic	1.0.13	Copyright © 2017 QOS.ch[1288]
ch.qos.logback.core	1.0.13	Copyright © 2017 QOS.ch[1289]
com.ibm.icu	4.4.2	Copyright © Eclipse Foundation, Inc
com.ibm.icu	58.2.0	Copyright © Eclipse Foundation, Inc
common	2.1.0	Copyright © Eclipse Foundation, Inc
common	2.11.0	Copyright © Eclipse Foundation, Inc
eclipse.core.contenttype	3.6.0	Copyright © 2014 The Eclipse Foundation
eclipse.core.jobs	3.9.2	Copyright © 2014 The Eclipse Foundation
eclipse.core.runtime	3.13.0	Copyright © 2014 The Eclipse Foundation
eclipse.emf.common	2.10.1	Copyright © 2014 The Eclipse Foundation
eclipse.emf	2.13.0	Copyright © 2014 The Eclipse Foundation
eclipse.emf.common	2.2.1	Copyright © 2014 The Eclipse Foundation
eclipse.emf	2.8.0	Copyright © 2014 The Eclipse Foundation

1288 http://QOS.ch
1289 http://QOS.ch

IP Asset Name	IP Asset Version	Copyright notice
eclipse.emf.ecore	2.10.2	Copyright © 2014 The Eclipse Foundation
eclipse.emf.ecore	2.2.3	Copyright © 2014 The Eclipse Foundation
eclipse.emf.ecore	2.8.1	Copyright © 2014 The Eclipse Foundation
eclipse.equinox	1.3.400	Copyright © 2014 The Eclipse Foundation
eclipse.equinox.common	3.9.0	Copyright © 2014 The Eclipse Foundation
eclipse.equinox.ds	1.5.0	Copyright © 2014 The Eclipse Foundation
eclipse.equinox	3.7.0	Copyright © 2014 The Eclipse Foundation
eclipse.ocl.common	1.4.100	Copyright © 2014 The Eclipse Foundation
eclipse.ocl	3.6.100	Copyright © 2014 The Eclipse Foundation
eclipse.osgi	3.12.50	Copyright © 2014 The Eclipse Foundation
eclipse.osgi.services	3.6.0	Copyright © 2014 The Eclipse Foundation
eclipse.osgi.util	3.4.0	Copyright © 2014 The Eclipse Foundation
eclipse.uml2.common	1.2.4	Copyright © 2014 The Eclipse Foundation
eclipse.uml2.common	1.5.0	Copyright © 2014 The Eclipse Foundation

IP Asset Name	IP Asset Version	Copyright notice
eclipse.uml2.common	1.7.0	Copyright © 2014 The Eclipse Foundation
eclipse.uml2.common	2.0.1	Copyright © 2014 The Eclipse Foundation
eclipse.uml2.types	1.0.0	Copyright © 2014 The Eclipse Foundation
eclipse.uml2.types	2.0.0	Copyright © 2014 The Eclipse Foundation
eclipse.uml2.uml	2.0.4	Copyright © 2014 The Eclipse Foundation
eclipse.uml2.uml	3.0.1	Copyright © 2009The Eclipse Foundation
eclipse.uml2.uml	4.0.1	Copyright © 2014 The Eclipse Foundation
eclipse.uml2.uml	5.0.2	Copyright © 2014 The Eclipse Foundation
eclipse.uml2.uml.profile.l2	1.0.1	Copyright © 2014 The Eclipse Foundation
eclipse.uml2.uml.profile.l3	1.0.1	Copyright © 2014 The Eclipse Foundation
eclipse.uml2.uml.profile.standard	1.0.0	Copyright © 2014 The Eclipse Foundation
eclipse.uml2.uml.resources	2.0.3	Copyright © 2014 The Eclipse Foundation
eclipse.uml2.uml.resources	3.0.0	Copyright © 2014 The Eclipse Foundation
ecore	2.11.0	Copyright © Eclipse Foundation, Inc

IP Asset Name	IP Asset Version	Copyright notice
jruby.complete	1.7.26	Copyright © 2007-2018 The JRuby project
javax.mail	1.6.1	Copyright © 2005-2018 Oracle and/or its affiliates. All rights reserved.
javax.persistence	1.0.0	Copyright © 2005-2018 Oracle and/or its affiliates. All rights reserved.
javax.servlet.jsp	2.2.0	Copyright © 2005-2018 Oracle and/or its affiliates. All rights reserved.
javax.servlet	3.0.0	Copyright © 2005-2018 Oracle and/or its affiliates. All rights reserved.
javax.servlet	3.1.0	Copyright © 2005-2018 Oracle and/or its affiliates. All rights reserved.
javax.xml	1.3.4	Copyright © 2005-2018 Oracle and/or its affiliates. All rights reserved.
sisu-guava-0.9.9.jar	0.9.9	Copyright © Eclipse Foundation, Inc. All Rights Reserved.
sisu-guice-3.1.0-no_aop.jar	3.1.0	Copyright © Eclipse Foundation, Inc. All Rights Reserved.
sisu-inject-plexus-2.3.0.jar	2.3.0	Copyright © Eclipse Foundation, Inc. All Rights Reserved.
Under Eclipse Public License 2.0[1290]		

1290 https://opensource.org/licenses/EPL-2.0

IP Asset Name	IP Asset Version	Copyright notice
jaxrpc		Copyright © 2006, 2018 Oracle and/or its affiliates. All rights reserved.
Under LGPL 2.1[1291]		
jgrapht	0.8.0	© Copyright 2003-2018, by Barak Naveh and Contributors. All rights reserved.
trove	3.0.3	Copyright © 2013, NVIDIA Corporation
trove4j	3.0.3	Copyright © 2013, NVIDIA Corporation
freehep.base		© Maplesoft, a division of Waterloo Maple Inc. 1981 - 2015.
freehep.graphics2d		© Maplesoft, a division of Waterloo Maple Inc. 1981 - 2015.
freehep.graphicsio		© Maplesoft, a division of Waterloo Maple Inc. 1981 - 2015.
freehep.graphicsio.emf		© Maplesoft, a division of Waterloo Maple Inc. 1981 - 2015.
freehep.graphicsio.ps[1292]		© Maplesoft, a division of Waterloo Maple Inc. 1981 - 2015.
c3p0	0.9.1.1	© Steve Waldman

1291 https://opensource.org/licenses/LGPL-2.1
1292 http://freehep.graphicsio.ps

IP Asset Name	IP Asset Version	Copyright notice
Under LGPL 3.0[1293]		
fmu-wrapper	2.20.2	Copyright © 2013-2016 SIANI - ULPGC
fmu-wrapper	2.20.0	Copyright © 2013-2016 SIANI - ULPGC
Alfresco Community Edition	6.1	© 2019 Alfresco Software, Inc. All Rights Reserved.
Under CDDL 1.0[1294]		
activation	1.0	Copyright © 2006, 2018 Oracle and/or its affiliates. All rights reserved.
annotation	1.2	Copyright © 2006, 2018 Oracle and/or its affiliates. All rights reserved.
javahelp	2.0.02	Copyright © 2006, 2018 Oracle and/or its affiliates. All rights reserved.
jhall		Copyright © 2006, 2018 Oracle and/or its affiliates. All rights reserved.
jsp.api	2	Copyright © 2006, 2018 Oracle and/or its affiliates. All rights reserved.
jsr250	1	Copyright © 2006, 2018 Oracle and/or its affiliates. All rights reserved.

1293 https://opensource.org/licenses/LGPL-3.0
1294 https://opensource.org/licenses/CDDL-1.0

IP Asset Name	IP Asset Version	Copyright notice
mail	1.4.1	Copyright © 2006, 2018 Oracle and/or its affiliates. All rights reserved.
servlet.api	2.4	Copyright © 2006, 2018 Oracle and/or its affiliates. All rights reserved.
servlet-api	3	Copyright © 2006, 2018 Oracle and/or its affiliates. All rights reserved.
javax.annotation	1.2.0	Copyright © 2005-2018 Oracle and/or its affiliates. All rights reserved.
Under Mozilla Public License 1.0[1295]		
Saxon9	9	© 2009 Saxonica

Restricted Rights

This clause applies to all acquisitions of Dassault Systèmes Offerings by or for the United States government, or by any prime contractor or subcontractor (at any tier) under any contract, grant, cooperative agreement or other activity with the United States government. The software, documentation and any other technical data provided hereunder is commercial in nature and developed solely at private expense. The Software is delivered as “Commercial Computer Software” as defined in DFARS 252.227-7014 or as a “Commercial Item” as defined in FAR 2.101(a) and as such is provided with only such rights as are provided in Dassault Systèmes standard commercial end user license agreement. Technical data is provided with limited rights only as provided in DFAR 252.227-7015 or FAR 52.227-14, whichever is applicable. The terms and conditions of the Dassault Systèmes standard commercial end user license agreement shall pertain to the United States government's use and disclosure of this software, and shall supersede any conflicting contractual terms and conditions. If the DS standard commercial license fails to meet the United States government's needs or is inconsistent in any respect with United States Federal law, the United States government agrees to return this software, unused, to DS. The following additional statement applies only to acquisitions governed by DFARS Subpart 227.4: "Restricted Rights - use, duplication and disclosure by the Government is subject to restrictions as set forth in subparagraph ©(l)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252-227-7013.

1295 https://opensource.org/licenses/MPL-1.0

Index

2
2-way-merge 1083

3
3-way-merge 1083

A
Abstract-matrix-cell 2330
Accepting-changes 1107
Actions-configurator-manager 2325
Active-validation 1306
Active-validation-suite 2333
Actors 876
Addition-change 1086, 1292
Advanced-numbering 774
Aggregation 654
All-mode 917
All-property-display-mode 917
Alternative-condition 889
Alternative-flow 882
Alternative-flow-step 889
Amconfigurator 2325, 2326
Analyze-menu 25
Analyzing-changes 1108, 1303
Analyzing-comparison-results 1295
Analyzing-merge-results 1095
Annotation 2333
Annotation-manager 2332
Application 2209
Association-class 652
Association-end-type 654
Association-navigability 654
Asynchcall 756
Asynchsignal 756
Attribute 2090
Auto-number 2347
Automatic-synchronization 1457

B
Base 1291
Base-element 2223
Basic-flow 882
Behavior-type 662
Bezier 560
Brand 2160
Break-point 2278
Browser 2308
Browser-tree 2308
Button 2196

C
Can-create 2326
Can-edit 2326
Cell-renderer 2330
Change-activator 777
Change-concept 1086, 1292
Change-details 1103, 1300
Change-states 1086, 1097
Change-type 1086
Change-types 1292
Changing-states 852
Class 297
Classifier-behavior 662
Client 2278
Client-point 2278
Close-session 2224
Code-engineering-manager 2357, 2399
Collaborate-menu 25
Column-header-cell-renderer 2330
Command 2196
Compare-concepts 1291
Compare-specification-panel 1298
Compared 1291
Comparison-result 1295, 1296, 1296, 1303
Comparison-result-navigation 1301
Comparison-summary 1296
Compatibility-range 108
Composite-state 848
Concepts 1083
Configure-dependency-handlers 2326
Conflicting-changes 1086, 1106
Connection-point-reference 853
Container 2226
Count-elements 108
Create-add-actions 2326
Create-button 2197
Create-command 2197
Create-edit-actions 2326
Create-operation 758
Create-path 558
Create-region 851
Create-relation-map 365
Create-relationship 558
Create-session 2224
Createmessage 756
Creating-dependency-matrix-type 2052
Creating-diagram-type 2046
Custom-button 2197
Custom-command 2197
Customize-diagram-wizard 2045
Customizing-dependency-matrix 2052
Customizing-diagram 2045
Customizing-diagrams 2046
Customiztion-attribute 2090

D
Decision 698
Define-rules 2412
Delete-path 560
Delete-relationship 560
Deletemessage 756
Deletion-change 1086, 1292
Dependency-editor 2326
Dependency-entry 2326
Dependency-extractor 2326, 2331
Dependency-matrix-action-registry 2326

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Dependency-matrix-configurator 2325, 2326, 2330
Dependency-matrix-customization 2052
Dependency-matrix-diagram-descriptor 2325
Dependent-changes 1086, 1292
Detailed-statistics 108
Diagram 297
Diagram-comparator 2175
Diagram-customization 2045, 2046
Diagram-listener-adapter 2298
Diagram-stereotype 608
Diagram-surface 2301
Diagram_diff 1431
Diagram_diffrerences 1431
Diagrams-menu 25
Difference-viewer 1108, 1300, 1303
Differences 1300, 1431
Directly-used-project 1443
Disconnected-collaboration 1386
Display-parts 491
Display-ports 491
Displaying-changes 1104, 1301
Distributed-resources 2187
Distributing-diagrams 2187
Distributing-documentation 2187
Distributing-plugins 2187
Distributing-profiles 2187
Distributing-samples 2187
Distributing-templates 2187
Documentation-space-sample 1547
Dot-notation 654
Dragging-in-activity-diagram 487
Dragging-in-diagram 487
Dragging-in-sequence-diagram 487
Dragging-in-state-machine-diagram 487
Dsl 2034

E

Ea-import 117
Eclpise 2178
Edit-menu 25
Element 2223
Element-attribute 2090
Element-decorations 1097, 1296
Element-diff 1431
Element-differences 1431
Element-numbering 1040
Elements-name-change-event 2249
Enterprise-architect-import 117
Equivalent-changes 1086
Exception-flow-step 891
Exception-type 891
Exceptional-flow 882
Expert-mode 917
Expert-property-display-mode 917
Export-branches 1372
Export-projects-to-teamwork-cloud 1372
Export-property-values 201
Export-trunk 1372
Exporting-uml 201
Extending-use-case 887

F

Featured 1083, 1547
File-menu 25
Finish-merging 1112
For-row 2326
Fork-node 847
Forward-compatibility 108
Forward-engineering 2356

G

Generating-report 1111, 1305
Get-dependencies 2326
Get-plugin-name 2168
Get-plugin-version 2168

H

Header-cell-renderer 2330
Help-menu 25
Helper-class 2255
Hidden-packages 1457
Hide-ports 491
Hide-region 851
History 1437
History-dialog 1428
Horizontal-line-jump 563
How-to-update-the-last-id 1042
Hyperlink-model 2305
Hyperlink-model-active 2305
Hyperlink-text 2305
Hyperlink-text-active 2305

I

Implementation 318, 319
Implementation-diagram 318, 319
Importing-packages 2406
Importing-uml 116
Importing-xmi 117
Included-use-case 887
Indirectly-used-project 1443
Init 2157
Insert-shape-on-path 563
Insert-state 851
Intellij 2178
Is-plugin-required 2168
Ispecification-node-configurator 2310

J

Jar 2160
Java-code-engineering-manager 2357
Java-manager 2357
Java-packages 2406
Java-profile 2428
Join-node 847
Jython 2384, 2384

L

Layout-menu 25
Line-jump 563

M

Magicdraw-test-case 2170
Managing-association 2412
Managing-attribute 2412
Managing-merge-results 1095
Managing-regions 851
Manual-synchronization 1457
Matrix-cell-view 2330
Matrix-data-helper 2331
Menu-command 2196
Merge 698
Merge-legend 1096
Merge-result 1103

Merge-specification-panel 1101
Merge-summary 1096
Merge-type 1083
Merged-result 1095, 1096, 1097, 1106, 1107, 1108
Merged-results-navigation 1104
Message-sort 756
Message-syntax 756
Meta-property 2090
Metaproperty 2090
Model 2209
Model-comparator 2175
Model-comparator-filter 2175
Model-merge 1083
Model-validation 1314
Modification-change 1086, 1292
Module 104
Movement-change 1086, 1292

N

Nested-activation 771
New-element-creation-event 2249
Node 2310
Non-hidden-model 1457
Non-hidden-part 1457
Non-hidden-project 1457
None-line-jump 563
Number-part 2347
Numbering 1040
Numbering-customization 1040
Numbering-property 1040
Numbering-scheme 1040, 2347

O

Oblique 560
Offline-project 1395
Offline-projects 1395
Omg-uml-metamodel 2223
Options-menu 25
Order-change 1086, 1292
Orthogonal-state 848
Out-of-sync 1428
Owned-behavior 662

P

Parsing-rules 760
Patch 2160
Path 557
Path-connector 2276
Path-corners-rounded 562
Path-direction 558
Path-style 560
Path-target 558
Plain-text-file-format 2381
Plug-in 2157
Plugin 2157, 2157
Plugin-descriptor 2162, 2384
Presentation-element 2298
Presentation-element-render-provider 2291
Presentation-element-renderer 2291
Presentation-element-renderer-manager 2291
Project 104, 2209
Project-descriptors-factory 2210
Project-statistics 108
Projectdescriptorsfactory 2210
Projects-comparator 2175
Property-changes 1101, 1298
Property-editor 2306
Property-manager 2306
Python 2384, 2384

Q

Qualifier 654

R

Rake-icon 876
Recognizing-changes 1097, 1296
Rectilinear 560
Reflective-association 652
Rejecting-changes 1107
Relate-source-and-target 558
Relation-map 364
Relation-map-change-context 370
Relation-map-colors 371
Relation-map-colors-legend 371
Relation-map-context 370
Relation-map-create-element 373
Relation-map-criteria 366
Relation-map-criteria-layout-depth 366
Relation-map-cut-element-name 370
Relation-map-decompose 373
Relation-map-depth 366
Relation-map-element-numbering 371
Relation-map-hide 373
Relation-map-layout 366
Relation-map-legend 371
Relation-map-remove 373
Relation-map-restore-refresh 373
Relation-map-single-node 370
Relation-map-suppress-expand 373
Relation-map-validate 373
Relation-map-zoom 373
Relationship 557
Relationship-direction 558
Remove-region 851
Reply 756
Report 1111, 1305
Report-templates 1111, 1305
Reporting 1111, 1305
Resize 555
Resize-shape 555
Resource-builder 2185
Resource-manager 2184, 2185
Reusing-model 1457
Reusing-project 1457
Reverse-engineering 2356
Root-model 2209
Round-trip-set 2399, 2406
Route-paths 561
Route-paths-rectilinearly 561
Row 2330

S

Scope-history 1437
Selection-events 2298
Set 2276
Set-as-latest 1428
Setter 2276
Setup-eclipse-environment 2178
Setup-intellij-idea-environment 2182
Shortcut-menu-command 2196
Signature 758, 768
Simple-state 848
Slot 735
Slot-specification 735

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.

Specification-window 913
Specifying-use-case-subject 878
Split-path 563
Split-transition 563
Sset 2276
Standard-mode 917
Standard-property-display-mode 917
Starting-merge 1090
Starting-project-compare 1293
State 848
State-behavior 849
State-condition 855
State-invariant 855
Stereotypes-helper 2255
Stripping-project 1457
Style-engine 988
Submachine-state 848
Summary-and-legend 1096, 1296
Suplier-piont 2278
Supplier 2278
Supported-languages 2426
Synchcall 756
Synchronization 1457

T

Table-cell-renderer 2330
Teamwork-utils 2210
Teamworkutils 2210
Terms 452
Tools-menu 25
Transaction-commit-listener 2253
Tree 2310

U

Uml 297
Uml-export 201
Uml-model 2267
Uml-profiling 2034
Update-offline-project 1395
Use-case 876
Use-case-extension-points 877
Use-case-subject 878
Use-cases 876
Used-project 104
Used-projects 245

V

Validate-diagram 1314
Validate-entire-model 1314
Validation 1306, 1315
Validation-results-marker-bar 1315
Validation-rule 2333
Validation-rules 1306
Validation-suite 2333
Validation-suites 1306
Vertical-line-jump 563
View-menu 25

W

Window-menu 25

X

Xmi 2381

Z

Zip-based-file-format 2381

Copyright © 1998 – 2020 No Magic, Incorporated, a Dassault Systèmes company.