



Developer Guides 2024x Refresh3

User Guide

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 1998-2025 by No Magic, Incorporated, a Dassault Systèmes company. All Rights Reserved.

Open API Deprecation Policy 9

Modeling Tools Developer Guide 9

Plugins 11

- How plugins work 11
- Writing plugins 12
- Compilation classpath 14
- Testing plugin 21
- Detail information 22
- Resource dependent plugin 28
- Identifying plugin dependencies 29
- Creating test cases 29
- Developing plugins using IDE 36

Distributing Resources 41

- How to distribute resources 41
- Creating required files and folders structure 43
- Resource manager descriptor file 48
- Building a resource distribution file 51

Adding new functionality 52

- Invoking actions 53
- Creating new actions 54
- Actions' hierarchy 61
- Working with icons 62
- Running action with progress 65

Working with project 65

- Project concept 66
- Project descriptor 67
- Projects management 67
- Working with projects from Teamwork Cloud server 73
- Project structure decomposition 73
- Project options 74

Contents

Project life-cycle events 76

UML model 76

Working with UML model 79

Event support 99

Working with stereotypes and tagged values 106

UML Model Implementation Using EMF 115

Working with diagrams 117

Diagram types 118

Creating a diagram 123

Presentation elements (symbols) 123

Selection in diagrams 141

Diagram events 142

Custom diagram painters 145

Creating diagram images 146

Working with Generic tables 146

Working with Dependency Matrix 150

Working with Relation Map 158

Working with Gantt Chart 159

Managing navigation in model 159

Customizing double-click behavior 159

Managing hyperlinks 159

Properties 161

Information logging 161

Environment Options 162

Example of adding custom environment options 163

Example of accessing environment options 163

Browser 163

Adding custom panel or tree into browser 164

Configuring node text and icon 165

Using and extending other UI components 165

Configuring element Specification window 165

Contents

- Showing Element selection dialog 166
- Showing question, error, warning dialogs 166
- Showing notifications, adding text into Message Window 167
- Creating PopupMenu or other menu 168
- Adding custom project window 169

Annotating the elements 169

Creating validation rules 169

- Validation rule developer's roadmap 170
- Create OCL2.0 validation rule 171
- Binary validation rule 173
- Implementing a binary validation rule 174
- Binary validation rule in the plugin 178
- Script validation rule 179
- How to provide a solution for a problem found during validation? 179

Custom Legend Item Implementation 180

Custom elements numbering 180

- IJavaNumberPart interface 181
- INumberingAction interface 182

Creating Use Case Scenario 187

Merging and differencing 188

Code engineering using API 189

- Code engineering set 190
- Managing code engineering sets 191
- Samples of the forward and reverse engineering 192

Oracle DDL generation and customization 194

- Understanding the Oracle DDL template structure 194
- Customizing templates 195
- Utility class 196
- Example 206

Running programs in batch mode 207

Contents

- Specifying batch mode program classpath and required system properties 208

- Implementing command line launchers 224

- Starting MagicDraw or other modeling tool as part of another application 287

- MagicDraw file format 288

 - File format in MagicDraw 3.6-17.0 288

 - File format in MagicDraw 17.0.1 and up 288

- Jython Scripting 290

 - Creating script 291

- JavaScript migration from Nashorn to Rhino engine 293

 - Class object and .class property 294

 - Accessing Java packages and classes from script 295

 - Creating Java arrays from script 296

 - Java exceptions 297

- Plugins migration 298

 - Plugins migration to MagicDraw 18.2 and later Open API 298

 - Plugins migration to MagicDraw 18.1 and later Open API 298

 - Plugins migration to MagicDraw 18.0 and later Open API 298

 - Plugins migration to MagicDraw 17.0.1 and later Open API 304

 - Plugins migration to MagicDraw 15.0 and later Open API 314

- Licensing information 316

 - Providing licensing information 316

 - Getting licensing information 361

- Multi-threading 361

 - Idle job service 362

- Application environment 363

- Attached Files 363

- Element Referencing in Texts 363

 - Creating text that refers an element 363

 - Setting text with element references 364

Contents

[Creating custom drag and drop handlers 364](#)

SysML Developer Guide 365

[SysML Profile 366](#)

[MD Customization for SysML Profile 367](#)

[SysML Profile API Changes 367](#)

[SysML classes for open API 368](#)

Cameo Simulation Toolkit Developer Guide 368

[Introduction 368](#)

[Java APIs 368](#)

[Introduction 368](#)

[SimulationOptionProvider interface 369](#)

[Java APIs 370](#)

[Execution 371](#)

[Engine 374](#)

[fUML Helper 379](#)

[Parametric Helper as executing parametric simulation from an Activity 379](#)

[Display Values in Diagrams 380](#)

[API Changes 383](#)

Teamwork Cloud Developer Guide 390

[REST APIs 390](#)

[General convention 391](#)

[Authentication 392](#)

[Reusing Teamwork Cloud session 402](#)

[Model manipulation 404](#)

[MagicDraw-specific extensions 411](#)

[REST API Change Log 415](#)

[LDAP configuration description 418](#)

[Command Line Interface \(CLI\) for user administration 419](#)

[Assign a project role to a particular user 420](#)

[List all permissions 420](#)

Contents

- Create a role 421
- Edit a role 422
- Delete a role 422
- Create a user 422
- Edit a user 423
- List all LDAP configurations 424
- Import a user from LDAP 425
- Search a user in an LDAP server 426
- List all users 427
- Get user information 427
- List all roles 428
- List users assigned to a particular role 430
- Unassign a role from a particular user 431
- Unassign a project role from a particular user 431
- Assign a role to a particular user 431

OSLC API 432

- OSLC Configuration Management Support 433
- OAuth 1.0a authentication 433

Alf Plugin Developer Guide 434

Introduction to the Alf API 434

Compiler API 435

- Compiler API: Parsing 435
- Compiler API: Mapping 437
- Compiler API: Utilities 438

Importer API 439

- Importer API: Parsing 440
- Importer API: Mapping 440
- Importer API: Utilities 442

Action API 442

- [Open API Deprecation Policy](#) (see page 9)
- [Modeling Tools Developer Guide](#) (see page 9)
- [SysML Developer Guide](#) (see page 365)
- [Cameo Simulation Toolkit Developer Guide](#) (see page 368)
- [Teamwork Cloud Developer Guide](#) (see page 390)
- [Alf Plugin Developer Guide](#) (see page 434)

Open API Deprecation Policy

Overview: This policy outlines the procedures and guidelines for the deprecation of APIs within our system.

Deprecation Process:


1. **Notification:** APIs that are deprecated will be clearly marked as such in the API javadoc.
2. **Timeline:** Deprecated APIs can be removed in the next major version following their deprecation. For instance, if an API is marked as deprecated in version 2024x (or any 2024x RefreshX), it can be removed in version 2026x.
3. **Replacement APIs:** Deprecated APIs will have replacement APIs provided by the development team. Users are encouraged to migrate to the replacement APIs to ensure continued compatibility and support.

Exceptional Cases: In exceptional cases during major releases, Open APIs may be removed without prior deprecation. This may occur due to significant changes in data structures or profile modifications. While such instances are rare, users should be aware that, in exceptional cases, Open APIs may be subject to removal without prior notice.

Review: This policy will be periodically reviewed and updated to align with evolving technological and business requirements. Any changes to the policy will be communicated to relevant stakeholders in a timely manner.

Effective Version: This API deprecation policy is effective as of version 2024x Refresh1.

Modeling Tools Developer Guide

 In the generated javadoc (<https://jdocs.nomagic.com/2024xRefresh3/>), you will find a detailed description of classes, their attributes, and operations. JavaDocs are located in `<modeling tool installation directory>\openapi\docs`.

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, perform 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).

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.

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, you will find the detailed description of classes, their attributes, and operations. JavaDocs are 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

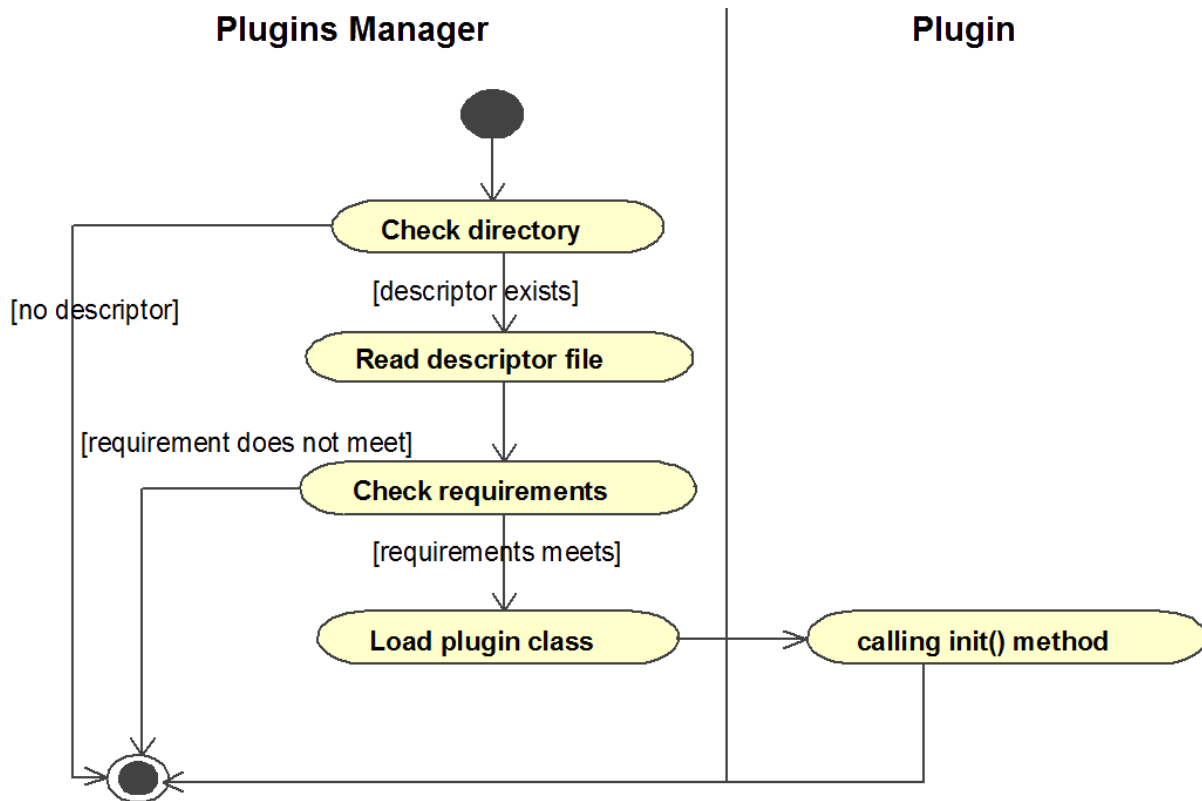
- [How plugins work](#) (see page 11)
- [Writing plugins](#) (see page 12)
- [Compilation classpath](#) (see page 14)
- [Testing plugin](#) (see page 21)
- [Detail information](#) (see page 22)
 - [Plugin descriptor](#) (see page 22)
 - [Plugin classes](#) (see page 25)
 - [Plugin class loading](#) (see page 27)
 - [Plugins directories](#) (see page 27)
- [Resource dependent plugin](#) (see page 28)
- [Identifying plugin dependencies](#) (see page 29)
- [Creating test cases](#) (see page 29)
 - [Creating JUnit test case](#) (see page 30)
 - [Comparing projects for testing purposes](#) (see page 34)
 - [Working with test resources](#) (see page 35)
 - [Configure test environment](#) (see page 35)
- [Developing plugins using IDE](#) (see page 36)
 - [Development in Eclipse](#) (see page 36)
 - [Development in IntelliJ IDEA](#) (see page 40)

How plugins work

On every startup, a modeling tool scans the [plugins directory](#) (see page 27), 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* class). Then a method *init()* of the loaded class is called. The *init()* method can add GUI components using the actions architecture or do other activities and return from the method. The *init()* method is called only if *isSupported()* returns *true*.

The following figure illustrates how do plugins work.



1 How plugins work

Related pages

- [Plugins directories \(see page 27\)](#)

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.


Step #2: Write a plugin code

The plugin must contain at least one class derived from the *com.nomagic.magicdraw.plugins.Plugin* 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 14).

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
```

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 \(see page 22\)](#).

Related pages

- [Compilation classpath \(see page 14\)](#)
- [Plugin descriptor \(see page 22\)](#)

Compilation classpath

Core libraries

The correct classpath must be specified when compiling your (plugin) code or running

***.jar**

files from

<modeli

ng tool

installa

tion

director

y>\lib

, as well as all its

subdirectories must be included into the classpath.

You can open the properties file in

<modelin

g tool

installa

tion

director

y>\bin

and check the

CLASSPA

TH

property to be sure what files must be included.

Make sure that following *.jar files are added in such order:

1.

brand.j

ar

(if the file exists). This file is needed if the Cameo Enterprise

- Architecture program is used. MagicDraw does not requires this file.
2. brand_api.jar (if the file exists)*

3. 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

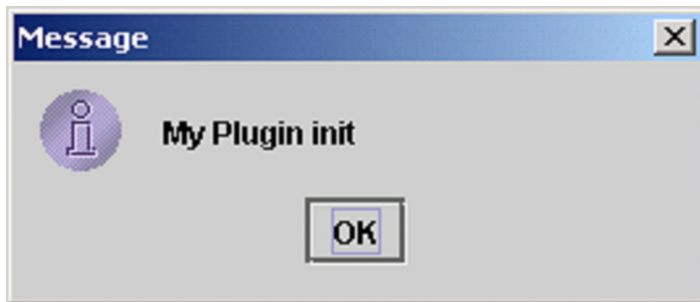
Keep in mind that some Open API classes may come from plugins (like the SysML plugin, for example). In this case, make sure that *jar* files from a plugin directory are added to the compiled classpath. For more information, see the [Identifying plugin dependencies](#) (see page 29) page.

Testing plugin

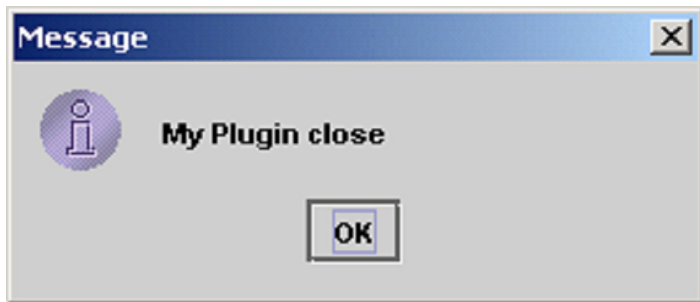
You can test if your plugin sample works by checking described messages or by checking the *magicdraw.log* file.

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 \(see page 22\)](#)
- [Plugin classes \(see page 25\)](#)
- [Plugin class loading \(see page 27\)](#)
- [Plugins directories \(see page 27\)](#)

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 descriptor \(see page 22\)](#)
- [Plugin descriptor \(see page 23\)](#)
- [Plugin descriptor \(see page 24\)](#)
- [required-plugin \(see page 24\)](#)
- [Plugin descriptor \(see page 24\)](#)
- [Plugin descriptor \(see page 25\)](#)

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, <i>my.first.plugin.0</i> .
name	A plugin name. There is no strict rules applied to this attribute. For example, <i>Example plugin</i> .

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: <i>No Magic</i> .
class	A full qualified class name. The class must be derived from <i>com.nomagic.magicdraw.plugins</i> . A plugin and stored in plugin runtime library. This class will be loaded and initialized by the plugin's manager. For example, <i>myplugin.MyPlugin</i> .
ownClassLoader	Optional; the default value is <i>false</i> . 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 <i>LocalFirst</i> , <i>GlobalFirst</i> , the default value is <i>GlobalFirst</i> . 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.

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, <i>1.0</i> .

required-plugin

Attribute	Description
id	The ID of a required plugin. For example, <i>my.first.plugin.0</i> .
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: <i>1.1</i> .
optional	<p>The default value of this attribute is <i>false</i>.</p> <p>If value is <i>false</i> (or attribute is skipped), the plugin will be loaded only if the required plugin is loaded.</p> <p>If value is <i>true</i>, the plugin will be loaded if the required plugin is loaded or required plugin is not installed.</p>

runtime

Nested elements

Element name	Description
library	A runtime library for a running plugin.

library

Attribute	Description
name	<p>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).</p> <p>For example, "pluginlib.jar", "lib/otherlib.jar".</p>

Plugin classes

com.nomagic.magicdraw.plugins.Plugin 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:

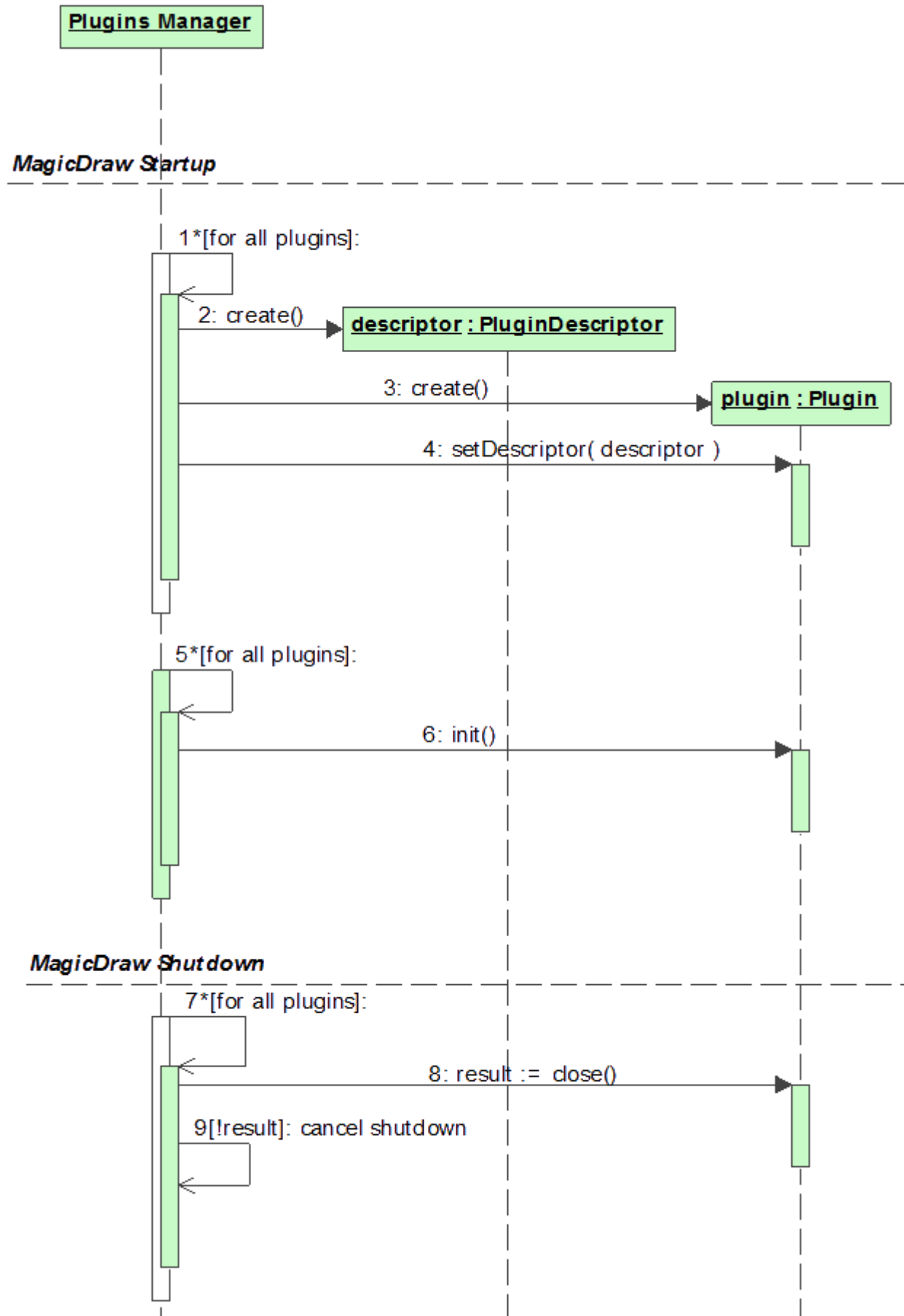
- a *init()* method is called on a program startup. The plugin must override this method and implement its own functionality there.
- a *close()* method is called on a program exit. The plugin must override this method and return the value *true* if the plugin is ready to exit. In other case, it should return the value *false*. If the plugin returns *false*, the program exit is canceled.
- a *isSupported()* method is called before the plugin initialization. If this method returns *false*, the plugin is not initialized. *isSupported()* may be used to check if the plugin can be started, for example, on this OS.

Use *com.nomagic.magicdraw.plugins.PluginUtils* 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 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()* to get the descriptor.

More information is available in javadoc.



2 Plugins manager workflow

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 change the location of the plugins directory

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"`




- If you are loading plugins from a custom location, make sure the location contains extracted plugin files. Compressed plugin files (.zip) cannot be loaded.
- Besides the plugin itself, any other files (profiles, modelLibraries, samples, etc.) contained in the plugin .zip file are not added to the installation directory like they are when [installing a resource](#)¹.

¹ <https://docs.nomagic.com/display/IL2024xR3/Installing+modeling+tools+and+plugins>

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.

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* interface.

ResourceDependentPlugin has three special methods:

- a *isPluginRequired*(²*Project*) method is called on saving a project. The plugin must return *true*, if a given project uses resources from the plugin.
- a *getPluginName*() method should return a plugin name.
- a *getPluginVersion*() method should return a version of the plugin.

The following example illustrates an implementation of *ResourceDependentPlugin*:

```
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;
    }
}
```

² <http://jdocs.nomagic.com/2024x/com/nomagic/magicdraw/plugins/ResourceDependentPlugin.html#isPluginRequired-com.nomagic.magicdraw.core.Project->

```

    }
    @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.

Identifying plugin dependencies

Some Open API classes may come from plugins (like the SysML plugin, for example). In this case, make sure that *jar* files from a plugin directory are added to the compiled classpath. Also, your *plugin.xml* must define the required plugin too.

To find dependencies

- In the Main toolbar, click Tools > Development Tools > Find Plugin Dependency by Class Name...

 The Development Tools menu item is available once the Development Tools Plugin is installed. See the [Installing plugins](#)³ page.

Creating test cases

Our modeling tools, including MagicDraw, provide a JUnit (www.junit.org)⁴-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 30)
- [JUnit 4 test cases](#) (see page 30)
- [JUnit 5 test cases](#) (see page 30)

³ <https://docs.nomagic.com/display/IL2024xR3/Installing+modeling+tools+and+plugins>

⁴ <http://www.junit.org/>

JUnit 3 test cases

The test framework consists of an abstract JUnit test case implementation (*com.nomagic.magicdraw.tests.MagicDrawTestCase*) and a number of tools (*com.nomagic.magicdraw.tests.common.TestEnvironment*, *com.nomagic.magicdraw.tests.common.comparators.ProjectsComparator*) 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*, *com.nomagic.magicdraw.tests.ParametrizedMagicDrawTestRunner*) that:

- Start the application
- Check program for memory leaks.

JUnit 5 test cases

Extend your test class with *com.nomagic.magicdraw.tests.MagicDrawApplication* 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 \(see page 30\)](#)
- [Comparing projects for testing purposes \(see page 34\)](#)
- [Working with test resources \(see page 35\)](#)
- [Configure test environment \(see page 35\)](#)

Creating JUnit test case

Let's use MagicDraw as an example to describe the creation of JUnit test cases.

JUnit 3 test case

The JUnit test case should be created by extending the abstract `com.nomagic.magicdraw.tests.MagicDrawTestCase` class from Open API (see the following *MyTest* sample). The *MagicDrawTestCase* class starts MagicDraw automatically for each test case method defined in *MagicDrawTestCase* and performs the memory leak test after each completed test case.

MagicDrawTestCase 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* initialization and tear down should be implemented in overridden *setUpTest()* and *tearDownTest()* 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.

```
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 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)* method, while the *setSkipMemoryTest(boolean)* 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()* method of *MagicDrawTestCase*. The overridden method implementation should return the list of required plugins IDs. Textual information about required plugins and their loading status can be obtained using *getRequiredPluginsDescription()* and *isRequiredPluginsLoaded()* 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()* method of *MagicDrawTestCase*. 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* from Open API (see a code sample bellow). The *MagicDrawTestRunner* 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
    }
}
```

JUnit 5 test case

The JUnit5 test class should be annotated to extend *com.nomagic.magicdraw.tests.MagicDrawApplication* from Open API (see a code sample bellow). The *MagicDrawApplication* 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
    }

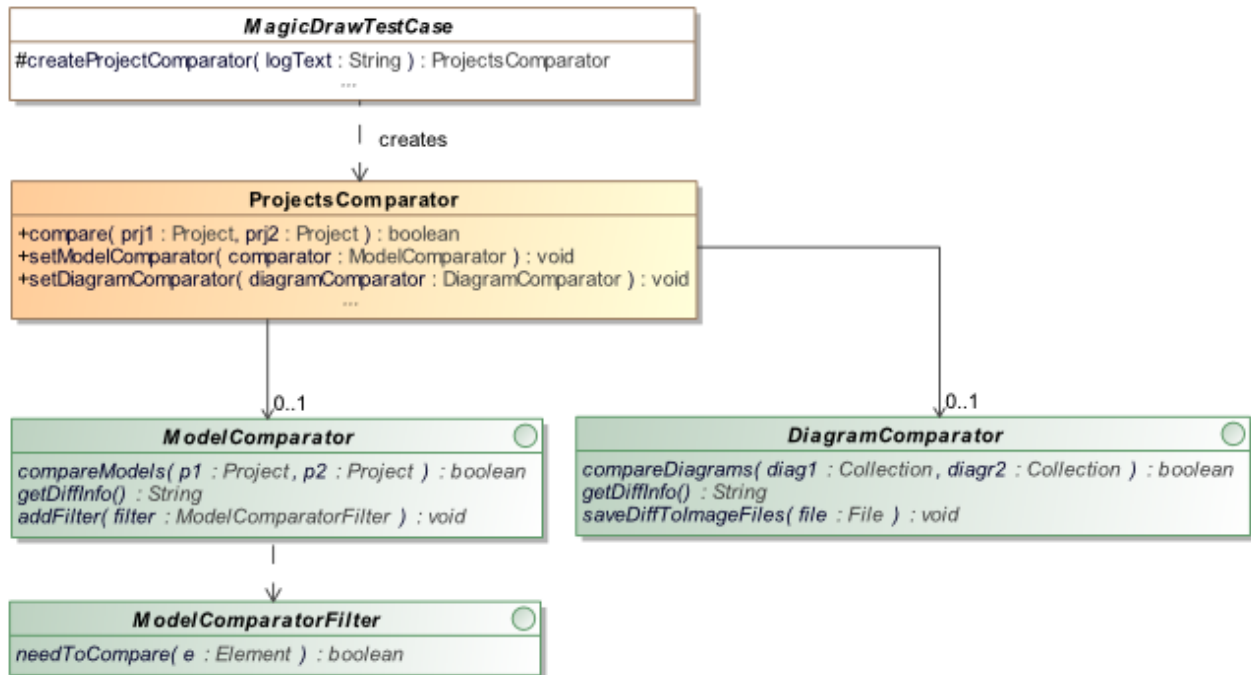
    @AfterEach
    void afterTestDone()
    {
        //do tear down here, e.g. close project
    }
}
```

On this page

- [JUnit 3 test case \(see page 31\)](#)
- [JUnit 4 test case \(see page 32\)](#)
- [JUnit 5 test case \(see page 33\)](#)

Comparing projects for testing purposes

To compare two projects for testing purpose, the test framework provides the *com.nomagic.magicdraw.tests.common.comparators.ProjectsComparator* class with the *compare(Project,⁵Project)⁶* method for comparing two projects (see the following figure). The default implementation of *ProjectComparator* can be created using the *createProjectComparator(java.lang.String)* method of the *MagicDrawTestCase* class. Projects can be compared by checking their model elements and element symbols in diagrams for equality.



3 Project comparator provided by test framework

The model elements comparison can be configured using the *com.nomagic.magicdraw.tests.common.comparators.ModelComparator* interface. Developers may exclude some model elements from the comparison by providing the custom *com.nomagic.magicdraw.tests.common.comparators.ModelComparatorFilter* interface implementation. Please note that the default implementation of *ProjectComparator* 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* 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)* method of the *DiagramComparator* interface.

Developers may create their own comparators for the custom diagrams and model elements comparison by implementing *ModelComparator* and *DiagramComparator* interfaces. The custom *Model*

⁵ <http://jdocs.nomagic.com/2024x/com/nomagic/magicdraw/tests/common/comparators/ProjectsComparator.html#compare-com.nomagic.magicdraw.core.Project-com.nomagic.magicdraw.core.Project>

⁶ <http://jdocs.nomagic.com/2024x/com/nomagic/magicdraw/tests/common/comparators/ProjectsComparator.html#compare-com.nomagic.magicdraw.core.Project-com.nomagic.magicdraw.core.Project>

and *Diagram* comparators implementation can be set to *ProjectComparator* 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()` 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 \(see page 35\)](#). The test framework also provides methods for collecting projects from the resource directory. Developers may use `TestEnvironment.getProjects(java.io.File)` 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)`. The class here specifies a class of the test case which output directory should be accessed.

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)`) or adding a special prefix to its name (`TestEnvironment.createFileNameWithPrefix(java.io.File, java.lang.String)`).

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 set up the program installation to use the floating license server or license file. For more information, see the "Providing licensing information" chapter at [Licensing information \(see page 316\)](#).

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 is not shown, instead of 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 \(see page 36\)](#)
- [Licensing information \(see page 316\)](#)

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 their favorite IDE for an agile modeling tool plugin coding, building, testing, and debugging. This chapter describes the plugin development in Eclipse (www.eclipse.org⁷) and IntelliJ IDEA (www.jetbrains.com⁸) IDEs.

Related pages

- [Development in Eclipse \(see page 36\)](#)
- [Development in IntelliJ IDEA \(see page 40\)](#)

Development in Eclipse

The pre-configured Eclipse projects for two sample plugins and a [batch mode \(see page 207\)](#) (command-line) tool are provided with a program installation. These projects can be found in *<modeling tool installation directory>/openapi/ide/eclipse.zip*.

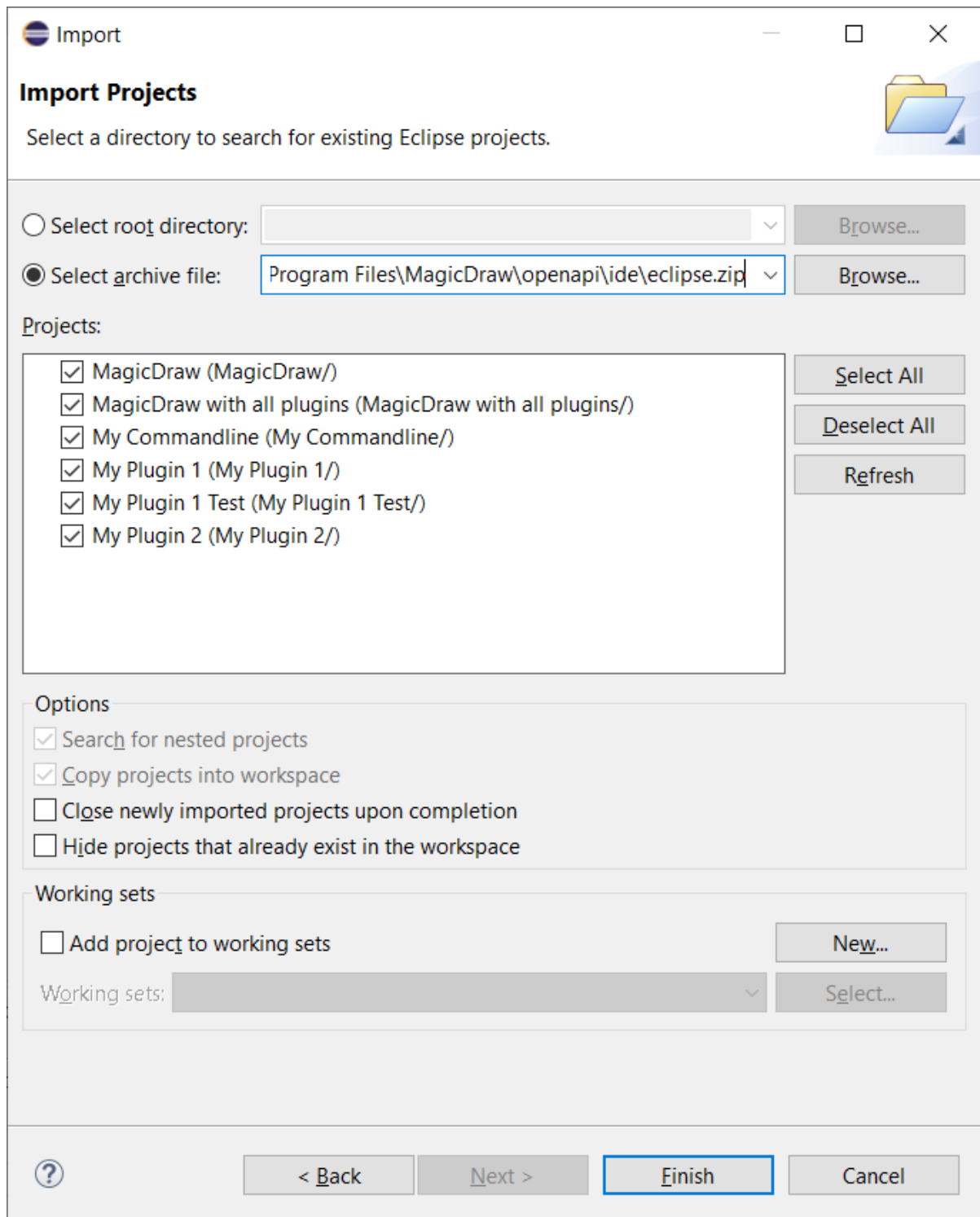
Let's use MagicDraw as an example to describe the following procedures.

To setup the Eclipse environment for the modeling tool development

1. Import java projects from *<modeling tool installation directory>/openapi/ide/eclipse.zip* into your Eclipse Workspace:
 - 1.1. In the main menu, click **File > Import**. The **Import** dialog opens.
 - 1.2. In the **Select an import wizard** list, select **General > Existing Projects into Workspace** and click **Next**.
 - 1.3. Copy the java projects zip location to the **Select archive file** box or click **Browse** and browse to the zip file. The project list appears in the **Projects** box.
 - 1.4. Select projects you need and click **Finish**.

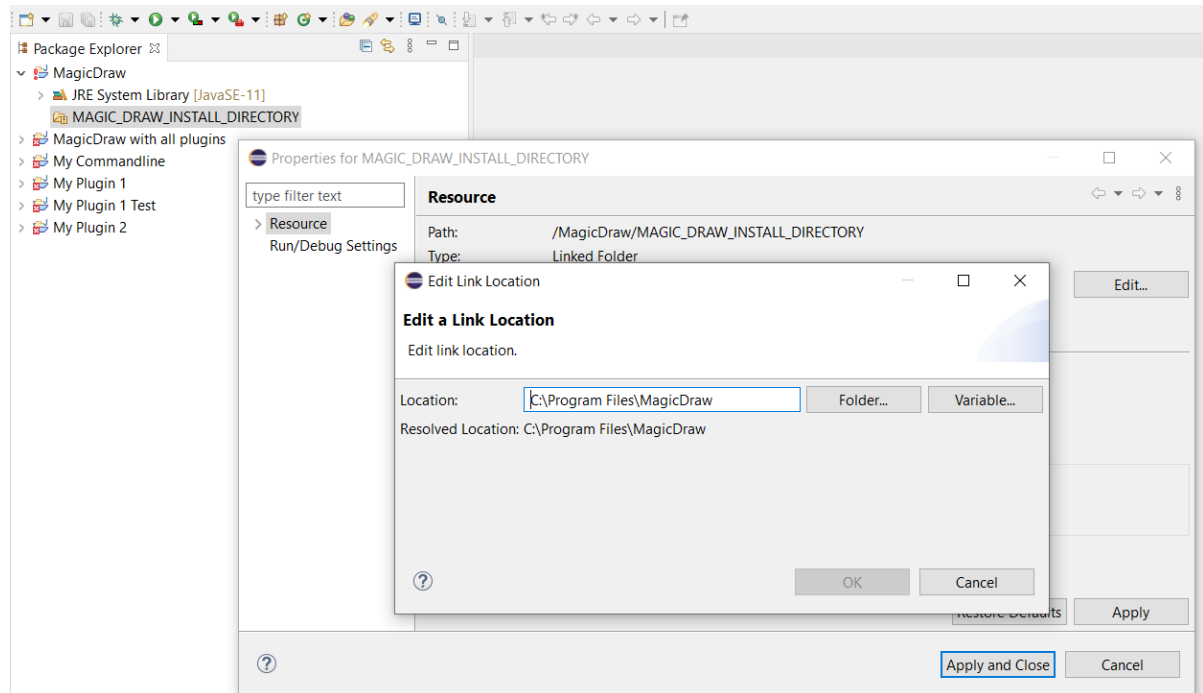
⁷ <http://www.eclipse.org/>

⁸ <http://www.jetbrains.com/>



2. Open the **Package Explorer** view, expand the MagicDraw project and edit the MAGIC_DRAW_INSTALL_DIRECTORY link:
 - 2.1. Right-click the link and select **Properties**.
 - 2.2. 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.
 - 2.3. 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).



3. Make sure that precisely Java 17 version is used by the **MagicDraw** project in your Eclipse IDE:
 - 3.1. Right-click the **MagicDraw** project and select **Properties**.
 - 3.2. On the left side of the open dialog, select **Java Build Path** and then select **Libraries** tab on the right.
 - 3.3. Double-click **JRE System Library [JavaSE-17]**.
 - 3.4. Click **Installed JREs** and check if Java 17 is available here.

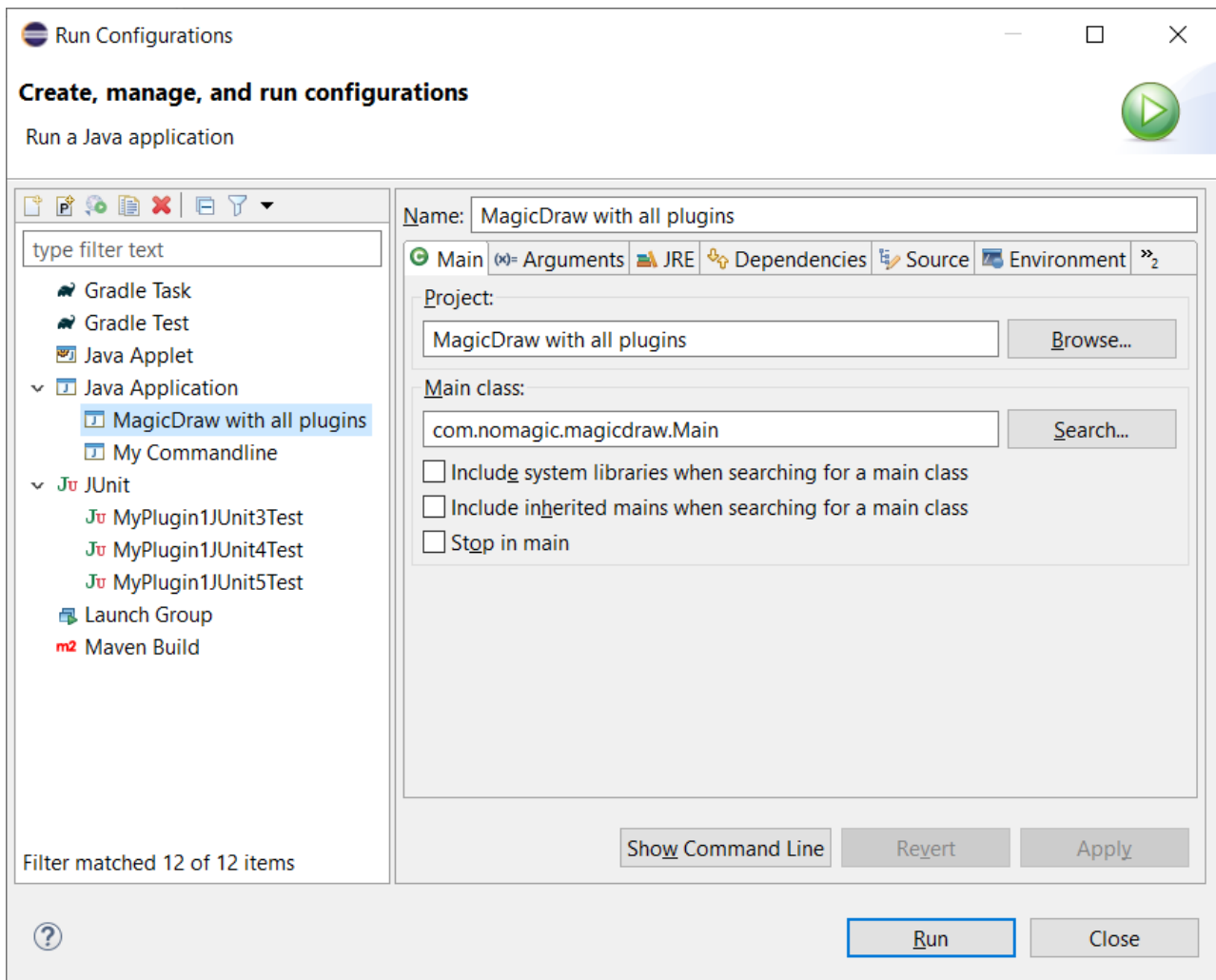
⚠️ If there is no Java 17, add it. *Java 17 version must be added, because later versions may not be supported by our tool.*

- 3.5. Click **Apply and Close**.
- 3.6. Select **Execution environment** check box and in the combo box, select **JavaSE-17**.
- 3.7. Click **Environments** button.
- 3.8. In the **Execution Environments** list, select **JavaSE-17** and in the **Compatible JREs** list, select Java 17.
- 3.9. Click **Apply and Close** and then **Finish** buttons.
- 3.10. Select **Order and Export** tab, click **Select All** button to select all check boxes.
- 3.11. Click **Apply and Close**.

Eclipse Workspace is ready for the source code development and running/debugging.

To use one of the prepared launch configurations


- On the main menu, click **Run > Run Configurations...** (or **Debug Configurations...**). The **Run Configurations** dialog opens.




⚠ 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 #2](#)) and two plugins from the Eclipse Workspace. Thus, if the `md.plugins.dir` java system property is not defined (see [Plugins directories](#) (see [page 27](#))), developing plugins are not loaded.

- *MagicDraw with all plugins* launches MagicDraw (or other modeling tool) with plugins available in the IDE workspace namely *My Plugin 1* and *My Plugin 2*.
- *My Commandline* launches batch mode tool *My Commandline*.
- *MyPlugin1JUnit<n>Test* launches sample *JUnit<n>* test.

⚠ 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. For more information, see the [Identifying plugin dependencies](#) (see [page 29](#)) page.

 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 \(see page 207\)](#) (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 activated licence) installation directory.
2. Extract `<MagicDraw installation directory>/openapi/ide/intellij.zip` to a chosen directory .
3. Open the extracted IntelliJ IDEA project `MagicDraw development.ipr` from that directory .

 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 \(see page 27\)](#)), 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. For more information, see the [Identifying plugin dependencies \(see page 29\)](#) page.

 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.

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](#)⁹.

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](#) (see page 41)
- [Creating required files and folders structure](#) (see page 43)
- [Resource manager descriptor file](#) (see page 48)
- [Building a resource distribution file](#) (see page 51)

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 43).

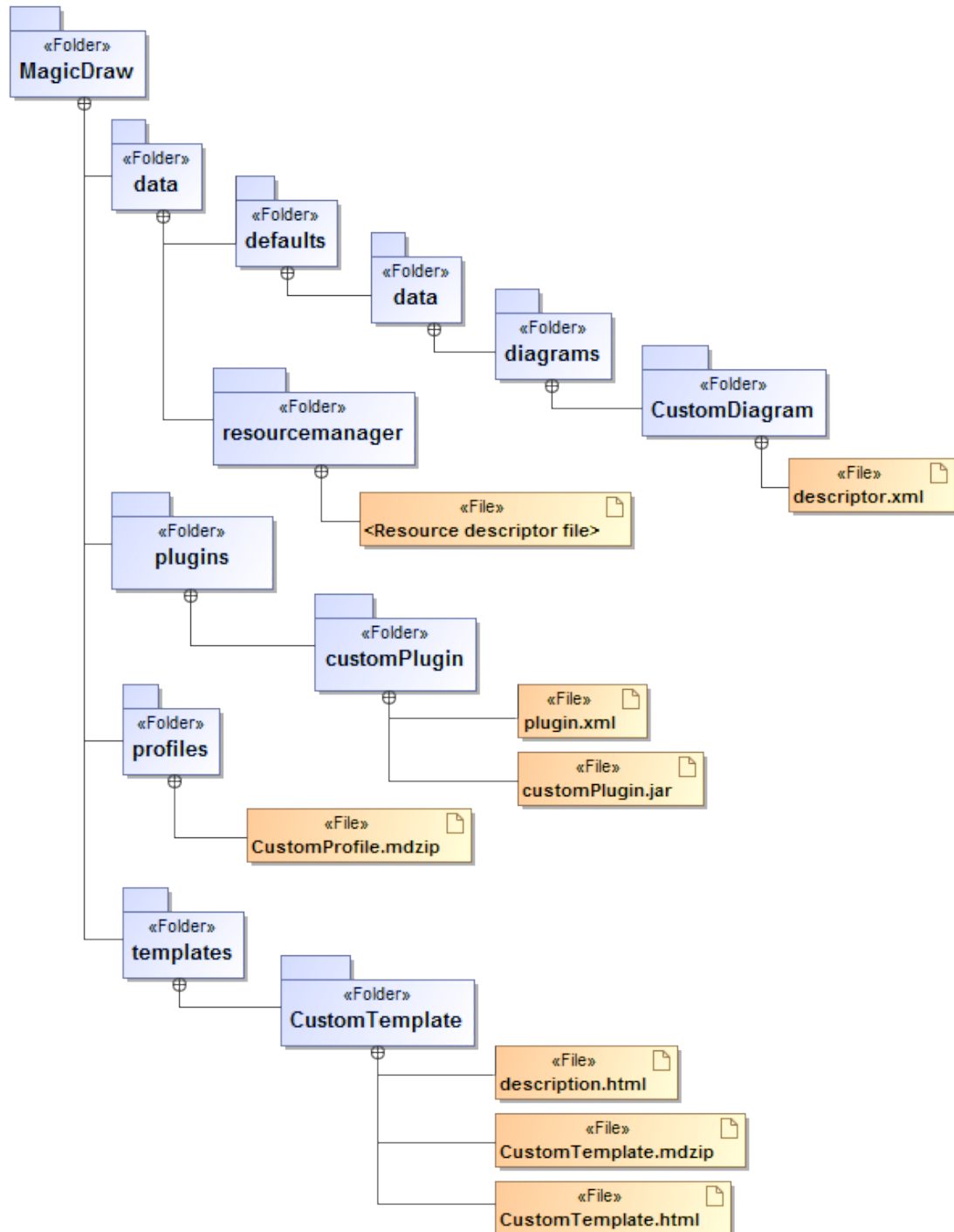
 The structure must not contain the "data/resourcemanager" folders yet.

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:


⁹ <https://docs.nomagic.com/display/MD2024xR3/Resource+Manager>

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 43).
2. Create a Resource descriptor file. For more information about the Resource descriptor file, see [Resource manager descriptor file](#) (see page 48).
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 \(see page 43\)](#)
- [Resource manager descriptor file \(see page 48\)](#)

Creating required files and folders structure

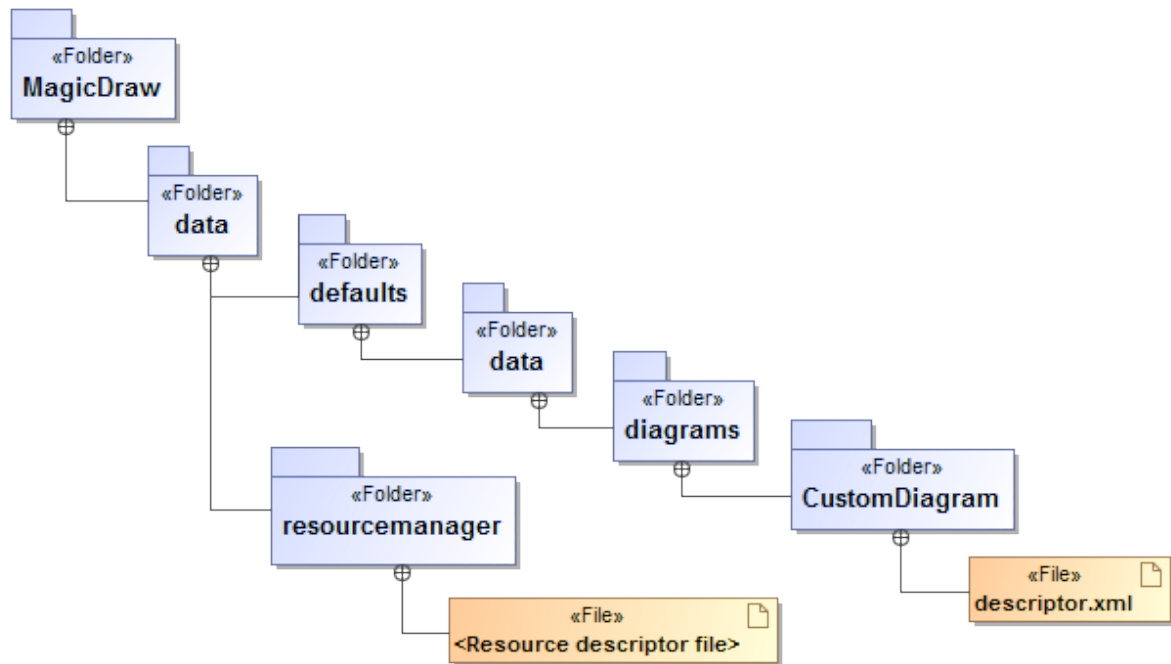
To distribute resources, you must create the required files and folders for a particular resource type. Some of the resource file names should match the standard names.

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 \(see page 48\)](#).

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:

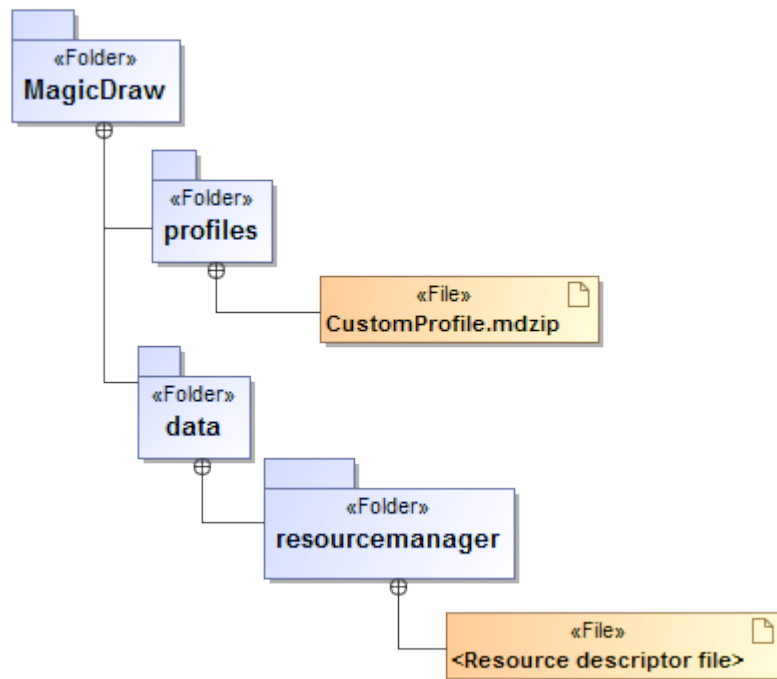


! The custom diagram type (in diagram descriptor.xml file) should match the folder name it is placed in.
For example, the custom diagram with type *My Custom Structure Diagram* should be placed in a folder named **My Custom Structure Diagram**.

i 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:



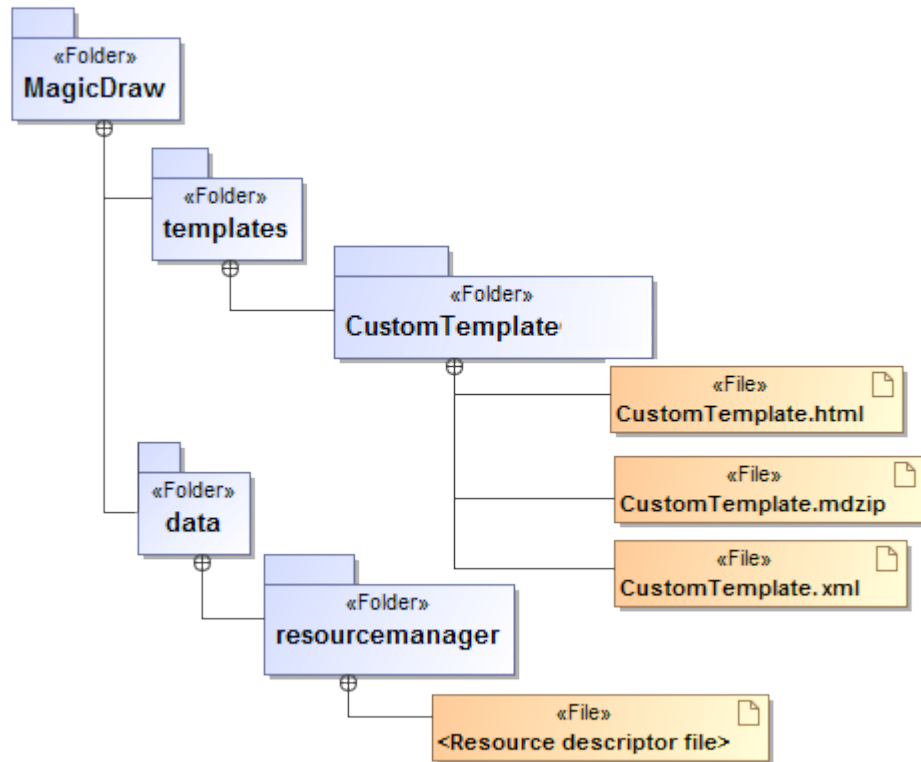
i For more information about working with Profiles see *MagicDraw UserManual.pdf* and *UML Profiling and DSL UserGuide.pdf*.

Distributing Templates

Files for the template distribution are as follows:

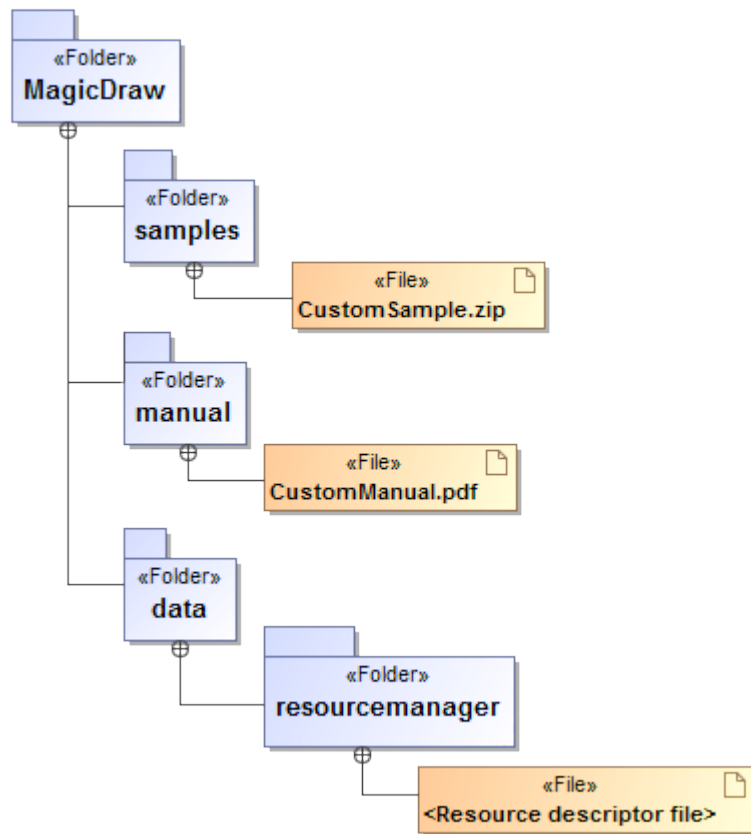
- *CustomTemplate.mdzip* (required) - the template project file.
- *CustomTemplate.html* - the description of the template project.
- *CustomTemplate.html* - the description of the template group.

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:

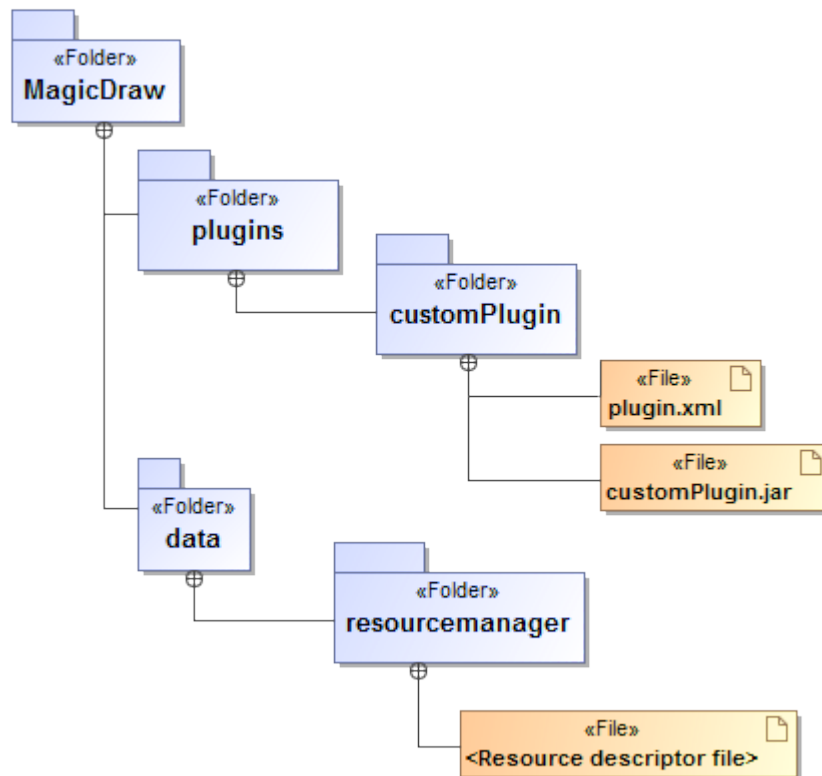


Distributing Plugins

Required files for a plugin distribution are as follows:

- *plugins.xml* - A [plugin description](#) (see page 22).
- *customPlugin.jar* - jarred [plugin class files](#) (see page 25). 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 48)
- [Plugin descriptor](#) (see page 22)
- [Plugin classes](#) (see page 25)

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.

i 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.

```
<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="data/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. <i>id</i> 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 .
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 <i>version internal</i> value multiplied by 10. <i>version resource</i> should be a number.
version internal	This version number is not visible to the user and may be used for an internal count. <i>version internal</i> should be a number.
version human	A human readable version number of the resource. This version number is visible to users. The <i>version human</i> number may consist of numbers and/or words.

Element	Description
edition	Supported program editions.
installation	<p><i>installation</i> includes files, which will be copied from the custom plugin archive to the program folder.</p> <div style="border: 1px solid red; padding: 10px; margin-top: 10px;"> <p> 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.</p> </div>
critical	If the resource is selected as critical, it cannot be removed because MagicDraw does not allow this.

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. Create a new empty folder, place the resources there.
4. In the modeling tool main menu, click **Tools > Development Tools > Build Resource Distribution File**.
5. Browse and select the created folder, choose the output location and click **OK**. A resource distribution file will be built in the output folder.

 The **Development Tools** plugin can be installed from [nomagic.com](https://www.nomagic.com/)¹⁰ or found in the resource distribution file (.rdzip).

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:


¹⁰ <https://www.nomagic.com/>

- Resource directory path
- Output directory path
- Result distribution file name

Example:

```
java -cp "lib/*;lib/bundles/*"

com.nomagic.magicdraw.resourcemanager.distribution.GenerateResourceDistributionFileTask "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 \(see page 43\)](#)

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

- [Invoking actions \(see page 53\)](#)
- [Creating new actions \(see page 54\)](#)
- [Actions' hierarchy \(see page 61\)](#)
- [Working with icons \(see page 62\)](#)
 - [MDP protocol \(see page 64\)](#)
- [Running action with progress \(see page 65\)](#)

[Using and extending other UI component \(see page 165\)](#)

- [Configuring element Specification window \(see page 165\)](#)
- [Showing Element selection dialog \(see page 166\)](#)
- [Showing question, error, warning dialogs \(see page 166\)](#)
- [Showing notifications, adding text into Message Window \(see page 167\)](#)
- [Creating PopupMenu or other menu \(see page 168\)](#)
- [Adding custom project window \(see page 169\)](#)

Invoking actions

Actions can be invoked from:


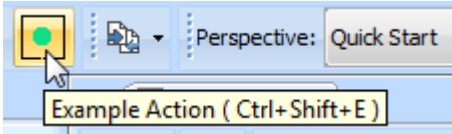
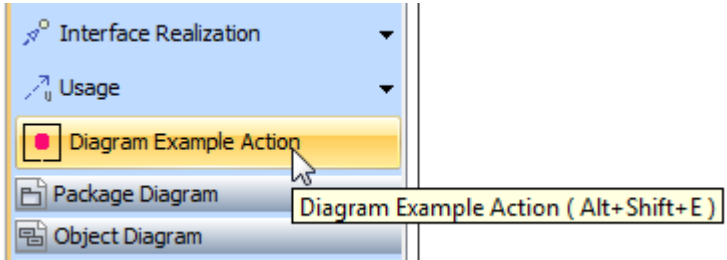
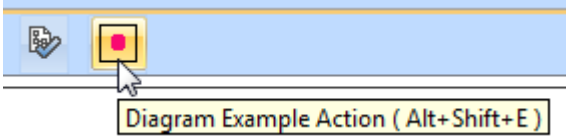
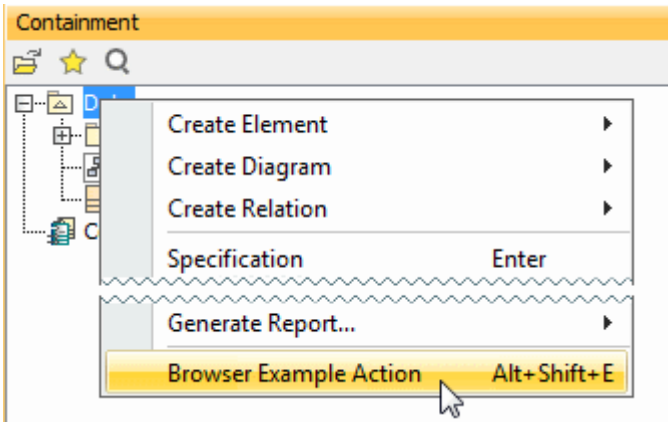
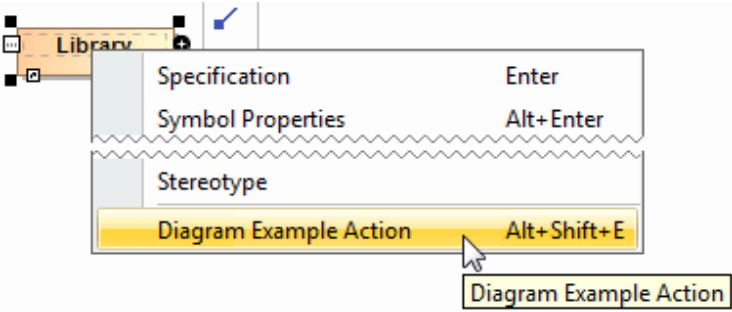
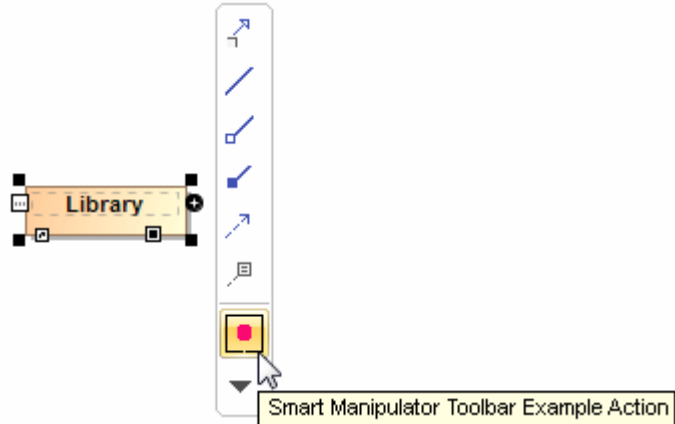
Main menu	
Main toolbar	
Diagram palette	
Diagram toolbar	
Browser shortcut menu	

Diagram shortcut menu	
Smart manipulator toolbar	
Keyboard shortcuts	<p>This action cannot be represented in GUI. Create a new action and assign a keyboard shortcut to invoke that action.</p>

Creating new actions

- i** 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`

Step #1: Create a new action class

All actions used in a modeling tool must be subclasses of the `com.nomagic.magicdraw.actions.MDAction` class (see javadoc for more details). The following `MDAction` subclasses that are used for basic purposes are already created in a modeling tool:

- `com.nomagic.magicdraw.ui.browser.actions.DefaultBrowserAction` – 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` – 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* – 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)* method and implement in it what this actions is going to do.

Example #1: A simple action

```
class SimpleAction extends MDAAction
{
    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());
    }
}
```

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();
            }
        }
    }
}
```


```

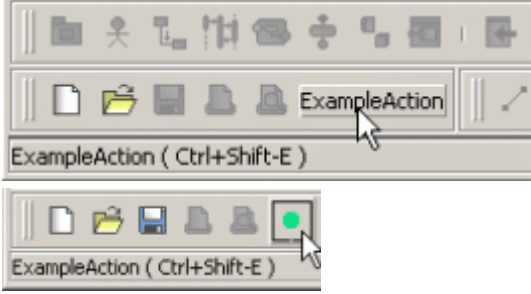
    }

    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 <i>null</i> , a new ID is generated. An action name can be used as ID. All MagicDraw (or another modeling tool developed by us) default actions IDs are defined in the <i>com.nomagic.magicdraw.actions.ActionsID</i> class (for more details, see JavaDoc). 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.
Shortcut and mnemonic	<p>Every action can have an assigned keyboard shortcut. Shortcuts can be customized from the Environment Options dialog box.</p> <p>(see MagicDraw User Manual, Section “Environment Options”.)</p> <div>  The action must have an ID assigned (not null). In other case, shortcuts cannot be restored after restarting an application. </div>

Property	Function
Icon	<p>Every action can have a small and large icon. A small icon is described as <code>javax.swing.Action.SMALL_ICON</code> 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.</p>  <pre>// 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_ico n.gif")));</pre>
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.

Example:

```
MAction action = new MAction("Example", "Example", KeyEvent.VK_E,
    ActionsGroups.PROJECT_OPENED_RELATED);
```

2. Implement the `updateState()` method for the action.

Here you may describe all conditions when an action must be enabled and disabled. Example of the `updateState()` 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()` 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` class. `ActionsCategory` 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` class, which is some kind of an actions container. One `ActionsManager` 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
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* - a configurator for a general purpose. It is used for menus, toolbars, browser, and diagrams shortcuts.
- *com.nomagic.magicdraw.actions.BrowserContextAMConfigurator* - 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* - 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;
    }
};
```

Example #2: add some action into main menu, after creating a new project

```
// Create an action.
```

```

final MDAAction 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* class. *ActionsConfiguratorsManager* enables to add or remove many configurators to every configuration predefined by a modeling tool.

All available configurations are accessible in the following way:

```

ActionsConfiguratorsManager.getInstance().add<configuration_name>Configurator(configu
rator);

```

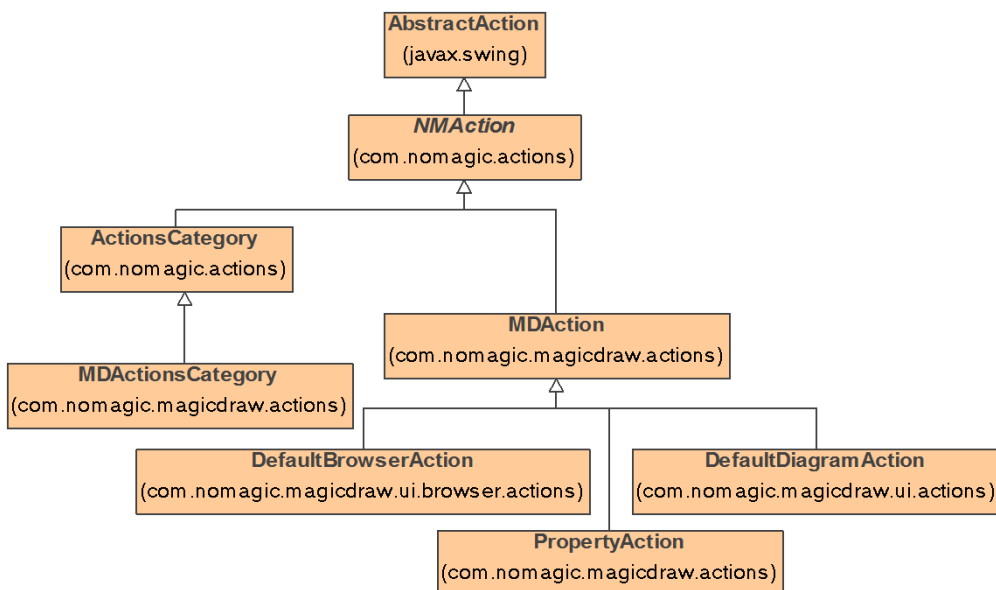
Example: Add new configurator for CONTAINMENT_BROWSER_CONTEXT configuration

```
//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:



4 Hierarchy of actions

Predefined Actions Configurations

MAIN_MENU

MAIN_TOOLBAR

MAIN_SHORTCUTS

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* 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*. *IconsFactory* 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.

Icon classes

Class	Description
<i>com.nomagic.ui.ResizableIcon</i>	The extended version of <i>javax.swing.Icon</i> . It can be scaled (resized) during the painting.
<i>com.nomagic.ui.ResizableIconAdapter</i>	An adapter of a simple <i>javax.swing.Icon</i> to a <i>com.nomagic.ui.ResizableIcon</i> .
<i>com.nomagic.ui.SquareIcon</i>	An icon which fits another icon into a square of a given size.

Class	Description
<i>com.nomagic.ui.ScalableImageIcon</i>	<p>The HiDPI (Retina) friendly <i>javax.swing.ImageIcon</i> 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.</p> <p>For example, <i>/com/product/icons/a.png</i> and <i>/com/product/icons/a.svg</i>. <i>ScalableImageIcon(Product.class, "/com/product/icons/a.png")</i> will load <i>a.png</i> on a regular dpi monitor, but <i>a.svg</i> 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 <i>a.png</i> icon and a scaling factor. For example, if a size of <i>a.png</i> 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.</p>

Utility classes

Class	Description
<i>com.nomagic.magicdraw.icons.IconsFactory</i>	A utility class to load icons from the MagicDraw icons package or other locations.
<i>com.nomagic.ui.IconUtilities</i>	A utility class to work with Icons.

Advanced icons classes

Class	Description
<i>com.nomagic.ui.AlphaCompositelIcon</i>	Paints a wrapped icon with an alpha composite.
<i>com.nomagic.ui.BorderIcon</i>	An icon for adding a border around a wrapped icon.
<i>com.nomagic.ui.DoubleIcon</i>	An Icon for painting two wrapped icons on a top of each other.

Class	Description
<i>com.nomagic.ui.RetinalImageIcon</i>	<p>A retina friendly image icon implementation. The icon (and the image returned by this icon) is twice smaller then the wrapped icon itself.</p> <p>A wrapped icon is used for painting. On painting, Graphics is scaled down by a retina scale factor and the wrapped icon is painted.</p>
<i>com.nomagic.ui.DoubleSizeImageIcon</i>	An icon combines two other icons and chooses which one to paint depending on the graphics scaling.
<i>com.nomagic.ui.ResizableIconImageIcon</i>	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 <i>java.awt.Image</i> (<i>sun.awt.image.MultiResolutionToolkitImage</i>) is returned which supports multi-resolution.

On this page

- [Icon classes \(see page 62\)](#)
- [Utility classes \(see page 63\)](#)
- [Advanced icons classes \(see page 63\)](#)

Related pages

- [MDP protocol \(see page 64\)](#)

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 <i>activitydiagram.png</i> icon from MagicDraw jar, use such URL: <code>mdp://resource/com/nomagic/magicdraw/icons/diagrams/activitydiagram.png</code>
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: <code>mdp://attachedFile#_18_2_8ca0285_1436339188295_24687_13288</code>

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.RunWithProgress* or *com.nomagic.magicdraw.ui.MagicDrawProgressStatusRunner*
- a background task - use *com.nomagic.magicdraw.task.BackgroundTaskRunner*



You can find the code examples in *<modeling tool installation directory>\openapi\examples\progressstatus*

Related pages

- [Multi-threading \(see page 361\)](#)

Working with project

- [Project concept \(see page 66\)](#)
- [Project descriptor \(see page 67\)](#)
- [Projects management \(see page 67\)](#)
- [Working with projects from Teamwork Cloud server \(see page 73\)](#)
- [Project structure decomposition \(see page 73\)](#)
- [Project options \(see page 74\)](#)
- [Project life-cycle events \(see page 76\)](#)

Project concept

The *com.nomagic.magicdraw.core.Project* 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* is accessible for any other *com.nomagic.magicdraw.uml.BaseElement* (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](#)¹¹. Locally stored projects or projects have 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();
```

Related pages

- [Project structure decomposition](#) (see page 73)
- [UML Model](#) (see page 76)
- [Working with diagrams](#) (see page 117)
- [Project options](#) (see page 74)


¹¹ <https://docs.nomagic.com/display/MD2024xR3/Using+Teamwork+Cloud>

Project descriptor

com.nomagic.magicdraw.core.project.ProjectDescriptor 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* 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 can create an appropriate *ProjectDescriptor* object,

com.nomagic.magicdraw.esi.EsiUtils helps to find an existing [Teamwork Cloud](https://docs.nomagic.com/display/MD2024xR3/Using+Teamwork+Cloud)¹² (remote) project descriptors.

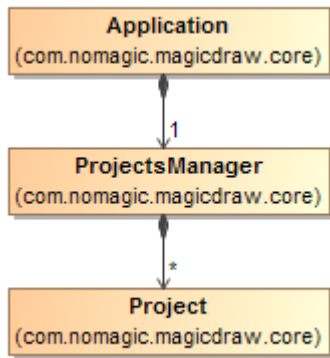
Projects management

The *com.nomagic.magicdraw.core.project.ProjectsManager* class is responsible for containment and management of projects.

Use the code below to access *ProjectsManager*

```
ProjectsManager projectsManager = Application.getInstance().getProjectsManager();
```

¹² <https://docs.nomagic.com/display/MD2024xR3/Using+Teamwork+Cloud>



ProjectsManager provides the API for *com.nomagic.magicdraw.core.Project* 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*:

```

//Gets an active project
Project project = Application.getInstance().getProject();
  
```

 You can find the code examples in `<installation_directory>\openapi\examples\projects`

Related pages

- [Project loading and saving \(see page 68\)](#)
- [Used project management \(see page 70\)](#)

Project loading and saving

Projects are saved and loaded by using two methods in the *com.nomagic.magicdraw.core.project.ProjectsManager* class.

- *saveProject(ProjectDescriptor, boolean)*
- *loadProject(ProjectDescriptor, boolean)*

 "Save" means "Commit" 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* to plugin into a save/commit operation.

Register it using *com.nomagic.magicdraw.core.Application.addSaveParticipant(SaveParticipant)*.

Implementing *SaveParticipant.isReadyForSave(Project, ProjectDescriptor)* you can "disable" the save/commit operation until some conditions are met.

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.getRemoteProjectDescriptorByQualified_name(remoteProjectQualified_name);  
if (projectDescriptor != null)  
{  
    projectsManager.loadProject(projectDescriptor, true);  
}
```

Related pages

- [Project descriptor](#) (see page 67)

Used project management

com.nomagic.magicdraw.core.project.ProjectsManager 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);
```

Example #2. Exporting a used server project

The *com.nomagic.magicdraw.teamwork.application.TeamworkUtils* class is used to export a used server project.

```
TeamworkUtils.exportTeamworkModule(project, packages, "My used remote project",  
remoteProjectQualified_name);
```

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:

```
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

```
ProjectManager projectManager = Application.getInstance().getProjectManager();
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
projectManager.sharePackage(project, Arrays.asList(aPackage), "my used project")
;

// Save a project
ProjectDescriptor projectDescriptor =
ProjectDescriptorsFactory.getDescriptorForProject(project);
projectManager.saveProject(projectDescriptor, true);
```

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* utility class for retrieving more project decomposition related information.

Use the *com.nomagic.magicdraw.core.modules.ModulesService* or *ProjectsManager* classes for managing attached projects.

Related pages

- [Project descriptor](#) (see page 67)
- [Project structure decomposition](#) (see page 73)

Working with projects from Teamwork Cloud server

Use the *com.nomagic.magicdraw.esi.EsiUtils* utility class to work with projects from [Teamwork Cloud](#)¹³.

Check javadoc for more details.

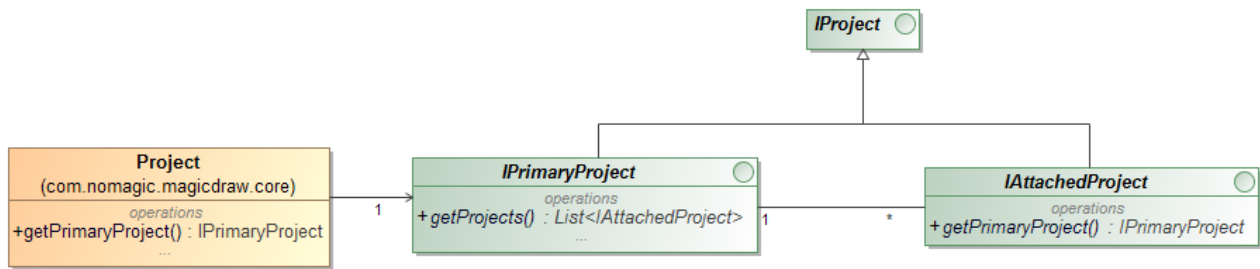


You can find the code examples in *<modeling tool installation directory>\openapi\examples\teamworkcloud*

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.

¹³ <https://docs.nomagic.com/display/MD2024xR3/Using+Teamwork+Cloud>



5 Project structure decomposition

Use the *com.nomagic.magicdraw.core.ProjectUtilities* utility class for retrieving more project decomposition related information.

Use *com.nomagic.magicdraw.core.modules.ModulesService* or *com.nomagic.magicdraw.core.project.ProjectsManager* classes for managing attached projects.

Related pages

- [Projects management \(see page 67\)](#)
- [Used project management \(see page 70\)](#)

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* and *com.nomagic.magicdraw.properties.PropertyManager* classes.

Project options belong to three categories:

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()*.

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()* method in order to have the additional project option "Test Property" for every project.

```

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 \(see page 161\)](#)

Project life-cycle events

Use *com.nomagic.magicdraw.core.project.ProjectsManager.addProjectListener(ProjectEventListener)* to register *com.nomagic.magicdraw.core.project.ProjectEventListener*. 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()* method.

Save participant

Use *com.nomagic.magicdraw.core.SaveParticipant* to plugin into a save/commit operation.

Register it using *com.nomagic.magicdraw.core.Application.addSaveParticipant(SaveParticipant)*¹⁴.

Related pages

- [Projects management \(see page 67\)](#)

UML model

In projects created with our modeling tools 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. 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

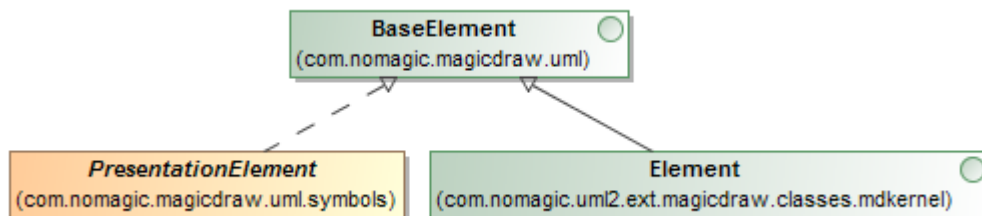
¹⁴ <http://jdocs.nomagic.com/2024x/com/nomagic/magicdraw/core/Application.html#addSaveParticipant-com.nomagic.magicdraw.core.SaveParticipant->

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* in your code!

The base interface of all model classes is *com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element*, it implements the *com.nomagic.magicdraw.uml.BaseElement* interface.



All structure derived from the *Element* you can see in the OMG-UML 2.5.1 Superstructure Specification.

All attributes and references 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 361).

Related pages

- [Working with UML model](#) (see page 79)
 - [Session management](#) (see page 80)
 - [Checking element editing permissions](#) (see page 80)
 - [Accessing and modifying model element properties](#) (see page 81)
 - [Creating new model elements](#) (see page 84)
 - [Creating new relationship objects](#) (see page 84)
 - [Editing model elements](#) (see page 85)
 - [Adding, moving, deleting model elements](#) (see page 86)
 - [Refactoring model elements](#) (see page 89)
 - [Copying elements and symbols](#) (see page 92)
 - [Identifying elements](#) (see page 93)
 - [Finding elements by name or by meta-type](#) (see page 95)
 - [Model traversing, Visitor pattern](#) (see page 95)
 - [Visitors](#) (see page 96)
 - [ModelHierarchyVisitor](#) (see page 97)
 - [Retrieving meta information about element](#) (see page 97)
 - [Creating textual element representation](#) (see page 98)
 - [Retrieving element icon](#) (see page 98)

- [Advanced utility functions \(see page 99\)](#)
- [Event support \(see page 99\)](#)
 - [Property change event concept \(see page 100\)](#)
 - [UML model properties names \(see page 101\)](#)
 - [Listening to property change events \(see page 101\)](#)
 - [Listening to related elements in hierarchy events \(see page 103\)](#)
 - [Listening to transaction commit events on session closing \(see page 104\)](#)
- [Working with stereotypes and tagged values \(see page 106\)](#)
 - [Using StereotypesHelper \(see page 106\)](#)
 - [Custom profile implementation \(see page 110\)](#)
 - [Standard profiles implementation \(see page 115\)](#)
- [UML Model Implementation Using EMF \(see page 115\)](#)
 - [UML model implementation in version 17.0 details \(see page 116\)](#)
 - [UML model implementation in 17.0.1 and up details \(see page 116\)](#)

Working with UML model

Session management

Checking element editing permissions

Accessing and modifying model element properties

Creating new model elements

Creating new relationship objects

Editing model elements

Adding, moving, deleting model elements

Refactoring model elements

Copying elements and symbols

Identifying elements

Finding elements by name or by meta-type

Model traversing, Visitor pattern

- [Visitors](#) (see page 96)
- [ModelHierarchyVisitor](#) (see page 97)

Retrieving meta information about element

Creating textual element representation

Retrieving element icon

Advanced utility functions

Session management

com.nomagic.magicdraw.openapi.uml.SessionManager is the singleton manager used for editing model Elements. All modifications to model elements should be performed between the *createSession(Project, java.lang.String)* and *closeSession(Project)*¹⁵ method calls.

To edit some *com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element*, a session with this manager must be created. After editing a model element, a session must be closed. After that, all changes will be 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(Project, java.lang.String)* 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 Cloud environment
- Some other restrictions are added in some specific modeling case

¹⁵ <http://jdocs.nomagic.com/2024x/com/nomagic/magicdraw/openapi/uml/SessionManager.html#closeSession-com.nomagic.magicdraw.core.Project->

Checking editing permission

Use *com.nomagic.magicdraw.uml.BaseElement.isEditable()* 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()* to check if children can be added into the Element.

Use *BaseElement.canAdd(com.nomagic.magicdraw.uml.BaseElement)* 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()* 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(ElementPermissions)* 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

Element properties can be accessed with simple setters and getters. For example *NamedElement.name* property:

```
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.NamedElement el = ...;
String name = el.getName();
el.setName("new name");
```

Container Properties

Our modeling tools use 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()` method. Owned children can be received with the `Element.getOwnedElement()` method. Different types of children are stored in separate container properties.

You can access these container properties by names that are described in the UML specification. The `getOwnedElement()` 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:

```
modelElement.get<SomeContainer>().add(child);
modelElement.get<SomeContainer>().remove(child);
```

Navigable opposite references

References between elements are defined in 'pairs' - for each primary navigable reference there is an opposite reference. Majority of those opposite references are not navigable (are not exposed in opposite metaclass).

For example *TypedElement* references it's *Type* by reference *type*, but *Type* does not expose *TypedElements* typed by it. Such approach is valid semantically, but makes model traversing challenging.

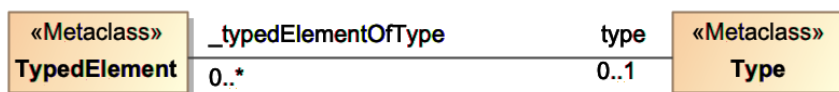
Our UML metamodel implementation does not follow this restriction and exposes all opposite references.

Some opposite references have 'friendly' names defined by UML specification.

For example *NamedElement.clientDependency* opposite reference name is *Dependency.client*.

Other opposite reference names are constructed using a pattern *_<primaryReferenceDefiningMetaclassName>Of<primaryReferenceName>*. Such naming pattern provides different names for these opposite references.

For example, there is a primary reference *type* defined in *TypedElement*. The opposite reference is defined in *Type* and has the name *_typedElementOfType*.



Setters and getters are provided for each opposite reference. For example:

```
Primary reference Element.getAppliedStereotype(),
opposite Stereotype.get_stereotypedElement()
```

```
Primary reference TypedElement.getType(),
opposite Type.get_typedElementOfType()
```

Related pages

- [Session management \(see page 80\)](#)
- [Checking element editing permissions \(see page 80\)](#)

Creating new model elements

Use the `com.nomagic.uml2.impl.ElementsFactory` class for creating model elements. To create a model element, a session with the `com.nomagic.magicdraw.openapi.uml.SessionManager` 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);
```

i You can find the code examples in `<modeling tool installation directory>\openapi\examples\accessors`

Related pages

- [Session management](#) (see page 80)
- [Creating a diagram](#) (see page 123)

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`,
- `com.nomagic.uml2.ext.magicdraw.activities.mdbasicactivities.ActivityEdge`,
- `com.nomagic.uml2.ext.magicdraw.statemachines.mdbehaviorstatemachines.Transition`,
- `com.nomagic.uml2.ext.magicdraw.activities.mdextrastructuredactivities.ExceptionHandler`,
- `com.nomagic.uml2.ext.magicdraw.compositestructures.mdinternalstructures.Connector`.

For checking if the model element is a relationship, call the `com.nomagic.uml2.ext.jmi.helpers.CoreHelper.isRelationship(Element)`¹⁶ method.

For getting supplier and client elements of a relationship, use *CoreHelper* class `getSupplierElement(Element)`¹⁷, `getClientElement(Element)`¹⁸ methods.

Use *CoreHelper* class `setSupplierElement(Element, Element)`, `setClientElement(Element, 19Element)`²⁰ 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* class `setSupplierElement(Element, Element)`, `setClientElement(Element, 21Element)`²² methods.
3. Add a new relationship into some parent by using the `com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element.setOwner(Element)` method.

Related pages

- [Session management \(see page 80\)](#)
- [Checking element editing permissions \(see page 80\)](#)

Editing model elements

To edit some `com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element`, a session with the `com.nomagic.magicdraw.openapi.uml.SessionManager` 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())
```

¹⁶ [http://jdocs.nomagic.com/2024x/com/nomagic/uml2/ext/jmi/helpers/CoreHelper.html#isRelationship-](http://jdocs.nomagic.com/2024x/com/nomagic/uml2/ext/jmi/helpers/CoreHelper.html#isRelationship-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-)

¹⁷ [http://jdocs.nomagic.com/2024x/com/nomagic/uml2/ext/jmi/helpers/CoreHelper.html#getSupplierElement-](http://jdocs.nomagic.com/2024x/com/nomagic/uml2/ext/jmi/helpers/CoreHelper.html#getSupplierElement-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-)

¹⁸ [http://jdocs.nomagic.com/2024x/com/nomagic/uml2/ext/jmi/helpers/CoreHelper.html#getClientElement-](http://jdocs.nomagic.com/2024x/com/nomagic/uml2/ext/jmi/helpers/CoreHelper.html#getClientElement-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-)

¹⁹ [http://jdocs.nomagic.com/2024x/com/nomagic/uml2/ext/jmi/helpers/CoreHelper.html#setClientElement-](http://jdocs.nomagic.com/2024x/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-)

²⁰ [http://jdocs.nomagic.com/2024x/com/nomagic/uml2/ext/jmi/helpers/CoreHelper.html#setClientElement-](http://jdocs.nomagic.com/2024x/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-)

²¹ [http://jdocs.nomagic.com/2024x/com/nomagic/uml2/ext/jmi/helpers/CoreHelper.html#setClientElement-](http://jdocs.nomagic.com/2024x/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-)

²² [http://jdocs.nomagic.com/2024x/com/nomagic/uml2/ext/jmi/helpers/CoreHelper.html#setClientElement-](http://jdocs.nomagic.com/2024x/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-)

```

{
    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 \(see page 80\)](#)
- [Checking element editing permissions \(see page 80\)](#)

Adding, moving, deleting model elements

There are two ways to add/move/delete model elements:

- Use *com.nomagic.magicdraw.openapi.uml.ModelElementsManager*
- Call direct functions on *Elements*

The *ModelElementsManager* is the singleton utility class which checks *com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element* 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* is not used, a programmer must perform these checks in the code explicitly.

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* is moved. The general rule would be to use an adding function for newly created *Element(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 \(see page 89\)](#) page.

Adding element to owner with *ModelElementsManager*

For adding a new model *Element* into a model, use the *addElement(Element, Element)* method provided by the *ModelElementsManager*.

```

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* cannot be added into a *com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Package* or an *Operation* cannot be added into a not locked for editing *com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class* (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* is thrown.

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*, use the *moveElement(Element, Element)* method provided by the *ModelElementsManager*.

```
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* cannot be moved into a *Package* or an *Operation* cannot be moved into a not locked for editing *Class* (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* 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);
```

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 \(see page 86\)](#)
- [Adding element to owner using a direct function \(see page 87\)](#)
- [Moving element with ModelElementsManager \(see page 87\)](#)
- [Moving element using a direct function \(see page 88\)](#)
- [Removing element with ModelElementsManager \(see page 88\)](#)
- [Removing element using direct call \(see page 88\)](#)

Related pages

- [Session management \(see page 80\)](#)
- [Checking element editing permissions \(see page 80\)](#)
- [Refactoring model elements \(see page 89\)](#)

Refactoring model elements

To refactor an element in a model, use the *com.nomagic.magicdraw.uml.Refactoring* class.

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);
```

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* 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* and *com.nomagic.magicdraw.uml.refactor.extract.ExtractTarget*. Invoke the refactoring with *com.nomagic.magicdraw.uml.refactor.extract.ExtractManager.extract()*. Review refactoring results by inspecting *ExtractSource* and *ExtractTarget*.

```
// 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*([Element](#)²³) 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.

```

²³ <http://jdocs.nomagic.com/2024x/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Element.html>

```

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<Element>, Element)` to move `com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element` and all connected `com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Relationship(s)` to a new owner.

 You can find the code examples in *<modeling tool installation directory>\openapi\examples\refactor*

Related pages

- [Session management](#) (see page 80)
- [Checking element editing permissions](#) (see page 80)

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);

```

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 `<installation_directory>\openapi\examples\copypaste`

Related pages

- [Session management](#) (see page 80)
- [Checking element editing permissions](#) (see page 80)

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()*
2. *com.nomagic.magicdraw.foundation.MDObject.getID()*

The meaning of the returned identifiers depends on the project type.

Local (.mdzip) 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.

✓ 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 projects

If a local project is added to the Teamwork Cloud (or vice versa), *getLocalID()* consistently returns the same identifier value. Note that several elements with the same local ID may exist in the scope of a single Teamwork Cloud project.

getID() returns UUID, which is unique both in the project and in the scope of the whole Teamwork Cloud 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()* returns the same value for different copies, *getID()* will return a unique value for each copy (project usage).

✓ Example

If a project *A* is copied as a project *A1*, and a Teamwork Cloud project *B* uses both *A* and *A1*, *getLocalID()* returns the same value for corresponding elements of *A* and *A1*, but *getID()* returns different identifiers for the same elements.

i Tip

Both identifiers can be used to retrieve the element using the method `com.nomagic.magicdraw.core.Project.getElementByID(java.lang.String)`. 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 \(see page 79\)](#)

Finding elements by name or by meta-type

Use *com.nomagic.magicdraw.uml.Finder* 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.class, "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*

Related pages

- [Retrieving meta information about element \(see page 97\)](#)

Model traversing, Visitor pattern

Here is an example of how to collect all children from the
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element and avoid recursion using a simple *for* cycle:

```
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* utility:

- *byScope()*
- *byType()*
- *byTypeRecursively()*

Related pages

- [Visitors \(see page 96\)](#)
- [ModelHierarchyVisitor \(see page 97\)](#)

Visitors

Visitor design pattern is implemented in UML model structure. Every *com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element* has the *accept(AbstractVisitor)* method for visiting it.

The *com.nomagic.uml2.impl.ModelVisitor* 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* (for example, save/load, copy/paste, or a specific properties setting).

Just derive your class from *com.nomagic.magicdraw.uml.InheritanceVisitor* and override some *visit* methods and call *accept(AbstractVisitor)* for *Elements*.

```
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 `<installation_directory>\openapi\examples\statistics`

ModelHierarchyVisitor

The `com.nomagic.uml2.impl.ModelHierarchyVisitor` is an enhanced implementation of the *Visitor* pattern and a subclass of `com.nomagic.uml2.impl.ModelVisitor`, 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` 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(Classifier)` and it will be executed for all subclasses of a `com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Classifier`.

```
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 `<installation_directory>\openapi\examples\statistics`

Retrieving meta information about element

Every Element is identified by its "metaclass". Element metaclass is mapped into java interface (for example `com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class`).

Use *com.nomagic.magicdraw.uml.BaseElement.getClassType()* to retrieve the element metaclass.

Use *com.nomagic.magicdraw.uml.ClassTypes* for managing the metaclasses - collecting all available, collecting derived and etc.

Creating textual element representation

Use *com.nomagic.magicdraw.uml.RepresentationTextCreator* 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);
```

Providing a custom representation text

You can override default custom text creation for the specific element.

Use *RepresentationTextCreator* class method *addProvider(RepresentationTextCreator.RepresentationTextProvider)* to register a custom representation text provider.

Retrieving element icon

Use *com.nomagic.magicdraw.uml.ElementIcon* 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.Package.class);
```

Related pages

- [Retrieving meta information about element \(see page 97\)](#)
- [Working with icons \(see page 62\)](#)

Advanced utility functions

Use *com.nomagic.uml2.ext.jmi.helpers.ModelHelper* 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
 - collecting connected relationships of element.
 - working with Associations - getting member ends, changing navigability
 - 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* for advanced Information Flows related functions

Use *com.nomagic.uml2.ext.jmi.helpers.InteractionHelper* for Interaction domain related advanced functions

Use *com.nomagic.uml2.ext.jmi.helpers.StateMachineHelper* for State Machine related advanced functions

Use *com.nomagic.uml2.ext.jmi.helpers.UseCaseHelper* for Use Case related advanced functions

Use *com.nomagic.magicdraw.uml.ConnectorsCollector* to collect connected Connector elements

Event support


Our modeling tools provide an API for listening to 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.

 You can find the code examples in `<program installation directory>\openapi\examples\events`.

Related pages

- [Property change event concept \(see page 100\)](#)
- [UML model properties names \(see page 101\)](#)
- [Listening to property change events \(see page 101\)](#)
- [Listening to related elements in hierarchy events \(see page 103\)](#)
- [Listening to transaction commit events on session closing \(see page 104\)](#)

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 \(see page 101\)](#) 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 \(see page 101\)](#)

UML model properties names

Constants for UML model properties names are defined in several classes.

Use *com.nomagic.uml2.impl.PropertyNames* for standart UML metamodel property names.

Use *com.nomagic.uml2.impl.PrivatePropertyNames* for auto-generated opposite property names.

The events' names related with model creation/ deletion are listed in the *com.nomagic.uml2.ext.jmi.UML2MetamodelConstants* 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.

- 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().isEnabledEventFiring()
```

- To stop/ start event firing, use:

```
project.getRepository().getEventSupport().setEnabledEventFiring(...)
```

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
    }
});
```

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* 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*, which will provide the chain of the property names to listen to.

The *SmartListenerConfig* class provides static methods for the default configuration of property chains. This is useful, when listening for common property change events.

Use the *SmartPropertyChangeListener* class *createSmartPropertyListener(...)* method to create and register such listener for the particular element, which will be notified with events provided by the *SmartListenerConfig*.

 If the listener is not needed anymore, unregister it using the *removePropertyChangeListener(...)* method.

Example #1. Listen to the element owner's name

```
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.createSmartPropertyChangeListener(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* is a special listener, which is notified when the all changes inside a transaction are done and the transaction is closed.

The *TransactionCommitListener* contains *transactionCommitted(java.util.Collection<java.beans.PropertyChangeEvent>)* method, which provides a collection of all *java.beans.PropertyChangeEvent(s)* that were executed in a transaction.

Create a custom transaction commit listener:

```

public class MyTransactionListener implements TransactionCommitListener
{
    public Runnable transactionCommitted(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);

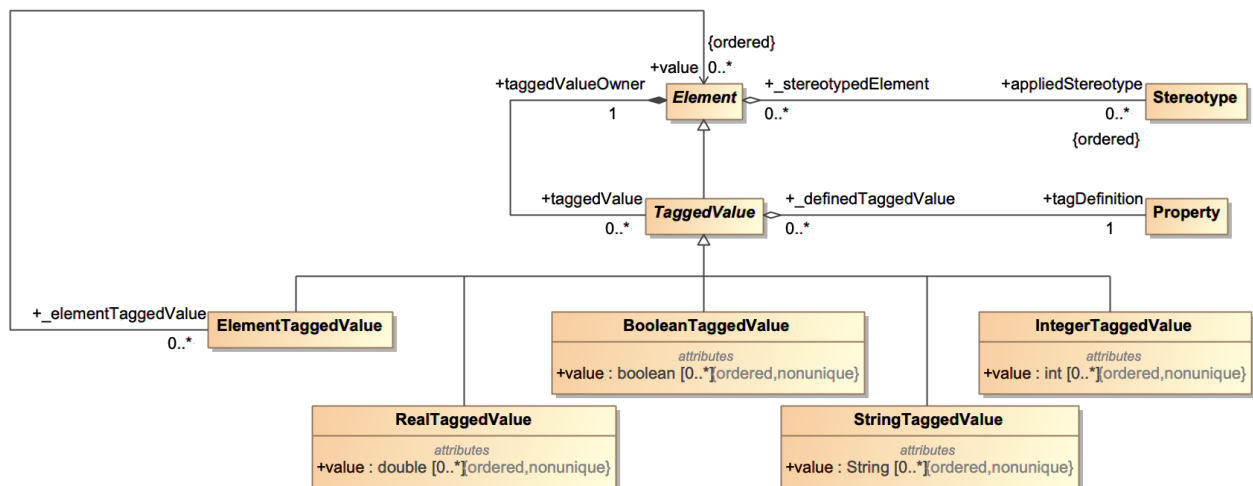
```

Related pages

- [Session management \(see page 80\)](#)

Working with stereotypes and tagged values

Standard UML metamodel is extended with additional meta-classes and meta-properties to support stereotypes applications.



There are two ways to work with stereotypes applications

- We provide a *com.nomagic.uml2.ext.jmi.helpers.StereotypesHelper* helper class for hiding this mapping complexity. It has set of useful methods for assigning or unassigning stereotypes, creating *TaggedValues*
- Use profile implementation classes


Related pages

- [Using StereotypesHelper](#) (see page 106)
- [Custom profile implementation](#) (see page 110)
- [Standard profiles implementation](#) (see page 115)

Using StereotypesHelper

Use utility class *com.nomagic.uml2.ext.jmi.helpers.StereotypesHelper* to work with profiles, stereotypes, tagged values.

StereotypesHelper extends *com.nomagic.uml2.ext.jmi.helpers.TagsHelper*. Keep in mind that *Tagged Values* in these helpers are also called stereotype properties.

 *StereotypeHelper* still provides deprecated 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* methods instead of passing a stereotype name

Find a stereotype by name

Find profile by URI (or by name) first using

StereotypesHelper.getProfileByURI(Project, java.lang.String)

StereotypesHelper.getProfile(Project, java.lang.String)

Find stereotype by name inside a profile using

*StereotypesHelper.getStereotype(Project, java.lang.String, Profile)*²⁴

Check if stereotype is applied

Check if any stereotype is applied

*StereotypesHelper.hasStereotype(Element)*²⁵

or

check if *Element.getAppliedStereotype()* collection is empty

Check if concrete stereotype is assigned

*StereotypesHelper.hasStereotype(Element, ²⁶Stereotype)*²⁷

or

check if *Element.getAppliedStereotype()* collection contains stereotype

²⁴ <http://jdocs.nomagic.com/2024x/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#getStereotype-com.nomagic.magicdraw.core.Project-java.lang.String-com.nomagic.uml2.ext.magicdraw.mdprofiles.Profile->

²⁵ <http://jdocs.nomagic.com/2024x/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#hasStereotype-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element->


²⁶ <http://jdocs.nomagic.com/2024x/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#hasStereotype-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype->

²⁷ <http://jdocs.nomagic.com/2024x/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#hasStereotype-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype->

Check if concrete stereotype or derived stereotype is applied

StereotypesHelper.hasStereotypeOrDerived(Element,²⁸Stereotype)²⁹

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.

TagsHelper.getStereotypePropertyValue(Element, Property)

TagsHelper.getStereotypePropertyValue(Element, Stereotype, java.lang.String)

TagsHelper.getStereotypePropertyFirst(Element, Property)

TagsHelper.getStereotypePropertyFirst(Element, Stereotype, java.lang.String)

TagsHelper.getTaggedValue(Element, Property)

TagsHelper.getTaggedValue(Element, Stereotype, java.lang.String)

Set value of tagged value

TagsHelper.setStereotypePropertyValue(Element, Stereotype, Property, java.lang.Object)

TagsHelper.setStereotypePropertyValue(Element, Stereotype, java.lang.String, java.lang.Object)

TagsHelper.clearStereotypeProperty(Element, Property)

TagsHelper.clearStereotypeProperty(Element, Property)

Collect all elements with applied stereotype

²⁸<http://jdocs.nomagic.com/2024x/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#hasStereotypeOrDerived-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype->
²⁹<http://jdocs.nomagic.com/2024x/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#hasStereotypeOrDerived-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype->

For a concrete stereotype use

StereotypesHelper.getStereotypedElements(Stereotype)

For a concrete stereotype and derived stereotypes from it use

StereotypesHelper.getStereotypedElementsIncludingDerived(Stereotype)

Check if stereotype can be applied to Element

StereotypesHelper.canApplyStereotype(Element,³⁰Stereotype)³¹

Remove stereotype from element

StereotypesHelper.removeStereotype(Element,³²Stereotype)³³

or

remove from collection Element.getAppliedStereotype(). TaggedValues of removed stereotype will not be removed immediately using this approach, but will be removed on transaction commit.

Apply stereotype to element

StereotypesHelper.addStereotype(Element,³⁴Stereotype)³⁵

or

add into collection Element.getAppliedStereotype()

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");
```

³⁰<http://jdocs.nomagic.com/2024x/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#canApplyStereotype-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype>

³¹<http://jdocs.nomagic.com/2024x/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#canApplyStereotype-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype>

³²<http://jdocs.nomagic.com/2024x/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#removeStereotype-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype>

³³<http://jdocs.nomagic.com/2024x/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#removeStereotype-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype>

³⁴<http://jdocs.nomagic.com/2024x/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#addStereotype-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype>

³⁵<http://jdocs.nomagic.com/2024x/com/nomagic/uml2/ext/jmi/helpers/StereotypesHelper.html#addStereotype-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element-com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype>

```
Stereotype stereotype = StereotypesHelper.getStereotype(project,
"StereotypeNameExample", profile);
StereotypesHelper.addStereotype(element, stereotype); //We add stereotype 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* 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 115) 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 (wrappers) ...

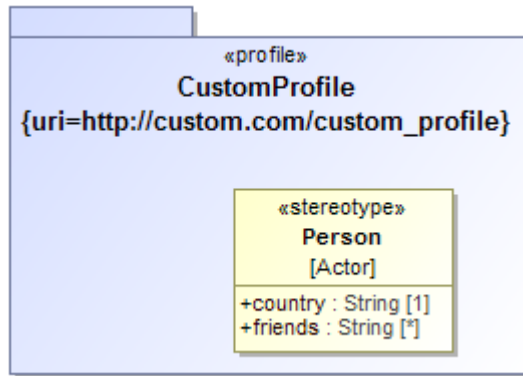
Profile implementation class has these features

- There is a constant defined for every DataType, Enumeration name
- There is a getter for every DataType element defined in the profile
- Java enumeration is generated for every Enumeration element defined in the profile. This class has following features
 - There are methods to get enumeration literals by generated enumeration and vice versa
- Inner java class is generated 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.
 - There is a check method to check if Element has specific stereotype applied
 - There is a getter for Stereotype element itself



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 = ...;
CustomProfile customProfile = CustomProfile.getInstance(element);
//check if element has Person stereotype applied
customProfile.person().is(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

```

import com.nomagic.magicdraw.uml.BaseElement;
import com.nomagic.magicdraw.uml2.Profiles;
import com.nomagic.profiles.ProfileCache;
import com.nomagic.profiles.ProfileImplementation;
import com.nomagic.profiles.ProfilesBridge;
import com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element;
import com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Property;
import com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype;

import javax.annotation.CheckForNull;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.HashSet;

@SuppressWarnings("WeakerAccess, unused")
public class CustomProfile extends ProfileImplementation
{
    public static final String PROFILE_URI = "http://custom.com/custom_profile";
  
```

```

public static final String PROFILE_NAME = "CustomProfile";

private final PersonStereotype personStereotype;

public static CustomProfile getInstance(BaseElement baseElement)
{
    CustomProfile profile = ProfilesBridge.getProfile(CustomProfile.class,
baseElement);
    if (profile == null)
    {
        return ProfilesBridge.createProfile(CustomProfile.class, baseElement,
CustomProfile::new, PROFILE_NAME, PROFILE_URI);
    }
    return profile;
}

public CustomProfile(ProfileCache cache)
{
    super(cache);
    personStereotype = new PersonStereotype(this);
}

public PersonStereotype person()
{
    return personStereotype;
}

public static class PersonStereotype extends StereotypeWrapper
{
    //stereotype Person and its tags
    public static final String STEREOYPE_NAME = "Person";
    public static final String COUNTRY = "country";
    public static final String FRIENDS = "friends";

    private final CustomProfile _p;
    @CheckForNull
    private Property country;
    @CheckForNull
    private Property friends;

    protected PersonStereotype(CustomProfile profile)
    {
        super(profile);
        _p = profile;
    }

    @Override
    @SuppressWarnings("ConstantConditions")
    public Stereotype getStereotype()
    {
        return getElementByName(STEREOYPE_NAME);
    }

    @CheckForNull

```

```

public Property getCountryProperty()
{
    if (country == null)
    {
        country = getTagByName(getStereotype(), COUNTRY);
    }
    return country;
}

@CheckForNull
public Property getFriendsProperty()
{
    if (friends == null)
    {
        friends = getTagByName(getStereotype(), FRIENDS);
    }
    return friends;
}

public void setCountry(Element element, @CheckForNull String value)
{
    Profiles.setValue(element, getStereotype(), getCountryProperty(), value);
}

public void clearCountry(Element element)
{
    Profiles.clearValue(element, getCountryProperty());
}

@CheckForNull
public String getCountry(Element element)
{
    return toString(Profiles.getFirstValue(element, getCountryProperty()));
}

public void setFriends(Element element, @CheckForNull
java.util.Collection<String> value)
{
    Profiles.setValue(element, getStereotype(), getFriendsProperty(), value);
}

public void clearFriends(Element element)
{
    Profiles.clearValue(element, getFriendsProperty());
}

public void addFriends(Element element, String value)
{
    Profiles.addValue(element, getStereotype(), getFriendsProperty(), value);
}

public void removeFriends(Element element, String value)
{
    java.util.List<String> values = getFriends(element);
    values.remove(value);
    setFriends(element, values);
}

```

```

    }

    @SuppressWarnings("unchecked")
    public java.util.List<String> getFriends(Element element)
    {
        return (java.util.List<String>) Profiles.getValue(element,
getFriendsProperty());
    }

    @Override
    protected void clear()
    {
        super.clear();
        country = null;
        friends = null;
    }

    @Override
    public boolean is(@CheckForNull Element element)
    {
        return element instanceof com.nomagic.uml2.ext.magicdraw.mdusecases.Actor &&
            _p.isTypeOf(element, getStereotype());
    }

    public static boolean isInstance(@CheckForNull Element element)
    {
        if (element instanceof com.nomagic.uml2.ext.magicdraw.mdusecases.Actor)
        {
            CustomProfile instance = getInstance(element);
            return instance.isTypeOf(element, instance.person().getStereotype());
        }
        return false;
    }
}

@Override
protected Collection<ProfileElementWrapper> generatedGetAllElementWrappers()
{
    Collection<ProfileElementWrapper> wrappers = new ArrayList<>();
    wrappers.add(personStereotype);
    return wrappers;
}

@Override
protected Collection<Stereotype> generatedGetAllStereotypes()
{
    if (getProfile() != null)
    {
        final Collection<Stereotype> stereotypes = new HashSet<>();
        stereotypes.add(personStereotype.getStereotype());
        return stereotypes;
    }
    return Collections.emptyList();
}
}

```

Related pages

- [Standard profiles implementation](#) (see page 115)

Standard profiles implementation

Standard profiles implementation are available as part of API.

Use *com.nomagic.uml2.MagicDrawProfile* to access "MagicDraw Profile" elements

Use *com.nomagic.uml2.StandardProfile* to access "StandardProfile" elements

Use *com.nomagic.uml2.ValidationProfile* 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

- [UML model implementation in version 17.0 details \(see page 116\)](#)
- [UML model implementation in 17.0.1 and up details \(see page 116\)](#)

UML model implementation in version 17.0 details

The MagicDraw (or other modeling tool developed by us) 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 version 17.0.1 of MagicDraw (or other modeling tool developed by us), 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*. 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*. 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();
```

There is no strict definition how EMF resources are organized in MagicDraw project. It depends on the project type (local 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](#) (see page 118)
 - [Adding new diagram types](#) (see page 119)
- [Creating a diagram](#) (see page 123)
- [Presentation elements \(symbols\)](#) (see page 123)
 - [Diagram presentation element](#) (see page 125)
 - [Shapes](#) (see page 125)
 - [Paths](#) (see page 125)
 - [Working with presentation elements](#) (see page 126)
 - [Creating shape elements](#) (see page 127)
 - [Creating path elements](#) (see page 127)
 - [Reshaping shape elements](#) (see page 128)
 - [Moving symbols on diagram to another location](#) (see page 128)
 - [Moving symbols on diagram to another location and to different parent](#) (see page 129)
 - [Changing path break points](#) (see page 130)
 - [Deleting presentation elements](#) (see page 130)
 - [Changing properties of presentation elements](#) (see page 131)
 - [Managing compartments](#) (see page 132)
 - [Collecting presentation elements, finding usages in diagrams](#) (see page 132)
 - [Displaying related elements in the diagram](#) (see page 134)
 - [Layouting symbols in the diagram](#) (see page 134)
 - [Displaying ports](#) (see page 135)
- [Custom presentation elements rendering](#) (see page 136)
 - [Custom renderer provider](#) (see page 137)
 - [Custom symbol renderer](#) (see page 137)
 - [Custom renderers sample](#) (see page 137)
 - [Registering custom symbol renderer provider](#) (see page 140)
- [Selection in diagrams](#) (see page 141)

- [Diagram events \(see page 142\)](#)
 - [Presentation elements modification events \(see page 142\)](#)
 - [Presentation element selection events \(see page 144\)](#)
 - [Notification of presentation element draw \(see page 144\)](#)
- [Custom diagram painters \(see page 145\)](#)
- [Creating diagram images \(see page 146\)](#)
- [Working with Generic tables \(see page 146\)](#)
- [Working with Dependency Matrix \(see page 150\)](#)
 - [Creating a custom dependency matrix type \(see page 151\)](#)
 - [Customizing a dependency matrix \(see page 153\)](#)
 - [Configuring the command bar and shortcut keys \(see page 155\)](#)
 - [Configuring shortcut menus of a row or column header element and a filter panel \(see page 155\)](#)
 - [Configuring cell entries - dependencies \(see page 156\)](#)
 - [Specifying the appearance of cells and the row or column headers \(see page 157\)](#)
 - [Accessing dependency matrix data \(see page 158\)](#)
- [Working with Relation Map \(see page 158\)](#)
- [Working with Gantt Chart \(see page 159\)](#)

Diagram types

There are two groups of diagrams in our modeling tools – 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.

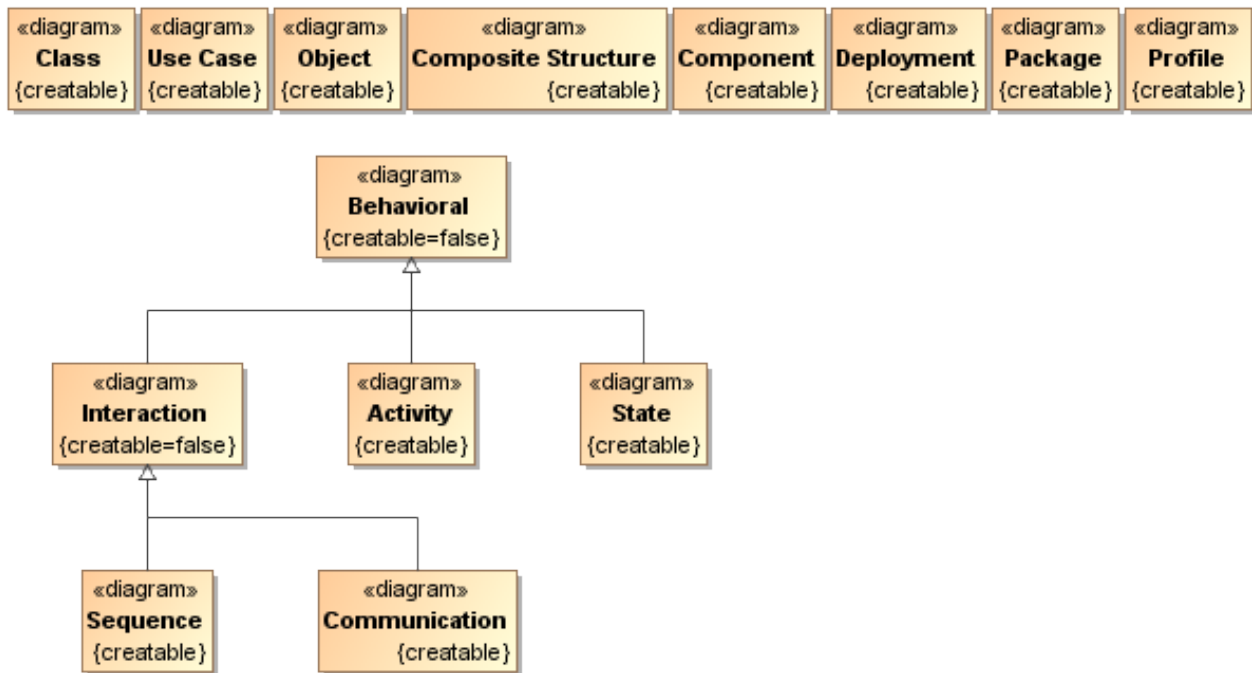
There are 17 predefined types of diagrams (13 creatable and 4 not). Creatable diagrams are:

- 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.



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 \(see page 119\)](#).

Related pages

- [Adding new diagram types \(see page 119\)](#)

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 \(see page 119\)](#)
2. [Registering a new diagram type \(see page 122\)](#)

Step #1. Override the abstract *com.nomagic.magicdraw.uml.DiagramDescriptor* class

Override the abstract class *DiagramDescriptor* and implement the following abstract methods:

- *getDiagramTypeId()* – this method must return a diagram type id, which is used to identify the diagram
- *getSingularDiagramTypeHumanName()* – the method returns the diagram type human name
- *getPluralDiagramTypeHumanName()* – the method returns the diagram type human name in a plural form
- *getSuperType()* – this method must return a super type (super diagram) of this diagram type

- `isCreatable()` – returns the flag indicating if this diagram type is creatable
- `getSVGIcon()` – returns a large icon for this type of the diagram (see [Adding New Functionality](#) (see page 52))
- `getSmallIconURL()` – returns a small icon URL for this type of the diagram (see [Adding New Functionality](#) (see page 52))
- `getDiagramActions()` – returns the manager of actions used in the diagram
- `getDiagramToolbarConfigurator()` – returns the action manager configurator which configures the described diagram toolbar (see [Adding New Functionality](#) (see page 52))
- `getDiagramShortcutsConfigurator()` – returns the action manager configurator which configures described diagram shortcuts (see [Adding New Functionality](#) (see page 52))
- `getDiagramContextConfigurator()` – returns the action manager configurator which configures described diagram shortcut menu actions (see [Adding New Functionality](#) (see page 52)).

The diagram descriptor example

i For the full source code, see the Open API examples in `<program installation directory>\openapi\examples`.

```
/**
 * 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(DiagramDescriptor)` method of the modeling tool you are using. This method can be invoked when a [plugin](#) (see [page 11](#)) is initialized.

The diagram descriptor registration example

i For the full source code, see the Open API examples in *<program installation directory>\openapi\examples\specificdiagram*.

```

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_DIAGRAM, parent);
    //open a diagram
    project.getDiagram(diagram).open();
}
catch (ReadOnlyElementException e)
{
}
SessionManager.getInstance().closeSession(project);
```

Related pages

- [Session management \(see page 80\)](#)

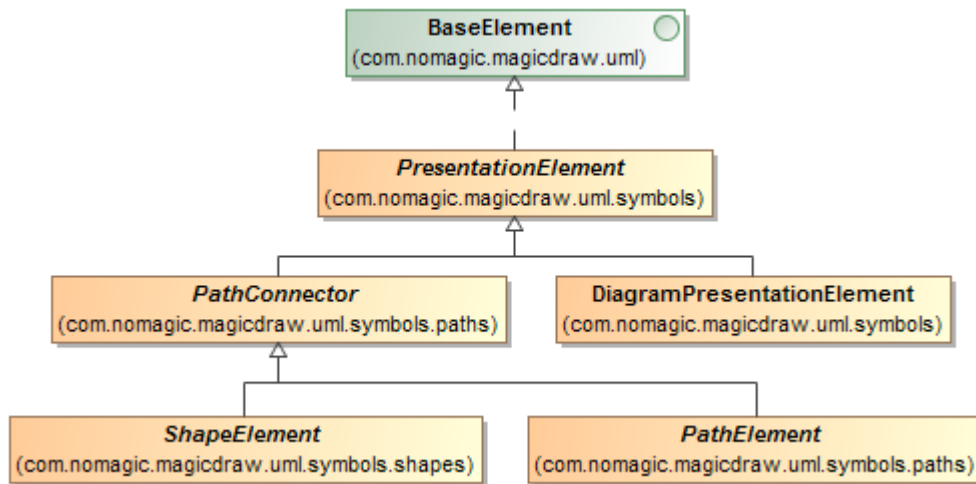
Presentation elements (symbols)

The UML semantic defines only the UML metamodel. The modeling tool 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*. 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* is the *com.nomagic.magicdraw.uml.BaseElement* 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*.

A current version of Open API provides just a basic structure of presentation elements:



Every presentation element can have children. For example, *com.nomagic.magicdraw.uml.symbols.DiagramPresentationElement* has a collection of inner presentation elements. *PresentationElement* 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()*
- Access a model *com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element* of a presentation element *PresentationElement.getElement()*. A presentation element can have no *Element*, for example, a *com.nomagic.magicdraw.uml.symbols.shapes.TextBoxView*.
- Access children *PresentationElement.getPresentationElements()*
- Access properties of the element *PresentationElement.getProperty(java.lang.String)*, *PresentationElement.getPropertyManager()*. 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*.
- Select/unselect or access a selection state of the presentation element (*PresentationElement.isSelected()*, *PresentationElement.setSelected(boolean)*).

A subclass of presentation elements *com.nomagic.magicdraw.uml.symbols.paths.PathConnector* 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()*.

Related pages

- [Diagram presentation element](#) (see page 125)
- [Shapes](#) (see page 125)
- [Paths](#) (see page 125)
- [Working with presentation elements](#) (see page 126)
 - [Creating shape elements](#) (see page 127)
 - [Creating path elements](#) (see page 127)
 - [Reshaping shape elements](#) (see page 128)
 - [Moving symbols on diagram to another location](#) (see page 128)
 - [Moving symbols on diagram to another location and to different parent](#) (see page 129)

- Changing path break points (see page 130)
- Deleting presentation elements (see page 130)
- Changing properties of presentation elements (see page 131)
- Managing compartments (see page 132)
- Collecting presentation elements, finding usages in diagrams (see page 132)
- Displaying related elements in the diagram (see page 134)
- Layouting symbols in the diagram (see page 134)
- Displaying ports (see page 135)
- Custom presentation elements rendering (see page 136)
 - Custom renderer provider (see page 137)
 - Custom symbol renderer (see page 137)
 - Custom renderers sample (see page 137)
 - Registering custom symbol renderer provider (see page 140)

Diagram presentation element

com.nomagic.magicdraw.uml.symbols.DiagramPresentationElement is a one to one mapping to UML *com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram*. It means for every *Diagram* in a model exists one and only one *DiagramPresentationElement*.

Use *com.nomagic.magicdraw.core.Project.getDiagram(Diagram)*³⁶ 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()* method. This method returns only direct children of the diagram.
- To open a diagram in the program UI, call *DiagramPresentationElement.open()*.
- To create/modify/delete inner presentation elements of the diagram, use *com.nomagic.magicdraw.openapi.uml.PresentationElementsManager*.

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:

- **Supplier** – the presentation element of a Relationship supplier. Use method *com.nomagic.magicdraw.uml.symbols.paths.PathElement.getSupplier()* to access this element.
- **Client** – the presentation element of a Relationship client. Use method *PathElement.getClient()* to access this element.
- **Supplier point** – the connection point of a path and supplier element. Use method *PathElement.getSupplierPoint()* to access this point.

³⁶ <http://jdocs.nomagic.com/2024x/com/nomagic/magicdraw/core/Project.html#getDiagram-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram->

- **Client point** – the connection point of a path and client element. Use method *PathElement.getClientPoint()* 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()* to access this list.

Related pages

- [Working with paths and relationships](#)³⁷
- [Displaying paths](#)³⁸

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*.

PresentationElementManager can be used only inside the created session with *com.nomagic.magicdraw.openapi.uml.SessionManager*. If a session is not created, *java.lang.IllegalStateException* is thrown.

PresentationElementManager can be used only in already loaded and active project. In other cases, results can be unpredictable.

PresentationElementManager is implemented as a singleton. Use method *PresentationElementManager.getInstance()* to get a shared instance of this manager.

Related pages

- [Session management](#) (see page 80)
- [Creating shape elements](#) (see page 127)
- [Creating path elements](#) (see page 127)
- [Reshaping shape elements](#) (see page 128)
- [Moving symbols on diagram to another location](#) (see page 128)
- [Moving symbols on diagram to another location and to different parent](#) (see page 129)
- [Changing path break points](#) (see page 130)
- [Deleting presentation elements](#) (see page 130)
- [Changing properties of presentation elements](#) (see page 131)
- [Managing compartments](#) (see page 132)
- [Collecting presentation elements, finding usages in diagrams](#) (see page 132)
- [Displaying related elements in the diagram](#) (see page 134)
- [Layouting symbols in the diagram](#) (see page 134)
- [Displaying ports](#) (see page 135)

³⁷ <https://docs.nomagic.com/display/MD2024xR3/Working+with+paths+and+relationships>

³⁸ <https://docs.nomagic.com/display/MD2024xR3/Displaying+paths>

Creating shape elements

To create a *com.nomagic.magicdraw.uml.symbols.shapes.ShapeElement* for given *ModelElement* in the given *com.nomagic.magicdraw.uml.symbols.DiagramPresentationElement*, 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

- *<installation_directory>\openapi\examples\symbolcreation*
- *<installation_directory>\openapi\examples\sequencecreation*

Related pages

- [Session management \(see page 80\)](#)

Creating path elements

To create a *com.nomagic.magicdraw.uml.symbols.paths.PathElement* for given *ModelLink* between given client and supplier elements, use method *com.nomagic.magicdraw.openapi.uml.PresentationElementsManager.createPathElement(Element, PresentationElement, PresentationElement)*:

```
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(Element, PresentationElement, PresentationElement)* method. A new path element is created in the same diagram as client or supplier presentation elements.

- i** You can find the code examples in
- `<installation_directory>\openapi\examples\symbolcreation`
 - `<installation_directory>\openapi\examples\sequencecreation`

Related pages

- [Session management \(see page 80\)](#)

Reshaping shape elements

To change bounds for the existing `com.nomagic.magicdraw.uml.symbols.shapes.ShapeElement`, use the method `com.nomagic.magicdraw.openapi.uml.PresentationElementsManager.reshapeShapeElement(ShapeElement, java.awt.Rectangle)`:

```
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 \(see page 80\)](#)

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

- *<installation_directory>\openapi\examples\symbolmove*

Related pages

- [Session management \(see page 80\)](#)

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

- *<installation_directory>\openapi\examples\symbolmove*

Related pages

- [Session management](#) (see page 80)

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(PathElement, java.util.List)`. 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 80)

Deleting presentation elements

To remove the given `com.nomagic.magicdraw.uml.symbols.PresentationElement` from the diagram, use the method `com.nomagic.magicdraw.openapi.uml.PresentationElementsManager.deletePresentationElement(PresentationElement)`:

```
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 80)


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()`.
- To get all properties of this element, use the method `PresentationElement.getPropertyManager()`.
- To get a property of this element with a given ID, use the method `com.nomagic.magicdraw.uml.symbols.PresentationElement.getProperty(java.lang.String)`.
- To change element properties, use the method `com.nomagic.magicdraw.openapi.uml.PresentationElementsManager.setPresentationElementProperties(PresentationElement, PropertyManager)`. The given `com.nomagic.magicdraw.properties.PropertyManager` 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`.

Look at [Properties](#) (see page 161) page for more details about properties structure.

 There is no information about what exactly properties every presentation element has. Use java debugger and analyze `PresentationElement.getPropertyManager()` 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(element, properties);
SessionManager.getInstance().closeSession(project);
```

Related pages

- [Properties](#) (see page 161)
- [Session management](#) (see page 80)

Managing compartments

Some presentation elements display 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* for managing the visibility of compartment entries.

More information is available in javadoc.



You can find the code examples in

- `<installation_directory>\openapi\examples\compartmentedit`

Related pages

- [Session management](#) (see page 80)


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 suits your needs.

Keep in mind that diagrams in MagicDraw use 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()`. 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()`


Find element usages in loaded and not loaded diagrams (do not load diagrams)

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 API provides very useful methods for that. This API works both in loaded and not loaded diagrams.

DiagramPresentationElement.getUsedModelElements(boolean) to get all displayed elements in the diagram

Use *DiagramPresentationElement.isElementInDiagram(Element)* 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(Element)* to check if element is in diagram
 - Use *DiagramPresentationElement.ensureLoaded()* to load a diagram
 - Use *Project.getSymbolElementMap()* 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(Element, java.lang.Class)* method

More information is available in javadoc.

Displaying related elements in the diagram

Our modeling tools allow displaying symbols that are related to a given symbol via relationships. The *com.nomagic.magicdraw.uml.symbols.DisplayRelatedSymbols* 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

- `<installation_directory>\openapi\examples\displayrelated`

Related pages

- [Session management](#) (see page 80)

Layouting symbols in the diagram

We provide an API to layout presentation elements in a diagram using different strategies (layouter), see:


- *com.nomagic.magicdraw.uml.symbols.layout.Layouting*


```
1 //layout using the default diagram layouter
2 AbstractDiagramPresentationElement diagramView = ...;
3 diagramView.open();
```

```
4 Layouting.layout(diagramView);
```

```
1 //layout the selected symbols in the diagram using the default diagram
  //layouter
2 AbstractDiagramPresentationElement diagramView = ...;
3 diagramView.open();
4
5 List<PresentationElement> symbolsToLayout = ...;
6 diagramView.setSelected(symbolsToLayout);
7 Layouting.layout(diagramView);
```

```
1 //layout using any of the available layouters
2 AbstractDiagramPresentationElement diagram = ...;
3 diagram.open()
4 Layouting.layout(diagram, Layouting.HIERARCHIC_DIAGRAM_LAYOUTER);
```

 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 more code examples in

- `<installation_directory>\openapi\examples\testframework, com.nomagic.magicdraw.examples.testframework.DiagramLayoutTest` [class](#)

Related pages

- [Session management](#) (see page 80)

Displaying ports

We provide an API to simplify displaying of Ports on Symbols. Use `com.nomagic.magicdraw.uml.symbols.DisplayPorts#displayAllPorts` to display all ports or `com.nomagic.magicdraw.uml.symbols.DisplayPorts#collectAvailablePorts` to get available ports and then `com.nomagic.magicdraw.uml.symbols.DisplayPorts#displayPorts(com.nomagic.magicdraw.uml.symbols.PresentationElement, java.util.List<com.nomagic.uml2.ext.magicdraw.compositestructures.mdports.Port>)` or `com.nomagic.magicdraw.uml.symbols.DisplayPorts#displayPorts(com.nomagic.magicdraw.uml.symbols.PresentationElement, java.util.List<com.nomagic.uml2.ext.magicdraw.compositestructures.mdports.Port>, boolean, boolean, boolean)` to display just some of them.


Example: Displaying unnamed ports

```
SessionManager sessionManager = SessionManager.getInstance();
sessionManager.createSession(project, getName());

PresentationElement view = ...; // A symbol for which you want to display ports

List<Port> ports = DisplayPorts.collectAvailablePorts(view, false, false);
List<Port> unnamedPorts = ports.stream().filter(port ->
port.getName().isEmpty()).toList();
DisplayPorts.displayPorts(view, unnamedPorts);

sessionManager.closeSession(project);
```

-  You can find the code examples in
- `<installation_directory>\openapi\examples\displayports`

Related pages

- [Session management \(see page 80\)](#)

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.

-  You can find the code examples in `<installation_directory>\openapi\examples\symbolrendering`.

Related pages

- [Custom renderer provider \(see page 137\)](#)
- [Custom symbol renderer \(see page 137\)](#)
- [Custom renderers sample \(see page 137\)](#)
- [Registering custom symbol renderer provider \(see page 140\)](#)

Custom renderer provider

A *com.nomagic.magicdraw.uml.symbols.PresentationElementRendererProvider* is an object, which provides a custom symbol renderer for a given presentation element.

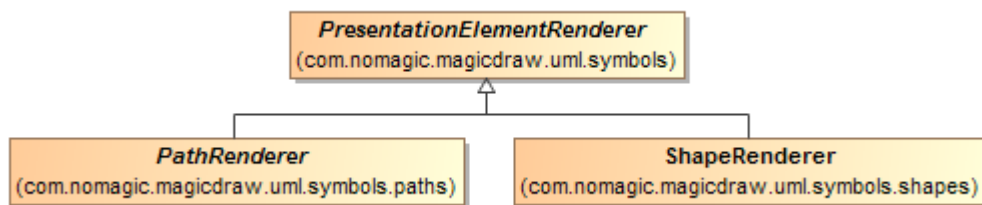
PresentationElementRendererProvider.getRenderer(PresentationElement) method provides a specific renderer for the given *com.nomagic.magicdraw.uml.symbols.PresentationElement* (Symbol) or null.

To make your provider start working, you must register it into the *com.nomagic.magicdraw.uml.symbols.PresentationElementRendererManager* using:

```
PresentationElementRendererManager.getInstance().addProvider(new RendererProvider());
```

Custom symbol renderer

To have a custom element renderer, extend abstract *com.nomagic.magicdraw.uml.symbols.PresentationElementRenderer* class.



Custom renderers sample

This example covers:

- [A custom renderer for the package.](#) (see page 137) The package without inner elements is filled with a green color.
- [A custom renderer for the slot.](#) (see page 138) The slot values are rounded.
- [A custom renderer for the dependency link.](#) (see page 139) 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*.

Step 1. Creating the RendererProvider class

```

/**
 * 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* can access selected elements or change their own selection status.

PresentationElement (com.nomagic.magicdraw.uml.symbols)
<i>operations</i> +setAllSelected(select : boolean) : void +setSelected(elements : List<PresentationElement>) : void +setSelected(select : boolean) : void +getSelected() : List<PresentationElement> +isSelected() : boolean ...

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 \(see page 144\)](#) page describes selection change events

Related pages

- [Presentation element selection events \(see page 144\)](#)

Diagram events

[Presentation elements modification events](#)


[Presentation element selection events](#)

[Notification of presentation element draw](#)

Presentation elements modification events

Our modeling tools provide an API for listening to all diagram changes in a single adapter:

1. The listener receives events from all *opened* diagrams.
2. The adapter delegates these events to your own listener.

 The adapter works in a project scope.

To listen to diagram change events, you need to create the *com.nomagic.magicdraw.uml.symbols.DiagramListenerAdapter* object, that is, to pass the property change listener as a delegator, which receives all events.

To create the *DiagramListenerAdapter* object, call:

```
new DiagramListenerAdapter(propertyChangeListener)
```

The *DiagramListenerAdapter* 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* 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);
```

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 notifies when the presentation element is drawn (added to a diagram):

```
// 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*. Only the opened diagram has *DiagramSurface*. A closed diagram returns *null*.

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);
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

        };
    });
}
}
});
}
});

```

i You can find the code examples in `<installation_directory>\openapi\examples\customdiagrampainter`

Creating diagram images

The Open API of a modeling tool provides the *com.nomagic.magicdraw.export.image.ImageExporter* class for creating images from the whole diagram or just a part of it.

Image formats are predefined as constants in *ImageExporter*.

To create an image from the whole diagram, use the *export(DiagramPresentationElement, int, java.io.File)* method.

To create an image from selected symbols in the diagram, use the *export(DiagramPresentationElement, int, java.io.File, boolean)* method.

i You can find the code examples in `<installation_directory>\openapi\examples\imagegenerator`

Working with 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](https://docs.nomagic.com/display/MD2024xR3/Generic+table)³⁹.

The Open API of modeling tools provides the *com.nomagic.generictable.GenericTableManager* 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.

³⁹ <https://docs.nomagic.com/display/MD2024xR3/Generic+table>

i 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(Project, java.lang.String)* method.

```
Diagram createdDiagram = GenericTableManager.createGenericTable(project, "Generic  
table name");
```

Setting generic table filter

To set a generic table element types, use the *GenericTableManager.setTableElementTypes(Diagram, java.util.List<java.lang.Object>)* 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(Diagram)* method:

```
List<Object> setElementTypes =  
GenericTableManager.getTableElementTypes(createdDiagram);
```

Adding/getting columns

To add columns to the generic table, use the *GenericTableManager.addColumnsByld(Diagram, java.util.List<java.lang.String>)* method.

To get all possible columns from the generic table, use the *GenericTableManager.getPossibleColumnIDs(Element)* method. To get set columns for the generic table, use the *GenericTableManager.getColumnIDs(Diagram)* method.

```
List<String> columnIDs = GenericTableManager.getPossibleColumnIDs(elementType);
```

getColumnIDs(Diagram) returns columns from the generic table:

```
List<String> columnIDs = GenericTableManager.getColumnIDs(createdDiagram);  
GenericTableManager.addColumnById(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(Diagram,⁴⁰Element, java.lang.String)* method:

```
Property mdProperty = GenericTableManager.getCellValue(createdDiagram, element,  
columnId);
```

Getting column names

To get column names from the generic table, use the *GenericTableManager.getColumnNames(Diagram)⁴¹* method.

To get a single column name from the generic table, use the *GenericTableManager.getColumnNameById(Diagram, java.lang.String)* method.

```
List<String> columnNames = GenericTableManager.getColumnNames(createdDiagram);  
String columnName = GenericTableManager.getColumnNameById(createdDiagram, columnID);
```

⁴⁰ <http://jdocs.nomagic.com/2024x/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->

⁴¹ <http://jdocs.nomagic.com/2024x/com/nomagic/generictable/GenericTableManager.html#getColumnNames-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram->

Getting row elements

To get row elements from the generic table, use the *GenericTableManager.getRowElements(Diagram)* method:

```
List<Element> rows = GenericTableManager.getRowElements(createdDiagram);
```

Getting visible row elements

To get visible row elements from the generic table, use the *GenericTableManager.getVisibleRowElements(Diagram)* method:

```
List<Element> rows = GenericTableManager.getVisibleRowElements(createdDiagram);
```

Adding row element

To add a new row for the model element to the generic table, use the *GenericTableManager.addRowElement(Diagram,⁴²Element)⁴³* 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(Diagram, Element)* method:

```
GenericTableManager.removeRowElement(createdDiagram, element);
```

Setting table scope

To set a scope for the generic table, use the *GenericTableManager.removeRowElement(Diagram, Element)* method:

```
GenericTableManager.removeRowElement(createdDiagram, element);
```

⁴²<http://jdocs.nomagic.com/2024x/com/nomagic/generictable/GenericTableManager.html#addRowElement-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element->

⁴³<http://jdocs.nomagic.com/2024x/com/nomagic/generictable/GenericTableManager.html#addRowElement-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram-com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element->

Refreshing table

To refresh the table, use the `GenericTableManager.refreshTable(Diagram)` method:

```
GenericTableManager.refreshTable(diagram)
```

On this page

- [Creating generic table](#) (see page 147)
- [Setting generic table filter](#) (see page 147)
- [Getting generic table element types](#) (see page 147)
- [Adding/getting columns](#) (see page 147)
- [Getting cell value](#) (see page 148)
- [Getting column names](#) (see page 148)
- [Getting row elements](#) (see page 149)
- [Adding row element](#) (see page 149)
- [Removing row element](#) (see page 149)

Related pages

- [Session management](#) (see page 80)

Working with Dependency Matrix

[Dependency Matrix](#)⁴⁴ is a special type of diagram, that is, a diagram with a specific stereotype applied. Our modeling tools 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

- `<installation_directory>\openapi\examples\dependencymatrix`

⁴⁴ <https://docs.nomagic.com/display/MD2024xR3/Dependency+Matrix>

Related pages

- [Creating a custom dependency matrix type \(see page 151\)](#)
- [Customizing a dependency matrix \(see page 153\)](#)
 - [Configuring the command bar and shortcut keys \(see page 155\)](#)
 - [Configuring shortcut menus of a row or column header element and a filter panel \(see page 155\)](#)
 - [Configuring cell entries - dependencies \(see page 156\)](#)
 - [Specifying the appearance of cells and the row or column headers \(see page 157\)](#)
- [Accessing dependency matrix data \(see page 158\)](#)

Creating a custom dependency matrix type

To create a custom dependency matrix type:

1. [Create a new plugin to register components of the new dependency matrix type. \(see page 151\)](#)
2. [Register the new dependency matrix type that it was added to modeling tool toolbars and menus \(see page 152\).](#)

Creating a plugin for custom dependency matrix type

Use the *com.nomagic.magicdraw.dependencymatrix.configuration.DependencyMatrixConfigurator* 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* 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*.

Registering a new dependency matrix type in UI

Once the [plugin](#) (see [page 11](#)) 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* class as the argument.

Customize Dependency Matrix Wizard

Specify basic diagram type information
In order to create or edit dependency matrix type specify the following: its type, abbreviation, category to which new matrix will belong in main menu and toolbar, custom matrix icons.

☒ **1. Specify Dependency Matrix type and icon**
☐ **2. Specify modules**
☐ **3. Specify Dependency Matrix properties**

Type:
Abbreviation:
Category:

Icon

Browser	Toolbar	SVG
<input data-bbox="917 1579 997 1624" type="button" value="..."/>	<input data-bbox="1077 1579 1157 1624" type="button" value="..."/>	<input data-bbox="1236 1579 1316 1624" type="button" value="..."/>
<input data-bbox="893 1646 1021 1680" type="button" value="Remove"/>	<input data-bbox="1053 1646 1181 1680" type="button" value="Remove"/>	<input data-bbox="1212 1646 1340 1680" type="button" value="Remove"/>


"Browser" icon is for Browser and Menu (16 x 16).
"Toolbar" icon is for Toolbars buttons (16 x 16).
"SVG" is for drawing.

< Back Next > Finish Cancel Help

Specifying name of new Dependency Matrix type

5. In the **Abbreviation** box, specify the abbreviation of the new dependency matrix type.
6. Click **Finish**.
7. Restart your modeling tool.

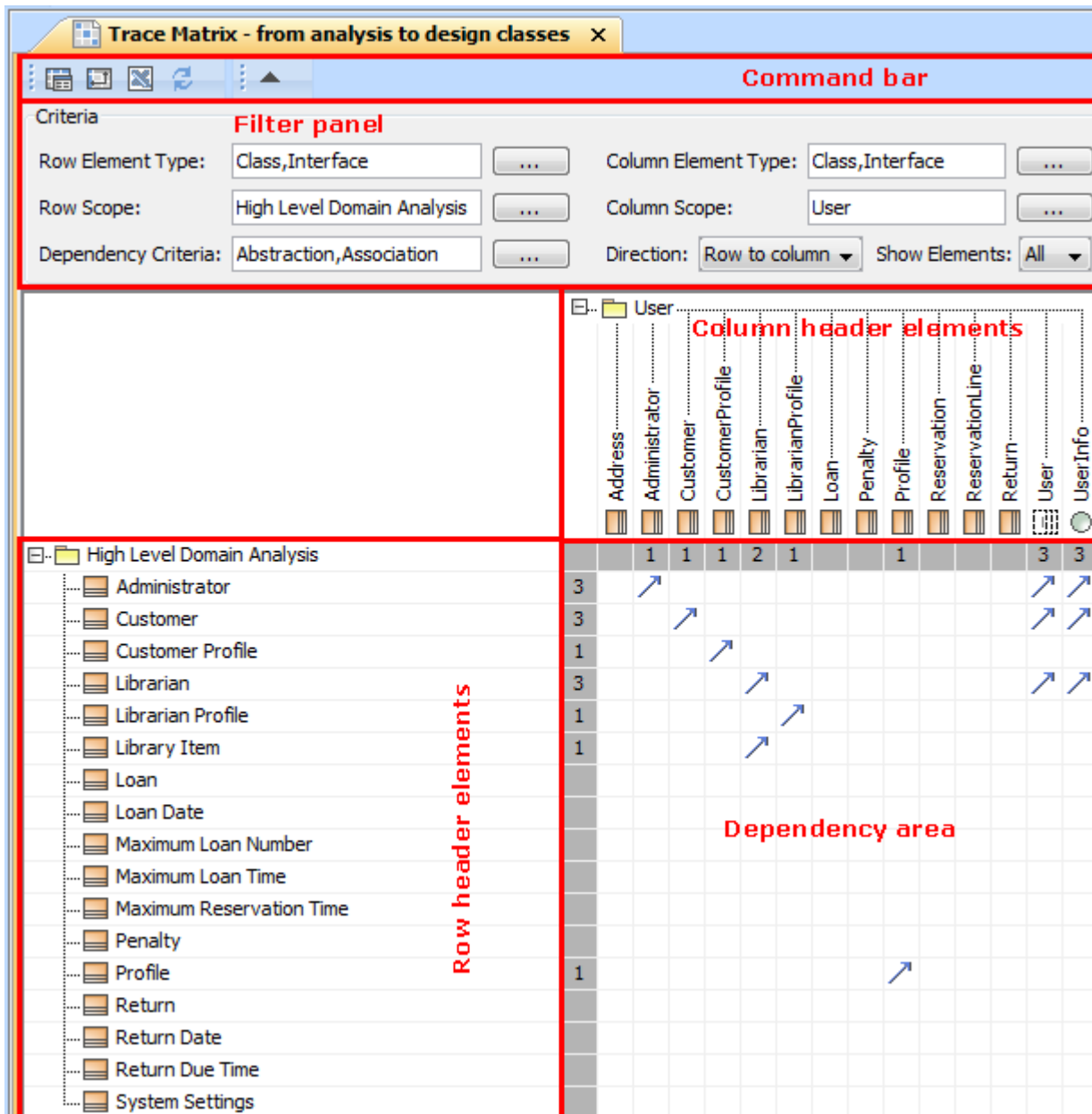
You can now create a custom dependency matrix.

 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



Components of dependency matrix pane

- ⚠** 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 \(see page 155\)](#)

- [Configuring shortcut menus of a row or column header element and a filter panel \(see page 155\)](#)
- [Configuring cell entries - dependencies \(see page 156\)](#)
- [Specifying the appearance of cells and the row or column headers \(see page 157\)](#)

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* interface to the *com.nomagic.magicdraw.actions.ActionsConfiguratorsManager* 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* constant. And when customizing a custom dependency matrix type, provide the name defined in the instance of the *DependencyMatrixConfigurator* class as well as in the **Customize Dependency Matrix Wizard** dialog.

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* class.

The *com.nomagic.magicdraw.dependencymatrix.configuration.DependencyMatrixAMConfigurator* 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* 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](#)⁴⁵ 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<DependencyExtractor>, java.util.Collection<DependencyEditor>) 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)*. 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* 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* 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.configureDependencyEditors(java.util.Collection<DependencyExtractor>, java.util.Collection<DependencyEditor>) 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.

⁴⁵ <https://docs.nomagic.com/display/MD2024xR3/Specification+window>

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.

When adding a new command, the *DependencyEditor.canCreate(PersistenceManager, ElementNode, ElementNode, AbstractMatrixCell)* method is first called and if it returns *true*, the *DependencyEditor.createAddActions(PersistenceManager, ElementNode, ElementNode, AbstractMatrixCell, ActionsCategory, ActionsCategory, ActionsCategory)* method is called to add the command.

When editing a command, the *DependencyEditor.canEdit(PersistenceManager, ElementNode, ElementNode, AbstractMatrixCell)* method is first called and if it returns *true*, the *DependencyEditor.createEditActions(PersistenceManager,⁴⁶ElementNode,⁴⁷ElementNode,⁴⁸AbstractMatrixCell,⁴⁹ActionsCategory,⁵⁰ActionsCategory)⁵¹* 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* 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* class. The *java.lang.Integer* class

⁴⁶<http://jdocs.nomagic.com/2024x/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->

⁴⁷<http://jdocs.nomagic.com/2024x/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->

⁴⁸<http://jdocs.nomagic.com/2024x/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->

⁴⁹<http://jdocs.nomagic.com/2024x/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->

⁵⁰<http://jdocs.nomagic.com/2024x/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->

⁵¹<http://jdocs.nomagic.com/2024x/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->

represents dependency count. And the *MatrixCellView* class has a reference to the *com.nomagic.magicdraw.dependencymatrix.datamodel.cell.AbstractMatrixCell* 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)* 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* 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* 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* implementation and the size of the matrix.

Working with Relation Map

A relation map is a diagram with a specific stereotype. It is used to represent the element properties in a relation based form. It can represent elements connected via various relation criteria

The Open API of our modeling tools provides the *com.nomagic.magicdraw.visualization.relationshipmap.RelationshipMapUtilities* to work with Relation maps.

Refreshing relation map

To refresh the relation map, use the `RelationshipMapUtilities.refreshMap(Diagram)` method:

```
RelationshipMapUtilities.refreshMap(diagram)
```

Working with Gantt Chart

Refreshing Gantt Chart

To refresh the gantt chart, use the `GanttChartUtilities.refreshChart(diagram)` method:

```
GanttChartUtilities.refreshChart(Diagram)
```

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](#) (see page 159)
- [Managing hyperlinks](#) (see page 159)

Customizing double-click behavior

The default application behavior on a double-click on the element is to open the [Specification window](#)⁵² or open a diagram. This behavior can be changed by registering custom element activators.

Use `com.nomagic.magicdraw.ui.ElementActivationManager.addActivator(ElementActivator)`⁵³ 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.

⁵² <https://docs.nomagic.com/display/MD2024xR3/Specification+window>

⁵³ <http://jdocs.nomagic.com/2024x/com/nomagic/magicdraw/ui/ElementActivationManager.html#addActivator-com.nomagic.magicdraw.ui.ElementActivator->

See the following pages for more information:

- [Editing hyperlinks](#) (see page 160)
- [MDEL hyperlink](#) (see page 160)
- [Registering custom hyperlink handler](#) (see page 160)

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 106) for more details.

Recommended way is to use utility class *com.nomagic.magicdraw.hyperlinks.HyperlinkUtils*. This class provides utility methods for adding/removing/retrieving *com.nomagic.magicdraw.hyperlinks.Hyperlink* from model Element,

Related pages

- [Working with stereotypes and tagged values](#) (see page 106)

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* 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(HyperlinkHandler)*⁵⁴ to add a custom handler.

⁵⁴ <http://jdocs.nomagic.com/2024x/com/nomagic/magicdraw/hyperlinks/HyperlinksHandlersRegistry.html#addHandler-com.nomagic.magicdraw.hyperlinks.HyperlinkHandler->

-  You can find the code examples in
- `<installation_directory>\openapi\examples\customhyperlink`

Properties

Our modeling tools include the Open API that 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* is *java.lang.Boolean*, the value of a *com.nomagic.magicdraw.properties.StringProperty* is *java.lang.String*.

The property ID identifies the specific property in the properties set.

You must set some *com.nomagic.magicdraw.properties.PropertyResourceProvider* 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* has just one method *getString(java.lang.String, Property)*⁵⁵, where a key is an ID of your property.

The collection of properties is grouped by *com.nomagic.magicdraw.properties.PropertyManager*. Every *PropertyManager* 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.

Related pages

- [Changing properties of presentation elements \(see page 131\)](#)
- [Project options \(see page 74\)](#)

Information logging

Our modeling tools, including MagicDraw, use the Log4j logger. The configuration file is `<program installation directory>/data/log4j2.xml`.

-  The `<program installation directory>/data/log4j2-test.xml` 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.

⁵⁵ <http://jdocs.nomagic.com/2024x/com/nomagic/magicdraw/properties/PropertyResourceProvider.html#getString-java.lang.String-com.nomagic.magicdraw.properties.Property->

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="warn" monitorInterval="30">
  <Appenders>
    <Console name="Console" target="SYSTEM_OUT">
      <PatternLayout pattern="%d [%t] [%X{id} %X{action_name}] %-5p %c - %m%n"/>
    </Console>
  </Appenders>
  <Loggers>
    <Root level="info">
      <AppenderRef ref="Console"/>
    </Root>
  </Loggers>
</Configuration>
```

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");

// set thread context values (optional)
ThreadContext.put("id", "My id");
ThreadContext.put("action_name", "My Action");

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] [My id My Action] INFO GENERAL - Logging info
//0000-00-00 00:00:00,000 [THREAD] [My id My Action] ERROR GENERAL - Logging error
```

More information about Log4j can be found at <https://logging.apache.org/log4j/2.x/>.

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\l.magicdraw\<version number>\data*. You can find this file for other modeling tools using the same location pattern.

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* 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

- *<installation_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
- 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*;
- Call *com.nomagic.magicdraw.ui.browser.Tree#openNode(com.nomagic.magicdraw.uml.BaseElement)*.

Related pages

- [Adding custom panel or tree into browser](#) (see page 164)
- [Configuring node text and icon](#) (see page 165)

Adding custom panel or tree into browser

Use *com.nomagic.magicdraw.ui.browser.Browser.addBrowserInitializer(Browser.BrowserInitializer)*⁵⁶ to register a custom browser initializer and add your custom panel or tree into a browser.



You can find the code examples in

- `<installation_directory>\openapi\examples\browserpanel`

⁵⁶ <http://jdocs.nomagic.com/2024x/com/nomagic/magicdraw/ui/browser/Browser.html#addBrowserInitializer-com.nomagic.magicdraw.ui.browser.Browser.BrowserInitializer->

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 98). 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(IconAdornment)` to register a node icon customizer.

Use `TreeNodeAdornmentManager.addTextAdornment(TextAdornment)`⁵⁷ to register a node text customizer.

Related pages

- [Creating textual element representation](#) (see page 98)

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](#) (see page 165)
- [Showing Element selection dialog](#) (see page 166)
- [Showing question, error, warning dialogs](#) (see page 166)
- [Showing notifications, adding text into Message Window](#) (see page 167)
- [Creating PopupMenu or other menu](#) (see page 168)
- [Adding custom project window](#) (see page 169)

Configuring element Specification window

The Open API provides a way to configure the elements' [Specification windows](#)⁵⁸. 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](#)⁵⁹.


The `com.nomagic.magicdraw.ui.dialogs.specifications.SpecificationDialogManager` class should be used for registering a specification dialog configurator `com.nomagic.magicdraw.ui.dialogs.specifications.configurator.ISpecificationNodeConfigurator`.

⁵⁷ <http://jdocs.nomagic.com/2024x/com/nomagic/magicdraw/ui/browser/TreeNodeAdornmentManager.html#addTextAdornment-com.nomagic.magicdraw.ui.browser.TextAdornment->

⁵⁸ <https://docs.nomagic.com/display/MD2024xR3/Specification+window>

⁵⁹ <https://docs.nomagic.com/display/MD2024xR3/Specification+window>

More information is available in javadoc.

 You can find the code examples in `<installation_directory>/openapi/examples/elementspecification`

Showing Element selection 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 `<installation_directory>/openapi/examples/elementselection`

Showing question, error, warning dialogs

Use `com.nomagic.magicdraw.core.GUILog` class to show various simple UI dialogs.

Error dialog

Use `GUILog.showError(java.lang.String)`

Use `GUILog.showError(java.awt.Frame, java.lang.String, java.lang.Throwable)` to show error dialog with exception stack trace

Warning dialog

Use `GUILog.showWarning(java.lang.String)`

Question dialog

Use `GUILog.showQuestion(java.lang.String)`

Message dialog

Use `GUILog.showMessage(java.lang.String)`

Use `GUILog.showHTMLMessage(java.lang.String, GUILog.URLActionHandler)` to show html message with some active links

Log messages into Message Window

Use `GUILog.log(java.lang.String)` to add some information into a Message Window

See [Showing notifications, adding text into Message Window](#) (see page 167) for more information about showing Notifications and appending text into a Message Window

For more information about other API choices see javadoc.

Related pages

- [Showing notifications, adding text into Message Window](#) (see page 167)

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` interface.

Use `com.nomagic.magicdraw.ui.notification.NotificationManage`.

Use `com.nomagic.magicdraw.core.GUILog.log(java.lang.String)` to add some information into a Message Window,

`com.nomagic.magicdraw.ui.notification.Notification.getContext()` controls where notification is logged in the Message Window - into Environment or Project tab.

GUILog

Use `GUILog.log(java.lang.String)` to put a text into Project Message Window message.

Use `GUILog.log(Notification, boolean)` to log information from notification into a Message Window.

Use `GUILog.openLog()` to open Message Window.

For more details about every specific kind of a property, see javadoc.



You can find the code examples in `<installation_directory>/openapi/examples/notifications`

Related pages

- [Showing question, error, warning dialogs \(see page 166\)](#)

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()` to get an instance of a creator.

Use `com.nomagic.actions.ActionsMenuCreator.createPopupMenu(ActionsManager)` to create a popup menu.

Steps to create a popup menu would be

1. Create an instance of `com.nomagic.actions.ActionsManager`.
2. Add `com.nomagic.actions.NMAction` actions and `com.nomagic.actions.ActionsCategory` categories into `ActionsManager`.
3. Call `ActionsMenuCreator.createPopupMenu(ActionsManager)` to create a popup for your actions.

Related pages

- [Creating new actions \(see page 54\)](#)

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.

- Use `com.nomagic.magicdraw.ui.ProjectWindowsManager.ConfiguratorRegistry.addConfigurator(ProjectWindowsConfigurator)` to register project window configurator.
- Implement `com.nomagic.magicdraw.ui.ProjectWindowsConfigurator` and add window using `ProjectWindowsManager.addWindow(ProjectWindow)`⁶⁰.

 You can find the code examples in `<installation_directory>/openapi/examples/projectwindow`


Annotating the elements

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`.

 Do not forget to call `AnnotationManager.update()`, when you are done with adding or removing in order to refresh UI.

More information is available in javadoc.

 You can find the code examples in

- `<installation_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.

⁶⁰ <http://jdocs.nomagic.com/2024x/com/nomagic/magicdraw/ui/ProjectWindowsManager.html#addWindow-com.nomagic.magicdraw.ui.ProjectWindow->

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.

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.

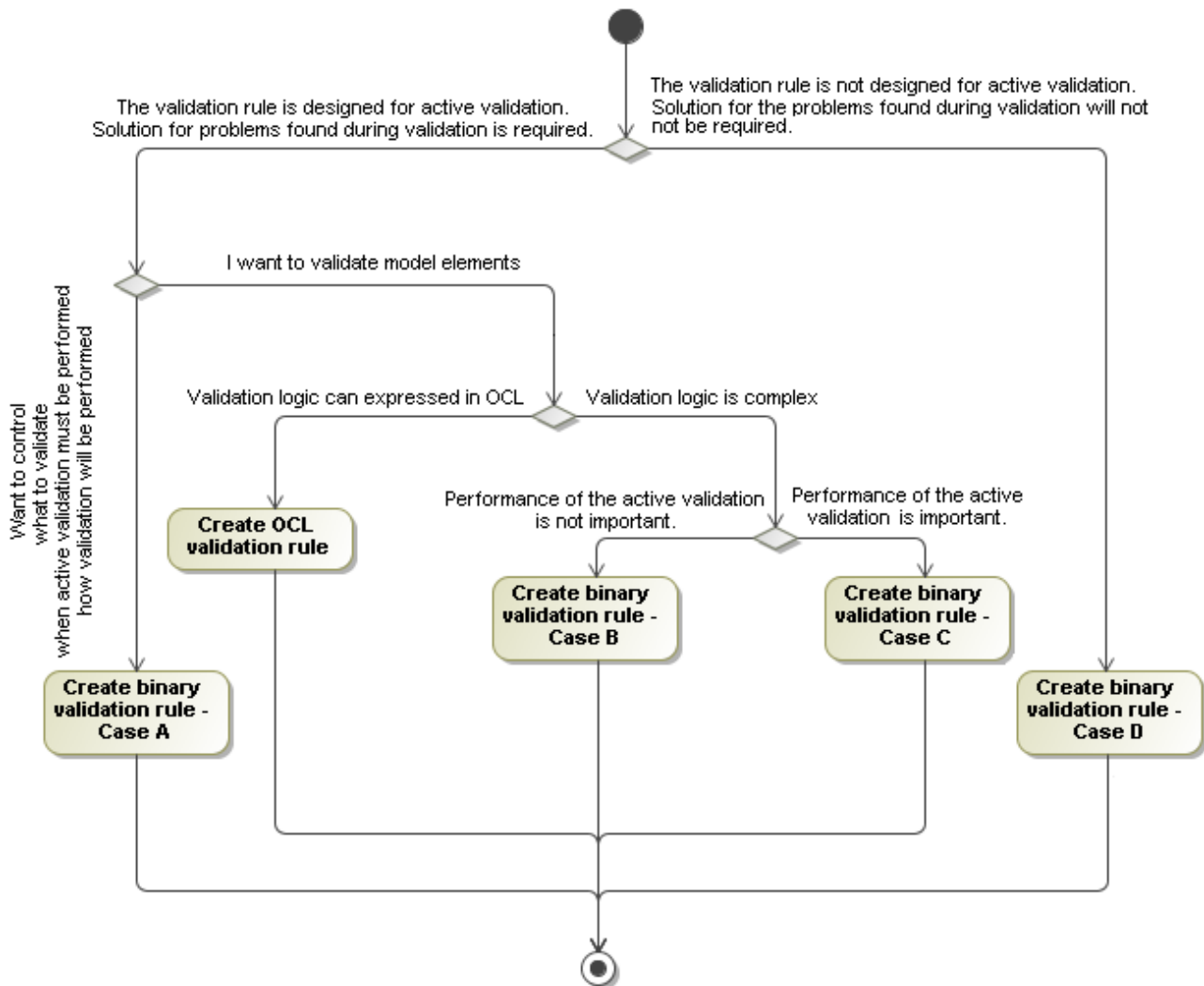
An **annotation** defines the validation result. It contains information about what is incorrect, severity of the problem, and possible actions that can be used to solve the problem.

Related pages

- [Annotating the elements \(see page 169\)](#)
- [Validation rule developer's roadmap \(see page 170\)](#)
- [Create OCL2.0 validation rule \(see page 171\)](#)
- [Binary validation rule \(see page 173\)](#)
- [Implementing a binary validation rule \(see page 174\)](#)
- [Binary validation rule in the plugin \(see page 178\)](#)
- [Script validation rule \(see page 179\)](#)
- [How to provide a solution for a problem found during validation? \(see page 179\)](#)

Validation rule developer's roadmap

Validation rule developer's roadmap allows faster understanding of steps required for creating a validation rule.



Related pages

- [Creating validation rules](#)
(see page 169)

Create OCL2.0 validation rule

The OCL2.0 validation rule describes the validation logic using OCL2.0.

To create the OCL2.0 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 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 \(see page 169\)](#)

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 OCL2.0 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* class
8. Override the method *DefaultValidationRuleImpl.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 \(see page 178\)](#)).

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 \(see page 174\)](#))
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 \(see page 178\)](#)).

Related pages

- [Creating validation rules \(see page 169\)](#)

Implementing a binary validation rule

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`. 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 169)

Implementing Rule to Validate Model Elements

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` 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<NMAAction> 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 to implement *ElementValidationRuleImpl* and *com.nomagic.uml2.ext.jmi.smartlistener.SmartListenerConfigProvider* interfaces. 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.

The *SmartListenerConfigProvider* 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* class to collection(s) of *com.nomagic.uml2.ext.jmi.smartlistener.SmartListenerConfig* objects. Another way would be to extend the *com.nomagic.magicdraw.validation.DefaultValidationRuleImpl* 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.class,
configs);
    return configMap;
}
```

 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* 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* 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* should be implemented. This rule can be 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()* method to get diagrams that are in the currently validated scope. To annotate individual symbols, create *com.nomagic.magicdraw.annotation.Annotation* object with the target being the *com.nomagic.magicdraw.uml.symbols.PresentationElement* which does not pass your validation conditions.

 Extend a subclass *com.nomagic.magicdraw.validation.OpenedDiagramValidator* instead for your rule to always restrict the validated diagrams to the currently opened ones only.

Also see: [Presentation elements \(symbols\)](#) (see page 123)

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 169)

Script validation rule

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](#)⁶¹.
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 (Groovy, Javascript, or Jython).
6. Enter the validation code in the selected language.

Related pages

- [Validation](#)⁶²
- [Creating validation rules](#) (see page 169)

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* class and implement the *actionPerformed(java.awt.event.ActionEvent)* 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* 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 169)

⁶¹ <https://docs.nomagic.com/display/MD2024xR3/Working+with+Constraints>

⁶² <https://docs.nomagic.com/display/MD2024xR3/Validation>

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 174). Each *com.nomagic.magicdraw.annotation.Annotation* 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](#)⁶³

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 181) 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 182) 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](#)⁶⁴).

These classes have to be specified with the complete package and class name, for example, *com.nomagic.magicdraw.autoid.sample.SquareNumberPart*.

Related pages

⁶³ <https://docs.nomagic.com/display/MD2024xR3/Legends>

⁶⁴ <https://docs.nomagic.com/display/MD2024xR3/Numbering+Scheme+property+values>

- [IJavaNumberPart interface](#) (see page 181)
- [INumberingAction interface](#) (see page 182)

JavaNumberPart 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` 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` 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.

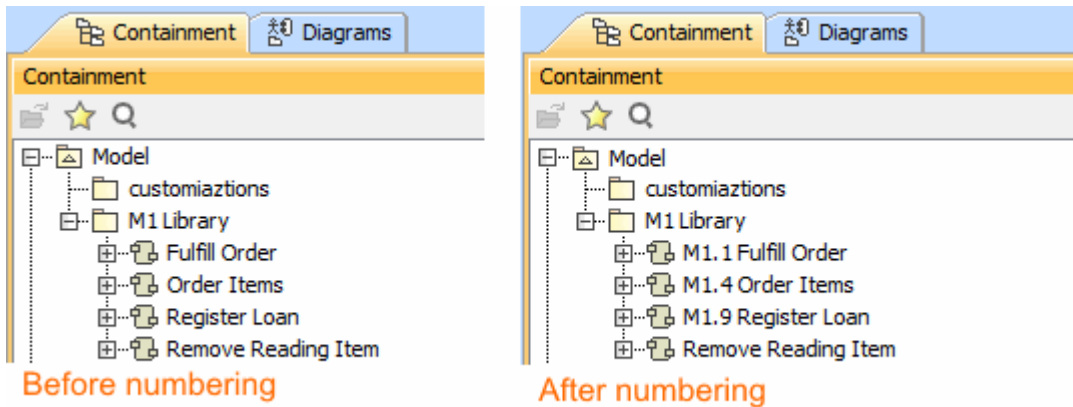
The return value of this method will be incorporated by the Numbering Framework to create a complete number together with all other *NumberParts* that are part of the same *NumberingScheme*.

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.

«NumberingScheme»
SquareScheme
«NumberPart»-part1(sequence = OwnerNumber)
«NumberPart»-part2(initialValue = ".", sequence = Separator)
«NumberPart»-part3(expression = "squaring;Binary;com.nomagic.magicdraw.autoid.SquareNumberPart;", sequence = 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.



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 $nextSquare = (\sqrt{square} + 1)^2$.

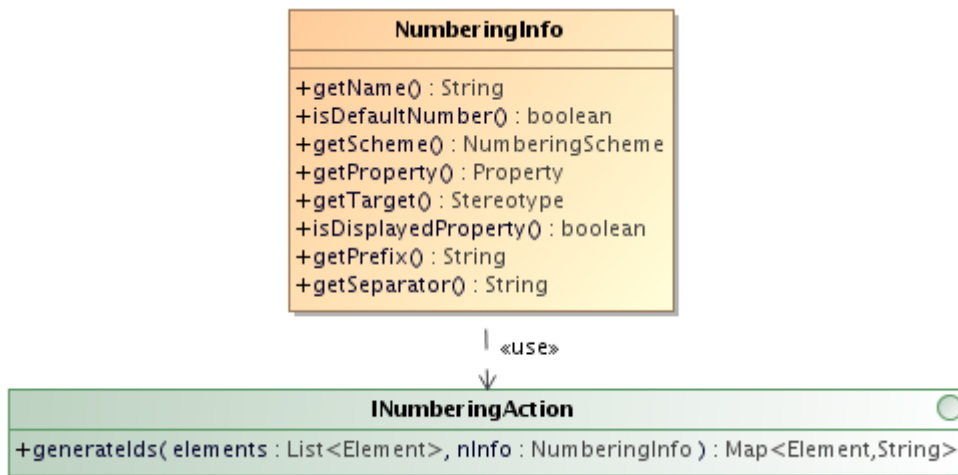
```
public class SquareNumberPart implements IJavaNumberPart
{
    private final int initialValue = 1;
    @Override
    public String generateNextId(String lastNumberPart)
    {
        int nextSquare = initialValue;
        try
        {
            if ("".equals(lastNumberPart))
            {
                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` interface, it is possible to create the numbering

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* 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* support object. This is an immutable Java Bean object which provides the implementer of the *INumberingAction* interface with all the information as defined in the *AutoNumber* and *NumberingScheme* objects:

- *getName()* returns the name of the *AutoNumber*.
- *isDefaultNumber()* returns the *AutoNumber.defaultNumber* value.
- *getScheme()* returns the *AutoNumber.numberingScheme* value, which is the *NumberingScheme* used to number this type of element.
- *getProperty()* returns the *AutoNumber.numberedProperty* value, which is the *Property* that will hold the value of the calculated number.
- *getTarget()* returns the *AutoNumber.getTarget()* value, which is the type of the elements to be numbered.
- *isDisplayedProperty()* returns true, if the value of the above *getProperty()* 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()* 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()* 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).

Filter Elements by Element Type : «» Stereotype packID

Separator	Prefix	Numbering Scheme
-	p	PackScheme

6 Modified Separator and Prefix values (fragment of Element Numbering dialog)

For more information on how to create Numbering customizations with «AutoNumber», «NumberingScheme», and «NumberPart» elements, see [Working with Generic Numbering Mechanism](#)⁶⁵.

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*.

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* interface, it should be set as the expression in the only *NumberPart* of this *NumberingScheme*.

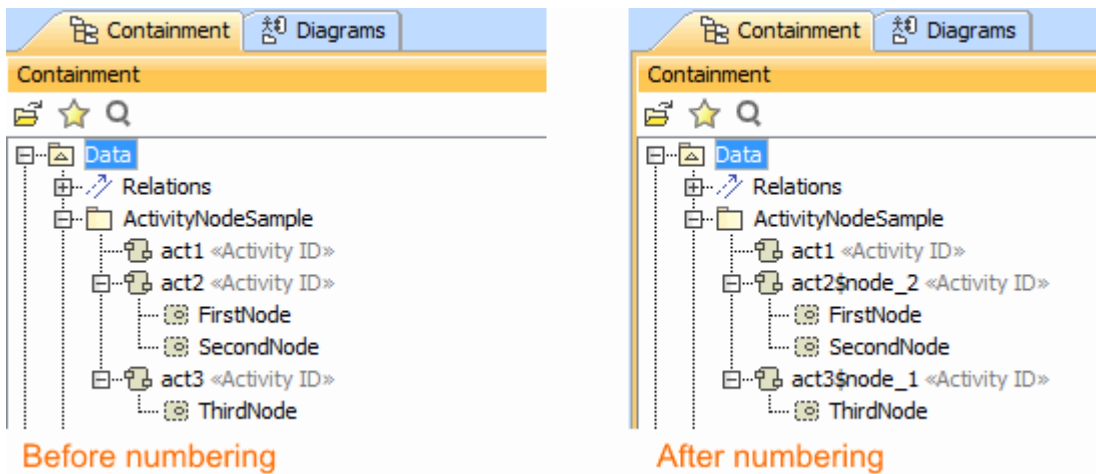


7 ActivityScheme numbering scheme (fragment)

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* children, we have to open the **Element Numbering dialog**⁶⁶ and renumber the Activities. This will generate new numbers that reflect the ownership relation.

⁶⁵ <https://docs.nomagic.com/display/MD2024xR3/Working+with+generic+numbering+mechanism>

⁶⁶ <https://docs.nomagic.com/display/MD2024xR3/Element+Numbering+dialog>



8 Activities and Structural Nodes before and after renumbering using Element Numbering dialog

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

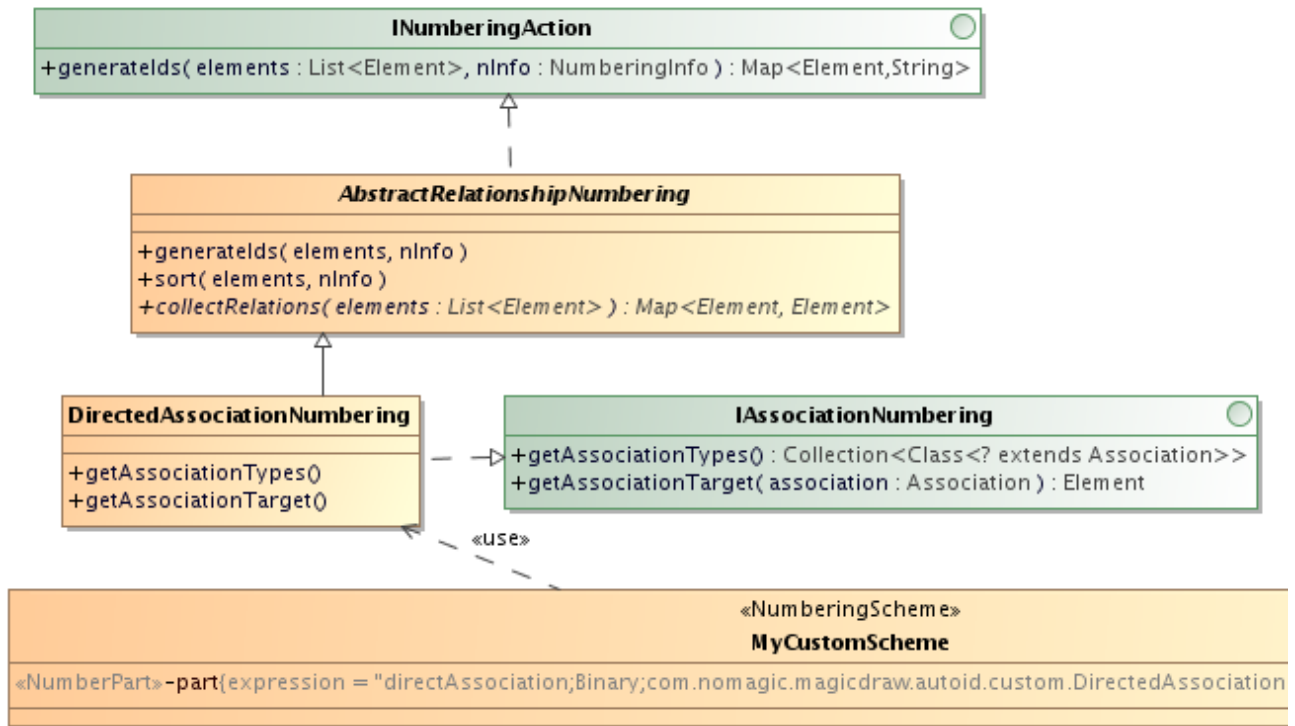
                if (nodes != null && !nodes.isEmpty()) {
                    id += "$node_" + nodes.size();
                }
                idMap.put(e, id);
            }
        }
    }
    // returning the mapping of elements and ids, so that the framework can
register them
    return idMap;
}
}

```

Example #2

A more complicated example can be found in 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* 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)

Creating Use Case Scenario

Modeling tools provide an API for creating and manipulating a [Use Case Scenario](#)⁶⁷ for a use case:

- The `com.nomagic.magicdraw.usecasescenarios.scenarios.Scenario` class represents a Use Case Scenario. It contains methods for manipulating it.
- The `com.nomagic.magicdraw.usecasescenarios.scenarios.ScenarioFactory` class is used for creating use case scenarios.
- The `com.nomagic.magicdraw.usecasescenarios.scenarios.ScenarioManager` 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.

```

SessionManager.getInstance().createSession(...);
// Creates a use case scenario.
Scenario scenario = ScenarioManager.createScenario(useCase);
// Sets the scenario name.
scenario.setName("Extract money from ATM.");
  
```

⁶⁷ <https://docs.nomagic.com/display/MD2024xR3/Use+Case+Scenario+sketch>

```

// 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.
SceneManager.displayScenario(scenario, true, true, "Open ATM Scenario");

```

More information is available in javadoc.



You can find the code examples in

- `<installation_directory>\openapi\examples\scenario`

For more information about the use case scenario, see Use Case Scenario in MagicDraw UserManual.pdf.

Merging and differencing

The modeling tools provide APIs for project merging and differencing. There are two separate APIs:

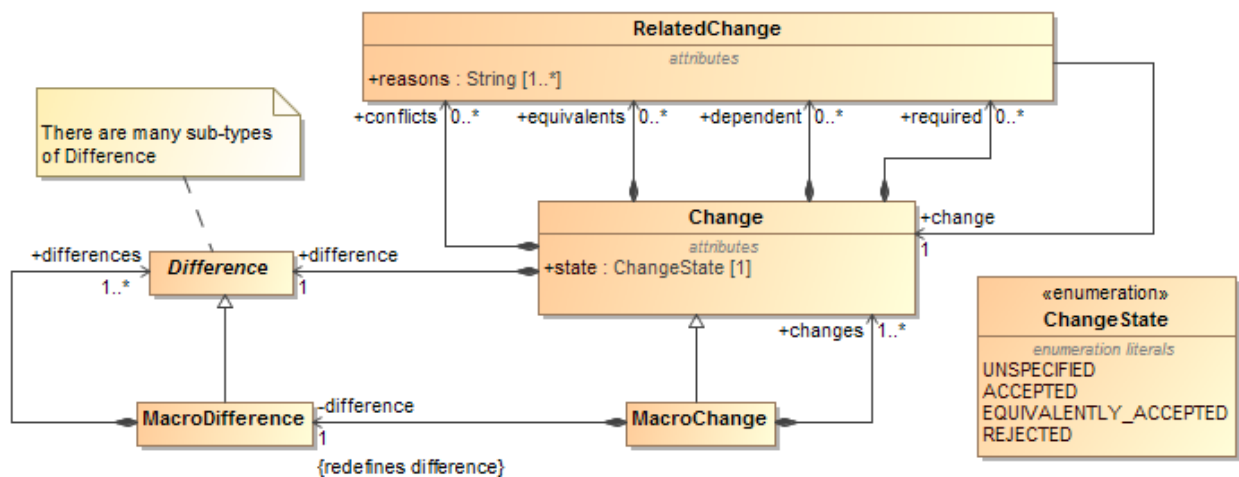
- **Merge** - allows for project merging. The Merge API comes together with the Project Merge plugin, which requires a license. You can access the Merge API javadoc at `<installation_directory>\openapi\docs\merge_javadoc.zip` once you acquire and install the Project Merge plugin.
- **Compare** - allows for project differences comparison. The Compare API comes together with the internal Merge Core plugin, which is available pre-installed on the modeling tool. You can access the Compare API javadoc at `<installation_directory>\openapi\docs\javadoc.zip` once you have the modeling tool installed.

Project merging consists of three steps:

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 APIs for 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 procedure.



You can find the code examples:

- for Merge API (requires the Project Merge plugin to be installed first):
`<installation_directory>\openapi\examples\merge`
- for Compare API: `<installation_directory>\openapi\examples\compare`

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.

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.



You can find the code examples in

- `<installation_directory>\openapi\examples\codeengineering`

Related pages

- [Code engineering set \(see page 190\)](#)
- [Managing code engineering sets \(see page 191\)](#)
- [Samples of the forward and reverse engineering \(see page 192\)](#)

Code engineering set

A *com.nomagic.magicdraw.ce.CodeEngineeringSet* object represents a code-engineering object as a resource for forward or reverse engineering.

Each *CodeEngineeringSet* 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* object. Use the *CodeEngineeringSet.addElementsToCodeEngineeringSet(java.util.List<BaseElement>)*⁶⁸ 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

Source code files are required to perform the Reverse Engineering. There are 2 methods available for adding files into the code engineering set:

⁶⁸ <http://jdocs.nomagic.com/2024x/com/nomagic/magicdraw/ce/CodeEngineeringSet.html#addElementsToCodeEngineeringSet-java.util.List->

1. `CodeEngineeringSet.addFilesToCodeEngineeringSet(java.util.List<java.io.File>)`; This method adds the given list of files to the code engineering set.
2. `CodeEngineeringSet.addAllFilesRecursivelyToCES(java.io.File)`; 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` class provides static API methods for creating, removing, reversing, generating, and adjusting code-engineering settings. The `CodeEngineeringManager` 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` class.
- The name of the [code engineering set](#) (see page 190).
- The project, where code engineering set will be added.
- The working package.
- The working directory.

New `com.nomagic.magicdraw.ce.CodeEngineeringSet` can be created directly via `CodeEngineeringManager` using the `CodeEngineeringManager.createCodeEngineeringSet(java.lang.String, java.lang.String, java.lang.String, Project, Package, java.lang.String)` method.

The language and dialect can be selected from the `CodeEngineeringConstants` class. The dialect is required for DDL and C++ languages.

-  • 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` use the `CodeEngineeringManager.removeCodeEngineeringSet(CodeEngineeringSet)` method.

All Code Engineering Sets can be retrieved by the `CodeEngineeringManager.getAllCodeEngineeringSets(Project)` method.


To perform generation of the `CodeEngineeringSet` object use the `CodeEngineeringManager.generate(Project, boolean)` method.

To perform the reverse engineering use the `CodeEngineeringManager.reverse(Project, boolean)` method.

Language Specific Options

`com.nomagic.magicdraw.ce.JavaCodeEngineeringManager` provides methods to set Java specific options. These options apply to all code engineering sets in the project.

To add a classpaths for the java code engineering sets, use
JavaCodeEngineeringManager.setJavaClasspath(Project, java.lang.String[])

 Setting a new classpath will override the old one.

To get applied classpaths for java code engineering sets, use
JavaCodeEngineeringManager.getJavaClasspath(Project).

To resolve collection generics when reversing the java code,
CodeEngineeringManager.setResolveCollectionGenerics(Project, boolean)

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*

```
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 explain how the Oracle DDL generation works in our modeling tools and how to change the template for specific elements.

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/sqlangu.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 merges 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 \(see page 194\)](#)
- [Customizing templates \(see page 195\)](#)
- [Utility class \(see page 196\)](#)
- [Example \(see page 206\)](#)

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, \$QUERY_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 installation 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 \(see page 190\)](#) 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<NamedElement>	The list of model elements that are added to the code engineering set (see page 190).
\$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.classes.mdkernel.Element	An element to check.
	\$stereotypeName	java.lang.String	A stereotype name to be checked.

	Name	Type	Description
Return	-	boolean	true if elements have an applied stereotype with a given name.

```
$oracleHelper.getBooleanValueFromDefaultProfile($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.m agicdraw. classes.mdkernel.Eleme nt	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.m agicdraw. classes.mdkernel.Eleme nt	An element to be tested.

	Name	Type	Description
	\$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 the 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.
	\$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 <i>String</i> .

```
$oracleHelper.getPropertiesListFromProfile($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 object to 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.

	Name	Type	Description
Return	-	boolean	<i>true</i> 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.m agicdraw. classes.mdkernel.Opera tion	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.m agicdraw. classes.mdkernel.Prope rty	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.m agicdraw. classes.mdkernel.Eleme nt	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.

	Name	Type	Description
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	<i>true</i> 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	<i>true</i> 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	<i>true</i> 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.

- i** 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.

x The application cannot be run on the headless device even in a batch mode. The graphical environment is required.

i For more information on the launching batch mode programs, see [Specifying batch mode program classpath](#) (see page 208).

i For more information on creation of batch mode programs, see [Implementing command line launchers](#) (see page 224).

More information is available in javadoc.

Related pages

- [Specifying batch mode program classpath and required system properties](#) (see page 208)

- [Implementing command line launchers \(see page 224\)](#)

Specifying batch mode program classpath and required system properties

MagicDraw or other modeling tools run as a simple Java program.

In order to launch the program from the command line, the launcher is started using a *java* command line. The classpath has to include jar files from the lib folder and its subfolders (additional requirements exist for different command line implementations).

There is one mandatory property that has to be specified as well:

```
.-Desi.s
```

```
ystem.c
```

onfig=d

ata/

applica

tion.co

nf

The property is specified as a relative path to *<modeling tool installation directory>*, therefore, you either have to make your current directory before launching your program or specify absolute paths for the above-listed properties.

There are two types of command lines.

1. Core related batch program:
A program that does not call/depend on any functionality from any plugins. Should only depend/call core tool functionality.
 - Must extend *com.nomagic.magicdraw.commandline.CommandLine*
 - Must contain the Java *main* method and implement *com.nomagic.magicdraw.commandline.CommandLine#execute*
2. Plugin related batch program:
A program that can call/depend on any running modeling tool plugin

com.no

magic.m

agicdra

w . comm

andLine

. action

- specifies the batch mode action (implementation of *com.nomagic.magicdraw.commandline.CommandLineAction*) you want to execute, for

example,



Dcom.no

magic.m

agticdr

aw . comm

andline

. actio

n=com.n

omagic.

magi cd

raw . exa

mples . c

ommand

lineplu

gin.Com

mandLi

neActio

nExamp

le

- Must launch using `com.nomagic.magicdraw.commandline.CommandLineActionLauncher`
- The `CommandLineAction` must be registered in `CommandLineActionManager.getInstance().addAction(new CommandLineActionExample\(\));` e.g. in your plugin `init()` method
- The plugin where `CommandLineAction` is registered must load and run, in order for the command line program to launch it. (this means plugin.xml and plugin.jar files must be in the /plugins directory)

Also, it is possible to set up the command line to use the floating license server or license file - see "Providing licensing information" at [Licensing information](#)⁶⁹.

The implemented examples can be found in :

- Core batch program: `<modeling tool installation directory>\openapi\examples\imagegenerator`
- Plugin batch program: `<modeling tool installation directory>\openapi\examples\commandlineplugin`

The examples of complete command lines that launch the batch mode programs:

Core batch program command line for Windows

```
C:\Program Files\MagicDraw>java -Xmx1200M -Xss1024K ^
```

⁶⁹ <https://docs.nomagic.com/display/MD2024xR3/Licensing+information>

```
-cp lib/brand.jar;lib/brand_api.jar;lib/*;lib/graphics/*;openapi/examples/
imagegenerator/imagegenerator.jar ^
-Desi.system.config=data/application.conf ^
-Dfile.encoding=UTF-8 ^
@bin/vm.options ^
com.nomagic.magicdraw.examples.imagegenerator.ExportDiagramImages
project=project.mzip destination_dir=out
```

Plugin batch program command line for Windows

```
C:\Program Files\MagicDraw>java -Xmx1200M -Xss1024K ^
-cp lib/brand.jar;lib/brand_api.jar;lib/*;lib/graphics/* ^
-Desi.system.config=data/application.conf ^
-Dfile.encoding=UTF-8 ^
@bin/vm.options ^
-Dcom.nomagic.magicdraw.commandline.action=com.nomagic.magicdraw.examples.commandline
plugin.CommandLineActionExample ^
com.nomagic.magicdraw.commandline.CommandLineActionLauncher argument1 argument2
```

Core batch program command line for Mac OS-X and Linux

```
> cd MagicDraw_dir
> java -Xmx1200M -Xss1024K \
-Desi.system.config=data/application.conf \
-Dfile.encoding=UTF-8 \
@bin/vm.options \
-cp lib/brand.jar\:lib/brand_api.jar\:lib/*\:lib/graphics/*\:openapi/examples/
imagegenerator/imagegenerator.jar \
com.nomagic.magicdraw.examples.imagegenerator.ExportDiagramImages
project=project.mzip destination_dir=out
```

Plugin batch program command line for Mac OS-X and Linux

```
> cd MagicDraw_dir
> java -Xmx1200M -Xss1024K \
-cp lib/brand.jar\:lib/brand_api.jar\:lib/*\:lib/graphics/* \
-Desi.system.config=data/application.conf \
-Dfile.encoding=UTF-8 \
@bin/vm.options \
-Dcom.nomagic.magicdraw.commandline.action=com.nomagic.magicdraw.examples.commandline
plugin.CommandLineActionExample \
com.nomagic.magicdraw.commandline.CommandLineActionLauncher argument1 argument2
```

You can modify the previous examples and create more sophisticated batch files. For example, you can define the directory to run the tool from, or specify to use the same java, as the tool uses itself. The exemplary scripts that can be used to start the command line program are shown in the following

script examples. The scripts read and use the same classpath as the modeling tool when launching it manually (this is the preferred way to define classpaths).

These scripts can be launched from any directory. Can be modified to read other properties files (e.g. csm.properties, cea.properties)

A script that starts the Windows Core batch command line program on Windows:

Command line Core batch program Windows batch file that is able to launch program from any directory

```
@echo off
setlocal EnableExtensions

:: either local script variable or environment variable MAGICDRAW_HOME should be
defined
set "MAGICDRAW_HOME=C:\Program Files\MagicDraw"

if "%MAGICDRAW_HOME%" == "" (
    echo MAGICDRAW_HOME environment variable not set, please set it to the MagicDraw
    installation folder
    exit /B 1
)

setlocal enableDelayedExpansion
:: change slashes if needed
set MAGICDRAW_HOME=!MAGICDRAW_HOME:\=\/!
setlocal disableDelayedExpansion

pushd "%MAGICDRAW_HOME%"

:: Reads CLASSPATH value from magicdraw.properties files (change to your properties
file name)
For /F "tokens=1* delims==" %%A IN (bin\magicdraw.properties) DO (
    IF "%%A"=="CLASSPATH" set MD_CLASSPATH=%%B
)

:: Modify classpath to a valid Windows style classpath. Replace \: with ;
set MD_CLASSPATH=%MD_CLASSPATH:\=;%

java -Xmx1200M -Xss1024K ^
    -cp "%MD_CLASSPATH%;openapi\examples\imagegenerator\imagegenerator.jar" ^
    -Desi.system.config="data\application.conf" ^
    -Dfile.encoding=UTF-8 ^
    @bin\vm.options ^
    com.nomagic.magicdraw.examples.imagegenerator.ExportDiagramImages %*

popd
```

A script that starts the Windows Plugin batch command line program on Windows:

Command line Plugin batch program Windows batch file that is able to launch program from any directory

```
@echo off
setlocal EnableExtensions

:: either local script variable or environment variable MAGICDRAW_HOME should be
defined
set "MAGICDRAW_HOME=C:\Program Files\MagicDraw"

if "%MAGICDRAW_HOME%" == "" (
    echo MAGICDRAW_HOME environment variable not set, please set it to the MagicDraw
    installation folder
    exit /B 1
)

setlocal enableDelayedExpansion
:: change slashes if needed
set MAGICDRAW_HOME=!MAGICDRAW_HOME:\=/!
setlocal disableDelayedExpansion

pushd "%MAGICDRAW_HOME%"

:: Reads CLASSPATH value from magicdraw.properties files (change to your properties
file name)
For /F "tokens=1* delims==" %%A IN (bin\magicdraw.properties) DO (
    IF "%%A"=="CLASSPATH" set MD_CLASSPATH=%%B
)

:: Modify classpath to a valid Windows style classpath. Replace \: with ;
set MD_CLASSPATH=%MD_CLASSPATH:\:=%;

java -Xmx1200M -Xss1024K ^
    -cp "%MD_CLASSPATH%" ^
    -Desi.system.config="data\application.conf" ^
    -Dfile.encoding=UTF-8 ^
    @bin\vm.options ^
    -
Dcom.nomagic.magicdraw.commandline.action=com.nomagic.magicdraw.examples.commandlinep
lugin.CommandLineActionExample ^
    com.nomagic.magicdraw.commandline.CommandLineActionLauncher %*

popd
```

The shell script that starts a Core batch command line program on Mac OS-X, Linux, and Windows cygwin/msys:

Command line Core batch program shell script that is able to launch program from any directory

```
#!/bin/bash

# either local script variable or environment variable MAGICDRAW_HOME should be
defined
MAGICDRAW_HOME="/home/myuser/Desktop/MagicDraw"

if [ -z "$MAGICDRAW_HOME" ]; then
    echo "MAGICDRAW_HOME environment variable not set, please set it to the MagicDraw
installation folder"
    return
fi

cd "$MAGICDRAW_HOME"

# Reads CLASSPATH value from magicdraw.properties files (change to your properties
file name)
MD_CLASSPATH=`grep "CLASSPATH" bin/magicdraw.properties | cut -d'=' -f 2`

if [ "$OS" = Windows_NT ]; then
    MD_CLASSPATH=$(echo "$MD_CLASSPATH" | sed "s/\\\:;/g")
    cp_delim=";"
else
    MD_CLASSPATH=$(echo "$MD_CLASSPATH" | sed "s/\\\:/:/g")
    cp_delim=":"
fi

java -Xmx1200M -Xss1024K \
    -cp "$MD_CLASSPATH${cp_delim}openapi/examples/imagegenerator/imagegenerator.jar" \
    -Desi.system.config="data/application.conf" \
    -Dfile.encoding=UTF-8 \
    @bin/vm.options \
    com.nomagic.magicdraw.examples.imagegenerator.ExportDiagramImages "$@"
```

The shell script that starts a Plugin batch command line program on Mac OS-X, Linux, and Windows cygwin/msys:

Command line Plugin batch program shell script that is able to launch program from any directory

```
#!/bin/bash
```

```
# either local script variable or environment variable MAGICDRAW_HOME should be
defined
MAGICDRAW_HOME="/home/myuser/Desktop/MagicDraw"

if [ -z "$MAGICDRAW_HOME" ]; then
    echo "MAGICDRAW_HOME environment variable not set, please set it to the MagicDraw
installation folder"
    return
fi

cd "$MAGICDRAW_HOME"

# Reads CLASSPATH value from magicdraw.properties files (change to your properties
file name)
MD_CLASSPATH=`grep "CLASSPATH" bin/magicdraw.properties | cut -d=' ' -f 2`

if [ "$OS" = Windows_NT ]; then
    MD_CLASSPATH=$(echo "$MD_CLASSPATH" | sed "s/\\\:;/g")
else
    MD_CLASSPATH=$(echo "$MD_CLASSPATH" | sed "s/\\\:;/g")
fi


java -Xmx1200M -Xss1024K \
    -cp "$MD_CLASSPATH" \
    -Desi.system.config="data/application.conf" \
    -Dfile.encoding=UTF-8 \
    @bin/vm.options \

-Dcom.nomagic.magicdraw.commandline.action=com.nomagic.magicdraw.examples.commandline
plugin.CommandLineActionExample \
    com.nomagic.magicdraw.commandline.CommandLineActionLauncher "$@"
```

Related pages

- [Licensing information \(see page 316\)](#)

Implementing command line launchers

 While launching the command line action, the single sign-on connection to the server is not supported.

Implementation of command line launchers depends on the type of the batch mode program (Core related or Plugin related).

Both modes provide two ways of implementation:

Command

Line

and

Command

LineAct

ion

(for Core and Plugin respectively) are general

launchers meant for developers that will not be opening projects or needs full control of argument parsing and opening of projects.

Project

Command

Line

and

Project

Command

LineAct

ion

(for Core and Plugin respectively) are

convenience launchers simplifying work with projects as they provide list of arguments that will be parsed and used to open one or more projects removing great deal of boilerplate code needed.

To create command line launcher:

1. Core related batch program:
 - For general command line extend

com.no

magic.m

agticdr

aw . comm

andline

. Comma

ndLine

- Override

com.no

magic.

magi cd

raw . co

mmandl

ine.Co

mmandL

ine#pa

rseArg

S

to parse your arguments

- Override

com.no

magic.

magi cd

raw.co

mmandl

ine.Co

mmandL

ine#ex

ecute

to execute

your action

- For project command line action extend

com.no

magic.m

agicdra

W . comm

andline

.Projec

tComma

ndLine

- Override

com.no

magic.

magi cd

raw . co

mmandl

ine.Pr

objectC

ommand

Line#p

arseAr

gument

S

to parse custom arguments you might be using

- Override

com.no

magic.

magi cd

raw . co

mmandl

ine.Pr

objectC

ommand

Line#e

xecute

(java.

util.P

roper

ies,

com.no

magic.

magi cd

raw . co

re . Pro

ject)

to

execute your action. This method will be called for every project opened using available project command line arguments.

2. Plugin related batch program:
 - For general command line implement

com.no

magic.m

agticdr

aw . comm

andline

. Comma

ndLineA

ction

- Override

com.no

magic.

magi cd

raw.co

mmandl

ine.Co

mmandL

ineAct

ion#ex

ecute

to

- parse your arguments and do your action
• For project command line action extend

com.no

magic.m

agticdr

aw . comm

andline

.Proje

ctComma

ndLine

Action

- Override

com.no

magic.

magi cd

raw . co

mmandl

ine.Pr

objectC

ommand

LineAc

tion#e

xecute

(java.

lang.S

tring[

],

java.u

til.Pr

operti

es,

com.no

magic.

magi cd

raw.co

re.Pro

ject)

to execute

your action. This method will be called for every project opened using available project command line arguments. Full array of originally passed arguments and parsed project command line arguments specific to the project (default arguments that were parsed from the command line or project-specific properties file) are also passed along with opened project.

Available parameters for project command line launcher:

. project

- Project name or path

. project

Descrip

tor

- Project descriptor (The description is a copy

of parameter name. Describe how to obtain it, add an example how it looks like. All cases.)

server

Server URL (Describe all cases(TWC, PowerBy). In TWC case it is a host:port which is not a URL)

.

username



e

- Username on server

password

d

- Password for the provided username

- By default provided password should be encrypted. (**3DEXPERIENCE** projects accept only plain text passwords)
 - To generate an encrypted password, run any command line launcher with a single argument

genera

teServ

erPass

word=y

ourPas

sword

or
gener

ateSer

verPas

sword

without a

password for the interactive prompt.

- In order to use the password in plain text form argument

encrypt

tpasswo

rd

needs to be provided and set to

true

. (Not needed for

3DEXPERIENCE projects)

serverT

ype

- If launching a project stored in the **3DEXPERIENCE**

platform, must specify the platform deployment type

- OnPremise deployment

server

Type=PL

ATFORM_

ON_PRE

MISE

- Cloud deployment

server

Type=P

LATFORM

_CLOUD

. enables

SL

- To use SSL

encrypt

Password

d

- Set to true if the provided password is in plain text (Not needed for

3DEXPERIENCE projects)

properties

ies

- Path to a properties file containing properties.

Multiple properties files can be provided "properties=prop1.properties;prop2.properties"

project

Password

d

- Project password

version

- Project version

branch

Project branch

All parameters for project command line launchers, both in the command line and properties files, should be provided in the format

propert

$y_1 = \text{valu}$

e_1

property

2=value

2

To open multiple projects, you must define properties for each project (in case the project is local, path or name is enough) in separate properties files. These files are passed as "properties" arguments e.g. "properties=prop1.properties;prop2.properties".

Starting MagicDraw or other modeling tool as part of another application

To integrate MagicDraw or other modeling tool into your application you should launch MagicDraw directly from your Java application. This would allow you to interact with MagicDraw or other modeling tool classes directly.

In order to do that you must:

- Include lib/ folder and all of its subfolder into your project classpath
- Import class `com.nomagic.magicdraw.core.Application` and call the appropriate `start(...)` method. (see Javadoc)
- Make sure there is only one instance of the modeling tool running in the same JVM(multiple instances cannot be started in the same JVM)

For launching the modeling tool straight from the command line, please refer to [Running programs in batch mode](#) (see page 207) and [Specifying batch mode program classpath and required system properties](#) (see page 208).

MagicDraw file format

File format in MagicDraw 3.6-17.0

MagicDraw (or other modeling tool developed by us) 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 XML 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
- 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

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>CD0=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 us) 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>⁷⁰. Information about the python language you can find at <http://www.python.org>⁷¹.

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 \(see page 22\)](#). If *script.xml* contains valid information, the script file specified in *script.xml* is executed. The script file should contain a valid Jython 2.7.2 script.

Related pages

- [Creating script \(see page 291\)](#)

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

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"
```

⁷⁰ <http://www.jython.org/>

⁷¹ <http://www.python.org/>

```

    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"
	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"

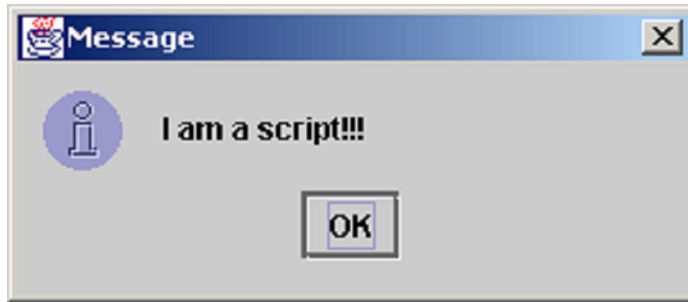
Step #3: Write Script Code

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* 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.

JavaScript migration from Nashorn to Rhino engine

The JavaScript *Nashorn* has been removed from the modeling tool ([learn more about deprecated JavaScript Nashorn](#)⁷²). Please use JavaScript *Rhino* instead.

i To find the usages of the JavaScript *Nashorn* in the project, run the **Deprecated JavaScript** validation suite (**Analyze > Validation > Validate**).

The key differences are listed below. For more information, please check [Nashorn/Rhino Migration Guide](#)⁷³.

⁷² <https://openjdk.java.net/jeps/335>

⁷³ <https://wiki.openjdk.java.net/display/Nashorn/Rhino+Migration+Guide>

Class object and .class property

If a java API accepts a

java.la

ng.Class

object, in *Nashorn* "

.class"

property (similar to Java) is used. In *Rhino* you pass script representation of class "as is".

```
// Nashorn
var types = [com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class.class]
// Rhino
var types = [com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class]
```

```
var project = com.nomagic.magicdraw.core.Application.getInstance().getProject()
var Finder = com.nomagic.magicdraw.uml.Finder
Finder.byTypeRecursively().find(project, types, false)
```

Accessing Java packages and classes from script

Nashorn's recommended way to access Java classes (by using

Java.type

pe

) is not supported in *Rhino*.

```
// Nashorn
var Vector = Java.type("java.util.Vector")
var JFrame = Java.type("javax.swing.JFrame")

// Rhino
var Vector = java.util.Vector
var JFrame = Packages.javax.swing.JFrame
```

Creating Java arrays from script

In *Nashorn*, you can resolve to a Java array class using the same

Java.type

pe

API. And array creation is done using

new

operator. In *Rhino*, you create a Java array using

Java reflection from script.

```
// Nashorn
var IntArray = Java.type("int[]")
var array = new IntArray(8)
```

```
// Rhino
var Array = java.lang.reflect.Array
var intClass = java.lang.Integer.TYPE
var array = Array.newInstance(intClass, 8)
```

Java exceptions

In *Nashorn* Java exception objects are thrown "as is". *Rhino* wraps Java exceptions as a script object. If you want underlying Java exception, use

"`javaExc`

`ption`

" property to


access it.

```
try
{
    <...>
}
catch (e)
{
    // Nashorn
    e.printStackTrace()
    // Rhino
    e.javaException.printStackTrace()
}
```

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 \(see page 298\)](#)
- [Plugins migration to MagicDraw 18.1 and later Open API \(see page 298\)](#)
- [Plugins migration to MagicDraw 18.0 and later Open API \(see page 298\)](#)
 - [Supported Changes from UML 2.4.1 to UML 2.5 \(see page 299\)](#)
- [Plugins migration to MagicDraw 17.0.1 and later Open API \(see page 304\)](#)
 - [Project \(decomposition\) structure API changes \(see page 306\)](#)
 - [Supported UML Specification Changes from Version 2.3 to 2.4.1 \(see page 308\)](#)
- [Plugins migration to MagicDraw 15.0 and later Open API \(see page 314\)](#)
 - [UML metamodel changes \(see page 315\)](#)

Plugins migration to MagicDraw 18.2 and later Open API

The 18.2 version Open API changes are described [here](#)⁷⁴.

Plugins migration to MagicDraw 18.1 and later Open API

18.1 Open API changes are described [here](#)⁷⁵.

Plugins migration to MagicDraw 18.0 and later Open API

Open API Changes

UML 2.5 support related changes

- UML metamodel interfaces have been adjusted to UML 2.5 support related changes.
- *com.nomagic.uml2.StandardProfileL2* and *com.nomagic.uml2.StandardProfileL3* classes have been merged into *com.nomagic.uml2.StandardProfile*.

⁷⁴ <http://www.nomagic.com/news/new-noteworthy/magicdraw-noteworthy/magicdraw-18-2-fr/magicdraw-18-2-fr-all.html#api>

⁷⁵ http://www.nomagic.com/news/new-noteworthy/magicdraw-noteworthy/magicdraw-18-1-fr/magicdraw-18-1-fr-all.html#API_changes

UI refactoring related changes

UI of diagrams' windows management, symbols' compartments management, and elements creation in compartments have been refactored. As a consequence, some actions have been removed from the *com.nomagic.magicdraw.actions.ActionsID* interface and new actions have been added.

IDs of the removed actions are as follows:

- ActionsGroups.NEXT_DIAGRAM_RELATED
- ActionsGroups.PREVIOUS_DIAGRAM_RELATED
- ActionsID.DIAGRAM_NAVIGATION_GROUP
- ActionsID.INSERT_NEW_ATTRIBUTE
- ActionsID.INSERT_NEW_EXTENSION
- ActionsID.INSERT_NEW_LITERAL
- ActionsID.INSERT_NEW_OPERAND
- ActionsID.INSERT_NEW_OPERATION
- ActionsID.INSERT_NEW_PORT
- ActionsID.LINK_EDITING
- ActionsID.METRICS
- ActionsID.OPEN_IN_NEW_TAB_ACTION
- ActionsID.OPEN_LAST_ActionsID.RECENT_DIAGRAMS
- ActionsID.REDO_LIST
- ActionsID.UNDO_LIST

Project Merge related changes

Types of parameters have been changed from Set to Collection, because of performance related improvements in methods of the following interfaces:

- *com.nomagic.magicdraw.merge.Change*
- *com.nomagic.magicdraw.merge.macro.MacroChange*
- *com.nomagic.magicdraw.merge.RelatedChange*
- *com.nomagic.magicdraw.merge.macro.RelatedMacroChange*

Structured expressions related changes

The *com.nomagic.magicdraw.expressions.ExpressionHelper* class has been added. It provides utility methods to use in queries defined with the StructuredExpression language (used by smart packages, derived properties in DSL, and so forth).

UML Interactions related changes

The *com.nomagic.uml2.ext.jmi.helpers.InteractionHelper* class has been added. It provides utility methods to access Interactions part of UML metamodel.

Supported Changes from UML 2.4.1 to UML 2.5

- [Metamodel Changes](#) (see page 300)
 - [Notation Changes](#) (see page 303)
 - [Profile changes](#) (see page 303)
-

Metamodel Changes

All the changes are grouped by the categories:

- [New property names added](#) (see page 300)
- [Properties renamed](#) (see page 300)
- [Derivation changes](#) (see page 300)
- [Order changes](#) (see page 301)
- [Other changes](#) (see page 302)
- [Properties removed](#) (see page 302)

New property names added

Property	Description / Comment
Classifier::conveyingFlow	-
DeployedArtifact::deploymentForArtifact	
PackageableElement::import	

Properties renamed

Property	Description / Comment
NamedElement::classifierInheritingClassifier	-
Classifier::classNestingClass	
ParameterableElement::parameterSubstitutionOwnningTemplateParameterSubstitution	

Derivation changes

Property	Description / Comment
Property::/classifier : Classifier [0..1] {readOnly, union, subsets Feature::featuringClassifier, subsets RedefinableElement::redefinitionContext}	All these properties have been made automatically derived and read-only unions.

InputPin::/action : Action[0..1]{readOnly, union, subsets Element::owner}	
Action::/input : InputPin[*] {ordered, readOnly, union, subsets Element::ownedElement}	
OutputPin::/action : Action[0..1]{readOnly, union, subsets Element::owner}	
NamedElement::/namespace : Namespace[0..1] {readOnly, union, subsets Element::owner}	
DirectedRelationship::/source : Element [1..*] {readOnly, union, subsets Relationship::relatedElement}	Both properties have been made automatically derived and read-only unions. This feature has been supported in earlier versions of MagicDraw.
DirectedRelationship::/target : Element [1..*] {readOnly, union, subsets Relationship::relatedElement}	
NamedElement::/clientDependency	The Client Dependency property of a named element has been made automatically derived. This feature has been supported in earlier versions of MagicDraw.

Order changes

Property	Description / Comment
AcceptEventAction::result : OutputPin [0..*] {ordered, subsets Action::output}	The ability to order output pins holding the values received from an event occurrence, has been available in earlier versions of MagicDraw.
Association::/endType : Type [1..*]{ordered, subsets Relationship::relatedElement}	The set of classifiers that are used as end types of the association, is no longer ordered.

Classifier::/attribute : Property [0..*]{ordered, union, subsets Classifier::feature}	The ability to order the attributes, which are direct properties of a classifier has been available in earlier versions of MagicDraw. Though the Attribute property cannot be edited, its value displays the same order of the attributes as is defined in other places of the classifier specification, such as the Owned Attribute property or the Attributes property group. Thus to change the Attribute property value, change the value of a relevant property.
ConnectableElement::/end : ConnectorEnd [0..*]{ordered}	A set of connector ends that attach to the particular connectable element, is no longer ordered.
DurationObservation::event : NamedElement [1..2]{ordered}	Event elements are now automatically ordered.
ReplyAction::♦ replyValue : InputPin [0..*]{ordered, subsets Action::input}	The set of input pins can now be ordered. In the Reply Value property value cell, click the <input type="checkbox"/> button to open the dialog for reordering.
UnmarshallAction::♦ result : OutputPin [1..*]{ordered, subsets Action::output}	The set of output pins can now be ordered. In the Result property value cell, click the <input type="checkbox"/> button to open the dialog for reordering.

Other changes

Property	Description / Comment
LoopNode::♦ loopVariable : OutputPin [0..*]{ordered, subsets Element::ownedElement}	The Loop Variable property has been made composite.
OutputPin::loopNode : LoopNode[0..1]{subsets Element::owner}	
Activity::structuredNode : StructuredActivityNode [0..*]{readOnly, subsets Activity::node, Activity::group}	The property has been made read-only in earlier versions of MagicDraw.

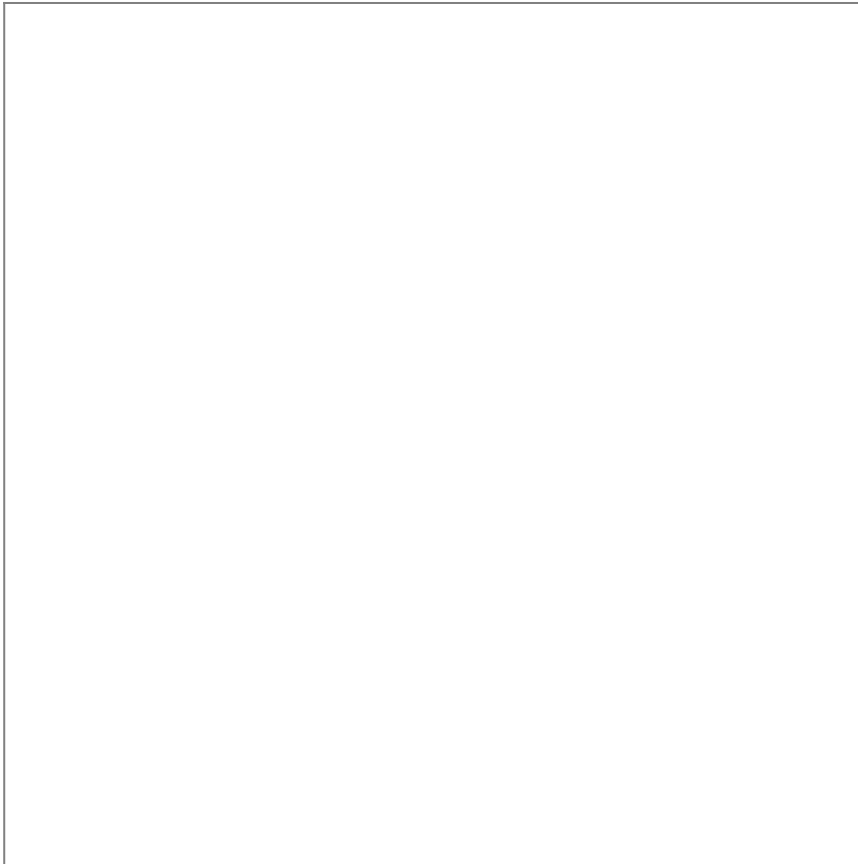
Properties removed

Property	Description / Comment
----------	-----------------------

Property::/default : String[0..1]	The Default property no longer appears in a property specification. The value of the Default Value property will be used instead of the removed property value.
-----------------------------------	---

Notation Changes

From now on, the inherited members are denoted with the caret "^" sign.



Profile changes

Standard/system profiles *StandardProfileL2* and *StandardProfileL3* have been merged into a single one.

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 \(see page 314\)](#) 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. \(see page 308\)](#)
- [Project \(decomposition\) structure API changes \(see page 306\).](#)

- Changes in methods:
 - **com.nomagic.magicdraw.core.options.ProjectOptionsConfigurator**. The method *afterLoad(ProjectOptions)* has been added. An empty implementation of this method must be added to a custom ProjectOptionsConfigurator implementation.
 - **com.nomagic.magicdraw.core.project.ProjectEventListener**. The following methods have been added:

*projectActivatedFromGUI(Project)*⁷⁶

*projectCreated(Project)*⁷⁷

*projectOpenedFromGUI(Project)*⁷⁸

*projectPreActivated(Project)*⁷⁹

*projectPreClosed(Project)*⁸⁰

*projectPreDeActivated(Project)*⁸¹

*projectPreReplaced(Project,⁸²Project)*⁸³

projectPreSaved(Project, boolean)

*projectPreClosedFinal(Project)*⁸⁴

An empty implementation of these methods must be added to the custom *ProjectEventListener* implementation. The code change is not required, if *com.nomagic.magicdraw.core.project.ProjectEventListenerAdapter* 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 14).

Related pages

- [Project \(decomposition\) structure API changes](#) (see page 306)

⁷⁶ <http://jdocs.nomagic.com/2024x/com/nomagic/magicdraw/core/project/ProjectEventListener.html#projectActivatedFromGUI-com.nomagic.magicdraw.core.Project-com.nomagic.magicdraw.core.Project>

⁷⁷ <http://jdocs.nomagic.com/2024x/com/nomagic/magicdraw/core/project/ProjectEventListener.html#projectCreated-com.nomagic.magicdraw.core.Project-com.nomagic.magicdraw.core.Project>

⁷⁸ <http://jdocs.nomagic.com/2024x/com/nomagic/magicdraw/core/project/ProjectEventListener.html#projectOpenedFromGUI-com.nomagic.magicdraw.core.Project-com.nomagic.magicdraw.core.Project>

⁷⁹ <http://jdocs.nomagic.com/2024x/com/nomagic/magicdraw/core/project/ProjectEventListener.html#projectPreActivated-com.nomagic.magicdraw.core.Project-com.nomagic.magicdraw.core.Project>

⁸⁰ <http://jdocs.nomagic.com/2024x/com/nomagic/magicdraw/core/project/ProjectEventListener.html#projectPreClosed-com.nomagic.magicdraw.core.Project-com.nomagic.magicdraw.core.Project>

⁸¹ <http://jdocs.nomagic.com/2024x/com/nomagic/magicdraw/core/project/ProjectEventListener.html#projectPreDeActivated-com.nomagic.magicdraw.core.Project-com.nomagic.magicdraw.core.Project>

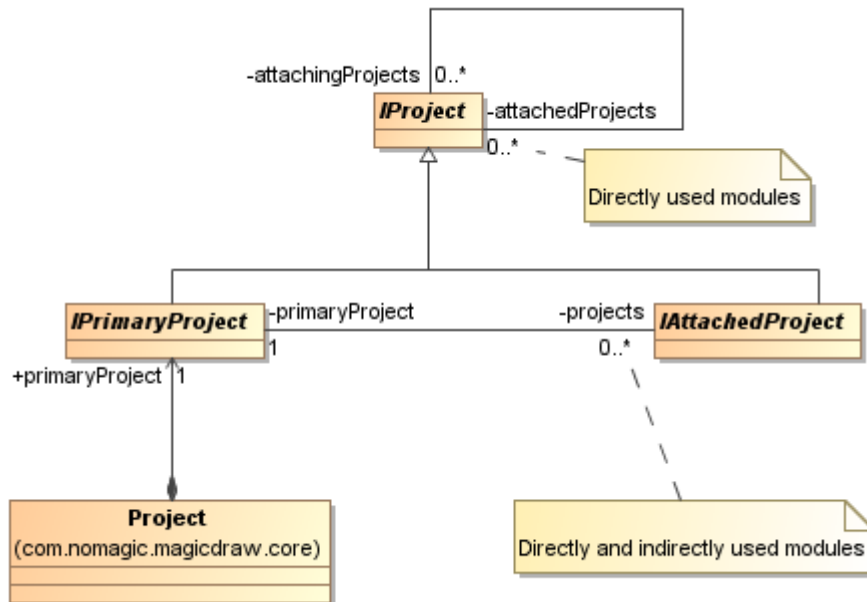
⁸² <http://jdocs.nomagic.com/2024x/com/nomagic/magicdraw/core/project/ProjectEventListener.html#projectPreReplaced-com.nomagic.magicdraw.core.Project-com.nomagic.magicdraw.core.Project-com.nomagic.magicdraw.core.Project>

⁸³ <http://jdocs.nomagic.com/2024x/com/nomagic/magicdraw/core/project/ProjectEventListener.html#projectPreReplaced-com.nomagic.magicdraw.core.Project-com.nomagic.magicdraw.core.Project-com.nomagic.magicdraw.core.Project>

⁸⁴ <http://jdocs.nomagic.com/2024x/com/nomagic/magicdraw/core/project/ProjectEventListener.html#projectPreClosedFinal-com.nomagic.magicdraw.core.Project-com.nomagic.magicdraw.core.Project>

Project (decomposition) structure API changes

The following figure shows the domain model of the project decomposition.



9 Domain model of project decomposition

The core change is an introduction of *com.nomagic.ci.persistence.IProject* - *com.nomagic.ci.persistence.IPrimaryProject* and *com.nomagic.ci.persistence.IAttachedProject*. *IPrimaryProject* is a single instance in the scope of a project. *IPrimaryProject* can have attached (used) any number of *IAttachedProject* (used projects), while *IAttachedProject* itself can attach other used projects (*IAttachedProject*).

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

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

com.nomagic.ci.persistence.IProject

com.nomagic.ci.persistence.IPrimaryProject

com.nomagic.ci.persistence.IAttachedProject

com.nomagic.ci.persistence.decomposition.ProjectAttachmentConfiguration

com.nomagic.ci.persistence.mounting.IMountPoint

com.nomagic.ci.persistence.sharing.ISharePoint

ProjectUtilities and *com.nomagic.magicdraw.core.modules.ModulesService* 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* and *com.nomagic.ci.persistence.decomposition.ProjectAttachmentConfiguration*. There is separate *ModuleDescriptor* for every (same) project usage in MagicDraw 17.0 or earlier versions, while there is a single *IAttachedProject* for the used project, but a separate *ProjectAttachmentConfiguration* for every project usage in the current version.

The example of the MagicDraw 17.0 and earlier version code:

```
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();
}
```

Supported UML Specification Changes from Version 2.3 to 2.4.1

Supported UML specification changes include:

- [Metamodel Changes](#) (see page 308)
- [Notation changes](#) (see page 313)

Metamodel Changes

Metamodel Change	Description / Comment
DATA TYPES ADDED	
Real	Added to the PrimitiveTypes package. You can now use values of a real type in your models.
METACLASSES ADDED	
LiteralReal	Specifies the real value.
METACLASSES CHANGED	

Metamodel Change	Description / Comment
DestructionEvent → DestructionOccurrenceSpecification	The DestructionEvent class has been renamed to DestructionOccurrenceSpecification. It represents the destruction of an instance described by the lifeline that contains this instance. DestructionOccurrenceSpecification is a specialization of MessageOccurrenceSpecification.
METACLASSES REMOVED	
ExecutionEvent	These metaclasses have been removed from the Interactions package due to unsupported event types.
CreationEvent	
ReceiveOperationEvent	
ReceiveSignalEvent	
SendOperationEvent	
SendSignalEvent	
PROPERTIES ADDED	
Package::URI:String[0..1] {id}	Provides the package with an identifier that can be used for many purposes. URI is the universally unique identification of the package following the IETF URI specification, RFC 2396 http://www.ietf.org/rfc/rfc2396.txt and it must comply with those syntax rules. You can now specify this property for packages, profiles, and models. For the notation of the URI property, see Notation Changes (see page 313).
Property::isID:Boolean=False	Indicates that the property can be used to uniquely identify an instance of the containing class, when the value is set to true. The default value is false. For the notation of the isID property modifier, see Notation Changes (see page 313).

Metamodel Change	Description / Comment
InteractionUse::returnValue:ValueSpecification [0..1]	Specifies the value returned by the executed interaction.
InteractionUse::returnValueRecipient:Property [0..1]	Specifies the recipient of the value which is returned by the executed interaction.
EnumerationLiteral::/classifier:Enumeration[1]	The classifier of this EnumerationLiteral should now be equal to the enumeration that contains this enumeration literal. Redefines InstanceSpecification::classifier.
ASSOCIATIONS REMOVED	
ExecutionOccurrenceSpecification::event: ExecutionEvent[1]	The association from ExecutionOccurrenceSpecification to ExecutionEvent has been removed, as the ExecutionEvent metaclass does not exist in the UML 2.4.1
OccurrenceSpecification::event:Event[1]	The specification of the occurring event is not referenced any more.
BehavioredClassifier::ownedTrigger:Trigger[0..*]	Trigger descriptions owned by a classifier are not referenced any more.
SUBSETTING CHANGES	
Interface::redefinedInterface {subsets <i>redefinedElement</i> → <i>redefinedClassifier</i> }	An interface now references all the interfaces that are redefined by this interface.
SUBSETTING ADDED Note: These features were already available in earlier versions of MagicDraw.	
Duration::expr:ValueSpecification[0..1] {subsets <i>Element::ownedElement</i> }	Element or elements can be selected or created only under the owned element's scope.
LinkEndData::qualifier:QualifierValue[*] {subsets <i>Element::ownedElement</i> }	

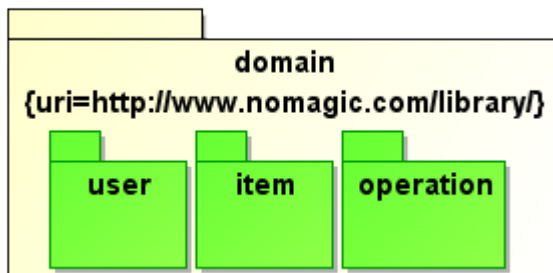
Metamodel Change	Description / Comment
LinkAction::endData:LinkEndData[2..*] {subsets <i>Element::ownedElement</i> }	
CreateLinkAction::endData:LinkEndCreationData[0..*] {subsets <i>Element::ownedElement</i> }	
DestroyLinkAction::endData:LinkEndDestructionData[2..*] {subsets <i>Element::ownedElement</i> }	
TimeExpression::expr:ValueSpecification[0..1] {subsets <i>Element::ownedElement</i> }	
ValuePin::value:ValueSpecification[1] {subsets <i>Element::ownedElement</i> }	
State::deferrableTrigger:Trigger[0..*] {subsets <i>Element::ownedElement</i> }	
StructuredActivityNode::edge:ActivityEdge[0..*] {subsets <i>Element::ownedElement</i> }	
StructuredActivityNode::node:ActivityNode[0..*] {subsets <i>Element::ownedElement</i> }	
ValueSpecificationAction::value:ValueSpecification[1] {subsets <i>Element::ownedElement</i> }	
AcceptEventAction::trigger:Triger[1..*] {subsets <i>Element::ownedElement</i> }	
Stereotype::icon:Image[0..*] {subsets <i>Element::ownedElement</i> }	
TimeEvent::when:TimeExpression[1] {subsets <i>Element::ownedElement</i> }	

Metamodel Change	Description / Comment
InteractionUse::argument:ValueSpecification[*] {subsets <i>Element::ownedElement</i> }	
StructuredActivityNode::node:ActivityNode [0..*] {subsets <i>Element::ownedElement</i> }	
SequenceNode::executableNode: ExecutableNode[0..*] {subsets <i>Element::ownedElement</i> }	
Transition::trigger:Trigger[0..*] {subsets <i>Element::ownedElement</i> }	
Classifier::/feature {subsets <i>Namespace::member</i> }	The /feature property of Classifier is now included as a part of the member property.
Feature::/featuringClassifier {subsets <i>NamedElement::memberNamespace</i> }	The /featuringClassifier property of Feature is now included as a part of the memberNamespace property.
Property::owningAssociation {subsets <i>RedefinableElement::redefinitionContext</i> }	A property now references the owning association of this property.
DEFAULT VALUE CHANGES	
DurationObservation::firstEvent:Boolean[0..2] =True	The default value has been removed. In effect, the value of the firstEvent property is now by default empty.
DurationConstraint::firstEvent:Boolean[0..2] =True	
PackageableElement::visibility:VisibilityKind[1] =False → Public	The default value of this property has been changed to Public. This means that packageable elements are now by default public.
MULTIPLICITY CHANGES	

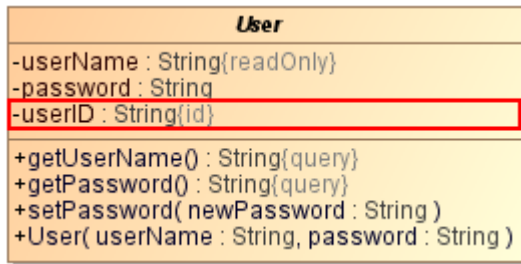
Metamodel Change	Description / Comment
EnumerationLiteral::enumeration:Enumeration [0..1] → [1]	The Enumeration must now be specified. It has become mandatory.
Multiplicity of ExecutionSpecification association with ExecutionOccurrenceSpecification: [1] → [0..2]	<p>ExecutionSpecification is now associated with two ExecutionOccurrenceSpecifications:</p> <ul style="list-style-type: none"> • The start ExecutionOccurrenceSpecification that designates the start of an action or behavior. • The finish ExecutionOccurrenceSpecification that designates the finish of an action or behavior.
DERIVATION CHANGES	
Constraint::/context → context	The context property of the constraint is no longer derived. It is now a settable property.
Message::/signature → signature	The signature of the message is no longer derived. However, it remains read-only.

Notation changes

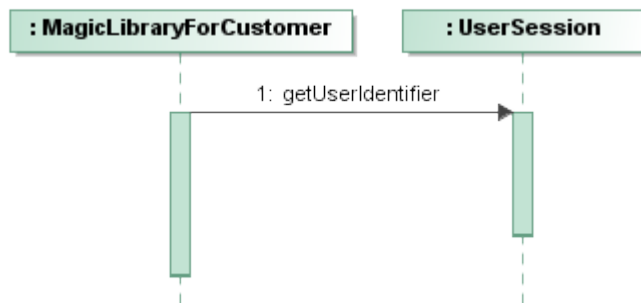
1. If there is a value defined for the URI property of a package, model, or profile, it is automatically displayed on the corresponding shape. The following figure depicts an example of the package notation with the URI property value defined:



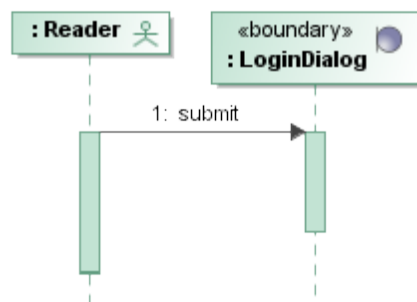
2. If the isID property is set to true for the selected attribute, the {id} modifier is displayed in the property modifiers group on the class shape. The following figure depicts an example of the attribute notation when the isID property is set to true:



3. The name of a synchronous message is now displayed on a diagram pane, even if the message does not have a signal or an operation assigned. The following figure depicts an example of the synchronous message name notation:



4. Brackets “()” are no more added to a call message name if the message does not have a signal or an operation assigned. The following figure depicts an example of the call message name notation:



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 315).
- Some **deprecated methods** have been removed:

Removed deprecated method	Substitution
<code>BaseElement.getProject()</code>	<code>Project.getProject(BaseElement)</code>
<code>MainFrame.getDialogParent()</code>	<code>MDDialogParentProvider.getProvider().getDialogParent()</code>
<code>MainFrame.setDialogParent(Frame)</code>	<code>MDDialogParentProvider.getProvider().setDialogParent(Frame)</code>
<code>ElementListProperty.setStereotype(Stereotype)</code>	<code>ElementListProperty.setSelectableRestrictedElements(Collection)</code>

- 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 \(see page 315\)](#)

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 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.

Licensing information



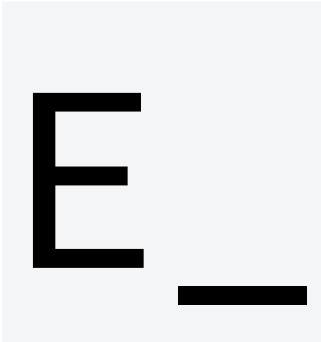


Providing licensing information

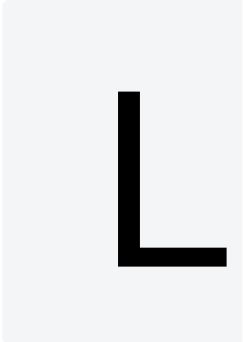
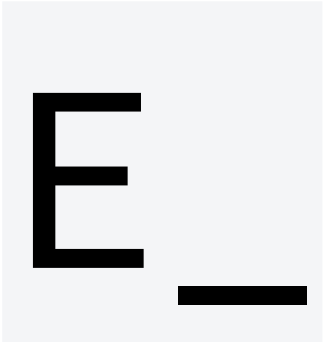

It is possible to provide the tool with the licensing information using the Java (JVM) system properties.



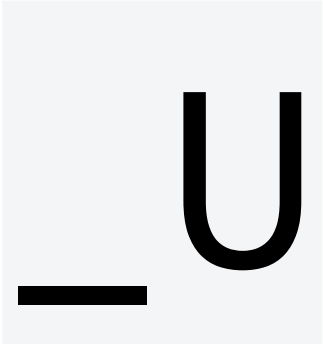
You can specify either seat or floating license.




- Seat license (license file) properties:

Property	Value	Description
L	t	when specified, the tool tries using the license file specified in LI
IC	ru	CEN
EN	e	SE_

Property	Value	Description
		
		 property. Allows changing the license.
		

Property	Value	Description
		
		
		




Property	Value	Description
		
		
		



Property	Value	Description
		
		
		

Property	Value	Description
L	path to the license file or directory with the license files	when the tool has no license specified yet, or
LI		
IC		
EN		CEN
		SE_

Property	Value	Description
S		FI
E_		LE_
		FOR



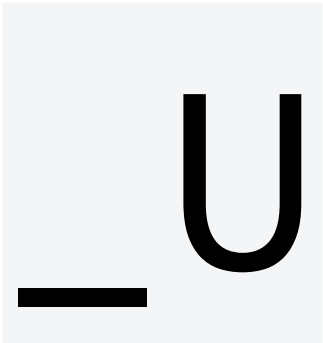
Property	Value	Description
F		CE
IL		_US
E		AGE
		is specified, the tool uses a license specified by




Property	Value	Description
		
		
		

Property	Value	Description
		  property. <i>Surround the path with double quotes if it contains spaces.</i>



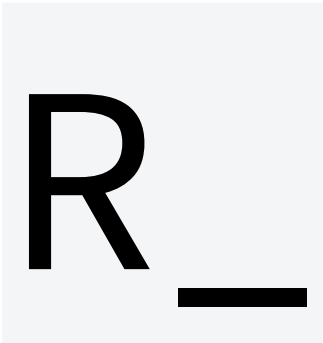
- Floating license (FlexNet license server) properties:


Property	Value	Description
F	t	when specified, the tool uses a floating license.
L_	ru	
FO	e	

Property	Value	Description
		
		
		

Property	Value	Description
		
		
		



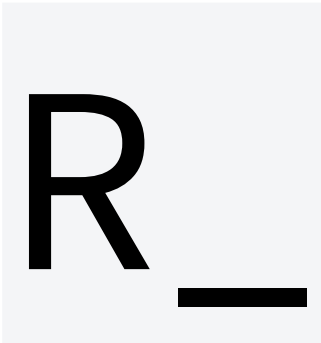
Property	Value	Description
	FlexNet license server name or IP address	
		
		



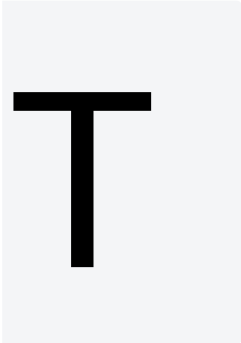
Property	Value	Description
		
		
		

Property	Value	Description
		
		

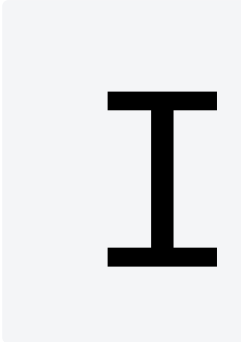
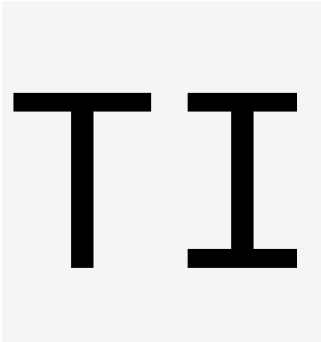

Property	Value	Description
R		
ES		
S		

Property	Value	Description
	FlexNet license server port number	
		
		

Property	Value	Description
		
		
		

Property	Value	Description
		
		
		

Property	Value	Description
	tool edition	specify if the tool supports editions; otherwise, omit.
		
		

Property	Value	Description
		
		
		

-D

must be prepended to the property name if the Java system

property is specified through java command-line arguments/parameters.

bin/

E.g., in the tool launcher properties file (

<launcher

r

name> .pr

operties

), add floating license properties:

JAVA_AR

GS=< . . . >

-DFL_FOR

CE_USAG

E\=true

-DFL_SER

VER_ADD

RESS\=my

Server

-DFL_SE

RVER_POR

T\=1101

-DFL_ED

ITION\=E

nterpri

se

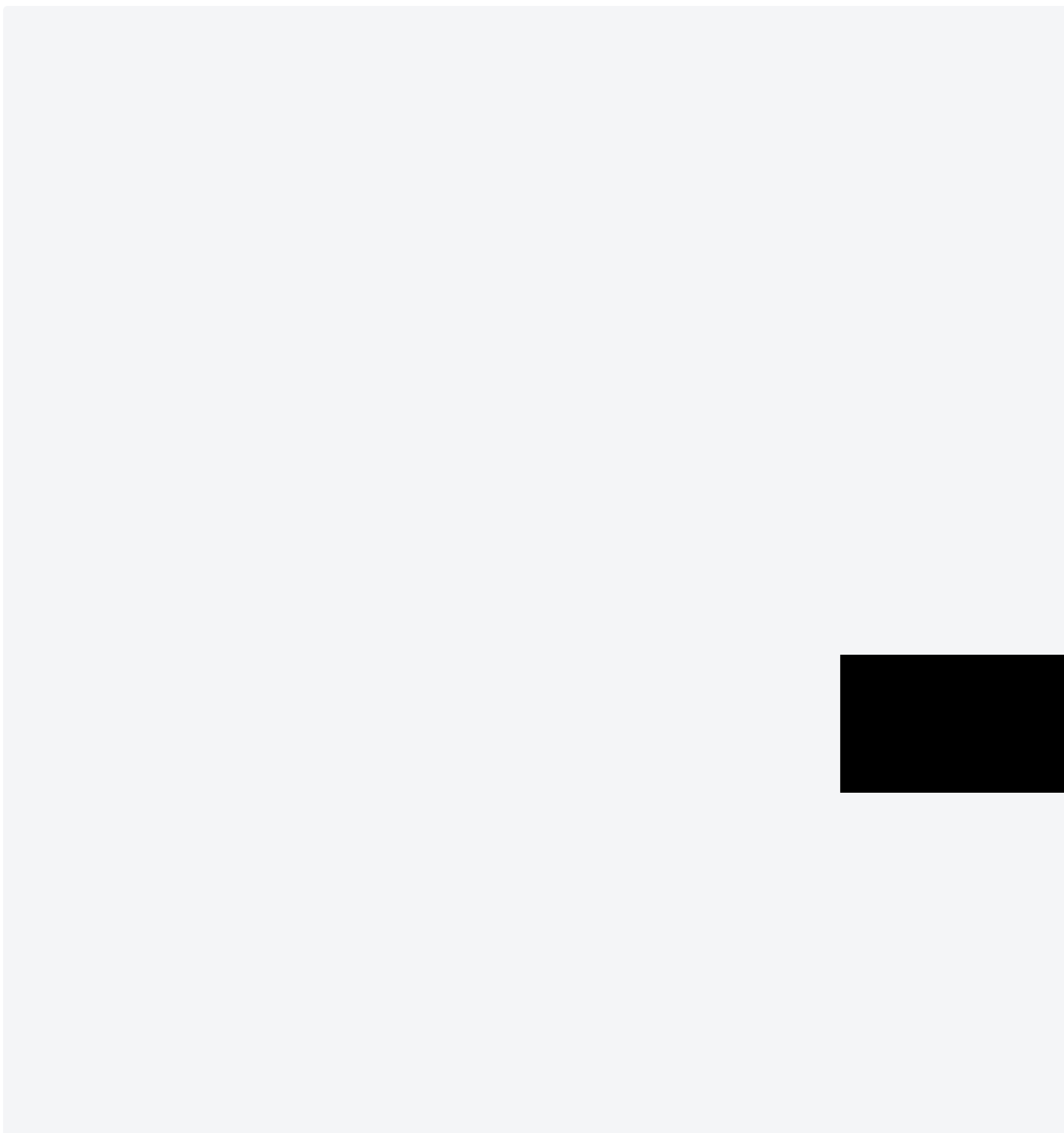
or specify license file:

JAVA_ARG

$S = \langle \dots \rangle$

-D









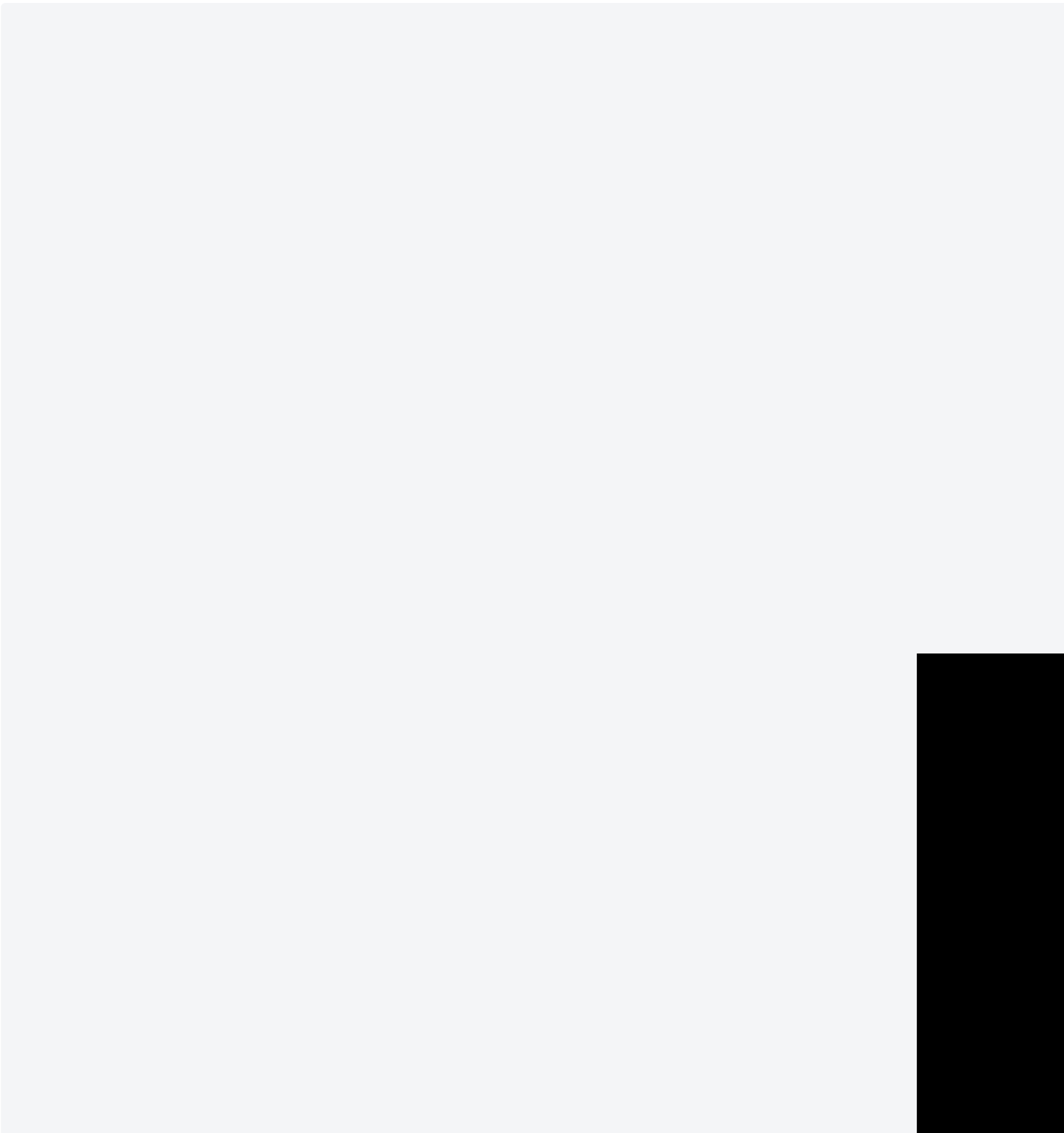














\="c:/my

licenses

/

licenseF

ile.txt"

Getting licensing information

Use the `com.nomagic.magicdraw.license.utils.LicenseUtils` to check some licensing information (edition, is it a trial or evaluation version).

Related pages

- [Configure test environment \(see page 35\)](#)
- [Specifying batch mode program classpath and required system properties \(see page 208\)](#)

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

- [Idle job service \(see page 362\)](#)

- [Running action with progress \(see page 65\)](#)

Idle job service

The *com.nomagic.magicdraw.job.IdleJobService* allows to add *com.nomagic.magicdraw.job.Job(s)* which can be executed when a program is idle. The program decides when it is the most appropriate to execute the *Job* 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 `<installation_directory>\openapi\examples\JobExample`.

Related pages

- [Multi-threading \(see page 361\)](#)

Application environment

Use `com.nomagic.magicdraw.core.Application.runtime()` to retrieve a `com.nomagic.magicdraw.core.Application.Runtime` object. It contains an application name, version, and file format information.

Use `Application.environment()` to retrieve an `com.nomagic.magicdraw.core.Application.Environment` object. It contains application environment directory information like install, configuration, data, templates, profiles, etc.

Attached Files

Use `com.nomagic.magicdraw.fileattachments.FileAttachmentsHelper` to create, remove or save attached files to disk.



You can find the code examples in

- `<installation_directory>\openapi\examples\fileattachments`

Element Referencing in Texts

Open API provides the `com.nomagic.magicdraw.elementreferenceintext.ElementReferencingInTexts` 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(BaseElement, java.lang.String)`,
- `ElementReferencingInTexts.createReferencingText(BaseElement, java.lang.String, DisplayMode)`⁸⁵,
- `ElementReferencingInTexts.createReferencingText(BaseElement, java.lang.String, DisplayMode, UpdateMode)`

```
//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.
```

⁸⁵<http://jdocs.nomagic.com/2024x/com/nomagic/magicdraw/elementreferenceintext/ElementReferencingInTexts.html#createReferencingText-com.nomagic.magicdraw.uml.BaseElement-java.lang.String-com.nomagic.magicdraw.elementreferenceintext.DisplayMode->

```
//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)*.

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 the text 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);
```

Creating custom drag and drop handlers

MagicDraw itself has many default drag and drop handlers for doing various things. Such handlers consist of :

- Dropping elements from various UI components(trees, specification windows, etc.) diagram or its presentation elements.
- Dropping external application items onto trees or diagrams.

Default drag and drop handlers for example create new presentation elements, change the properties of elements, external application image drop on diagram creates an image and etc. Same is true if presentation elements are dropped on top of other presentation elements.

API allows to register a custom drag and drop handler for doing some specific things using drag and drop mechanism.

OpenAPI drag and drop handler registrations:

- Use `com.nomagic.magicdraw.ui.dnd.CustomDropDiagramHandlerFactory.register(com.nomagic.magicdraw.ui.dnd.CustomDragAndDropHandlerFactory)` to register a factory which creates a handler for dropping elements from browser(trees) into diagram.
- Use `com.nomagic.magicdraw.ui.dnd.CustomShapeMoveHandlerFactory.register(com.nomagic.magicdraw.ui.dnd.CustomDragAndDropHandlerFactory)`⁸⁶ to register a factory which creates a handler for dropping one presentation element on other presentation element in the same diagram
- Use `com.nomagic.magicdraw.ui.dnd.BrowserTabTreeDragAndDropHandlerRegistry#register(com.nomagic.magicdraw.ui.dnd.BrowserTabTreeDragAndDropHandlerFactory)` to register a factory which creates a handler for dropping external application items into the browser tree nodes(elements).
- Use `com.nomagic.magicdraw.ui.dnd.DiagramTransferableDragAndDropHandlerRegistry#register(com.nomagic.magicdraw.ui.dnd.DiagramTransferableDragAndDropHandlerFactory)` to register a factory which creates a handler for dropping external application items into the diagram.



You can find the code examples in

- `<installation_directory>\openapi\examples\customdraganddrop`

SysML Developer Guide

Standard stereotypes in SysML plugin are defined in SysML Profile and MD Customization for SysML Profile. Both profiles have their corresponding API classes:

[com.nomagic.magicdraw.sysml.util.SysMLProfile](#)⁸⁷ and

[com.nomagic.magicdraw.sysml.util.MDCustomizationForSysMLProfile](#)⁸⁸, respectively. Each class allows you to:

- Get a string constant for each property of stereotype (tag).
- Get a stereotype element.
- Check if an element is stereotyped.

Find JavaDoc of SysML plugin in `<modeling tool installation directory>/openapi/docs/sysml_javadoc.zip`



For example, `<modeling_tool_version_number>_no_install\openapi\docs\sysml_javadoc.zip`

Find JavaDoc of Requirements plugin in `<modeling tool installation directory>/openapi/docs/requirements_javadoc.zip`



For example, `<modeling_tool_version_number>_no_install\openapi\docs\requirements_javadoc.zip`

Extract a .jar file and double-click the `index.html` to open it.

⁸⁶ <http://jdocs.nomagic.com/2024x/com/nomagic/magicdraw/ui/dnd/CustomShapeMoveHandlerFactory.html#register-com.nomagic.magicdraw.ui.dnd.CustomDragAndDropHandler->

⁸⁷ <https://jdocs.nomagic.com/2024xRefresh3/com/nomagic/magicdraw/sysml/util/SysMLProfile.html>

⁸⁸ <https://jdocs.nomagic.com/2024xRefresh3/com/nomagic/magicdraw/sysml/util/MDCustomizationForSysMLProfile.html>

Related pages

- [SysML Profile \(see page 366\)](#)
- [MD Customization for SysML Profile \(see page 367\)](#)
- [SysML Profile API Changes \(see page 367\)](#)
- [SysML classes for open API \(see page 368\)](#)

Related docs

- [Modeling Tools Developer Guide \(see page 9\)](#)

SysML Profile

You need to import *com.nomagic.magicdraw.sysml.util.SysMLProfile* to use this API class.

Get a string constant for each property of stereotype (tag)

Usage includes "SysMLProfile.Stereotype.STEREOTYPE_NAME".

For example, SysMLProfile.RequirementStereotype.ID returns a string of "Id".

Get a stereotype element

Usage includes:

"SysMLProfile.getInstance(project).stereotype().getStereotype()" where project refers to a project that uses SysML Profile.

"SysMLProfile.getInstance(element).stereotype().getStereotype()" where element refers to the element in a project that uses SysML Profile.

For example, SysMLProfile.getInstance(project).block().getStereotype() returns the reference to the «Block» stereotype object.

Check if an element is stereotyped

Usage includes SysMLProfile.Stereotype.isInstance(Elem) where "Elem" is the element you want to check.

For example, given an element "Elem", SysMLProfile.BlockStereotype.isInstance(Elem) returns *True* if the element "Elem" has «Block» stereotype applied and returns *False* otherwise.

MD Customization for SysML Profile

You need to import *com.nomagic.magicdraw.sysml.util.MDCustomizationForSysMLProfile* to use this API class.

Get a string constant for each property of stereotype (tag)

Usage includes "MDCustomizationForSysMLProfile.STEREOTYPE_PROPERTY_NAME".

For example, MDCustomizationForSysMLProfile.NUMBEROWNER_PREFIX_PROPERTY returns a string of "prefix".

Get a stereotype element

Usage includes:

- "MDCustomizationForSysMLProfile.getInstance(project).getStereotype()" - where project refers to the project which uses MD Customization for SysML Profile.
- "MDCustomizationForSysMLProfile.getInstance(element).getStereotype()" - where element refers to the element in the project which uses MD Customization for SysML Profile.

For example, MDCustomizationForSysMLProfile.getInstance(project).getPartProperty() returns the reference to the «PartProperty» stereotype object.

Check if an element is stereotyped

Usage includes "MDCustomizationForSysMLProfile.isStereotype(Elem)" - where Elem is the element you would like to check.

For example, given an element "Elem", MDCustomizationForSysMLProfile.isValueProperty(Elem) returns *True* if the element "Elem" has «ValueProperty» stereotype applied, and returns *false* otherwise.

SysML Profile API Changes

SysML Profile API changes were made in relation to the SysML 1.4 support.

The following constants were moved from the *com.nomagic.magicdraw.sysml.util.SysMLProfile* to *com.nomagic.magicdraw.sysml.util.MDCustomizationForSysMLProfile*:

```
public static final String CONSTRAINTPROPERTY_STEREOTYPE = "ConstraintProperty";  
  
public static final String QUANTITYKIND_STEREOTYPE = "QuantityKind";
```

```
public static final String QUANTITYKIND_DEFINITIONURI_PROPERTY = "definitionURI";

public static final String QUANTITYKIND_DESCRIPTION_PROPERTY = "description";

public static final String QUANTITYKIND_SYMBOL_PROPERTY = "symbol";

public static final String UNIT_STEREOTYPE = "Unit";
```

The following methods were moved from the *com.nomagic.magicdraw.sysml.util.SysMLProfile* to the *com.nomagic.magicdraw.sysml.util.MDCustomizationForSysMLProfile*:

```
getConstraintProperty()

getQuantityKind()

getUnit()

isQuantityKind()

isUnit()

isConstraintProperty()
```

The constant `NESTEDCONNECTOREND_PROPERTYPATH_PROPERTY` changed to `ELEMENTPROPERTYPATH_PROPERTYPATH_PROPERTY`.

SysML classes for open API

Classes which are available for open API are included in SysML plugin open API documentation. Find these in *<SysML plugin installation directory>\openapi\docs*.

The *com.nomagic.magicdraw.sysml.util.SysMLUtilities* class was added to the open APIs. It provides utility methods for easier work with SysML projects.

Methods and classes marked as deprecated do not support the development of external plugins.

Cameo Simulation Toolkit Developer Guide

Introduction

Java APIs

Introduction

This document describes the Application Programming Interface (API) of Cameo Simulation Toolkit, which is categorized into two fundamental APIs: (i) Java API and (ii) Action Scripts API.

The Java API is mostly used together with the MagicDraw plugin and the Action Script API is used independently for each supported model element in a MagicDraw project. All of the Cameo Simulation Toolkit API classes are packaged in the jar file *simulation_api*, which is located in

- *<MagicDraw installation directory>/plugins/com.nomagic.magicdraw.simulation/simulation_api.jar*

You can find the Cameo Simulation Toolkit JavaDoc file in

- URL: <https://jdocs.nomagic.com/2024xRefresh3/com/nomagic/magicdraw/simulation/package-summary.html>
- *<MagicDraw installation directory>/openapi/docs/simulation/SimulationJavaDoc.zip*

For more information about creating a new MagicDraw plugin, see the MagicDraw Open API user guide located in

- *<MagicDraw installation directory>/openapi/docs/MagicDraw OpenAPI UserGuide.pdf*

Note

Before going through each example given in this document, make sure that *simulation.jar* and *simulation_api.jar* has been added to your IDE classpath.

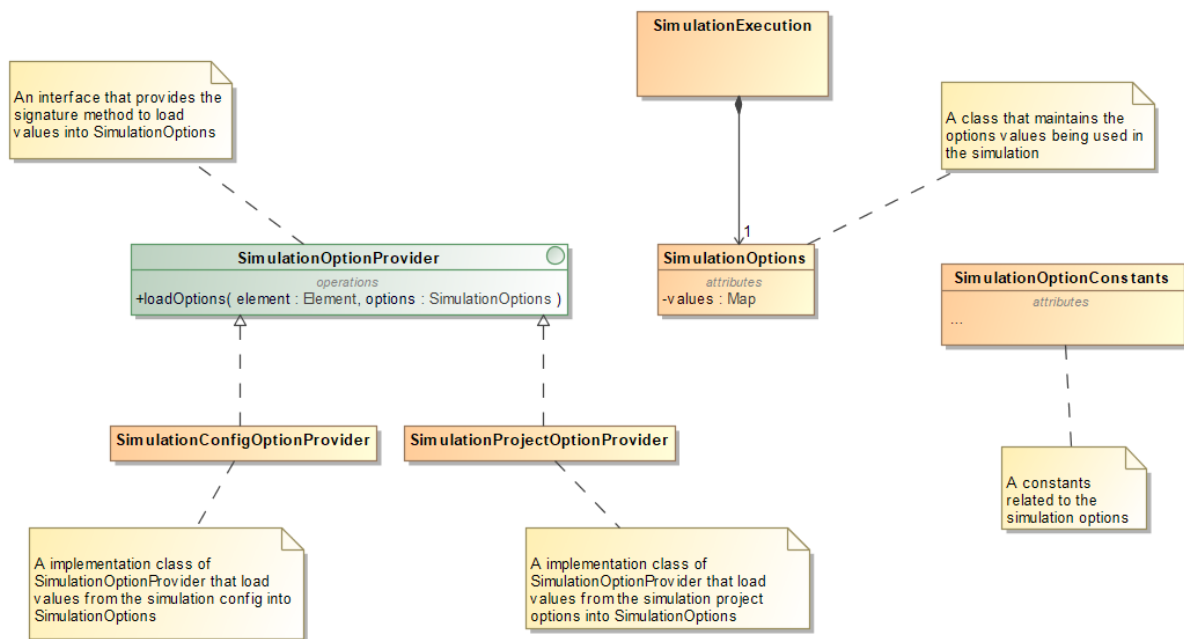
SimulationOptionProvider interface

Simulation provides an internal API so that execution can have all simulation options from the **SimulationOptionProvider** interface as shown in the figure below. You can use the Simulation API independently in Teamwork Cloud or other platforms without modeling tool environment, e.g., MagicDraw.

Implementation Classes of the **SimulationOptionProvider** interface are **SimulationConfigOptionProvider** (for [Simulation Config](#)⁸⁹) and **SimulationProjectOptionProvider** (for [project options](#)⁹⁰), both of which will be used in the **SimulationOptions** Class being deployed in the **SimulationExecution** Class. The **SimulationOptionConstants** Class also contains all option constant variables used in Simulation, e.g., Active Color, Silent, etc.

⁸⁹ <https://docs.nomagic.com/display/CST2024xR3/SimulationConfig+stereotype>

⁹⁰ <https://docs.nomagic.com/display/CST2024xR3/Project+options>



SimulationOptionProvider interface.

Information

You can find more details about the Simulation JavaDoc file at *<MagicDraw installation directory>/openapi/docs/simulation/SimulationJavaDoc.zip*.

Java APIs

With Java APIs you can [start the execution](#) (see page 371), [stop the execution](#) (see page 371), and [create and register a new Simulation Execution Listener](#) (see page 371) to listen to various events during the execution. [fUML Helper](#) (see page 379) is a convenience class for methods related to the fUML structures.

Information

For more information about Cameo Simulation Toolkit JavaDoc files, visit <https://jdocs.nomagic.com/2024xRefresh3/com/nomagic/magicdraw/simulation/package-summary.html>.

Related pages

- [Execution](#) (see page 371)
- [Engine](#) (see page 374)
- [fUML Helper](#) (see page 379)
- [Parametric Helper as executing parametric simulation from an Activity](#) (see page 379)
- [Display Values in Diagrams](#) (see page 380)
- [API Changes](#) (see page 383)

Execution

A simulation execution represents a model execution of a specific element. It is the top level of the model execution which contains running execution sessions and execution engines. A simulation execution will start when the specific element is executed (through the graphical user interface or Java API). The following APIs are available for handling executions.

- Start Execution
- Stop Execution
- Creating and Registering a New Execution Listener

Related pages

- [Starting execution \(see page 371\)](#)
- [Stopping execution \(see page 371\)](#)
- [Creating and registering a new SimulationExecutionListener \(see page 371\)](#)

Starting execution

With Cameo Simulation Toolkit, you can start executing an element using the following Open APIs

- `SimulationManager.execute(Element element) : SimulationSession`
- `SimulationManager.execute(Element element, Boolean isMainSession, Boolean start) : SimulationSession`

Once you use the aforementioned Open APIs, Cameo Simulation Toolkit creates corresponding simulation sessions. The conditions when using the Open APIs to execute an element are as follows

- The specified element could be any executable element, such as a Class, a Behavior, or an simulation configuration element.
- If the executed element is intended to be the main element of the execution, set the "isMainSession" flag to true. Then, the created session will become the main session of the execution. The main session is the first session created once the execution starts. If the main session is terminated, all executions will be terminated accordingly.
- If the "start" flag is set to true, the created session will start itself automatically (similar to the 'autorun' option in an SimulationConfig).

Stopping execution

You can stop an execution by terminating its main session. Cameo Simulation Toolkit allows you to terminate any simulation session using the following Open API

- `SimulationManager.terminateSession(SimulationSession session) : void`

Terminating a session will also cause all children of the specific session to be terminated.

Creating and registering a new SimulationExecutionListener

A *SimulationExecutionListener* Class is a listener for Events that will be activated during execution of a model. All available Events are listed below.

- An execution is started.
- An element is activated.

- An element is deactivated.
- A signal event is triggered.
- An operation is called.
- A behavior is called.
- A runtime object is created.
- An execution is terminated.

```
public class SimulationExecutionListener {
    void executionStarted(SimulationExecution execution);
    void elementActivated(Element element, Collection<?> values);
    void elementDeactivated(Element element, Collection<?> values);
    void eventTriggered(SignalInstance signal);
    void operationCalled(Operation operation, ParameterValueList pvl, Object_
caller, Object_ target, boolean isSynchronous);
    void behaviorCalled(Behavior behavior, ParameterValueList pvl, Object_
caller, Object_ target, boolean isSynchronous);
    void objectCreated(Object_ sender, Object_ object);
    void valueChange(StructuredValue context, FeatureValue feature, Object
oldValue, Object newValue);
    void executionTerminated(SimulationExecution execution);
    void configLoaded(Element config);
    void busyStatusChange(StructuredValue context, Object oldValue, Object
newValue);
}
```

Once you have created the execution listener, you can register it to a list of global execution listeners.

To register a new execution listener to a global list, type the following code

```
SimulationManager.registerSimulationExecutionListener(listener); //listener is an
instance of SimulationExecutionListener.
```

All registered listeners will be cleared and removed automatically when the execution is terminated.

Note

ExecutionAdapter and *ExecutionListener* are not valid in Cameo Simulation Toolkit 19.0 and in later versions.

All open APIs below are used by the Simulation Dashboard plugin and can be added for particular purposes.

Open API	Function
beforeObjectDestroyed(Object_ object)	Works with an object-destroying Listener when added into SimulationExecution Listener.
objectStateActivated(StructuredValue context, State newState)	Works with a State activation Listener when added into SimulationExecution Listener.
SimulationExecutionListener.contextInitialized(SimulationExecution execution)	Gets an Event when objects have been initialized but not started the execution yet.
SimulationHelper.findFeatureValue(Object_ runtimeContext, List<Property> propertyPath, Property target) : List<FeatureValue>	Gets featureValue objects that match the given nested property path.
SimulationHelper.getAvailableSignals(StructuredValue target, ExecutionEngine engine)	Retrieves available Signals from the specified structured value.
SimulationHelper.getConfigElement(SimulationExecution execution)	Gets a Config element.
SimulationHelper.isFeatureValueInvalid(FeatureValue featureValue, int index)	Checks the validity of FeatureValue.
SimulationManager.getContext(SimulationSession session) : Object_	Gets the execution context.

Open API	Function
SimulationManager.registerSimulationExecutionListener(SimulationExecutionListener listener, SimulationExecution execution)	Registers SimulationExecution Listener dynamically during the execution.
SimulationManager.unregisterSimulationExecutionListener(SimulationExecutionListener listener) SimulationManager.unregisterSimulationExecutionListener(SimulationExecutionListener listener, SimulationExecution execution)	Unregisters SimulationExecution Listener.

Engine

ExecutionEngine defines how a Behavior e.g. State Machine is executed, including the full logic how their inner elements (e.g. States) are activated.

Implementing custom ExecutionEngine and registering it's ExecutionEngineDescriptor enables overriding the default behavior of the Cameo Simulation Toolkit for specific:

- StateMachine
- Interaction
- Activity

Related pages:

- [Creating a new execution engine \(see page 374\)](#)
- [Creating an execution engine descriptor \(see page 375\)](#)
- [Creating an execution engine listener \(see page 376\)](#)
- [Registering an execution engine to the Simulation Manager \(see page 377\)](#)
- [Activating and deactivating an element \(see page 378\)](#)
- [Triggering an Event \(see page 378\)](#)
- [Printing messages in Simulation Console \(see page 378\)](#)

Creating a new execution engine

When creating an execution engine, you are required to implement three abstract methods in the Subclass of ExecutionEngine (see JavaDoc for more details). These abstract methods are as follows

```
public abstract class ExecutionEngine {
    ...
    public abstract void init(Element element);
    public abstract void execute(Element element);
    public abstract void onClose();
    ...
}
```

API developers can insert any desired code into the above abstract methods and get the events when the execution engine is initialized, executed, and terminated. You can execute the following code to create a MyExecutionEngine class

```
public class MyExecutionEngine extends ExecutionEngine {

    public MyExecutionEngine(ExecutionEngineDescriptor engineDescriptor) {
        super(engineDescriptor);
    }

    @Override
    public void execute(Element element) {
        ...
    }

    @Override
    public void init(Element element) {
        Project project = Project.getProject(element);
        // add engine listener if needed
        EngineListener listener = new MyEngineListener(project);
        addEngineListener(listener);
    }

    @Override
    public void onClose() {
        ...
    }
}
```

The created class MyExecutionEngine must be returned from createEngine() of its descriptor as defined in [Creating an execution engine descriptor](#) (see page 375).

Creating an execution engine descriptor

The following example shows how to create an Execution Engine Descriptor

```
public interface ExecutionEngineDescriptor {
    String getEngineName();
    ImageIcon getEngineIcon();
    boolean canExecute(Element element);
    ExecutionEngine createEngine();
    boolean canAnimate(PresentationElement element);
    boolean isAutoDiagramOpened();
    boolean canDebug();
    boolean canUserTriggerEvents();
    boolean isDiagramPerSession();
    boolean isHasIdle();
    boolean isFindEngine(Element element);
}
```

```
List<? Extends ValidationSuite> getValidationSuite();
}
```

The new execution engine descriptor must implement the `ExecutionEngineDescriptor` or extend the `AbstractExecutionEngineDescriptor` abstract Class. You can find details about these Classes in JavaDoc.

```
public class MyExecutionEngineDescriptor extends
AbstractExecutionEngineDescriptor {
    @Override

    public boolean canExecute(Element element) {
        return element instanceof Interaction;
    }

    @Override
    public ExecutionEngine createEngine() {
        return new MyExecutionEngine(this);
    }

    @Override
    public String getEngineName() {
        return "My Execution Engine";
    }
}
```

The created execution engine descriptor must then be registered to the simulation manager (see [Registering an execution engine to the Simulation Manager \(see page 377\)](#)).

Creating an execution engine listener

The following example shows how to create an engine listener

```
public class MyEngineListener implements EngineListener {

    private Project project;
    public MyEngineListener(Project project) {
        this.project = project;
    }

    @Override
    public void elementActivated(Element element, Collection<?> values) {
        System.out.println("--Activate Element-- : element name =
" ((NamedElement)element).getName());
    }

    @Override
    public void elementDeactivated(Element element, Collection<?> values) {
```

```

        System.out.println("--Deactivate Element-- : element name = " +
((NamedElement)element).getName());
    }

    @Override
    public void eventTriggered(String eventID) {
        NamedElement element = (NamedElement)project.getElementByID(eventID);
        System.out.println("--Event Trigger-- : event name = " +
((NamedElement)element).getName());
    }

    @Override
    public void executionTerminated() {
        System.out.println("--Engine Terminated--");
    }
}

```

Once you have created an `EngineListener`, you can register it to the specified `ExecutionEngine` if you want to receive events occurring in each execution engine. All engine listeners of a specific engine will be activated under the conditions as follows

- An element is activated.
- An element is deactivated.
- An event is triggered.
- An engine is terminated.

```

public interface EngineListener {
    void elementActivated(Element element, Collection<?> values);
    void elementDeactivated(Element element, Collection<?> values);
    void eventTriggered(String eventID);
    void executionTerminated();
}

```

See [Creating a new execution engine \(see page 374\)](#) for more information about adding execution engine listeners to the `MyExecutionEngine` class.

Note

You can add more than one execution engine listener to an execution engine.

Registering an execution engine to the Simulation Manager

To register your new execution engine to the Simulation Manager, pass a new instance of your execution engine descriptor to the Simulation Manager by typing the following code

```

SimulationManager.registerEngine(new MyEngineDescriptor());

```

In general, a new execution engine is registered at <Plugin Class>.init().

Activating and deactivating an element

An execution engine can activate a specific element with values by calling an `activateElement(Element element, Collection values)` of the `ExecutionEngine`. To deactivate the element, it will call a `deactivateElement(Element element, Collection values)` of the same class.

These two methods are mostly called in the `ExecutionEngine.execute(element)`. For example, if an Activity element on an Activity diagram is passed from the `ExecutionEngine.execute(element)`, we use the `activateElement()` and `deactivateElement()` methods to activate and deactivate the Actions in that particular Activity respectively.

If `activateElement()` or `deactivateElement()` is called to a specific element, it will trigger all registered engine listeners including both predefined and user-defined listeners. For example, an `AnimationListener` is a predefined listener that is the Subclass of an `EngineListener`. It is used to display a specific element animation (in highlight) on a diagram in `MagicDraw`.

Triggering an Event

An execution engine can trigger a given Event by calling the `triggerEvent(String event)` of an `ExecutionEngine`. Just like the `activateElement()` and `deactivateElement()` methods, this method is mostly used in the `ExecutionEngine.execute(element)`.

If the `triggerEvent(String event)` is called, the `eventTriggered(String eventID)` of all registered engine listeners will be activated.

Printing messages in Simulation Console

You can print different types of messages on the **Simulation Console** pane by using the following codes

- `SimulationManager.logConsoleDebug (Project project, String message)`
Displays debugging information.
- `SimulationManager.logConsoleInfo (Project project, String message)`
Displays information in general.
- `SimulationManager.logConsoleWarn (Project project, String message)`
Displays warning messages.
- `SimulationManager.logConsoleError (Project project, String message)`
Displays error messages.

fUML Helper

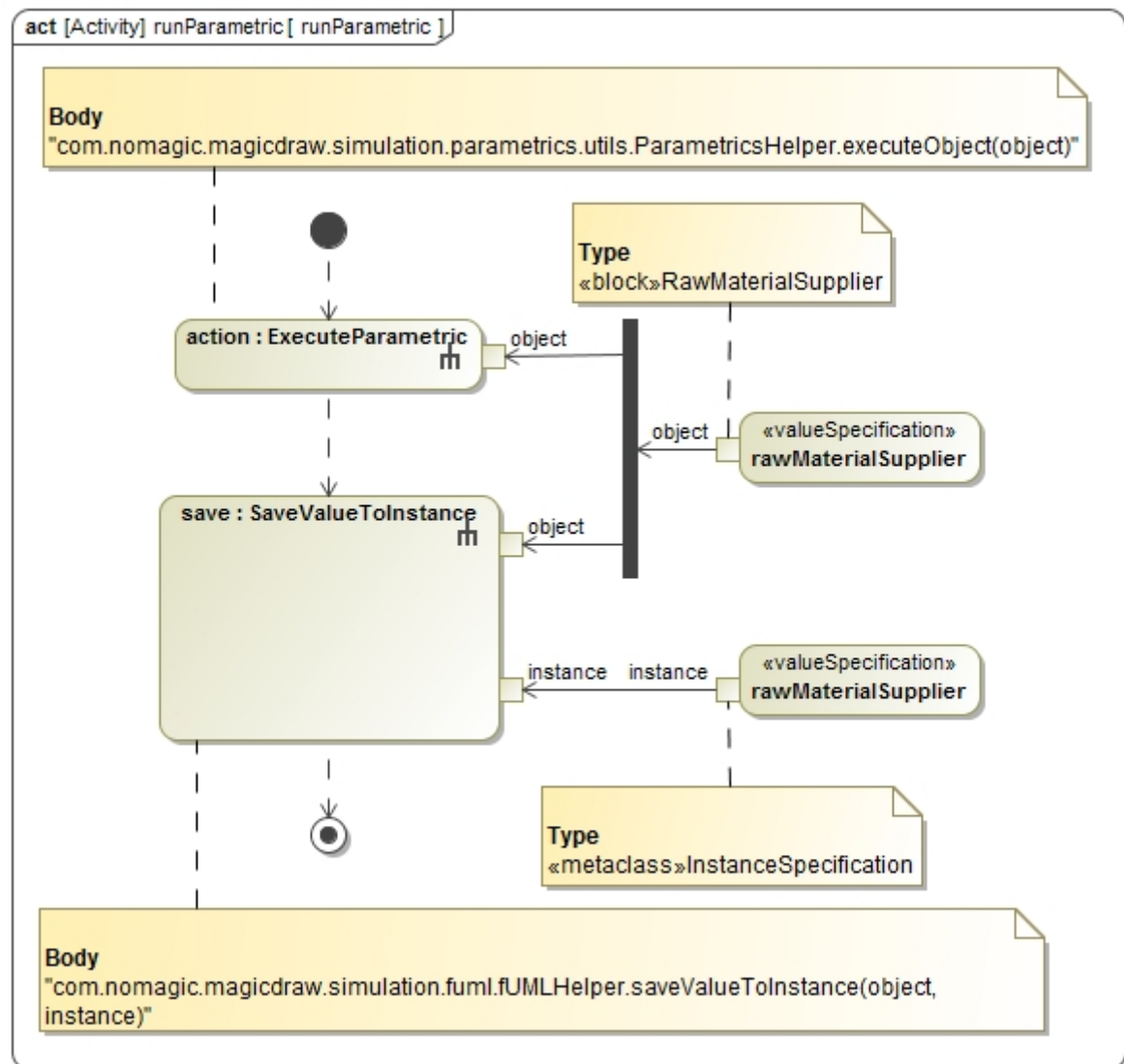
An fUMLHelper is a Class that provides Helper methods related to the fUML structures. See JavaDoc for more details about this Class.

Parametric Helper as executing parametric simulation from an Activity

The Parametric engine provides an API for a parametric execution with a runtime object of a Classifier. The runtime object of the Classifier will be passed to the API as an argument and the engine will execute the given object. With this API, you can use a scripting language to execute the parametric simulation, shown below:

```
com.nomagic.magicdraw.simulation.parametrics.utility.ParametricsHelper.executeObject(  
Object object);
```

An argument object is a runtime object of a classifier to be executed. To obtain this particular runtime object, you can use some UML actions, e.g., ReadSelfAction, ReadStructuralFeatureValueAction, ValueSpecificationAction, or the Cameo Simulation Toolkit Open API. The following figure shows the Parametric Activity diagram in the *CylinderPipe.mdzip* sample. The **action : ExecuteParametric** is used to run the parametric execution. The runtime object, which will be executed, is obtained from the value Specification Action rawMaterialSupplier.



The Parametric Activity diagram in the CylinderPipe.mdzip sample.

Display Values in Diagrams

Displaying Runtime Values

When any native model execution is not running in the Cameo Simulation Toolkit, an external tool executes the model. There is a capability to dynamically display any custom value string on parts and/or ports in structure diagrams, without modifying the model in Cameo Simulation Toolkit. Currently, only parts and ports that are typed with a primitive data type are supported.

Start by extending *com.nomagic.magicdraw.simulation.dashboard.api.ExternalValueProvider* class to create a custom implementation. The returned custom values will be displayed on the diagrams representing the *context* for which the *ExternalValueProvider* instance was constructed. The model is not modified in any way when these values are displayed. The color in which the values are displayed is configurable via the **Runtime Value Text Color on Shapes** project option. In the image below, two

such values are provided and displayed in a diagram of the *Transmission.mdzip* sample project: **30.0 mi/h** for **carSpeed** and **19.0** for **wheelDiameter**.

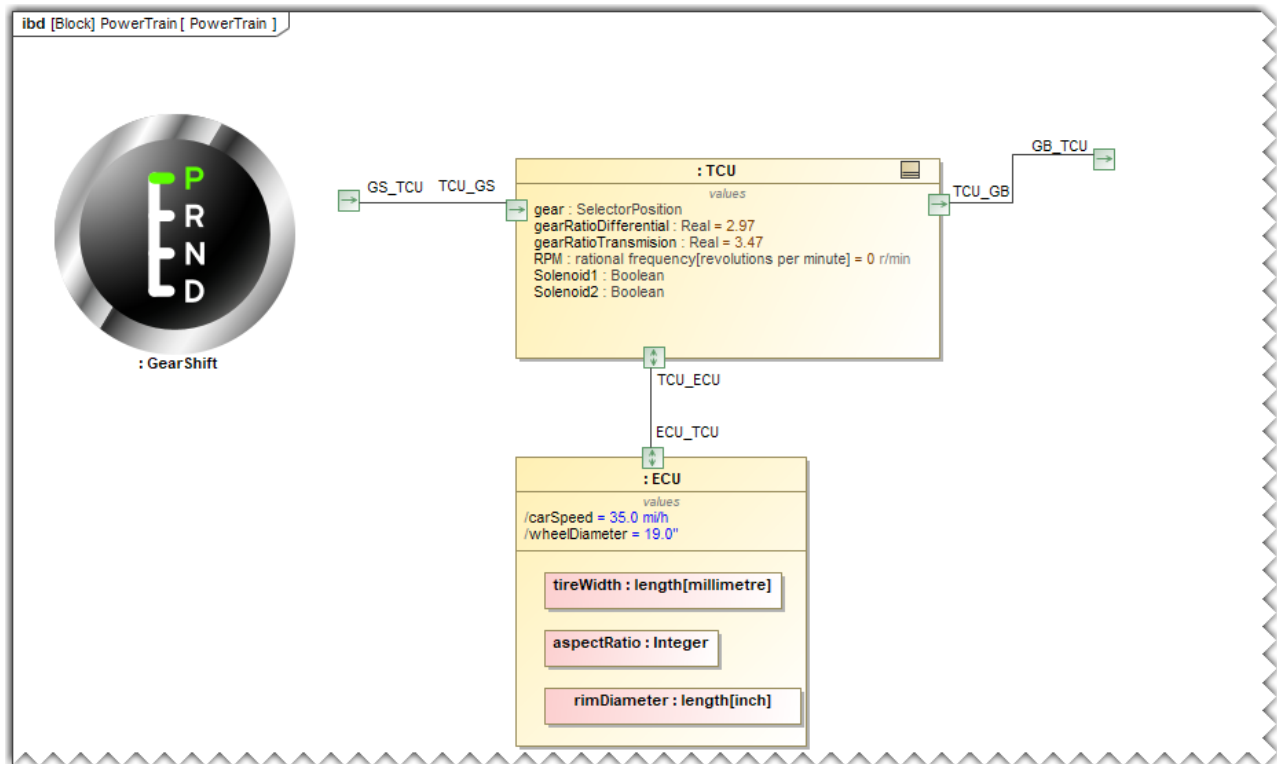


Image displaying the runtime values

Below is the implementation that was designed specifically for the *Transmission.mdzip* sample and provides just the two values seen in the above image:

```
//Engine Control Unit refers to ECU block in the Transmission sample project
public class EngineControlUnitValueProvider extends ExternalValueProvider
{
    private final Element carSpeedElement;
    private final Element wheelDiameterElement;

    //Mimics some model execution process that happens outside this tool and produces
    values to display
    private final MockExternalExecution externalExecution;

    public EngineControlUnitValueProvider(Element context)
    {
        super(context);
        Project project = Project.getProject(context);
        carSpeedElement = FinderByQualifiedNames.getInstance().find(project,
"Transmission::Structure::ECU::carSpeed", Property.class);
        wheelDiameterElement = FinderByQualifiedNames.getInstance().find(project,
"Transmission::Structure::ECU::wheelDiameter", Property.class);
        externalExecution = new MockExternalExecution(this);
    }
}
```

```

    }

    @CheckForNull
    @Override
    public String getValueText(Element element, List<? extends Element>
pathInContext)
    {
        // values should be pre-collected for diagram related performance reasons.
        // Avoid time demanding implementations (e.g. remote calls) of this method
        if (element == carSpeedElement)
        {
            return externalExecution.getCarSpeed() + " mi/h";
        }
        else if (element == wheelDiameterElement)
        {
            return externalExecution.getWheelDiameter() + "\""; // double quote to
            // represent inches
        }
        return null; // do not provide custom value for other elements in this
        // context
    }

    /**
     * To be called every time speed changes to call {@link #getValueText(Element,
     * List)} for that element again
     * @see #invalidate(Element)
     * @see #invalidateAll()
     */
    public void invalidateCarSpeed()
    {
        invalidate(carSpeedElement);
    }
}

```

To have the effect, an instance of the implemented provider must be registered to the *ExternalValueProviders* service, as illustrated here:

```

Project project = <...>
// The root context of the external execution. Values will only be visible in the
// Diagrams that have this element set as context
Classifier context = Finder.byQualifiedName().find(project,
"Transmission::Structure::PowerTrain", Class.class);
if (context != null)
{
    EngineControlUnitValueProvider valueProvider = new
    EngineControlUnitValueProvider(context);
    ExternalValueProviders.getInstance(project).register(valueProvider);

    // if later the provider needs to be unregistered, it should be done by
    // calling:
    // ExternalValueProviders.getInstance(project).unregister(valueProvider);
    // Since providers are registered for specific open project, there is no
    // obligation to unregister manually on project close.
}

```

i The full code example is available in `<installation_directory>\openapi\examples\externalvalueindiagrams`.

Animating flowing information

Additionally, it is possible to display animation for the information that flows via connector from one end to another. For a fixed amount of time, a banner with the custom value will move from the source towards the target. During the full duration of the animation, the connector presentation element will be highlighted in color which is controlled by a project option **Active Color**.

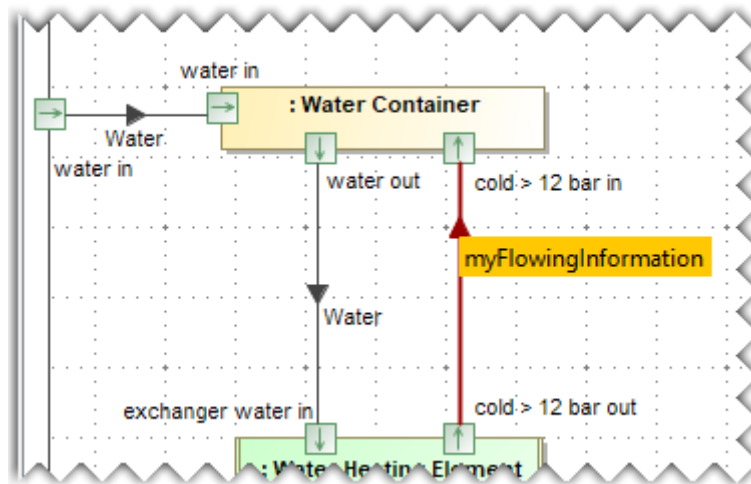


Image displaying the animation in between the connectors

```
Connector connector = <..> // select connector via which to animate
ConnectorEnd sourceEnd = selectSourceEnd(connector); // pick end from
which to start the animation
String flowingInformation = "myFlowingInformation"; // hardcoded for
example, should be some actual data compatible with end/connector/flow types
FlowingInformationDescriptor descriptor = new
FlowingInformationDescriptor(flowingInformation, connector, sourceEnd);

long duration = 4000; // 4 seconds to reach the other end

FlowAnimation.getInstance(Project.getProject(connector)).animate(descriptor,
duration, finished -> { /* do something when animation completes */ });
```


i The full code example for this feature is available in `<installation_directory>\openapi\examples\flowanimation`.

API Changes

The following table lists the old and new values of the constants for SimulationProfile, SimulationOptions, and SysMLConstants.

Old values up to 2022x Refresh2	New values from 2024x
com.nomagic.magicdraw.simulation.SimulationOptionConstants were removed altogether, instead, use com.nomagic.magicdraw.simulation.execution.SimulationOptions	
SimulationOptionConstants.ACTIVE_COLOR_NAME	SimulationOptions.ACTIVE_COLOR
SimulationOptionConstants.ADD_CONTROL_PANEL_NAME	SimulationOptions.ADD_CONTROL_PANEL
SimulationOptionConstants.ANIMATION_SPEED_NAME	SimulationOptions.ANIMATION_SPEED
SimulationOptionConstants.AUTOSTART_ACTIVE_OPTIONS_NAME	SimulationOptions.AUTO_START_ACTIVE_OPTIONS
SimulationOptionConstants.AUTO_CREATE_FUML_OBJECT_OF_OUTPUT_PIN_NAME	SimulationOptions.AUTO_CREATE_FUML_OBJECT_OF_OUTPUT_PIN
SimulationOptionConstants.AUTO_OPEN_DIAGRAMS_NAME	SimulationOptions.AUTO_OPEN_DIAGRAMS
SimulationOptionConstants.AUTO_START_NAME	SimulationOptions.AUTO_START
SimulationOptionConstants.BREAKPOINT_COLOR_NAME	SimulationOptions.BREAKPOINT_COLOR
SimulationOptionConstants.CHECK_MODEL_BEFORE_EXECUTION	SimulationOptions.CHECK_MODEL_BEFORE_EXECUTION
SimulationOptionConstants.CLONE_REFERENCES_NAME	SimulationOptions.CLONE_REFERENCES
SimulationOptionConstants.CONSTRAINT_FAILURE_AS_BREAKPOINT_NAME	SimulationOptions.CONSTRAINT_FAILURE_AS_BREAKPOINT

SimulationOptionConstants.DECIMAL_PLACES_NAME	SimulationOptions.DECIMAL_PLACES
SimulationOptionConstants.DEFAULT_LANGUAGE_NAME	SimulationOptions.DEFAULT_LANGUAGE
SimulationOptionConstants.DEFAULT_PARAMETRIC_EVALUATOR_NAME	SimulationOptions.DEFAULT_PARAMETRIC_EVALUATOR
SimulationOptionConstants.DURATION_SIMULATION_MODE_NAME	SimulationOptions.DURATION_SIMULATION_MODE
SimulationOptionConstants.ENDTIME_NAME	SimulationOptions.ENDTIME
SimulationOptionConstants.ENGINES_PRIORITY_NAME	SimulationOptions.ENGINES_PRIORITY
SimulationOptionConstants.EXCLUDED_ELEMENTS_NAME	SimulationOptions.EXCLUDED_ELEMENTS
SimulationOptionConstants.EXECUTION_LISTENERS_NAME	SimulationOptions.EXECUTION_LISTENERS
SimulationOptionConstants.EXECUTION_TARGET_NAME	SimulationOptions.EXECUTION_TARGET
SimulationOptionConstants.EXTERNAL_LIBRARIES_NAME	SimulationOptions.EXTERNAL_LIBRARIES
SimulationOptionConstants.EXTERNAL_SOLVER_TIMEOUT_NAME	SimulationOptions.EXTERNAL_SOLVER_TIMEOUT

SimulationOptionConstants.FIRE_VALUE_CHANGE_EVENT_NAME	SimulationOptions.AUTOMATIC_PARAMETRIC_RECALCULATION <div>  The tag was changed to control only the recalculation of constraints, ensuring that the values are consistent on both sides when there is a binding connector. Set to true for parametric models to be automatically recalculated when there is a change in any structural feature value. Set to false to recalculate parametric models manually. </div>
SimulationOptionConstants.INITIALIZE_NUMERICAL_VALUE_NAME	SimulationOptions.INITIALIZE_NUMERICAL_VALUE
SimulationOptionConstants.INITIALIZE_REFERENCES_NAME	SimulationOptions.INITIALIZE_REFERENCES
SimulationOptionConstants.LASTVISITED_COLOR_NAME	SimulationOptions.LAST_VISITED_COLOR
SimulationOptionConstants.LOG_NAME	SimulationOptions.LOG
SimulationOptionConstants.NUMBER_OF_RUNS_NAME	SimulationOptions.NUMBER_OF_RUNS
SimulationOptionConstants.NUMBER_OF_STEPS_NAME	SimulationOptions.NUMBER_OF_STEPS
SimulationOptionConstants.OPEN_SIMULATION_PANE_NAME	SimulationOptions.OPEN_SIMULATION_PANE
SimulationOptionConstants.PASS_CALLER_CONTEXT_NAME	SimulationOptions.PASS_CALLER_CONTEXT
SimulationOptionConstants.RECORD_STATE_CHANGE_NAME	SimulationOptions.RECORD_STATE_CHANGE
SimulationOptionConstants.RECORD_TIMESTAMP_NAME	SimulationOptions.RECORD_TIMESTAMP

SimulationOptionConstants.RECORD_VALUE_CHANGE_NAME	SimulationOptions.RECORD_VALUE_CHANGE
SimulationOptionConstants.REMEMBER_FAILURE_STATUS_NAME	SimulationOptions.REMEMBER_FAILURE_STATUS
SimulationOptionConstants.RESULT_LOCATION_NAME	SimulationOptions.RESULT_LOCATION
SimulationOptionConstants.RUNTIME_VALUE_TEXT_COLOR_ON_PART_SHAPES_NAME	SimulationOptions.RUNTIME_VALUE_TEXT_COLOR_ON_PART_SHAPES
SimulationOptionConstants.RUN_FORKS_IN_PARALLEL_NAME	SimulationOptions.RUN_FORKS_IN_PARALLEL
SimulationOptionConstants.SHOW_ACTIVE_STATES_ON_PART_SHAPES_NAME	SimulationOptions.SHOW_ACTIVE_STATES_ON_PART_SHAPES
SimulationOptionConstants.SHOW_ACTIVE_STATE_IMAGES_NAME	SimulationOptions.SHOW_ACTIVE_STATE_IMAGES
SimulationOptionConstants.SHOW_ACTIVE_STATE_IMAGES_ON_PART_SHAPES_NAME	SimulationOptions.SHOW_ACTIVE_STATE_IMAGES_ON_PART_SHAPES
SimulationOptionConstants.SHOW_FLOWING_INFORMATION_NAME	SimulationOptions.SHOW_FLOWING_INFORMATION
SimulationOptionConstants.SHOW_HELD_TOKENS_IN_ACTIVITY_DIAGRAM_NAME	SimulationOptions.SHOW_HELD_TOKENS_IN_ACTIVITY_DIAGRAM
SimulationOptionConstants.SHOW_RUNTIME_VALUES_ON_PART_SHAPES_NAME	SimulationOptions.SHOW_RUNTIME_VALUES_ON_PART_SHAPES
SimulationOptionConstants.SILENT_NAME	SimulationOptions.SILENT
SimulationOptionConstants.SOLVE_AFTER_INITIALIZATION_NAME	SimulationOptions.SOLVE_AFTER_INITIALIZATION

SimulationOptionConstants.STARTUP_DIAGRAM_NAME	SimulationOptions.STARTUP_DIAGRAM
SimulationOptionConstants.START_TIME_NAME	SimulationOptions.START_TIME
SimulationOptionConstants.START_WEB_SERVER_NAME	SimulationOptions.START_WEB_SERVER
SimulationOptionConstants.STEP_DELAY_NAME	SimulationOptions.STEP_DELAY
SimulationOptionConstants.STEP_SIZE_NAME	SimulationOptions.STEP_SIZE
SimulationOptionConstants.TERMINATE_BEHAVIOR_ON_EXCEPTION_THROWN_NAME	SimulationOptions.TERMINATE_BEHAVIOR_ON_EXCEPTION_THROWN
SimulationOptionConstants.TERMINATE_NESTED_BEHAVIOR_NAME	SimulationOptions.TERMINATE_NESTED_BEHAVIORS
SimulationOptionConstants.TERMINATE_STREAMING_BEHAVIORS_BY_OUTPUT_PARAMETER_MULTPLICITY_NAME	SimulationOptions.TERMINATE_STREAMING_BEHAVIORS_BY_OUTPUT_PARAMETER_MULTPLICITY
SimulationOptionConstants.TIME_UNIT_NAME	SimulationOptions.TIME_UNIT
SimulationOptionConstants.TIME_VALUE_NAME	SimulationOptions.TIME_VALUE
SimulationOptionConstants.TIME_VARIABLE_NAME	SimulationOptions.TIME_VARIABLE_NAME
SimulationOptionConstants.TREAT_ALL_OBJECTS_AS_ACTIVE_NAME	SimulationOptions.TREAT_ALL_CLASSIFIERS_AS_ACTIVE
SimulationOptionConstants.UI_NAME	SimulationOptions.UI

SimulationOptionConstants.USE_FUML_DECISION_SEMANTICS_NAME	SimulationOptions.USE_FUML_DECISION_SEMANTICS
SimulationOptionConstants.VISITED_COLOR_NAME	SimulationOptions.VISITED_COLOR
com.nomagic.magicdraw.sysml.util.SysMLConstants	
SysMLConstants.DERIVED_QUANTITY_KIND_QUALIFIED_NAME	removed
SysMLConstants.DERIVED_QUANTITY_KIND_QUALIFIED_NAME_SYSML_1_4	removed
SysMLConstants.DERIVED_UNIT_QUALIFIED_NAME	removed
SysMLConstants.DERIVED_UNIT_QUALIFIED_NAME_SYSML_1_4	removed
com.nomagic.magicdraw.simulation.SimulationProfile.SimulationConfigStereotype	
SimulationProfile.SimulationConfigStereotype.FIREVALUECHANGEEVENT	SimulationProfile.SimulationConfigStereotype.AUTOMATICPARAMETRICRECALCULATION
SimulationProfile.SimulationConfigStereotype.RECORDTIMESTAMP	SimulationProfile.SimulationConfigStereotype.CAPTURETIMESTAMP
SimulationProfile.SimulationConfigStereotype.clearFireValueChangeEvent(com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element)	SimulationProfile.SimulationConfigStereotype.clearAutomaticParametricRecalculation(com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element)
SimulationProfile.SimulationConfigStereotype.clearRecordTimestamp(com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element)	SimulationProfile.SimulationConfigStereotype.clearCaptureTimestamp(com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element)

SimulationProfile.SimulationConfigStereotype.getFireValueChangeEventProperty	SimulationProfile.SimulationConfigStereotype.getAutomaticParametricRecalculationProperty
SimulationProfile.SimulationConfigStereotype.getRecordTimestampProperty	SimulationProfile.SimulationConfigStereotype.getCaptureTimestampProperty
SimulationProfile.SimulationConfigStereotype.isFireValueChangeEvent(com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element)	SimulationProfile.SimulationConfigStereotype.isAutomaticParametricRecalculation(com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element)
SimulationProfile.SimulationConfigStereotype.isRecordTimestamp(com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element)	SimulationProfile.SimulationConfigStereotype.isCaptureTimestamp(com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element)
SimulationProfile.SimulationConfigStereotype.setFireValueChangeEvent(com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element,java.lang.Boolean)	SimulationProfile.SimulationConfigStereotype.setAutomaticParametricRecalculation(com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element,java.lang.Boolean)
SimulationProfile.SimulationConfigStereotype.setRecordTimestamp(com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element,java.lang.Boolean)	SimulationProfile.SimulationConfigStereotype.setCaptureTimestamp(com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element,java.lang.Boolean)

Teamwork Cloud Developer Guide

In this section:

- [REST APIs \(see page 390\)](#)
- [Command Line Interface \(CLI\) for user administration \(see page 419\)](#)
- [OSLC API \(see page 432\)](#)

REST APIs

Teamwork Cloud provides a set of REST APIs for server management. REST APIs are essential properties in Teamwork Cloud that allow you to query data about user accounts, roles, projects, and that provide access and basic operations for the resources. REST API also provides an administrator-friendly way to manage the server because it can be run in the text mode and in a shell script.

The documentation of REST API is provided in Swagger, available with Teamwork Cloud installation, where you can find the description of the APIs in both machine-readable and human-readable formats. Teamwork Cloud REST API documentation in Swagger is accessible at <https://>

osmc.nomagic.com⁹¹. The API document is also bundled with Teamwork Cloud installation. You can access it from <https://<Teamwork Cloud IP>:8111/osmc/swagger>.

Having Swagger come with the Teamwork Cloud installation makes it the most convenient way to issue REST API to the server. It allows logging in and calling the server from a web browser. While calling to the server, Swagger also shows a cURL command that the user can copy to and execute in the command line or use in a batch script instead.

Related pages

- [General convention \(see page 391\)](#)
- [Authentication \(see page 392\)](#)
- [Reusing Teamwork Cloud session \(see page 402\)](#)
- [Model manipulation \(see page 404\)](#)
- [MagicDraw-specific extensions \(see page 411\)](#)
- [REST API Change Log \(see page 415\)](#)
- [LDAP configuration description \(see page 418\)](#)

General convention

Unless specified otherwise, this document assumes that the following specific operations call HTTP methods.

Operation	HTTP method	Meaning
Get or read	GET	Get data.
Create	POST	Create new data.
Replace attributes	PUT	Replace all existing attributes with the new ones. Attributes with unspecified new values will be reset to null values.
Update a (sub)set of attributes	PATCH	Replace some attributes only with the new ones. Unspecified attributes remain unchanged.
Delete	DELETE	Delete data.

Related pages

- [Authentication \(see page 392\)](#)
- [Model manipulation \(see page 404\)](#)
- [MagicDraw-specific extensions \(see page 411\)](#)

⁹¹ <https://osmc.nomagic.com/>

Authentication

In this section:

- [OAuth 2.0 authentication \(see page 392\)](#)
- [OpenID Connect authentication \(see page 394\)](#)
- [Token-based authentication \(see page 397\)](#)
- [Token-based authentication sample for Teamwork Cloud REST API \(see page 399\)](#)
- [Personal access token authentication sample for Teamwork Cloud REST API \(see page 401\)](#)

OAuth 2.0 authentication

On this page

- [OAuth 2.0 integration \(see page 392\)](#)
- [OAuth 2.0 client registration \(see page 393\)](#)
- [Teamwork Cloud REST API access \(see page 394\)](#)

Other web applications can integrate with Authentication server through OAuth 2.0 protocol to authenticate users and access Teamwork Cloud REST API on behalf of those users.

Authentication server behaves as OAuth 2.0 Authorization Server. Web applications that need access to Teamwork Cloud REST API on behalf of an authenticated user should implement OAuth 2.0 Client functionality according to OAuth 2.0 protocol specification.

OAuth 2.0 integration

 See [OAuth 2.0 specification](#)⁹² for more details.

Authentication server provides a JSON with all endpoints required for the integration. This JSON can be retrieved using https://<server_host>:8443/authentication/.well-known/oauth2.0-configuration endpoint. This JSON also returns supported response types and grant types.

Example JSON:

```
{  
  "response_types_supported": [  
    "token",  
    "code"  
  ],  
}
```

⁹² <https://datatracker.ietf.org/doc/html/rfc6749>

```

    "device_authorization_endpoint": "https://localhost:8443/authentication/api/oauth2/
device_code",

    "jwks_uri": "https://localhost:8443/authentication/jwks.json",

    "grant_types_supported": [

        "urn:ietf:params:oauth:grant-type:device_code",

        "refresh_token",

        "password",

        "authorization_code"

    ],

    "registration_endpoint": "https://localhost:8443/authentication/api/oauth2/
register",

    "token_endpoint_auth_methods_supported": [

        "client_secret_basic"

    ],

    "scopes_supported": [],

    "issuer": "https://localhost:8443/authentication",

    "authorization_endpoint": "https://localhost:8443/authentication/oauth2/authorize",

    "token_endpoint": "https://localhost:8443/authentication/api/oauth2/token"

}

```

Authentication server supports the following authorization grants:

- Authorization code
- Implicit
- Resource Owner Password Credentials
- Device Code (OAuth 2.0 extension, see [OAuth 2.0 Device Authorization Grant](https://oauth.net/2/grant-types/device-code/)⁹³ for more information)

OAuth 2.0 client registration

To register OAuth 2.0 clients:

⁹³ <https://oauth.net/2/grant-types/device-code/>

- a. **Use the web UI:** go to the **Settings** app of Web Application Platform and search for **OAuth 2.0** tab in the **OAuth clients** (see page 392) section.
- b. **Use *registration_endpoint API*:** (see the example JSON).
Example client registration request when using API:

```
POST https://localhost:8443/authentication/api/oauth2/register

BODY:

{
    client_name: "Client name",
    redirect_uris: ["redirect uri"] // comma separated URIs
}
```



Limitations of the client registration endpoint:

- You cannot provide supported grant types or response types – default options are set.
- Authentication server generates the client secret automatically.

However, you can update all of these attributes any time in the **OAuth clients** (see page 392) section of the **Settings** app (i.e. using the web UI).

Teamwork Cloud REST API access

After the OAuth 2.0 client gets an access token after user authentication, it can call Teamwork Cloud REST API endpoints by providing the following header:

```
Authorization: Bearer <access token>
```

OpenID Connect authentication

On this page

- [OpenID Connect integration](#) (see page 395)
- [OIDC client registration](#) (see page 396)
- [Teamwork Cloud REST API access](#) (see page 396)

Other web applications can integrate with Authentication server through OpenID Connect (further referred to as OIDC) protocol to authenticate users and access Teamwork Cloud REST API on behalf of these users.

Authentication server behaves as the OIDC Identity Provider. Web applications that need to access Teamwork Cloud REST API on behalf of an authenticated user should implement OIDC Service Provider functionality according to OIDC protocol specification.

OpenID Connect integration

 See [OpenID Connect specification](#)⁹⁴ for more details.

Authentication server provides a JSON with all endpoints required for the integration. This JSON can be retrieved using `https://<server_host>:8443/authentication/.well-known/oidc-configuration` endpoint. This JSON also returns supported response types and grant types.

Example JSON:

```
{
  "response_types_supported": [
    "code token",
    "code id_token",
    "code",
    "id_token",
    "id_token token",
    "code id_token token"
  ],
  "device_authorization_endpoint": "https://localhost:8443/authentication/api/oidc/device_code",
  "jwks_uri": "https://localhost:8443/authentication/jwks.json",
  "grant_types_supported": [
    "urn:ietf:params:oauth:grant-type:device_code",
    "refresh_token",
    "password",
    "authorization_code"
  ],
  "subject_types_supported": [
    "public"
  ],
  "id_token_signing_alg_values_supported": [
    "RS256"
  ],
  "registration_endpoint": "https://localhost:8443/authentication/api/oidc/register",
  "token_endpoint_auth_methods_supported": [
    "client_secret_basic"
  ],
  "scopes_supported": [
    "openid"
  ],
  "issuer": "https://localhost:8443/authentication",
  "authorization_endpoint": "https://localhost:8443/authentication/oidc/authorize",
  "token_endpoint": "https://localhost:8443/authentication/api/oidc/token"
}
```

⁹⁴ https://openid.net/specs/openid-connect-basic-1_0.html

Authentication server supports the following authorization flows:

- Authorization code flow
- Implicit flow
- Hybrid flow
- Device code flow (extension, see [Device Code Flow - Authorization](#)⁹⁵ for more information)

OIDC client registration

To register OIDC clients:

- Use the web UI:** go to the **Settings** app of Web Application Platform and search for **OpenID Connect** tab in the **OAuth clients** (see page 394) section.
- Use *registration endpoint API*:** (see the example JSON).
Example client registration request when using API:

```
POST https://localhost:8443/authentication/api/oidc/register

BODY:

{
    client_name: "Client name",
    redirect_uris: ["redirect uri"] // comma separated URIs
}
```



Limitations of the client registration endpoint:

- You cannot provide supported grant types or response types – default options are set.
- Authentication server generates the client secret automatically.

However, you can update all of these attributes any time in the **OAuth clients** (see page 394) section of the **Settings** app (i.e. using the web UI).

Teamwork Cloud REST API access

After the OIDC client gets ID token after user authentication, it can call Teamwork Cloud REST API endpoints by providing the following header:

⁹⁵ <https://developers.onelogin.com/openid-connect/api/device-code-auth>

Authorization: Token <ID token>

Token-based authentication

On this page

- [Pre-configuring Authentication server \(see page 397\)](#)
- [Authentication with user interaction \(see page 398\)](#)
- [Authentication without user interaction \(see page 398\)](#)

Use the following procedures to authenticate your application or script with Teamwork Cloud.

Pre-configuring Authentication server

Authentication server implements the OpenID Connect standard with several customizations. To access the OpenID Connect configuration, go to `https://<auth_server_host>:<port>/authentication/.well-known/openid-configuration`. Here is a sample of a returned configuration:

```
{
  "response_types_supported":["id_token","code"],
  "validation_endpoint":"https://<hostname>:8443/authentication/api/validate",
  "jwks_uri":"https://<hostname>:8443/authentication/jwks.json",
  "subject_types_supported":["public"],
  "id_token_signing_alg_values_supported":["RS256"],
  "signout_endpoint":"https://<hostname>:8443/authentication/api/signout",
  "issuer":"https://<hostname>:8443/authentication",
  "authorization_endpoint":"https://<hostname>:8443/authentication/authorize",
  "token_endpoint":"https://<hostname>:8443/authentication/api/token"
}
```

Configure Authentication server to accept new client applications by changing these parameters in the **authserver.properties** file:

1. Add URL of the client app to the whitelist, separating URLs with commas: **authentication.redirect.uri.whitelist**. This can be either a full URL where users should be redirected back from the Authentication server, or just the beginning of it. The authorization endpoint will not accept redirect uri parameters that cannot be found in the whitelist.
2. Add new client IDs separated with commas: **authentication.client.ids**. You might need to uncomment this line first. The authorization endpoint will not accept **client_id** parameters that cannot be found in this list.

There are a few deviations from the standard OpenID Connect specification:

- When invoking the token endpoint, HTTP header X-Auth-Secret with secret must be passed with the value from **authserver.properties**, parameter **authentication.client.secret**.
- ID tokens have expiration time (configuration property **authentication.token.expiry**), they must be refreshed through the token endpoint by passing refresh tokens.

To call TWC REST API with a generated authentication token, the token should be sent in the header of the request:

```
Authorization: Token <received_id_token>
```

Authentication with user interaction

The basic Authorization flow should be as follows:

1. Redirect the user to Authentication server (**authorization_endpoint** from OpenID Connect configuration) with HTTP GET parameters:

```
scope=openid  
redirect_uri=<your_app_url>  
client_id=<your_client_id>  
response_type=code
```

2. After the user signs in, receive HTTP GET request with a code parameter.
3. Send an HTTP POST request to the token endpoint of the Authentication server (**token_endpoint** from OpenID Connect configuration) with HTTP header X-Auth-Secret and parameters:

```
scope=openid  
redirect_uri=<your_app_url>  
client_id=<your_client_id>  
grant_type=authorization_code  
code=<code_received_after_user_signs_in>
```

4. Receive the JSON response with ID Token that can be used to authorize with Teamwork Cloud and refresh token that later should be used to refresh ID Token.
5. Refresh the ID Token by sending HTTP POST request to the token endpoint of the Authentication server (**token_endpoint** from OpenID Connect configuration) with HTTP header X-Auth-Secret and parameters:

```
scope=openid  
redirect_uri=<your_app_url>  
client_id=<your_client_id>  
grant_type=refresh_token  
refresh_token=<refresh_token_value>
```

Authentication without user interaction

To get a token without user interaction, i.e. using some predefined username and password and make only server-server calls, the system needs to send an HTTP POST request to the token endpoint of the Authentication server with HTTP headers X-Auth-Secret and parameters:

- Headers:

```
X-Auth-Secret: <secret from authentication.client.secret >
```

Authorization: Basic xxxxxxxx, where xxxxxx is base64 encoded username:password

- Query parameters:

```
grant_type=client_credentials  
client_id=<your_client_id>
```

If your client ID is added into the **authentication.client.permanent** list, the returned token will have a longer expiration time, configured in the parameter **authentication.permanent.token.expiry**.

Token-based authentication sample for Teamwork Cloud REST API

On this page

- [Setting up token-based authentication](#) (see page 399)
- [Using token-based authentication](#) (see page 399)
- [Limitations](#) (see page 401)

Teamwork Cloud REST API has an endpoint, which implements token-based authentication described in the [Token-based authentication](#) (see page 397) page.

Setting up token-based authentication

To set up Teamwork Cloud and Authentication server for token-based authentication using REST API endpoint

1. Set up Authentication server to work with your SAML/SSO, if needed.
2. Open `<install_root>/WebAppPlatform/shared/conf/authserver.properties`.
3. Find key **authentication.client.ids** and add **,twc-rest-api** at the end of the value. Save and close the file.

 The comma before **twc-rest-api** is a separator.

4. Open `<install_root>/TeamworkCloud/configuration/application.conf`.
5. Find the **esi.auth** block and set the server value to your authentication server IP. Save and close the file.
6. Restart the Authentication server and Teamwork Cloud server.

Using token-based authentication

To use token-based authentication


1. Open REST API at the following URL: `https://<ip>:8111/osmc/authn/login` on a browser.
2. You should be redirected to the Authentication server login page.
3. Enter your credentials. The browser shows you a token, usually starting with `eyJ/...`

```

Login token is
eyJhbGciOiJSUzI1NiJ9.eyJzdWIiOiJvdW0iLCJhdWQiOiJ0d2MtcVzdC1hcGkiLCJpc3MiOiJodHRwczpcLlwwMTkyLjE2OC4yMjguMTMzOjg1NTVcL2FldGhlbnRyY2F0aW9uLiwiZXhwIjoxNjA4MTE0MTgxLCJpYXQiOiJlPeODQyYiEH6xCxIYagZtMHRh0fJB_eGgZ5B-87Lw3T3iu8zKLIHtCa8Agg0Ytm55Y3Q6WbCMs41_XfqU6G_4bpuSzGu2bVFP3L31M7ZdOc7uZQfHbMCri8G1EnhG1NLqGCXt_a3tv00m_44i6-9JzivMAvTkDNwJyvpeeAYcL3au7xd8ElxOpz3JfbpFx-1qgD-Fw17oJwQ9xQXVjHrRjixgV_DdwIVueRrNVt80yC2hLPqhEOlqNr9IgL9pmcGGZpIFvDcGcb36eVe1JKB52Xaarly7SDvDU1fwPxm2gAsadsQkxNuXSW9qcy0v5Lg
All attributes: {org.restlet.startTime=1608113281959, Metric-Start-Time-In-Milliseconds=1608113281959, org.restlet.https.keySize=0, Metric-Start-Time-In-Milliseconds-com.nomagic.esi.rest.osmc.a.h.b.b=1608113281959, esi.dm.query.cors.request=false, e.session=Session(User=oum, ID=0ae19c80-65f0-4c6b-818a-26e3891d346e), com.nomagic.esi.rest.osmc.a.j.login_token=eyJhbGciOiJSUzI1NiJ9.eyJzdWIiOiJvdW0iLCJhdWQiOiJ0d2MtcVzdC1hcGkiLCJpc3MiOiJodHRwczpcLlwwMTkyLjE2OC4yMjguMTMzOjg1NTVcL2FldGhlbnRyY2F0aW9uLiwiZXhwIjoxNjA4MTE0MTgxLCJpYXQiOiJlPeODQyYiEH6xCxIYagZtMHRh0fJB_eGgZ5B-87Lw3T3iu8zKLIHtCa8Agg0Ytm55Y3Q6WbCMs41_XfqU6G_4bpuSzGu2bVFP3L31M7ZdOc7uZQfHbMCri8G1EnhG1NLqGCXt_a3tv00m_44i6-9JzivMAvTkDNwJyvpeeAYcL3au7xd8ElxOpz3JfbpFx-1qgD-Fw17oJwQ9xQXVjHrRjixgV_DdwIVueRrNVt80yC2hLPqhEOlqNr9IgL9pmcGGZpIFvDcGcb36eVe1JKB52Xaarly7SDvDU1fwPxm2gAsadsQkxNuXSW9qcy0v5Lg, org.restlet.http.headers=[Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8], [Upgrade-Insecure-Requests: 1], [User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:78.0) Gecko/20100101 Firefox/78.0], [Connection: keep-alive], [Referer: https://192.168.228.133:8555/authentication/authorize?scope=openid&response_type=code&redirect_uri=https:%2F%2F192.168.228.133:8111%2Fosmc%2Fauthen%2Fauthenticate&client_id=twc-rest-api], [Host: 192.168.228.133:8111], [Accept-Language: en-US,en;q=0.5], [Accept-Encoding: gzip, deflate, br], [DNT: 1]], org.restlet.https.sslSessionId=297ce4c17b679f53966a05e26800ce29Eae02F090E74c324b1e5937F541Bd42}

```

4. Copy the token and use it to log on to REST API.

 The token is used in an Authorization header with the **Token Type**.

For example (using a token with cURL):

```

curl -v -k -H "Authorization: Token
eyJhbGciOiJSUzI1NiJ9.eyJzdWIiOiJBZG1pbmlzdHJhdG9yIiwiaXVkiOiJ0d2MtcVzdC1hcGkiLCJpc3MiOiJodHRwczpcLlwwMTkyLjE2OC4yMjguMTMzOjg1NTVcL2FldGhlbnRyY2F0aW9uLiwiZXhwIjoxNjA4MTE0MTgxLCJpYXQiOiJlPeODQyYiEH6xCxIYagZtMHRh0fJB_eGgZ5B-87Lw3T3iu8zKLIHtCa8Agg0Ytm55Y3Q6WbCMs41_XfqU6G_4bpuSzGu2bVFP3L31M7ZdOc7uZQfHbMCri8G1EnhG1NLqGCXt_a3tv00m_44i6-9JzivMAvTkDNwJyvpeeAYcL3au7xd8ElxOpz3JfbpFx-1qgD-Fw17oJwQ9xQXVjHrRjixgV_DdwIVueRrNVt80yC2hLPqhEOlqNr9IgL9pmcGGZpIFvDcGcb36eVe1JKB52Xaarly7SDvDU1fwPxm2gAsadsQkxNuXSW9qcy0v5Lg"
https://127.0.0.1:8111/osmc/login

```

The result is as follows:

```

> GET /osmc/login HTTP/1.1
> Host: 127.0.0.1:8111
> User-Agent: curl/7.55.1
> Accept: */*
> Authorization: Token
eyJhbGciOiJSUzI1NiJ9.eyJzdWIiOiJBZG1pbmlzdHJhdG9yIiwiaXVkiOiJ0d2MtcVzdC1hcGkiLCJpc3MiOiJodHRwczpcLlwwMTkyLjE2OC4yMjguMTMzOjg1NTVcL2FldGhlbnRyY2F0aW9uLiwiZXhwIjoxNjA4MTE0MTgxLCJpYXQiOiJlPeODQyYiEH6xCxIYagZtMHRh0fJB_eGgZ5B-87Lw3T3iu8zKLIHtCa8Agg0Ytm55Y3Q6WbCMs41_XfqU6G_4bpuSzGu2bVFP3L31M7ZdOc7uZQfHbMCri8G1EnhG1NLqGCXt_a3tv00m_44i6-9JzivMAvTkDNwJyvpeeAYcL3au7xd8ElxOpz3JfbpFx-1qgD-Fw17oJwQ9xQXVjHrRjixgV_DdwIVueRrNVt80yC2hLPqhEOlqNr9IgL9pmcGGZpIFvDcGcb36eVe1JKB52Xaarly7SDvDU1fwPxm2gAsadsQkxNuXSW9qcy0v5Lg"
>
< HTTP/1.1 204 No Content
< Content-Length: 0

```


```
< Content-Type: application/octet-stream
< Date: Wed, 25 Nov 2020 10:08:44 GMT
< Accept-Ranges: bytes
< Server: Restlet-Framework/2.2.3
< Vary: Accept-Charset, Accept-Encoding, Accept-Language, Accept
< Set-Cookie: twc-rest-current-user=Administrator; Path=/osmc; Expires=Wed, 25 Nov
2020 10:23:44 GMT
< Set-Cookie: twc-rest-session-id=f40ef933-5461-4058-a1e7-9b8d4021aa8a; Path=/osmc;
Expires=Wed, 25 Nov 2020 10:23:44 GMT
<
* Connection #0 to host 127.0.0.1 left intact
```

Limitations

This REST API endpoint only displays the ID token. Usually, ID token is not very long-living. You can configure the ID token expiration in *authserver.properties* file (using property **authentication.token.expiry**).

Usually the ID token needs to be refreshed as described in the page [Token-based authentication](#) (see page 397). However, this REST API does not display a refresh token, which is needed to refresh the ID token.


As a workaround, long-living ID tokens can be generated by adding **,twc-rest-api** to the *authserver.properties* file property **authentication.client.unlimited**. In such case, ID token expiration will be calculated using property **authentication.unlimited.token.expiry**.

 Use this feature with caution and make sure that such long-living ID token is adequately protected.

Personal access token authentication sample for Teamwork Cloud REST API

To use Teamwork Cloud REST API with personal access token authentication

1. [Generate your personal access token](#) (see page 401).
2. Copy the token and use it in the HTTP header attribute to perform REST API calls.

 The token is used in an Authorization header with the **Token** Type.

HTTP header attribute example

```
Authorization: Token
eyJhbGciOiJSUzI1NiJ9.eyJzdWIiOiJBZG1pbmlzdHJhdG9yIiwiaXVkiJoidHdjLXJlc3QtYXBp
IiwiaXNzIjoiaHR0cHM6XC9cLzEyNy4wLjAuMT04NTU1XC9hdXR0ZW50aWNhdGlvb2IsImV4cCI6M
TYwNjI5OTc3NywiawWF0IjoxNjA2Mjk4ODc3fQ.bA-
S5hHeSLV8AFoQVzzfIseC3qlmqQoBQREiapHN6I5CcvwetKdSVztWkssSGjm31Y1zqoULio7_1Ma
mtGBbbzvA1WWQYFRiYk0D612yNDv4uNHBbNLNEv61TYNLwdPwPh0atVRekh-
LSgjiPTvXj4mZViE0NHKIG9U7htA9Zzvxc2JDxe_eU2-
4TCNm8II89R0aEb1tZ5nD84ieRbzJWqrcVTdQU2YfbIUeew5Nir8obkLYgixBXFKWsTHi3jNuoBx3
KcAIyZqL6cjtscER4wbk4PEEDC57UVS0csXWr6yvXIoVdJMOiDHo_fJMKgOjDqSyIL-2B210-Y-GA
```

Token expiration and validation

The personal access token does not have an expiration date, so use it with caution and ensure it is adequately protected.

To invalidate the personal access token, [delete it in the My account application](#) (see page 401).

Reusing Teamwork Cloud session

When using cURL, user credentials (user name/password or a token) can be provided in a command-line argument. A new session is created during authentication. It is **very** important to note that a Teamwork Cloud session remains open after a REST API returns. This implies that a Teamwork Cloud session is not closed automatically. If the user's credentials are still in use (without sending a valid cookie), the license count will be used up, and new sessions cannot be created. The user needs to wait until the session is timed out. The default timeout is 15 minutes.

The correct way to use REST API is to log in only once, reuse the session for all subsequent calls, and log out once finished.

The following example is a Bash script used for creating a comma-separated list of email addresses for all users in Teamwork Cloud, to facilitate notification sending.

```
#!/bin/bash
# Script to export a unique list of email address for all users in Teamwork Cloud
# into a single CSV list (email_list.csv)
# Author: Benjamin Krajmalnik
#
#
SERVER=127.0.0.1
RESTBASEURL=https://$SERVER:8111/osmc
USERURL=${RESTBASEURL}/admin/users
LOGINURL=${RESTBASEURL}/login
LOGOUTURL=${RESTBASEURL}/logout
username=Administrator
password=Administrator
rm -f email*.*
curl -k -s -c cookie.txt --insecure -u $username:$password $LOGINURL
curl -k -s -c cookie.txt -b cookie.txt --insecure -X GET "$USERURL" -H "accept:
application/json" > users.json
sed 's/"/\\",\\"/n/g' users.json > users.txt
sed -i 's/\\["//g' users.txt
sed -i 's/\\["//g' users.txt
while read username; do
    curl -k -s -c cookie.txt -b cookie.txt --insecure -X GET "$USERURL/$username"
-H "accept: application/json" >> email.txt
done < users.txt
curl -s -c cookie.txt -b cookie.txt $LOGOUTURL
grep -E -o "\\b[a-zA-Z0-9.-]+@[a-zA-Z0-9.-]+\\. [a-zA-Z0-9.-]+\\b" email.txt > email.lst
sed -i 's/\\n;/g' email.lst
sort email.lst | uniq > email_unique.lst
tr '\\n' ',' < email_unique.lst > email_list.csv
truncate -s-1 email_list.csv
```

The first execution of cURL logs in to the server. The server returns a session ID in a cookie. The cookie is saved into the **cookie.txt** file. The remaining cURL calls send the cookie back to the server to reuse the open session.

The example below shows how to use PowerShell to create internal users from a CSV file.

```
# -----
# Script to import users from CSV file into Teamwork Cloud
# Author: Benjamin Krajmalnik
# Format: non-quoted CSV, no headers
# userLogin,userPassword,fullUserName,departmentName,emailAddress
#
# Usage:
# PowerShell.exe -ExecutionPolicy Bypass -File ImportUsersFromCSV.ps1
# -----
Add-Type @"
using System.Net;
using System.Security.Cryptography.X509Certificates;
public class TrustAllCertificatesPolicy : ICertificatePolicy
{
    public bool CheckValidationResult
    (
        ServicePoint srvPoint, X509Certificate certificate
        ,
        WebRequest request, int certificateProblem
    )
    {
        return true;
    }
}
"@
[System.Net.ServicePointManager]::CheckCertificateRevocationList = $false;
[System.Net.ServicePointManager]::ServerCertificateValidationCallback += { $true; };
[System.Net.ServicePointManager]::CertificatePolicy = New-Object
TrustAllCertificatesPolicy
[System.Net.ServicePointManager]::SecurityProtocol =
[System.Net.SecurityProtocolType]::Tls12
$Server = Read-Host -Prompt 'Input the Teamwork Cloud Server Name or IP address'
$Admin = Read-Host -Prompt 'Input the Teamwork Cloud Administrator User'
$Passwd = Read-Host -Prompt 'Input the Teamwork Cloud Server Administrator Password'
$Userfile = Read-Host -Prompt 'Input the full path name of the file containing the
users'
$RESTBASEURL = "https://" + $Server + ":8111/osmc/"
$ADDUSERURL = $RESTBASEURL + "admin/users"
$LOGINURL = $RESTBASEURL + "login"
$LOGOUTURL = $RESTBASEURL + "logout"
$pair="$($Admin):$($Passwd)"
$bytes = [System.Text.Encoding]::ASCII.GetBytes($pair)
$base64 = [System.Convert]::ToBase64String($bytes)
$basicAuthValue = "Basic $base64"
$headers = @{ Authorization = $basicAuthValue }
$RestError = $null
Try {
```

```

$response = Invoke-WebRequest -Uri "$LOGINURL" -Method Post -Headers $headers
-SessionVariable 'Session'
} Catch {
    $RestError = $_
}
foreach($line in [System.IO.File]::ReadLines($Userfile))
{
    $login,$password,$username,$department,$email =
$line.split(',').trim()

    $JSON=@{
        {
            "userName": "$login",
            "password": "$password",
            "otherAttributes": {
                "mobile": "",
                "name": "$username",
                "department":
"$department",
                "email": "$email"
            },
            "enabled": true
        }
    }

    $response = Invoke-WebRequest -Uri "$ADDUSERURL"
-Method Post -Body $JSON -ContentType "application/json" -WebSession $Session
}

$response = Invoke-WebRequest -Uri "$LOGOUTURL" -Method Get -WebSession $Session

```

Model manipulation

On this page

- [MagicDraw project \(see page 405\)](#)
- [MagicDraw primary project \(see page 405\)](#)
- [Traversing to a MagicDraw model \(see page 407\)](#)
- [Creating elements \(see page 408\)](#)
- [Applied stereotypes \(see page 409\)](#)
- [Tagged values \(see page 410\)](#)

Teamwork Cloud operates at the EMF level, while MagicDraw operates at the UML level, which is on top of EMF. REST API, which is the Teamwork Cloud API, also operates at the EMF level.

Therefore, Teamwork Cloud is not affected by UML-specific implementation, and all derived properties may not be saved in the Teamwork Cloud database. The best-known derived property is *owner*. This is the result when there is no such *"setOwner"* in REST API.

Even though all non-derived properties are saved in the Teamwork Cloud database, they are saved as raw EMF data. The user is required to have knowledge about UML models to manipulate data. For example, an applied stereotype is saved as a hierarchy of *InstanceSpecification*, *Slot*, and *ValueSpecification*.

This section describes how to traverse into a UML model and create some elements in it.

MagicDraw project

In Teamwork Cloud terminology, a MagicDraw project is referred to as a *resource*. Although REST API can be used to create a Teamwork Cloud resource, it is only a bare project MagicDraw cannot read. A MagicDraw project is composed of many MagicDraw-specific meta-data such as project options, primary project, and used projects configuration. The best way to create a MagicDraw project is to create it by MagicDraw and submit it to the Teamwork Cloud server.

MagicDraw primary project

To traverse through a UML model, the primary project must be identified. The model data starts at the revision level. Issuing GET to `/revisions/{revisionId}` shows the first-level object in the revision. UUIDs of the first-level object are listed in **rootObjectIDs**.

```
{
  "commitType": "NORMAL",
  "branchID": "../..",
  "resourceID": "../..../..",
  "@base": "http://127.0.0.1:8111/osmc/resources/4615e8fa-81e5-40e0-
a51b-8496a48caf18/revisions/5/elements",
  "author": "Administrator",
  "@type":
  [
    "RDFSSource",
    "kerm1:Revision"
  ],
  "pickedRevision": -1,
  "description": "Branch \"xx\" created",
  "@context": "http://127.0.0.1:8111/osmc/schemas/revision",
  "directParent": 3,
  "dependencies": [],
  "rootObjectIDs":
  [
    "429f969a-5c81-45f4-94af-8cf983f22950",
    "ec6060a3-f3d9-482b-93a3-32af9e19202c",
    "ca9a0235-f0f7-46b7-a142-e79a67c2d00d",
    "f7c3ae92-af44-4dab-8163-a199ca05c006",
    "ba3d0700-1062-4baf-a1df-a55a4f31ce54",
    "0f14cd2d-2fd0-4523-950c-627d59e1a43d",
    "7cd22dea-aaf8-4e08-bd67-5bd975c3f06a",
    "af1042fa-8b1b-4cf2-bb7d-98dd1b881da3",
    "6d24e5e7-cdff-4e9e-85b8-28b3088f85b6",
    "243020e5-da6c-4896-b32a-fcba0e93ac8d",
    "f7449238-5cd1-41eb-9025-040210b02d93",
    "4d2459a1-49dc-4eb7-aa82-9bbb4a76b038",
    "b242613d-957e-4aec-9333-e5938f50b2ab",
    "9b953064-e422-4391-b7d9-43a2d4f14a32",
    "fc997cfd-23c5-4d0b-9953-06667dcde0dd",
    "29d9416b-ead5-4a9d-b530-b23de836f1b8",
    "7af3f24b-2da9-4b31-94b3-a87f15747296"
  ]
}
```

```

    ],
    "createdDate": "1533051367",
    "ID": "",
    "artifacts": "artifacts"
  },

```

Among these entries, the first element with the **esiproject:EsiProject** type specifies the primary project named **main decomposition project**. Other **esiproject:EsiProject** objects are the models (used projects) in a MagicDraw project. The main decomposition project is element **429f969a**. You need to load element **429f969a** and all their children. An element is represented in the JSONLD format. The attributes of the elements are in **kerml:esiData**. There will be an element named **UML Model**. The below code fragment shows the attributes of this element.

```

"kerml:esiData":
{
  "name": "UML Model",
  "namespace": "com.nomagic.magicdraw.uml_model",
  "project":
  {
    "@id": "429f969a-5c81-45f4-94af-8cf983f22950"
  },
  "internalVersion": "1",
  "version": "17.0",
  "sections":
  [
    {
      "@id": "b3e68e8e-2253-4726-8d05-d8f74fd0ba5a"
    }
  ]
},

```

Take the first **esiproject:EsiDataSection** from the feature's sections list, which is **b3e68e8e**. The excerpt of the **b3e68e8e** data is shown below.

```

{
  "kerml:name": "model",
  "@base": "http://127.0.0.1:8111/osmc/resources/4615e8fa-81e5-40e0-a51b-8496a48caf18/revisions/5/elements",
  "kerml:nsURI": "http://www.nomagic.com/ns/magicdraw/esiproject/1.0",
  "@type": "esiproject:EsiDataSection",
  "kerml:owner":
  {
    "@id": "429f969a-5c81-45f4-94af-8cf983f22950"
  },
  "kerml:revision": "http://127.0.0.1:8111/osmc/resources/4615e8fa-81e5-40e0-a51b-8496a48caf18/revisions/5",
  "@context":
  {
    "esiproject:EsiDataSection": "http://127.0.0.1:8111/osmc/schema/esiproject/2014345/EsiDataSection",
    "kerml": "http://127.0.0.1:8111/osmc/schema/kerml/20140325"
  },
  "kerml:ownedElement": [],

```

```

"kerml:modifiedTime": "20180731223607ICT",
"kerml:esiData":
{
  "featuredBy":
  {
    "@id": "c9256728-4617-4097-8e9e-dc63e2823bf8"
  },
  "rootElements":
  [
    "ca9a0235-f0f7-46b7-a142-e79a67c2d00d"
  ],
  "name": "model",
  "project":
  {
    "@id": "429f969a-5c81-45f4-94af-8cf983f22950"
  },
  "properties": []
},
"kerml:resource": "http://127.0.0.1:8111/osmc/resources/4615e8fa-81e5-40e0-
a51b-8496a48caf18",
"kerml:esiID": "b3e68e8e-2253-4726-8d05-d8f74fd0ba5a",
"@id": "#b3e68e8e-2253-4726-8d05-d8f74fd0ba5a"
}

```

The first ID in the **rootElements** list of the data section will be the ID of the UML model root element in the resource. The root element should always be of the **uml:Package** type or derived from it.

Traversing to a MagicDraw model

Once a root project is retrieved, the whole hierarchy can be retrieved layer by layer. An element can be retrieved from `/elements/{elementId}`. Child UUIDs of the resulting element are in **kerml:ownedElement**.

Multiple elements can be retrieved by POST to `/elements`. In this case, you need to put UUID in the **uuid.txt** file to load elements.

```

4eeb2e6d-9cbc-4f59-9d74-69263e52ba54,520bc74d-
b5b4-40d5-87cb-9d0b8aba5d2e,7a057ae7-6281-462d-9dcd-de9931633f5c,a6212656-5ad1-4de6-
a47f-db656c2c25fd

```

The command to load those elements is:

```

1 curl -v -H "Content-Type: text/plain" -X POST -s --cookie cookie.txt --
insecure -d @uuid.txt --insecure "https://server:8111/osmc/resources/$
{projectId}/elements"

```

In most cases, loading multiple elements at once is substantially faster than loading them one by one.

Creating elements

To use REST API to create elements, you are required to have knowledge about UML models, especially how to set the owner. A combination of parent-child type yields differences in the owner attribute. For example, if you want to create a class under a class, you need to set the **UMLClass** attribute in the child class. For a class under a package, you need to set **owningPackage** in the child class. The easiest way to know which attribute is needed is to try creating it in MagicDraw and get it in REST API.

There are two URLs to create element(s), */resources/{resourceId}/elements* and */resources/{resourceId}/elements{elementId}*. Please note that the *elementId* in the latter form is not the parent. An Ecore is needed to create an element. It can be specified in the following ways:

- Specified from the *elementId* in */resources/{resourceId}/elements/{elementId}*. This form is used in most cases, except from creating the first element in the project. The new element data type will be in the same namespace of this specified element.

```
{
  "@type": "uml:Class",
  "kermi:nsURI": "http://www.nomagic.com/magicdraw/UML/2.5.1",
  "kermi:esiData":
  {
    "owningPackage":
    {
      "@id": "757be712-f397-404d-a5ff-b97567eb240f"
    },
    "name": "c3"
  }
}
```

- Specified from **kermi:nsURI**.
- Specifying the whole Ecore in **kermi:ecore**. This mode is rarely used.

```
{
  "@type": "ikml:Container",
  "kermi:esiData":
  {
    "name": "txdhhdhd",
    "uri": "http://www.chula.ac.th",
  },
  "kermi:ecore": "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\r\n<ecore:EPackage xmi:version=\"2.0\" xmlns:xmi=\"http://www.omg.org/XMI\" xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\" \r\n xmlns:ecore=\"http://www.eclipse.org/emf/2002/Ecore\" name=\"ikml\" nsURI=\"http://www.nomagic.com/ikml/1.0\" nsPrefix=\"ikml\">\r\n  <eClassifiers xsi:type=\"ecore:EClass\" name=\"Element\" abstract=\"true\"></eClassifiers>\r\n</ecore:EPackage>\r\n"
}
```

The following script creates an element:

```
cat file.txt | curl -v -H "Content-Type: application/ld+json" -X POST -s --cookie
cookie.txt --insecure -d @-https://server:8111/osmc/resources/${projectId}/elements
```

Multiple elements can be created at once by specifying the query parameter *batch=true* in the URL. The script is shown below:

```
cat file.txt | curl -v -H "Content-Type: application/ld+json" -X POST -s --cookie
cookie.txt --insecure -d @-"https://server:8111/osmc/resources/${projectId}/elements/
${elementId}?batch=true"
```

Applied stereotypes

Applied stereotypes stored in the **appliedStereotype** attribute are the arrays containing reference objects to stereotype objects. The code below shows an example of an applied stereotype.

```
{
  "@id": "#bdeffbc9-daae-407c-8663-d3b1a0ed6077",
  "kerml:esiID": "bdeffbc9-daae-407c-8663-d3b1a0ed6077",
  "@type": "uml:Class",
  "kerml:esiData": {
    "ID": "_2021x_2_3360135_1630482378802_586147_17",
    "mdExtensions": [],
    "_representationText": null,
    "taggedValue": [
      {
        "@id": "edaa1dd8-583d-4a8b-8974-3111aef8d47b"
      }
    ],
    "appliedStereotype": [
      {
        "@id": "8dbdc804-5eed-424d-9812-28e183a8dfdc"
      }
    ]
  }
}
```

To apply the stereotype, set PATCH to the element you are working on, as shown below.

```
{
  "@context": {
    "@base": "http://127.0.0.1:8111/osmc/workspaces/59068af0-a7ae-4663-
b972-3c58a2fde23f/resources/b0585059-c4e3-4d82-90d9-c5386045bb2e/branches/
5dfb978f-0984-47ea-aaec-bdbb5a945687/elements/",
    "kerml": "http://127.0.0.1:8111/osmc/schema/kerml/20140325",
    "uml:Class": "http://127.0.0.1:8111/osmc/schema/uml/2014345/Class"
  },
  "kerml:esiData": {
```

```

    "appliedStereotype": [
      {
        "@id": "8dbdc804-5eed-424d-9812-28e183a8dfdc"
      }
    ]
  }
}

```

The UUID **8dbdc804** is an ID of the applied stereotype whose content is applied via the following command where the UUID 8df9f306 is an ID of the element to which the stereotype is applied:

```

$ curl -v -H "Content-Type: application/ld+json" -X POST -s -c ../get\cookie.txt -b
../get\cookie.txt --insecure -d @- https://127.0.0.1:8111/osmc/resources/
2663059e-8ef5-4f13-97b6-864a0353d2d0/elements/8df9f306-c4dc-41fb-ac9f-4a87fe71610a

```

Tagged values

Tagged values stored in the **taggedValue** attribute are the arrays containing reference objects to tagged value objects. The code below shows an example of an applied stereotype and tagged values.

```

{
  "@id": "#bdeffbc9-daae-407c-8663-d3b1a0ed6077",
  "kerml:esiID": "bdeffbc9-daae-407c-8663-d3b1a0ed6077",
  "@type": "uml:Class",
  "kerml:esiData": {
    "ID": "_2021x_2_3360135_1630482378802_586147_17",
    "mdExtensions": [],
    "_representationText": null,
    "taggedValue": [
      {
        "@id": "d9388ec6-ddca-4c18-9a5c-409d8649731f"
      },
      {
        "@id": "edaa1dd8-583d-4a8b-8974-3111aef8d47b"
      }
    ],
    "appliedStereotype": [
      {
        "@id": "8dbdc804-5eed-424d-9812-28e183a8dfdc"
      }
    ]
  }
}

```

To add a tagged value, apply the stereotype to the target element first.

Note

The model is incorrect if tagged values do not have their corresponding stereotypes.

To apply the tagged value, use the POST request with the following code:

```
{
  "@type": "uml:StringTaggedValue",
  "kermldata": {
    "tagDefinition": {
      "@id": "657a3511-d012-4134-af5d-411a6df78745"
    },
    "taggedValueOwner": {
      "@id": "515073dd-fe25-4327-bfc2-6466375a50f7"
    },
    "value": [
      "somevaluefromrest"
    ]
  }
}
```

The command will be:

```
1 curl -v -H "Content-Type: application/ld+json" -X POST -s -c ..
2 \get\cookie.txt -b ..\get\cookie.txt
--insecure -d @- https://127.0.0.1:8111/osmc/resources/
2663059e-8ef5-4f13-97b6-864a0353d2d0/elements/8df9f306-c4dc-41fb-
ac9f-4a87fe71610a
```

Related pages

- [REST APIs \(see page 390\)](#)
- [General convention \(see page 391\)](#)
- [Authentication \(see page 392\)](#)
- [MagicDraw-specific extensions \(see page 411\)](#)
- [Teamwork Cloud Developer Guide \(see page 390\)](#)

MagicDraw-specific extensions

On this page

- [Inspecting model decomposition \(see page 412\)](#)
- [Decomposition model \(see page 412\)](#)
- [UML model \(see page 414\)](#)

[Model manipulation \(see page 405\)](#) describes how REST API can be used to manipulate data in Teamwork Cloud at the Eclipse modeling framework (EMF) level. However, using the API at the EMF level may become difficult when working with larger projects and more complex compositions.

Therefore, to facilitate the most common model manipulation tasks, Teamwork Cloud REST API has MagicDraw-specific extensions that help minimize the number of calls necessary to find the primary MagicDraw UML model within project decomposition.

Inspecting model decomposition

Normally, MagicDraw has different kinds of models within Teamwork Cloud resources. The term “models” here refers to the data structure required to store MagicDraw-specific project information that the tool needs when loading up projects on the client-side. Currently, Teamwork Cloud REST API exposes two kinds of such models – **decomposition** and **UML**, as shown in the code below. You can retrieve a list of available models within the master branch and chosen resource revision by issuing GET to `/resources/{resourceId}/revisions/{revision}/models`.

```
{
  "@type": [
    "ldp:Container",
    "ldp:BasicContainer"
  ],
  "ldp:contains": [
    {
      "@id": "decomposition"
    },
    {
      "@id": "uml"
    }
  ],
  "@id": "",
  "@context": {
    "ldp": "http://www.w3.org/ns/ldp#"
  }
}
```

You can further inspect the content of each of the models by issuing GET to `/resources/{resourceId}/revisions/{revision}/models/{modelId}`, where *modelId* is **decomposition** or **uml** respectively.

Decomposition model

This model lists all of the Used Project hierarchy within the chosen resource. The excerpt of the results after calling `/resources/{resourceId}/revisions/{revision}/models/decomposition` is shown below.

```
{
  "@type": "models:Decomposition",
  "models:usages": [
    {
      "models:project": {
        "models:name": "HSUV",
        "@type": "models:PrimaryProject",
        "@id": "../../../elements/8a91f0a9-cb83-4c64-865e-e4c797556807"
      },
      "models:uses": [
        {
          "models:requiredVersion": "33",
          "models:uri": "twcloud:/45d11019-c077-4d4e-84d3-75eaf21e1230/37b7511d-bb55-4280-99b9-2da1e1cad9d6",

```

```

        "models:name": "Electronic components library",
        "@id": "../elements/70abbe99-a551-4e46-9eda-d718c027a300"
    },
    {
        "models:requiredVersion": "19.0 v9",
        "models:uri": "file:/C:/Users/tomvil/Desktop/
Cameo_Systems_Modeler_190_sp1_no_install/profiles/SysML%20Profile.mdzip",
        "models:name": "SysML Profile",
        "@id": "../elements/7e6d87a8-790e-4415-b1d1-493a5a2c82ac"
    }
]
},
{
    "models:project": {
        "models:name": "Time & Performance_Profile",
        "@type": "models:UsedProject",
        "@id": "../elements/80f3fefb-77f9-49f1-afa8-6fe25951659c"
    },
    "models:uses": [
        {
            "models:requiredVersion": "19.0 v9",
            "models:uri": "file:/C:/Users/tomvil/Desktop/
Cameo_Systems_Modeler_190_sp1_no_install/profiles/UML_Standard_Profile.mdzip",
            "models:name": "UML_Standard_Profile",
            "@id": "../elements/280a000d-ec4a-4278-aeda-15f270c99a19"
        }
    ]
},
{
    "models:project": {
        "models:name": "Electronic components library",
        "@type": "models:UsedProject",
        "@id": "../elements/70abbe99-a551-4e46-9eda-d718c027a300"
    },
    "models:uses": [
        {
            "models:requiredVersion": "19.0 v9",
            "models:uri": "file:/C:/Users/tomvil/Desktop/
Cameo_Systems_Modeler_190_sp1_no_install/profiles/SysML%20Profile.mdzip",
            "models:name": "SysML Profile",
            "@id": "../elements/7e6d87a8-790e-4415-b1d1-493a5a2c82ac"
        },
        {
            "models:requiredVersion": "19.0 v9",
            "models:uri": "file:/C:/Users/tomvil/Desktop/
Cameo_Systems_Modeler_190_sp1_no_install/profiles/UML_Standard_Profile.mdzip",
            "models:name": "UML_Standard_Profile",
            "@id": "../elements/280a000d-ec4a-4278-aeda-15f270c99a19"
        },
        {
            "models:requiredVersion": "19.0 v9",
            "models:uri": "file:/C:/Users/tomvil/Desktop/
Cameo_Systems_Modeler_190_sp1_no_install/SysML%20constraints.mdzip",
            "models:name": "SysML constraints",
            "@id": "../elements/fcffda41-a4a4-45d1-85bf-ee033f449701"
        }
    ]
}

```

```

    ]
  }
],
"@id": "",
"@context": {
  "models": "https://localhost:8111/osmc/schema/kerml/234567/models#",
  "ldp": "http://www.w3.org/ns/ldp#"
}
}
}

```

The main root project which the call is executed against has a type of **models:PrimaryProject**. You can find all of its directly used projects within the **models:uses** array. Since the decomposition model covers the full hierarchy, it includes both locally used projects (usually concerning Standard/System profiles) and server projects.

Each of the directly used projects has a type of **models:UsedProject** that has its own decomposition, hence enabling the full breakdown of usage structure from the primary project perspective covering both, directly and indirectly, used projects.

 The **id** attribute refers to the project ID of **esiproject:EsiProject** type, not the UML model root element.

UML model

Different from **decomposition** model previously described, the **UML** model provides a flat view of the structure of the target model. Note that the term **project** should not be confused with the term **UML** model here. The purpose of the **UML** model is to expose direct links to root elements of UML models in primary projects, and both of its, directly and indirectly, used projects.

The excerpt of the results after calling `/resources/{resourceId}/revisions/{revision}/models/uml` is shown below.

```

{
  "@type": "models:Uml",
  "@id": "",
  "@context": {
    "models": "https://localhost:8111/osmc/schema/kerml/234567/models#",
    "ldp": "http://www.w3.org/ns/ldp#"
  },
  "models:roots": [
    {
      "models:name": "HSUV",
      "@type": "models:PrimaryModel",
      "models:root": "../../elements/2b36804f-b904-4b29-87a5-5d438e10b696"
    },
    {
      "models:name": "Electronic components library",
      "@type": "models:UsedModel",
      "models:root": "../../elements/42dda2bc-a1cf-4ad8-bb73-88b20caa3a74"
    },
    {
      "models:name": "SysML Profile",

```

```

    "@type": "models:UsedModel",
    "models:root": "../../../../elements/ee2da8c9-63dc-42ec-a68a-d96dddba72df"
  },
  {
    "models:name": "UML_Standard_Profile",
    "@type": "models:UsedModel",
    "models:root": "../../../../elements/f325ec2b-b17c-4e59-9ddf-2e7d74e578e1"
  },
  {
    "models:name": "Fuel Supply",
    "@type": "models:UsedModel",
    "models:root": "../../../../elements/c4adaad7-9884-4501-a1a1-b2f515b7e6bc"
  },
  {
    "models:name": "Steering system",
    "@type": "models:UsedModel",
    "models:root": "../../../../elements/f1af8807-676c-4087-86ac-6e962b5bae1f"
  },
  {
    "models:name": "Time & Performance_Profile",
    "@type": "models:UsedModel",
    "models:root": "../../../../elements/895a1228-b850-4d8b-93b8-b33917f19952"
  }
]
}

```

The **models:PrimaryModel** type object provides a direct link to the main project's UML model root element via the **models:root** attribute. Compared to the EMF-based approach described in [Model manipulation](#) (see page 405), using the above REST API extension is a matter of a single call and requires only the resource ID and revision number to be known in advance.

Related pages

- [Teamwork Cloud Developer Guide](#) (see page 390)
- [General convention](#) (see page 391)
- [Authentication](#) (see page 392)
- [Model manipulation](#) (see page 404)

REST API Change Log

On this page

- [Changed data types](#) (see page 416)
 - [arrayOfUuid](#) (see page 416)
 - [arrayOfString](#) (see page 416)
 - [osmc/admin/config/put](#) (see page 416)
 - [osmc/admin/roles/roleID/userGroups/post](#) (see page 417)
 - [osmc/admin/license/apply/put](#) (see page 417)
- [Changed endpoints](#) (see page 417)
 - [/osmc/admin/license/query/{licenseServer}](#) (see page 417)
 - [GET](#) (see page 417)

Changed data types

arrayOfUuid

2024x	2024x Refresh1
String	Array

Affected URLs:

- /osmc/workspaces
- /osmc/workspaces/{workspaceId}/resources/{resourceId}/tags
- /osmc/workspaces/{workspaceId}/resources/{resourceId}/branches/{branchId}/elements
- /osmc/workspaces/{workspaceId}/resources/{resourceId}/branches/{branchId}/revisions/{revision}/elements
- /osmc/workspaces/{workspaceId}/resources/{resourceId}/elements
- /osmc/workspaces/{workspaceId}/resources/{resourceId}/revisions/{revision}/elements
- /osmc/resources/{resourceId}/tags
- /osmc/resources/{resourceId}/elements
- /osmc/resources/{resourceId}/revisions/{revision}/elements
- /osmc/admin/usergroups
- /osmc/admin/ldaps/{ldapId}/resync

arrayOfString

2024x	2024x Refresh1
String	Array

Affected URLs:

- /osmc/resources/{resourceId}/locks
- /osmc/resources/{resourceId}/branches/{branchId}/locks
- /osmc/resources/{resourceId}/roles/{roleId}/users
- /osmc/workspaces/{workspaceId}/resources/{resourceId}/locks
- /osmc/workspaces/{workspaceId}/resources/{resourceId}/branches/{branchId}/locks
- /osmc/workspaces/{workspaceId}/resources/{resourceId}/roles/{roleId}/users
- /osmc/workspaces/{workspaceId}/roles/{roleId}/users
- /osmc/admin/ldaps/{ldapId}/resync

osmc/admin/config/put

2024x	2024x Refresh1
String	Array

Affected URLs:

- /osmc/admin/config/{key}

osmc/admin/roles/roleID/userGroups/post

2024x	2024x Refresh1
String	Array

Affected URLs:

- /osmc/admin/roles/{roleId}/usergroups
- /osmc/resources/{resourceId}/roles/{roleId}/usergroups
- /osmc/workspaces/{workspaceId}/resources/{resourceId}/roles/{roleId}/usergroups
- /osmc/workspaces/{workspaceId}/roles/{roleId}/usergroups

osmc/admin/license/apply/put

Parameter “port” removed.

Affected URLs:

- /osmc/admin/license/apply

Changed endpoints

/osmc/admin/license/query/{licenseServer}

GET

Query parameters added:

Name	Type	Remark
licenseFramework	String	The licensing framework (DSLS or FlexNet)
licensingMode	String	The licensing mode (ORGANIZATION_DEFINED or CUSTOM, for DSLS only)

LDAP configuration description

The following table describes the key-values for changing the LDAP configuration via REST API:

Key-value	Description
"java.naming.factory.initial"	An optional key-value describing the initial context factory to be used, such as "com.sun.jndi.ldap.LdapCtxFactory".
"com.sun.jndi.ldap.read.timeout"	The maximum amount of time in milliseconds for an LDAP request or a read timeout, e.g., "10000".
"weight"	An optional LDAP weight value used to order the LDAP realm for authentication, such as "1", "2", "3".
"enabled"	The key-value describing if the LDAP realm is enabled ("true") or disabled ("false").
"com.sun.jndi.ldap.connect.timeout"	The maximum amount of time in milliseconds for the LDAP provider to establish connection, e.g., "5000". If connection is not established within a timeout period, it is aborted.
"searchbase"	The starting point of the search in the LDAP directory tree, such as "dc=example,dc=com".
"query"	The LDAP search filter value for finding, retrieving, and importing users (used when "authen_dntype" : "query"). The value depends on the LDAP server, e.g., "(uid={0})" or "(&(cn={0})(objectClass=user))".
"usergroup_query"	The LDAP search filter value for finding, retrieving, and importing user groups. The value depends on the LDAP server, e.g., "(cn={0})" or "(&(cn={0})(objectClass=group))".
"authen_dntype"	The authentication type value. Use the "template" value when one-level search can be used to login. Use the "query" value when sub-level search can be used to login.
"userDNTemplate"	The user template value used to search for users by a specific user path in one-level scope. The user name in the user path will be "{0}", e.g., "uid={0},dc=example,dc=com".
"anonymousbind"	The anonymous binding allows to connect and search without logging in to the LDAP server. Normally, the value is "false". This value can be used only when the LDAP server allows it.

Key-value	Description
"ldap_realm_name"	The name of the LDAP realm, such as "Apache LDAP" or "AD1".
"java.naming.security.protocol"	The protocol for connecting to the LDAP server, such as "ssl" or "none".
"key_cer_file_content"	The SSL key file content. If "java.naming.security.protocol" is "none", the value can be "".
"protocol"	The LDAP protocol, e.g., "ldap".
"port"	The LDAP port, e.g., "10389".
"ip"	The server IP address, e.g., "127.0.0.1".
"id"	The ID of the LDAP realm. This value is generated from the Teamwork Cloud server.
"authen"	The authentication type. Use the "simple" value for a clear-text password. Use the "sasl_mech" value for a space-separated list of the SASL mechanism names. You can use one of the SASL mechanisms, e.g., "CRAM-MD5" which means that the CRAM-MD5 SASL mechanism described in RFC 2195 will be used.
"userName"	The user name used to log in to the LDAP server, such as "uid=admin,ou=system".
"password"	The password used to log in to the LDAP server.
"url"	The URL constructed from "ip", "port", and "protocol", e.g., "ldap://127.0.0.1:10389".

Command Line Interface (CLI) for user administration

On this page:

- [Assign a project role to a particular user \(see page 420\)](#)
- [List all permissions \(see page 420\)](#)
- [Create a role \(see page 421\)](#)
- [Edit a role \(see page 422\)](#)
- [Delete a role \(see page 422\)](#)
- [Create a user \(see page 422\)](#)
- [Edit a user \(see page 423\)](#)

- [List all LDAP configurations \(see page 424\)](#)
- [Import a user from LDAP \(see page 425\)](#)
- [Search a user in an LDAP server \(see page 426\)](#)
- [List all users \(see page 427\)](#)
- [Get user information \(see page 427\)](#)
- [List all roles \(see page 428\)](#)
- [List users assigned to a particular role \(see page 430\)](#)
- [Unassign a role from a particular user \(see page 431\)](#)
- [Unassign a project role from a particular user \(see page 431\)](#)
- [Assign a role to a particular user \(see page 431\)](#)

Besides using Teamwork Cloud Admin for administrative tasks, you can also use the Command Line Interface (CLI) to create and manage users, roles, and permissions. All of the CLI supported in Teamwork Cloud are properly defined on this page. You can also find them via Swagger where you can find the command-line description and issue it easily. We recommend you use CLI via [Swagger UI](#)⁹⁶ because executing the command via Swagger also shows you the correct curl command. This link <https://osmc.nomagic.com/#/Administrator> takes you to the Teamwork Cloud CLI documentation in Swagger.

Assign a project role to a particular user

Sending POST to `/osmc/workspaces/{workspaceId}/resources/{resourceId}/roles/{roleId}/users` assigns a project role to a specified user.

HTTP request header

Content-Type: text/plain

HTTP request body

user1,user2

HTTP Status

201 with empty body.

List all permissions

The `/osmc/admin/permissions` lists all permissions in the server.

⁹⁶ <https://osmc.nomagic.com/#/Administrator>

HTTP request header

Content-Type: application/json; charset=UTF-8

HTTP request body

```
[{
  "operationAssignableAccessScope": "GLOBAL_ONLY",
  "protectedType": "com.nomagic.esi.resource",
  "name": "com.nomagic.esi.resource_list.all.resources",
  "operationName": "list.all.resources",
  "ID": "d91a7ba9-a017-44ac-9ff8-4b35635cb7b9",
  "operationDisplayName": "List All Projects",
  "protectedTypeDisplayName": "Project"
}, {
  "operationAssignableAccessScope": "GLOBAL_OR_OBJECT",
  "protectedType": "com.nomagic.esi.resource",
  "name": "com.nomagic.esi.resource_read.resource",
  "operationName": "read.resource",
  "ID": "9649cb30-6933-49f1-b309-7aade63340cc",
  "operationDisplayName": "Read Projects",
  "protectedTypeDisplayName": "Project"
}, {
  "operationAssignableAccessScope": "GLOBAL_ONLY",
  "protectedType": "com.nomagic.esi.server",
  "name": "com.nomagic.esi.server_manage.user.permissions",
  "operationName": "manage.user.permissions",
  "ID": "8d7423b8-4e8c-4d32-8d3c-783504bef044",
  "operationDisplayName": "Manage User Permissions",
  "protectedTypeDisplayName": "Server"
}]
```

Create a role

Sending POST to `/osmc/admin/roles` creates a row in the server.

HTTP request header

Content-Type: application/json

HTTP request body

```
{
```

```
{
  "permissions": ["9649cb30-6933-49f1-b309-7aade63340cc"],
  "name": "new role name",
  "description": "new row description",
}
```

HTTP response header

Content-Type: application/json; charset=UTF-8

HTTP response body

```
{
  "permissions": [{
    "operationAssignableAccessScope": {},
    "protectedType": "com.nomagic.esi.resource",
    "name": "com.nomagic.esi.resource_edit.resource",
    "operationName": "edit.resource",
    "ID": "0b972f77-368c-4511-9285-0069a1a8bf07",
    "operationDisplayName": "Edit Projects",
    "protectedTypeDisplayName": "Project"
  }],
  "name": "Project Contributor",
  "description": "Project-specific role. Users who are assigned to this role can modify content of selected project.",
  "ID": "417494bc-d0e8-449a-a8ac-5476dc2e6537"
}
```

Edit a role

Sending PATCH to `/osmc/admin/roles/{roleId}` edits a row in the server. For the format of the request and the response, see [Create a role \(see page 421\)](#).

Delete a role

DELETE to `/osmc/admin/roles/{roleId}` deletes a row in the server.

Create a user

Sending POST to `/osmc/admin/users` creates a new user. The user must not be an LDAP user.

HTTP request header

Content-Type: application/json

HTTP request body (note that the otherAttributes map must not contain realmid key)

```
{
  "userName": "test2",
  "password": "a password",
  "otherAttributes": {
    "mobile": "456",
    "name": "a full name",
    "department": "department",
    "email": "user1@notset.com"
  },
  "enabled": true
}
```

HTTP response header

Content-Type: application/json; charset=UTF-8

HTTP response body

```
{
  "userName": "test2",
  "otherAttributes": {
    "mobile": "456",
    "name": "a full name",
    "department": "department",
    "email": "user1@notset.com"
  },
  "enabled": true
}
```

Edit a user

Sending PATCH to `/osmc/admin/users/{username}` edits a user.

HTTP request header

Content-Type: application/json

The key *userName* is ignored. By default, new **otherAttributes** are merged into the server. If the value of a key in **otherAttributes** is empty, the key will be removed from the server.

HTTP request body

```
{
  "userName": "test2",
  "password": "a password",
  "otherAttributes": {
    "mobile": "456",
    "name": "a full name",
    "department": "department",
    "email": "user1@notset.com"
  },
  "enabled": true
}
```

HTTP response header

Content-Type: application/json; charset=UTF-8

HTTP response body

```
{
  "userName": "test2",
  "otherAttributes": {
    "mobile": "456",
    "name": "a full name",
    "department": "department",
    "email": "user1@notset.com"
  },
  "enabled": true
}
```

List all LDAP configurations

The */osmc/admin/ldaps* returns a JSON array containing all LDAP configurations in the server.

HTTP response header

Content-Type: application/json; charset=UTF-8

HTTP response body

```
[{
  "environment": {
    "authetype": "simple",
    "searchbase": "ou=Users,dc=nomagicasia,dc=com",
    "authen_dntype": "template",
    "anoymousbind": "false",
    "ldap_realm_name": "ldap0",
    "java.naming.factory.initial": "com.sun.jndi.ldap.LdapCtxFactory",
    "java.naming.security.protocol": "none",
    "userDNTemplate": "uid={0},ou=Users,dc=nomagicasia,dc=com",
    "com.sun.jndi.ldap.read.timeout": "500",
    "java.naming.provider.url": "ldap://ldap.th.nomagic.com:389",
    "weight": "1073741823",
    "enabled": "true"
  },
  "protocol": "ldap",
  "port": "389",
  "IP": "ldap.th.nomagic.com",
  "ID": "36f8b736-f51c-43a2-a9a6-63905419838a",
  "authen": "simple",
  "userName": "cn=admin,dc=nomagicasia,dc=com",
  "url": "ldap://ldap.th.nomagic.com:389"
}]
```

Import a user from LDAP

Sending POST to `/osmc/admin/ldaps/{ldaplId}/import{username}` imports an LDAP user into Teamwork Cloud.

HTTP response header

Content-Type: application/json; charset=UTF-8

HTTP response body

```
{
  "userName": "test2",
```

```

    "password": "a password",
    "otherAttributes": {
      "mobile": "456",
      "realmid": "36f8b736-f56c-43a2-a9a6-54128919838a",
      "name": "a full name",
      "department": "department",
      "email": "user1@notset.com"
    },
    "enabled": true
  }
}

```

Search a user in an LDAP server

Sending GET to `/osmc/admin/ldaps/{ldapid}/search?<query string>` searches users from a specified LDAP server.

HTTP response header

Content-Type: application/json; charset=UTF-8

HTTP response body

```

[
  {
    "mobile": "0123456789",
    "fullName": "Bat",
    "department": "production",
    "userName": "bat",
    "email": "bat@nma.com",
    "userDN": "uid=bat,ou=Users,dc=nomagicasia,dc=com"
  },
  {
    "mobile": "",
    "fullName": "abc",
    "department": "",
    "userName": "abc",
    "email": "",
    "userDN": "uid=abc,ou=Users,dc=nomagicasia,dc=com"
  },
  {
    "mobile": "",
    "fullName": "aaaa",
    "department": "",
    "userName": "aaa",
    "email": "",
    "userDN": "uid=aaa,ou=Users,dc=nomagicasia,dc=com"
  }
]

```

List all users

The `/osmc/admin/users` returns a JSON array containing usernames in the server.

HTTP response header

Content-Type: application/json; charset=UTF-8

HTTP response body

```
["admin", "observer1", "user1", "user2", "worker1"]
```

Get user information

The `/osmc/admin/users/{username}` returns a JSON object containing information of a specified user.

HTTP response header

Content-Type: application/json; charset=UTF-8

HTTP response body

```
{
  "roleAssignments": [{
    "roleID": "3e4696d5-1937-4741-aacb-062378e0cda2",
    "protectedObjects": [{
      "protectedType": "com.nomagic.esi.resource",
      "ID": "cda926be-58c7-4d3f-acb2-e9cdb57d8aaf"
    }],
    "ID": "3871e058-65e6-40d9-bd23-f80bdc074394"
  }, {
    "roleID": "417494bc-d0e8-449a-a8ac-5476dc2e6537",
    "protectedObjects": [{
      "protectedType": "com.nomagic.esi.resource",
      "ID": "cda926be-58c7-4d3f-acb2-e9cdb57d8aaf"
    }],
    "ID": "54b8ef7f-f60c-410f-8a09-93a15c51bbaa"
  }, {
    "roleID": "a020ca32-22a2-4c90-8513-f6b9c4ea8513",
    "protectedObjects": [{
      "protectedType": "com.nomagic.esi.resource",
```

```

        "ID": "cda926be-58c7-4d3f-acb2-e9cdb57d8aaf"
    }],
    "ID": "51ae2287-405e-42a6-807f-ca5e6f2a41d0"
}, {
    "roleID": "e3cf7416-9cf3-42ff-84e6-e9d4283dfd18",
    "protectedObjects": [{
        "protectedType": "com.nomagic.esi.resource",
        "ID": "cda926be-58c7-4d3f-acb2-e9cdb57d8aaf"
    }],
    "ID": "a147bf72-1a51-49f5-baf9-ac3f2b6faf50"
}],
"userNAme": "user1",
"otherAttributes": {
    "realmid": "",
    "mobile": "",
    "name": "",
    "department": "",
    "email": ""
},
"enabled": true
}

```

List all roles

The `/osmc/admin/roles` lists all rows in the server.

HTTP response header

Content-Type: application/json; charset=UTF-8

HTTP response body

```

[ {
    "permissions": [ {
        "operationAssignableAccessScope": {},
        "protectedType": "com.nomagic.esi.resource",
        "name": "com.nomagic.esi.resource_create.resource",
        "operationName": "create.resource",
        "ID": "930c939c-6ec4-4c90-9458-92eefc73b11b",
        "operationDisplayName": "Create Project",
        "protectedTypeDisplayName": "Project"
    } ],
    {
        "operationAssignableAccessScope": {},
        "protectedType": "com.nomagic.esi.resource",
        "name": "com.nomagic.esi.resource_categorize.resources",
        "operationName": "categorize.resources",
        "ID": "9a223c45-71eb-4e45-b374-a8dd319afcea",

```

```

        "operationDisplayName": "Categorize Projects",
        "protectedTypeDisplayName": "Project"
    }, {
        "operationAssignableAccessScope": {},
        "protectedType": "com.nomagic.esi.resource",
        "name": "com.nomagic.esi.resource_list.all.resources",
        "operationName": "list.all.resources",
        "ID": "d91a7ba9-a017-44ac-9ff8-4b35635cb7b9",
        "operationDisplayName": "List All Projects",
        "protectedTypeDisplayName": "Project"
    }],
    "name": "Project Creator",
    "description": "Global or category-specific role. Users who are assigned to this
role can add projects to the server including the ability to categorize them: create
new categories or manage existing ones.",
    "ID": "15c045d8-44e1-4e14-8175-b209b6ae70a4"
}, {
    "permissions": [{
        "operationAssignableAccessScope": {},
        "protectedType": "com.nomagic.esi.resource",
        "name": "com.nomagic.esi.resource_edit.resource",
        "operationName": "edit.resource",
        "ID": "0b972f77-368c-4511-9285-0069a1a8bf07",
        "operationDisplayName": "Edit Projects",
        "protectedTypeDisplayName": "Project"
    }, {
        "operationAssignableAccessScope": {},
        "protectedType": "com.nomagic.esi.resource",
        "name": "com.nomagic.esi.resource_read.resource",
        "operationName": "read.resource",
        "ID": "9649cb30-6933-49f1-b309-7aade63340cc",
        "operationDisplayName": "Read Projects",
        "protectedTypeDisplayName": "Project"
    }, {
        "operationAssignableAccessScope": {},
        "protectedType": "com.nomagic.esi.resource",
        "name": "com.nomagic.esi.resource_edit.resource.properties",
        "operationName": "edit.resource.properties",
        "ID": "a93ff74f-baae-4aea-8f79-1a9d423f35fa",
        "operationDisplayName": "Edit Project Properties",
        "protectedTypeDisplayName": "Project"
    }],
    "name": "Project Contributor",
    "description": "Project-specific role. Users who are assigned to this role can
modify content of selected project.",
    "ID": "417494bc-d0e8-449a-a8ac-5476dc2e6537"
}, {
    "permissions": [{
        "operationAssignableAccessScope": {},
        "protectedType": "com.nomagic.esi.server",
        "name": "com.nomagic.esi.server_list.all.users",
        "operationName": "list.all.users",
        "ID": "34e47503-ad58-401b-a3d9-fdb0e00ea651",
        "operationDisplayName": "List All Users",
        "protectedTypeDisplayName": "Server"
    }],
    "name": "Server Administrator",
    "description": "Server-specific role. Users who are assigned to this role can
manage the server including the ability to create new servers, delete servers,
and manage server properties.",
    "ID": "15c045d8-44e1-4e14-8175-b209b6ae70a4"
}

```

```

    }, {
      "operationAssignableAccessScope": {},
      "protectedType": "com.nomagic.esi.server",
      "name": "com.nomagic.esi.server_remove.user",
      "operationName": "remove.user",
      "ID": "3f7a74c4-9a95-40a6-837a-2aaf7f5f91ef",
      "operationDisplayName": "Remove User",
      "protectedTypeDisplayName": "Server"
    }, {
      "operationAssignableAccessScope": {},
      "protectedType": "com.nomagic.esi.server",
      "name": "com.nomagic.esi.server_create.user",
      "operationName": "create.user",
      "ID": "d616eb9e-d1d4-4f2d-ad24-3cfb6e57d08e",
      "operationDisplayName": "Create User",
      "protectedTypeDisplayName": "Server"
    }, {
      "operationAssignableAccessScope": {},
      "protectedType": "com.nomagic.esi.server",
      "name": "com.nomagic.esi.server_edit.user.properties",
      "operationName": "edit.user.properties",
      "ID": "d81818d4-0d98-4464-b05c-e54e4af82877",
      "operationDisplayName": "Edit User Properties",
      "protectedTypeDisplayName": "Server"
    }
  ],
  "name": "User Manager",
  "description": "Global role. Users who are assigned to this role can create and manage users in a server.",
  "ID": "1b3a3af6-887f-4891-a3df-b0e7b9141ff2"
}]

```

List users assigned to a particular role

The `_osmc/admin/roles/{roleId}/users` lists users assigned to a particular role.

HTTP response header

Content-Type: application/json; charset=UTF-8

HTTP response body

["admin","observer1"]

Unassign a role from a particular user

Sending HTTP DELETE to `/admin/roles/{roleId}/users/{username}` unassigns a particular role from a specified user.

HTTP Response

204 with empty body.

Unassign a project role from a particular user

Sending DELETE to `_/osmc/workspaces/{workspaceId}/resources/{resourceId}/roles/{roleId}/users/{username}_` unassigns a project role from a particular user.

HTTP Status

204 with empty body.

Assign a role to a particular user

Sending POST to `/osmc/admin/roles/{roleId}/users` assigns a role to a user.

HTTP request header

Content-Type: text/plain

HTTP request body

user1,user2

HTTP Status

201 with empty body

OSLC API

- ⚠** Since version 2022x Refresh2, Teamwork Cloud OSLC API implementation has been moved to the Web Application Platform, resulting in a change of port and API URI patterns:
- OSLC API no longer uses port 8111 but relies on port 8443, as well as other Web Application Platform applications.
 - OSLC root services document URI can be found using the following pattern - *http(s)://WEBAPP_IP:PORT/oslc/api/rootservices*.
 - Each of the model elements is exposed using the following URI pattern - *http(s)://WEBAPP_IP:PORT/oslc/api/oslc/am/{projectID}/{elementID}*.
Note that the *elementID* here is an Element Server ID.

Note regarding migration

The old pre-2022x Refresh2 root services URI and element URIs will be automatically redirected to the new URIs via HTTP 301 redirect. After migrating to 2022x Refresh2, you must set the WAP URI value in the *server* field of the Teamwork Cloud *esi.oslc* configuration block in the *application.conf* file.

Teamwork Cloud has OSLC support and exposes model element data following [OSLC Architecture Management \(AM\)](#)⁹⁷ and [OSLC Configuration Management](#)⁹⁸ vocabularies. Along with core OSLC services provided in Teamwork Cloud, this enables smooth integration with other OSLC-compatible tools by linking resources in Linked Data fashion.

We expose the following model element properties:

- `rdf:type` (<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>) - represents architecture resource type, which according to OSLC AM vocabulary is always a Resource (<http://open-services.net/ns/am#Resource>).
- `dcterms:modified` (<http://purl.org/dc/terms/modified>) - stands for the last element modification date.
- `dcterms:identifier` (<http://purl.org/dc/terms/identifier>) - the Element Server ID as used in element's URI pattern.
- `dcterms:title` (<http://purl.org/dc/terms/title>) - the name of the element

Sample RDF/XML representation of element data

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns:oslc="http://open-services.net/ns/core#"
  xmlns:oslc_am="http://open-services.net/ns/am#"
  xmlns:dc="http://purl.org/dc/terms/"
  xmlns:dcterms="http://purl.org/dc/terms/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
```

⁹⁷ <http://open-services.net/ns/am>

⁹⁸ <https://archive.open-services.net/wiki/configuration-management/index.html>

```

<rdf:Description rdf:about="https://localhost:8443/oslc/api/oslc/am/
b1acff2d-4396-4314-9dba-477d573ede16/50775427-bce2-4de4-b747-8ab82d294237">
  <rdf:type rdf:resource="http://open-services.net/ns/am#Resource"/>
  <oslc:serviceProvider rdf:resource="https://localhost:8443/oslc/api/oslc/am/
b1acff2d-4396-4314-9dba-477d573ede16/services"/>
  <dcterms:modified rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">May
16, 2018 2:08:52 PM</dcterms:modified>
  <dcterms:identifier rdf:datatype="http://www.w3.org/2001/XMLSchema#string">507754
27-bce2-4de4-b747-8ab82d294237</dcterms:identifier>
  <dcterms:title rdf:parseType="Literal">Engine</dcterms:title>
</rdf:Description>
</rdf:RDF>

```

- Currently, querying services are not supported. Model data can only be read through OSLC services and editing capabilities are not supported.
- We offer OSLC UI previews through integration with [C \(see page 432\)ameo Collaborator for Teamwork Cloud \(see page 432\)](#). For this, you need to [publish an OSLC resource \(see page 432\)](#).
- We provide OSLC delegated dialog support via Selection Dialog service for navigating the model tree and linking to AM resources from external tools.

OSLC Configuration Management Support

OSLC Configuration Management support makes Teamwork Cloud OSLC services configuration-aware. This enables Teamwork Cloud resource discovery at the branch or tagged commit levels. The entry point for discovery of OSLC Configuration Management services is via the `<oslc_config:cmServiceProviders>` property. Teamwork Cloud concepts map to OSLC Configuration Management concepts in the following way:

Teamwork Cloud concept	OSLC Configuration Management concept
Cameo Model (.Master Resource)	Component
Branch	Stream
Tagged Commit	Baseline
Used Project (Branch)	Contribution of a Stream to a Stream

OAuth 1.0a authentication

To ensure secure access to server resources via OSLC, OAuth 1.0a authentication protocol is used.

OAuth 1.0a requires consumer key and secret to be known before starting the authentication process flow. These can be created through Teamwork Cloud Admin as described in [Managing OAuth client keys \(see page 432\)](#).

Alternatively, a pair of consumer key and secret can be generated manually via a service exposed in the root services document. The following HTTP POST request should be made to a consumer key generation service (jfs:oauthRequestConsumerKeyUrl):

HTTP request body

```
{
  "name": "consumerNameGoesHere",
  "secret": "OAuthSecretGoesHere"
}
```

HTTP response body

```
{
  "key": "generatedConsumerKeyShouldBeHere"
}
```

Note, that keys generated following this approach will still need to be [approved by an administrator](#) (see [page 432](#)) through Teamwork Cloud Admin.

Alf Plugin Developer Guide

- [Introduction to the Alf API](#) (see [page 434](#))
- [Compiler API](#) (see [page 435](#))
- [Importer API](#) (see [page 439](#))
- [Action API](#) (see [page 442](#))

Introduction to the Alf API

The Alf Plugin provides a basic Application Programming Interface (API) that can be used to programmatically compile Alf code and import Alf files into a UML model. The Alf API classes are package in the *alf_api.jar* file located at:

- *<MagicDraw installation directory>/plugins/com.nomagic.magicdraw.alf/alf_api.jar*

You can also find JavaDoc for the Alf API at:

- *<MagicDraw installation directory>/openapi/docs/alf/AlfJavaDoc.zip*

For general information about creating a new MagicDraw plugin using Open APIs, see the [MagicDraw Developer Guide](#) (see [page 9](#)).



Before using the Alf Open API in your Java code, make sure that *alf-base.jar*, *alf.jar* and *alf_api.jar* have all been added to your IDE classpath.

Related Pages

- [Compiler API](#) (see page 435)
- [Importer API](#) (see page 439)

Compiler API

The *AlfCompiler* class provides the basic interface for compiling Alf code. The processing of a single piece of Alf text is called a *compilation*, which takes place in two steps:

1. [Parsing](#) (see page 435) (including constraint checking)
2. [Mapping](#) (see page 437)

The API allows you to carry out these steps individually (which is useful if, for example, you wish to parse but not map some code) or using a single call.

Related Pages

- [Compiler API: Parsing](#) (see page 435)
- [Compiler API: Mapping](#) (see page 437)
- [Compiler API: Utilities](#) (see page 438)

Compiler API: Parsing

Parsing is the process of creating an abstract syntax representation of some Alf code. If parsing is successful, the Alf compiler also performs constraint checking on the resulting abstract syntax tree. *Constraint checking* is the process of doing name resolution and validating the Alf code against the static-semantic constraints defined in the Alf specification.

A compilation takes place in the context of a specific Named Element in the UML model, called the *context element*. During parsing and constraint checking, name resolution is performed in the scope containing the context element (known, in Alf terms, as the *model scope*). After the successful mapping of Alf code to UML, the context element is updated with the results of the compilation. The context element should be an Activity, Opaque Behavior, Opaque Action or Opaque Expression



In UML, a Behavior may have a context Classifier (for example, the Class that owns a State Machine as its classifier behavior is the context Classifier for the State Machine). This use of the term "context" is unrelated to the term "context element" for an Alf compilation.

To compile some Alf code text, first set the context element using the *setContextElement* method and then call the *parse* method, passing the text. What Alf text can be validly parsed depends on what the context element is.

- If the context element is a Value Specification, then the text is assumed to be for an Alf Expression.
- If the context element is an Opaque Action, then the text may be an Alf Expression or Statement Sequence.
- Otherwise the text is assumed to be for an Alf Statement Sequence.

i The Alf code associated with an Activity, Opaque Behavior, Opaque Action or Opaque Expression is known as the *Alf body* of that element. The Alf code is saved in the model differently for different kinds of Elements. However, you can use the *AlfElementUtil.getAlfBody* operation to get the Alf body of an Element (if it has one).

After the parsing process completes, you can check if it was successful by calling the *isSuccessful* method. If the parse is successful, then the resulting abstract syntax tree is rooted in an Alf Unit Definition obtained by calling the *getUnit* method, which in turn always contains an Activity Definition (which may be obtained by calling the *UnitDefinition.getDefinition* method). If there were errors, then you can retrieve them using the *getCompilerErrors* method. A single *AlfCompiler* instance may be used to compile multiple Alf code texts, for the same or different context elements.

i Alf abstract syntax classes are found in sub-packages of the package *org.modeldriven.alf.syntax*, which is from the Alf Reference Implementation, with source available [here](#)⁹⁹. These abstract syntax classes allow the abstract syntax tree to be navigated based on the structure of the abstract syntax metamodel defined in the Alf specification. For instance, the value of the *definition* property of a *UnitDefinition* is obtained by calling *getDefinition*, the *name* of that definition is obtained using *getName*, etc. (Note that use of the Reference Implementation outside of MagicDraw tooling is subject to its separate open-source licensing terms.)

For the purposes of triggering automatic compilation, the Alf compiler also [manages a record of the dependencies](#)¹⁰⁰ of Alf code in a model on other model elements. After successfully parsing some Alf code, if you want to update the dependency record based on the parse, you need to call the *AlfActionUtil.registerDependencies* method. If you do multiple parses with the same *AlfCompiler* instance, then you need to call *registerDependencies* after each parse operation.

```
AlfCompiler compiler = new AlfCompiler();
compiler.setContextElement(element);
compiler.parse(AlfElementUtil.getAlfBody(element));
if (compiler.isSuccessful()) {
    AlfActionUtil.registerDependencies(compiler);
    UnitDefinition unit = compiler.getUnit();
    ...
} else {
    for (CompilerError error: compiler.getCompilerErrors()) {
        ...
    }
}
```

99 <https://github.com/ModelDriven/Alf-Reference-Implementation/tree/master/org.modeldriven.alf/src/org/modeldriven/alf/syntax>

100 <https://docs.nomagic.com/display/ALFP/Dependency+Management>

Compiler API: Mapping

Mapping is the process of generating a UML activity model from the abstract syntax representation of some Alf code. If this mapping is successful, then the Alf compiler updates the context element using the generated elements. How this update is done depends on what the context element is:

- If the context element is an Activity, the nodes and edges of the Activity are replaced with those generated by the compilation.
- If the context element is an Opaque Behavior or Opaque Action, a *CompiledRepresentation* stereotype is applied with its *behavior* tag pointing to the compiled Behavior.
- If the context element is an Opaque Expression, its *behavior* property is set to the compiled Behavior.

In addition, there may be some generated elements that cannot be stored in conjunction with the context element. In particular, generated Instance Specifications are stored in the nearest Package containing the context element, and generated template instantiations are stored in the [\\$\\$Template Bindings](#) (see page 437) package.

If you have successfully parsed some Alf code text, then you can map that code by calling the *map* method. (If the parse was not successful, then calling *map* has no effect.) Normally, unless there is a system error, mapping should always complete successfully. Since mapping results in an update to the UML model, the *map* method should be called within a MagicDraw session. This can be done conveniently using the *AlfActionUtil.executeSession* method, which also ensures that any automatic Alf compilation is turned off during the session, so new compilations are not accidentally triggered by updates from the mapping process.

```
AlfCompiler compiler = new AlfCompiler();
compiler.setContextElement(element);
compiler.parse(text);
if (compiler.isSuccessful()) {
    AlfActionUtil.executeSession("Map Alf", new Runnable() {
        public void run() {
            compiler.map();
            AlfActionUtil.registerDependencies(compiler);
        }
    });
}
```

Rather than separately parsing and mapping some Alf text, you can attempt to do both with one call to the *compile* method. This method first parses some text for a given context element and, if that is successful, maps it and updates the UML model appropriately with the results of the mapping. If the parse is not successful, then compilation errors are available using the *getCompilerErrors* method, as after a call to the *parse* method. As for the *map* method, the *compile* method should be called within a MagicDraw session, since it may potentially update the UML model if a mapping is carried out.

```
Compiler compiler = new AlfCompiler();
AlfActionUtil.executeSession("Compile Alf", new Runnable() {
    public void run() {
        compiler.compile(element, text);
        AlfActionUtil.registerDependencies(compiler);
    }
})
```

```
});
```

Related Pages

- [Compiler API: Parsing](#) (see page 435)

Compiler API: Utilities

Instead of explicitly creating an *AlfCompiler* object and using its methods, you can alternatively parse and map Alf code using static methods provided by the *AlfActionUtil* class.

The *AlfActionUtil.parse* method takes a Named Element and, using this as the context element, parses and constraint checks the Alf body of the Element, if it has one. If the parsing and constraint checking succeed, then the method also automatically registers dependencies. The method returns a Boolean indicating whether the parse succeeded or not. If the parse fails, Compiler Errors are reported in an error message that is attached to the context element using an error Annotation (which then becomes available as an Active Validation Result, as described in [the Alf editor](#)¹⁰¹).

```
if (AlfActionUtil.parse(element)) {  
    // Parse was successful.  
    ...  
}
```

The *AlfActionUtil.compile* method takes a Named Element and, using this as the context element, both parses and maps the Alf body of the Element, if it has one. If the compilation is successful, then the method updates the context element and automatically registers dependencies. The method returns a Boolean indicating whether the parse succeeded or not. As with the *parse* method, if the compilation fails, Compiler Errors are recorded in an error message that is attached to the context element using an error Annotation. Unlike the *AlfCompiler.map* and *AlfCompiler.compile* methods, you do not need to explicitly start a MagicDraw session to call the *AlfActionUtil.compile* method, because it ensures internally that a session is started for mapping, if necessary.

```
if (AlfActionUtil.compile(element)) {  
    // Compilation was successful.  
    ...  
}
```

101 <https://docs.nomagic.com/display/ALFP/The+Alf+editor>

Related Pages

- [Compiler API: Parsing](#) (see page 435)
- [Compiler API: Mapping](#) (see page 437)

Importer API

The *AlfImporter* class provides the basic interface for importing Alf code from external files. An *AlfImporter* is instantiated with two pieces of information:

1. A string giving the path to the *model directory*, which is the root directory for all files being imported.
2. An optional [ProgressStatus](#)¹⁰² object, the description of which is updated with the names of files being imported. (This argument may be null, if you do not wish to display progress status.)

Importing of Alf code then takes place in two steps:

1. [Parsing](#) (see page 440) one or more Alf model units (along with their subunits) from files in the model directory, caching their resulting abstract syntax representations.
2. [Mapping](#) (see page 440) (after constraint checking) all parsed units into the UML model in MagicDraw.

The API allows you to carry out these steps individually (which is useful if, for example, you wish to import multiple model unit files before mapping) or using a single call (for a single model unit file).



Importation takes place into the currently open and active Project. Once you instantiate an *AlfImporter* object in a Project, it is no longer consistent to use an *AlfCompiler* object in that Project. Instead, once you have completed any importation, you must use the *AlfActionUtil.resetActiveProject* method to return the currently active project to a state in which the *AlfCompiler* can be used (see [Importer API: Mapping](#) (see page 440)). Alternatively, you can use the *AlfActionUtil.importFrom* method to do the importation (as described in [Importer API: Utilities](#) (see page 442)), in which case the Project will be restored for normal Alf operations at the completion of the method.

Related Pages

- [Importer API: Parsing](#) (see page 440)
- [Importer API: Mapping](#) (see page 440)
- [Importer API: Utilities](#) (see page 442)

¹⁰² <https://docs.nomagic.com/display/MD2024xR3/Running+action+with+progress>

Importer API: Parsing

An Alf model unit file may be initially imported using the *AlfImporter.parse* method, passing in a Java *Path* object for the file. The file must be in the model directory identified when the *AlfImporter* object was [created](#) (see [page 439](#)). The Alf code in the given file is then parsed and the resulting abstract syntax tree is cached. Any subunits of the Alf unit in the given file are also imported. In addition, when the import process identifies a reference that cannot be resolved within the file being imported (or within the UML model being imported into), it will attempt to find another associated file in which the reference can be resolved. The importer uses the conventions of the Alf Reference Implementation when resolving Alf unit names to file names, starting from the model directory identified for the importation (for more information, see the Alf Reference Implementation documentation [here](#)¹⁰³).

The *parse* method returns a Boolean indicating whether all parsing completed successfully or not. You can also check the success of the last parse by calling the *isSuccessful* operation. You can call the *parse* method multiple times to load multiple different files (which must all be from the same model directory) into the import cache. Note that the *isSuccessful* flag is reset after each parse operation.

Unlike parsing using the *AlfCompiler*, the parse process in the *AlfImporter* does *not* include constraint checking. Instead, constraint checking is done as part of the [mapping](#) (see [page 440](#)) process, on all cached model units and their subunits together. This means that any errors reported from parsing imported code will be syntactic errors. All such errors are reported as messages in the MagicDraw [Message Window](#) (see [page 440](#)). It is also possible for the parse to be unsuccessful without reporting any syntax errors, for one of the following reasons:

- The identified file is not found or attempting to read it results in an IO exception (though no exception is propagated).
- The identified file has a valid Alf unit in it, but the name of that unit does not match the name of the file.

```
AlfImporter importer = new AlfImporter(modelDirectory, progressStatus);
if (importer.parse(Paths.get(modelDirectory, "Utilities.alf"))) {
    importer.parse(Paths.get(modelDirectory, "Main.alf"));
}
```


Importer API: Mapping

Once one or more Alf model unit files (and their subunits, if any) have been [parsed](#) (see [page 440](#)), you can use the *AlfImporter.compile* method to constraint check all cached units and, if there are no constraint violations, map them into the UML model. If there are constraint violations, then these are reported in the MagicDraw [Message Window](#) (see [page 440](#)). The *compile* method returns a Boolean indicating whether constraint checking was successful or not. You can also check the success of the last compilation by calling the *isSuccessful* operation.

The *compile* method takes a Java Path as a parameter, but this is only used to update the *ProgressStatus* object provided when the *AlfImporter* was [created](#) (see [page 439](#)) (if any), and it may be null (in which case the progress status is not updated). Since mapping results in an update to the UML model, the *compile* method should be called within a MagicDraw session. This can be done conveniently using the *AlfActionUtil.executeSession* method, which also ensures that any automatic Alf compilation is turned off

¹⁰³ <https://github.com/ModelDriven/Alf-Reference-Implementation/wiki/Unit-Management#the-model-directory>

during the session, so new compilations are not accidentally triggered by updates from the mapping process.

 Using an *AlfImporter* in a Project is incompatible with subsequently using an *AlfCompiler* in that Project. Once you have completed an importation, you must use the *AlfActionUtil.resetActiveProject* method to return the currently active project to a state in which the *AlfCompiler* can be used, or use the *AlfActionUtil.resetProject* method to similarly reset a specific project. You should do this even if the importation was not successful. Alternatively, you can use the *AlfActionUtil.importFrom* method, which allows for subsequent Alf compilation after importation (see [Importer API: Utilities](#) (see page 442)).

```
AlfImporter importer = new AlfImporter(modelDirectory, progressStatus);
Path path = Paths.get(modelDirectory, modelFileName);
if (importer.parse(path)) {
    AlfActionUtil.executeSession("Import Alf", new Runnable() {
        public void run() {
            importer.compile(path);
        }
    });
}
// It is unsafe to use the AlfCompiler API at this point.

AlfActionUtil.resetActiveProject();

// It is safe to use the AlfCompiler API from here on...
```

Rather than separately parsing and then compiling imported Alf files, you can do both with one call to the *importFile* method. Given a Java *Path* object for a single file in the model directory identified when the *AlfImporter* object was [created](#) (see page 439), the *importFile* method parses the model unit in the given file (and its subunits, if any), performs constraint checking and, if that is all successful, compiles the unit into the UML model. The method returns a Boolean indicating whether the import was successful or not. If not, any syntactic errors or constraint violations are reported in the MagicDraw [Message Window](#) (see page 440). As for the *compile* method, the *importFile* method should be called within a MagicDraw session, since it may potentially update the UML model if a mapping is carried out.

```
AlfImporter importer = new AlfImporter(modelDirectory, progressStatus);
AlfActionUtil.executeSession("Import Alf", new Runnable() {
    public void run() {
        importer.importFile(Paths.get(modelDirectory, modelFileName));
    }
});
AlfActionUtil.resetActiveProject();
```

Related Pages

- [Importer API: Parsing \(see page 440\)](#)

Importer API: Utilities

AlfActionUtil.importFrom is a static method that takes a directory path (as a string) and a file name, and carries out the functionality of the **Import From > Alf File** command (see [Importing Alf \(see page 442\)](#)). That is, it does the following:

1. If there is no Project currently open, it creates a new Project using the [Alf project template \(see page 442\)](#) and makes it the active Project.
2. It imports the given file into the currently active Project, with progress status, in a MagicDraw session.
3. It marks the active Project as dirty (but does not save it or close it).

When the method completes, it is safe to continue to perform Alf compiler operations on the Project. Any parsing, constraint checking or mapping errors (but not IO exceptions) are reported to the MagicDraw [Message Window \(see page 442\)](#).

```
AlfActionUtil.importFrom(modelDirectory, modelFileName);
```

Related Pages

- [Importer API: Parsing \(see page 440\)](#)
- [Importer API: Mapping \(see page 440\)](#)

Action API

The *AlfActionUtil* class also includes a set of static methods that perform operations over the entire active Project. These are:

- *parse* (with no argument), which parses the Alf bodies of all Elements in the active Project that are annotated as needing recompilation.
- *parseAll*, which parses all Alf text in the active Project, but does not do any mapping to UML.
- *build*, which does a [normal build \(see page 442\)](#) of the Alf bodies of all elements in the active project that are annotated as needing recompilation and, optionally, also elements annotated with compilation errors (depending on its *includeErrorAnnotated* argument).
- *clean*, which deletes all auxiliary template bindings and does a [clean build \(see page 442\)](#) of all Alf text in the active project.

In all cases, dependencies are registered for all parsed Elements and any errors are recorded as error Annotations on relevant Elements (see also [The Alf editor \(see page 442\)](#) and [Dependency Management \(see page 442\)](#)).

A

Abstract-matrix-cell [157](#)
Actions-configurator-manager [155](#)
Active-validation-suite [169](#)
Amconfigurator [155](#), [155](#)
Annotation [169](#)
Annotation-manager [169](#)
Application [66](#)
Auto-number [182](#)

B

Base-element [76](#)
Brand [14](#)
Break-point [125](#)
Browser [163](#)
Browser-tree [163](#)
Button [52](#)

C

Can-create [156](#)
Can-edit [156](#)
Cell-renderer [157](#)
Client [125](#)
Client-point [125](#)
Close-session [80](#)
Code-engineering-manager [191](#)
Column-header-cell-renderer [157](#)
Command [52](#)
Configure-dependency-handlers [156](#)
Container [81](#)
Create-add-actions [156](#)
Create-button [54](#)
Create-command [54](#)
Create-edit-actions [156](#)
Create-session [80](#)
Custom-button [54](#)
Custom-command [54](#)

D

Dependency-editor [156](#)
Dependency-entry [156](#)
Dependency-extractor [156](#), [158](#)
Dependency-matrix-action-registry [155](#)
Dependency-matrix-configurator [155](#), [156](#), [157](#)
Dependency-matrix-diagram-descriptor [155](#)
Developer-guide [368](#)
Diagram-comparator [34](#)
Diagram-listener-adapter [142](#)
Diagram-surface [145](#)
Distributed-resources [43](#)
Distributing-diagrams [43](#)
Distributing-documentation [43](#)
Distributing-plugins [43](#)
Distributing-profiles [43](#)
Distributing-samples [43](#)
Distributing-templates [43](#)

E

Eclipse [36](#)
Element [76](#)

Elements-name-change-event [100](#)

F

For-row [155](#)
Forward-engineering [190](#)

G

Get-dependencies [156](#)
Get-plugin-name [28](#)
Get-plugin-version [28](#)

H

Header-cell-renderer [157](#)
Helper-class [106](#)
Hyperlink-model [160](#)
Hyperlink-model-active [160](#)
Hyperlink-text [160](#)
Hyperlink-text-active [160](#)

I

Init [11](#)
IntelliJ [36](#)
Is-plugin-required [28](#)
Ispecification-node-configurator [165](#)

J

Jar [14](#)
Java-code-engineering-manager [191](#)
Java-manager [191](#)
Javascript [293](#)
Js [293](#)
Jython [290](#), [291](#)

M

Magicdraw-test-case [30](#)
Matrix-cell-view [157](#)
Matrix-data-helper [158](#)
Menu-command [52](#)
Model [66](#)
Model-comparator [34](#)
Model-comparator-filter [34](#)

N

Nashorn [293](#)
New-element-creation-event [100](#)
Node [165](#)
Number-part [182](#)
Numbering-scheme [182](#)

O

Omg-uml-metamodel [76](#)
Open-api [365](#)

P

Patch [14](#)
Path-connector [123](#)
Plain-text-file-format [288](#)
Plug-in [11](#)
Plugin [11](#), [11](#)
Plugin-descriptor [22](#), [291](#)
Presentation-element [141](#)
Presentation-element-render-provider [137](#)

Presentation-element-renderer [137](#)
Presentation-element-renderer-manager [137](#)
Project [66](#)
Project-descriptors-factory [67](#)
Projectdescriptorsfactory [67](#)
Projects-comparator [34](#)
Property-editor [161](#)
Property-manager [161](#)
Python [290](#), [291](#)

R

Resource-builder [41](#)
Resource-manager [41](#), [41](#)
Reverse-engineering [190](#)
Rhino [293](#)
Root-model [66](#)
Row [157](#)

S

Selection-events [141](#)
Set [123](#)
Setter [123](#)
Setup-eclipse-environment [36](#)
Setup-intellij-idea-environment [40](#)

Shortcut-menu-command [52](#)
Sset [123](#)
Stereotypes-helper [106](#)
Suplier-piont [125](#)
Supplier [125](#)

T

Table-cell-renderer [157](#)
Teamwork-utils [67](#)
Teamworkutils [67](#)
Transaction-commit-listener [104](#)
Tree [165](#)

U

Uml-model [115](#)

V

Validation-rule [169](#)
Validation-suite [169](#)

X

Xmi [288](#)

Z

Zip-based-file-format [288](#)