Poseidon for UML

poseidon for uml

user guide

Dr. Marko Boger

Elizabeth Grass

Matthias Koester

Poseidon for UML

by Dr. Marko Boger, Elizabeth Grass, and Matthias Koester

Table of Contents

List of Tables

Chapter 1. About Gentleware and Poseidon for UML

1.1. About Gentleware and Poseidon for UML

According to Greek mythology, the hero Jason built a ship and named it the Argo. With his comrades, the Argonauts, he left on a quest for the golden fleece. Poseidon, the god of the seas, protected and safely guided their journey.

About 4000 years later, Jason Robbins started an open source project for a UML modeling tool and named it ArgoUML. Many others joined him in this adventurous undertaking, including a group of software developers led by Marko Boger, who was at that time a researcher at the University of Hamburg. Together they greatly advanced the tool. After Jason Robbins shifted his focus to other tasks, the developer group evolved to become leaders of the project. Under their guidance and with their advances, ArgoUML became very popular. They realized the great demand for a tool like ArgoUML, as well as the amount of work necessary to shape it into a professionally usable tool. They finally took the risk of starting a company with the goal of bringing the most usable tool to a broad audience. With respect to their open-source origin, the company is called Gentleware and their tool is called Poseidon for UML.

That is who we are and how our quest started. Today, Poseidon for UML is one of the most popular UML modeling tools on the market. Our special focus is on usability and on making the job of modeling a joy.

1.1.1. Our Vision

Software development is a creative process. It requires a deep understanding of the problems to be solved, the involved users and stake holders and their requirements, the ability to find the right level of abstraction from reality, and the creativity to shape a software solution. At the heart of software development is the human being. Our goal is to provide tools to increase his creativity and productivity. Tom DeMarco found a word for this: Peopleware. This point of view is engraved in our name. Gentleware is the connection between humans and the software they develop. Our main subject is the development of tools for UML, Java, MDA and XML with a strong focus on usability and high productivity. We also offer training, consulting and individual solutions.

1.1.2. Innovation

New tools require new ideas. Innovation drives our development. We want to prove this to our customers through improved usability and productivity. Founded with a strong university background, Gentleware maintains our ties to the University of Hamburg. With roots in academia and the community process of

open projects, we continuously seek dialog with researchers along with the open source community and users.

1.1.3. Cooperation

The tools we build are used in a wide range of industries, and the pace of development is high and always increasing. To stay ahead, we cooperate with leading experts and companies. Together with our partners we are building a rich set of development tools and extensions that will fit the needs of our users exactly.

1.1.4. Contact

We are always very happy to get feedback on our tools and services. If you want to contact us, there are several ways to get in touch.

Email

The easiest way to contact us is via the web form. We offer addresses for different purposes.

General information, feature requests, or suggestions:

http://www.gentleware.com/index.php?id=contact (http://www.gentleware.com/index.php?id=contact)

Customer support (for premium Editions):*

Please keep in mind that our customer support is available during our normal working hours - Monday through Friday, 9 am to 6 pm Central European Time. Consider checking the forums and the FAQ to answer your question first.

http://www.gentleware.com/index.php?id=supportreq
(http://www.gentleware.com/index.php?id=supportreq)

Questions on purchase process, quotes, or volume sales:

sales@gentleware.com (mailto:sales@gentleware.com)

* Community Edition users are encouraged to use our forum for support questions.

http://www.gentleware.com/fileadmin/forum.html (http://www.gentleware.com/fileadmin/forum.html)

Web Site

For general discussion we have installed an open forum (http://www.gentleware.com/fileadmin/forum.html) in which users of Poseidon for UML can freely discuss topics related to our tools. Typically these are questions on how to do something, discussions on what other features would be nice, or comments on what people like or dislike about our tool. Our staff is actively taking part in these discussions, but you might also get a response from other users.

To order our products you can use the online shop (http://www.gentleware.com/index.php?id=121) , which requires a credit card. If you do not have a credit card or you hesitate to use it over the web, send us an email at sales@gentleware.com (mailto:sales@gentleware.com) .

Phone

Our preferred payment method is credit card. However, if you do not have a credit card or you hesitate to use it over the web, you can also send a fax, send email to sales@gentleware.com (mailto:sales@gentleware.com) or call us.

There is a fax order sheet provided on our web site. Begin the order process as usual, then select 'Order by Fax' at the bottom of the Checkout page. Our fax number is +49 40 2442 5331.

Please try to find an answer to your question on our web pages (http://www.gentleware.com/index.php?id=support) , the FAQ list (http://www.gentleware.com/index.php?id=faq) , or the Poseidon Users Guide (http://www.gentleware.com/index.php?id=userguides) .

Regular Mail

To send us mail or to visit us in person, our address is:

Gentleware AG

Ludwigstrasse 12

20357 Hamburg

Germany

1.2. New Features in Version 6.x

Version 6.x has been specifically redesigned with usability in mind. Now you can be more productive than ever with Poseidon, using particularly the following enhancements:

- HotSpots
- HTML documentation preview
- Image import for elements and stereotypes
- New zoom options
- Color gradients

A complete list of changes can be found at the change log section of the Gentleware web site (http://www.gentleware.com/index.php?id=142) .

1.3. About This Document

This document describes Poseidon for UML and how to use it. It is intended as a user guide. It is not a book about UML or Java. Basic knowledge about UML as well as Java is assumed.

We are working hard to make Poseidon for UML as intuitive as possible. You should be able to open up Poseidon for UML and start using it without looking into this documentation. However, you will find it useful to read through this document to get you up to speed faster and discover useful features earlier.

This version of the User Guide has been reorganized to help you find the information you need more quickly. The first four chapters help you get up and running with Poseidon for UML, chapters 5 through 13 are reference sections, and the final group of chapters discuss the more advanced features of Poseidon.

Chapter 2. Editions

Poseidon for UML is delivered in different editions. This section gives a rough overview of the editions so that you may decide which of these is most appropriate for you.

Poseidon for UML is directly based on ArgoUML (version 0.8) and you will find that what is described here closely resembles ArgoUML. However, Poseidon for UML is more mature, provides additional features, and is more stable. It is intended for daily work in commercial and professional environments. ArgoUML, on the other hand, is open source and lends itself to research, study of architectures, and extensibility. If you want to get your hands on the code and help advance the open source project, we greatly encourage you to do so. In that case, we recommend you to turn to the web site www.argouml.org.

Poseidon for UML is released in *Versions* as well as in *Editions* . All Editions are based on the same source base and therefore carry the same version number. New versions are released a couple of times per year. This document refers to version 6.x, although most of the information is relevant for versions 2.0 and later.

The Editions offer different features and come with different levels of support.

2.1. Community Edition

The Community Edition is the base version. Offered for free, it is the zero-barrier entry to the world of UML for the individual software developer as well as for large organizations. It makes learning and using UML a snap and enables the cost-effective exchange of models.

It is fully usable for modeling UML, and you may use it for any non-commercial purpose, particularly for educational uses. It contains all UML diagrams and all implemented diagram elements. You can create, save, and load projects, browse existing models, exchange models, generate Java code, export your diagrams to various formats and much more. You may freely distribute it, put it on local or Internet servers, and distribute it on CDs or DVDs. Gentleware does not provide support for the Community Edition.

Generally speaking, the Community Edition provides everything you need to learn and to use UML at a non-professional level. However, there are a few restrictions. Some of the features that are available in the commercial editions are not included in the free Edition. These features are nice to have to increase your productivity, but are not necessarily required to build UML models. Perhaps most important, the Community Edition does not support reverse or round-trip engineering, and it cannot load plug-ins. The other Editions meet the requirements of professional users.

The Community Edition has the following features:

- Fully implemented in Java, platform independent.
- All 9 diagrams of the UML are supported.
- Compliant to the UML 2.0 standard.
- XMI 1.2 is supported as a standard saving format. XMI 1.0, 1.1 and 1.2 can be loaded.
- Diagram export as gif, ps, eps and svg.
- Graphic formats jpeg and png supported for JDK 1.4.
- Copy/cut/paste.
- Some diagrams allow drag and drop within the tool.
- Internationalization and localization for English, German, French, Spanish, and Simplified Chinese.
- Forward engineering for Java.
- Simple installation and updates with Java Web Start.
- Full Undo and Redo.

2.2. Standard Edition

The Standard Edition is the extendable base tool for the professional. It comes with all features of the Community Edition plus productivity features like printing, drag-and-drop to the Windows clipboard (copy diagrams to Word or Powerpoint), and full zoom. Through a plug-in mechanism you can pick and choose from a growing set of plug-ins that allow you to further extend its functionality. Additionally, we provide e-mail support for this edition.

UMLdoc, the HTML documentation generator, allows you to export your models to an HTML format an share it with others over the web or intranet. The outcome is similar to Javadoc, but includes all the information of a UML model including the diagrams; thus, we named it UMLdoc.

Poseidon for UML is constructed in a highly modular way so that additional features can be purchased from our technology partners and added to Poseidon by introducing new modules as plug-ins. The Standard Edition allows you to load (and unload) plug-ins at runtime. This functionality turns Poseidon for UML into a highly flexible and extensible platform of UML tools. The Standard Edition is the foundation for this.

The Standard Edition has some of the following additional features over the Community Edition:

- Forward and reverse engineering for Java
- Plug-in mechanism to load and unload plug-ins from our technology partners, even at runtime.
- Comfortable printing with fit-to-page print or multiple page split-up.
- Direct copy to the Windows clipboard, drag-and-drop to Word, Powerpoint, etc.
- HTML documentation generation into UMLdoc.
- Support from the Gentleware help desk via email.

2.3. Professional Edition

The Professional Edition is the prime version of Poseidon for UML. To meet the needs of the professional software developer, we have bundled the worlds most flexible code generation mechanism with a set of productivity features. This Edition includes round-trip engineering, JAR import, and HTML documentation generation.

One of the most valuable features of Poseidon for UML is its code generation technology, and the Professional Edition gives you full access to it. The code generation mechanism is based on a template technology, where the template defines the syntax of the outcome. This can be Java, C++, XML, HTML or what ever else you want it to be. The information from the model, like the names of classes and methods, are provided by Poseidon for UML. This Edition gives you access to the API and to the templates. As a developer you can edit and change these templates, even at runtime, and configure the outcome of the code generation yourself.

Sophisticated round-trip engineering for Java allows you to read in existing Java code and generate a UML model for it or to continuously synchronize your code with the model. You can change the generated code or redesign the model and never lose consistency between the two. With JAR import functionality you can read in existing libraries and use these in your models.

The Professional Edition has the following features over the Standard Edition:

- Template-based code generation with full access.
- Round-trip engineering for Java.
- JAR import to include existing libraries.
- Import of Rational Rose files (.mdl).

2.4. Embedded Edition

The Embedded Enterprise Edition is specifically designed for embedded systems development. In order to meet the needs of embedded systems engineers, this version bundles most of the features of the Professional Edition with optimized code generation for ANSI C and C++. Developers of these systems can now work together seamlessly while maintaining the embedded code generation support. The code generator has been uniquely created to fit the demanding criteria of embedded systems, such as memory resource and performance issues. It supports automatic code generation for UML state machine diagrams as well as class diagrams.

2.5. Edition Comparison

To give you a quick overview of the features in the different Editions, here is a table view of the available features:

Table 2-1. Edition Comparison

Feature	CE Community Edition	SE Standard Edition	PE Professional Edition	EmbEd Embedded Edition
UML 2.0 Diagram Interchange	✔	✔	✔	✔
All 9 Diagram Types	✔	✔	✔	✔
Forward Engineering Java	✔	✔	✔	✔

XMI Supported	✔	✔	✔	✔
Platform Independent	✔	✔	✔	✔
Export as GIF, JPG, PNG, PS, EPS, SVG, WMF	✔	✔	✔	✔
OCL Support	✔	✔	✔	✔
Printing	✔	✔	✔	✔
Undo/Redo	✔	✔	✔	✔
Copy/Cut/Paste	✔	✔	✔	✔
Complete Cut and Copy		✔	✔	✔
Clone Diagram		✔	✔	✔
UMLdoc (HTML Export)		✔	✔	✔
Reverse Engineering Java		✔	✔	✔
Plug-In Support		✔	✔	✔
Eclipse IDE Integration			✔	
JAR Import			✔	✔
MDL Import			✔	✔
Changeable Code Templates			✔	✔
C++ Generation			✔	✔
ANSI C Generation				✔
C# Code Generation			✔	
CORBA IDL Code Generation			✔	
Delphi Code Generation			✔	
Perl Code Generation			✔	
PHP Code Generation			✔	
SQL DDL Code Generation			✔	
VB.net Code Generation			✔	
Roundtrip Engineering Java			✔	
Concurrent License Option			✔	✔

Chapter 3. Installation and First Start

3.1. Prerequisites

Poseidon for UML is written entirely in Java and therefore is platform independent. It runs on almost any modern personal computer. To successfully start and run Poseidon for UML you need the following:

- Java 5 is required for Linux, Mac OS X, and Windows platforms when running Poseidon for UML. While it is possible to use Java Runtime Environment or Java Development Kit. JDK 1.4 or higher, errors may result. Poseidon for UML will not run with JDK 1.3 or older.
- A computer with reasonable memory and CPU power. For memory, 512 MB is recommended, more is helpful. For CPU, a Pentium III or equivalent is the recommended minimum.
- A specific operating system is not required. Poseidon for UML is known to run on Windows 98, 2000, NT, and XP, on Linux SuSe 6.X, 7.X, Red Hat, and MacOS X. It has been predominantly developed and tested on Linux. However, on Windows platforms performance is known to be superior due to a faster Java environment.

3.1.1. Additional Requirements for MacOS X Users

Certain combinations of Java and MacOS X have been known to cause problems. These are sometimes resolved by changing the Look And Feel used by Poseidon. For information on how to change the Look and Feel, refer to the Appearance section of the Settings chapter.

3.1.1.1. Jaguar (MacOS X 10.2.x)

There are two versions of Java 1.4.1, Original and Update 1. When using Original, Drag and Drop from the Navigation Tree into a diagram will not work. Update 1 includes a set of improvements and should be used whenever possible. In both cases, the Alloy Look and Feel should be used, as Aqua is known to have additional problems such as difficulty in opening dialogs.

3.1.1.2. Panther (10.3.x)

This version of MacOS X comes with Java 1.4.1 Update 2, which is an improvement over Update 1, yet there are still some issues with Drag and Drop and the Aqua Look and Feel. Be sure to use the Alloy Look and Feel with this version of Java.

Java 1.4.2 (which is equivalent to Sun JDK 1.4.2_03) has recently become available. This release appears to have addressed the speed and Look and Feel problems.

3.1.1.3. Tiger (10.4.x)

Tiger includes Java 2, Standard Edition, version 1.4.2, including the Java Developer Kit (JDK) and the HotSpot virtual machine (VM).

3.2. Moving from a Previous Poseidon Installation to Version 6.x

3.2.1. Models Created in Poseidon 3.0.1 and Lower

Older models can be opened in Poseidon 3.1 and later, but they will be saved in the Poseidon 3.1 format so they cannot subsequently be opened again in a version below 3.1. Before Poseidon opens the model, you will see a warning box.

Upon saving, Poseidon will automatically make a backup copy of your original model. This backup can be opened in pre-3.1 versions, but will not contain revisions made with 3.1. For extra peace of mind, you can always manually copy your original project.

3.3. Community, Standard, and Professional Editions

To install Poseidon for UML, you can choose any one of the following installation procedures:

- Install Poseidon for UML with InstallAnywhere.
- Install Java Web Start and start Poseidon for UML from the internet (Community Edition only).
- Download the compressed file (.zip file) over the internet and locally install Poseidon for UML.

A complete installation guide is available at http://www.gentleware.com/index.php?id=instguides (http://www.gentleware.com/index.php?id=instguides)

3.3.1. Install Using InstallAnywhere

The easiest way to install Poseidon for UML is to use the InstallAnywhere (http://www.installanywhere.com) installer for your platform. If you already have a recent Java version

installed, you can download the installer for your platform that does not include Java. If you do not have Java installed or if you are not sure of the version, download the installer that includes Java.

You will be asked to specify an installation folder and a location for shortcuts. Free disk space of about 20 MB is required.

3.3.1.1. Windows

The Windows installer functions similarly to other installation applications. It is an exe file that will prompt you through the entire process.

Following installation, start Poseidon for UML by selecting the icon placed in the 'Start' menu by the installer.

3.3.1.2. *NIX

The InstallAnywhere installer is known to run on RedHat, Caldera, TurboLinux, and SuSe. It will not work on Mandrake. If you are using Mandrake (or another *NIX flavor that appears not to like the installer), you should use the compressed file installation process.

Ensure that your installation file executable with:

chmod u+x <your-poseidon-installer>.bin

Then execute the file. If '.' is not in your path you should use:

./PoseidonCE_2_1_1Installer.bin to start the file.

3.3.2. Install from a ZIP File

If you prefer to install Poseidon for UML without an installer, you can download and install a platform independent zip file. Installation is very simple. In short, download the file, unzip it, open the created folder and run the start script. Follow these steps:

1. To locally install Poseidon, you first need to download the corresponding file over the internet. Make sure that you are connected to the internet, then open your favorite internet browser and go to http://www.gentleware.com/index.php?id=114 (http://www.gentleware.com/index.php?id=114) .
2. Follow the download instructions. You will then have a single file stored on your local hard drive in the location you have indicated.

- The file is compressed using zip format. Move this file to the folder where you would like to install Poseidon for UML. Then, to decompress it, call the zip program used on your platform. Here are some examples:
- On Linux or Unix, open a command shell, go to the folder where the downloaded file is stored (using the `cd` command), and call **unzip PoseidonPE-##.##.zip** . The file may be named differently, depending on the edition you downloaded.
- On Windows, start your Zip program. The Zip program should automatically start if you double-click on the downloaded file. Then extract the file to a folder of your choice by selecting extract and following the instructions.

3. All Poseidon files are extracted into a folder `PoseidonForUML_XX_##.##` .
4. Switch to the `/bin` subfolder.
5. Run the start script provided authentificationfor your platform:
   - On Linux or Unix, open a command shell, enter the `poseidon.sh` command, and press return.
   - On Windows, open the file explorer and double-click the start script `poseidon.bat` .

3.3.3. Silent Installation

Silent installation is a convenient way to install Poseidon without having to interact with the dialogs of the standard installation. There are a few differences based on the operating system in use: on Windows, the initial progress indicator and final 'clean up' screen will be visible during launch; and on Mac OS X supports silent mode only when running a UNIX installer. All default values except the installation directory are used in this mode.

You can perform a silent install via the regular installer using an Installer Properties file or the command line.

3.3.3.1. Installer Properties File

In order to install Poseidon in silent mode, you must first create a properties file. This file should be placed in the same directory as the installer, and given the same name as the installer with the file extension `.properties` . For example, if the downloaded installer is named `PoseidonPE_2_2Installer.exe` , then the properties file within the same directory should be named `PoseidonPE_2_2Installer.properties` .

Within this file, you can declare the installation directory and the installation mode through the variables `USER_INSTALL_DIR` and `INSTALLER_UI` .

Here are the contents of a sample properties file:

```
USER_INSTALL_DIR=/programs/PoseidonPE2/
```

```
INSTALLER_UI=silent
```

3.3.3.2. Command Line Parameter

To use the installer in silent mode from the command line, type:

installername -i silent

It is also possible to use the Properties file from the command line:

installername -f <properties file>

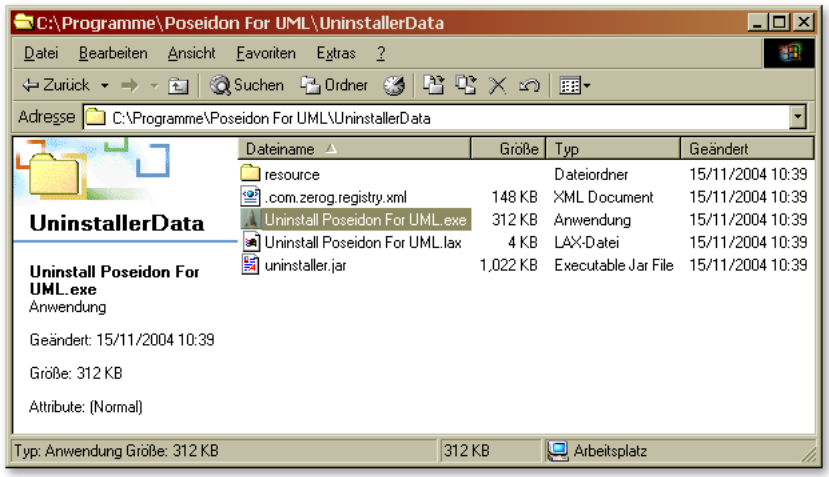
3.3.3.3. Uninstallation

When silent mode using a properties file has been used to install Poseidon and this file still exists, uninstallation is also completed silently.

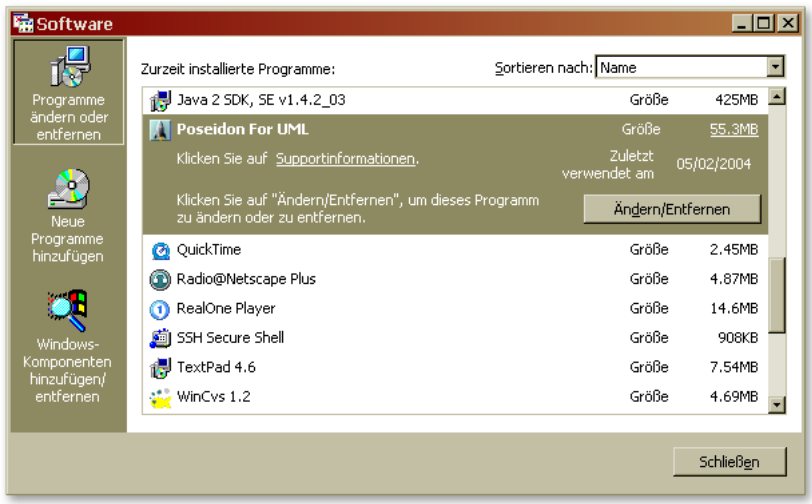
3.3.4. Uninstallation

The Poseidon installer places an uninstall utility within the program directory. You can uninstall

Poseidon by running this utility.



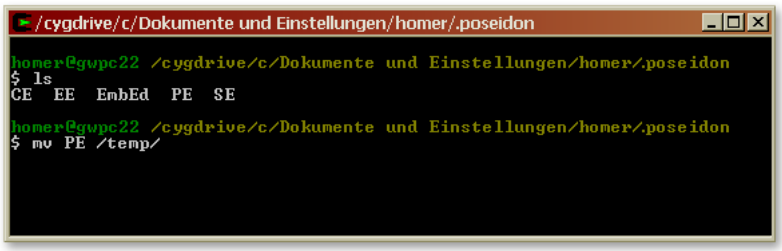
From Windows, you have the additional option of uninstalling Poseidon from the Software section of the Control Panel.



Once the uninstaller starts, follow the prompts to remove Poseidon from your system.

If you are planning to re-install Poseidon, you should copy your Poseidon settings directory to another location before starting the uninstall procedure in order to avoid losing your keys and registration data. In general, the Poseidon settings directory is located under your user's home directory in .poseidon/<

poseidon edition >, where < *poseidon edition* > is the abbreviation for the Edition. Below, the user 'homer' has moved his Professional Edition directory to a temporary location before proceeding with the uninstallation.



3.4. Environment Variables

The installation processes described above should enable Poseidon to run properly on your system. However, some adjustments can be made by using environment variables in order to make Poseidon fit even better in your personal environment. Please refer to the instructions of your operating system to see how environment variables can be set and persisted.

- `JAVA_HOME` - determines the path to the version of Java Poseidon should use for itself and the Java related tasks that can be done with it. Please remember that the code generation feature will need a complete SDK (i.e. a full install of Java that contains the compiler). If you don't want to generate and compile code from Poseidon, a runtime environment (JRE) is sufficient. Setting this variable is absolutely necessary for starting Poseidon using the batch scripts (poseidon.sh or poseidon.bat). The installer will search for the installed Java versions and let you choose which version to use for Poseidon. Once the installer has completed, this decision can be changed by a re-installation of Poseidon only.

- `POSEIDONxx_HOME` - determines the path to the folder where Poseidon can store user related settings and the log files. (Please note that xx stands for the edition you are using, i.e. CE for Community Edition, SE for Standard Edition and so on.) By default, Poseidon uses the home folder of the user. Please refer to the instructions of your operating system to see what folder is used as your home folder on your system. Some operating systems use rather strange settings for the home folder in networking environments, so it might be necessary to use a different folder than the default one. Defining this environment variable lets you choose a different folder. On Windows using the Standard Edition, you may want to set POSEIDONSE_HOME to `C:/Documents and Settings/yourname`.

3.5. Keys and Registration

To use Poseidon for UML you need a valid license key, which is a string of characters containing encrypted information. Obtaining this key is done through a simple registration process. Once you have the key, it is only a matter of pasting it into the application.

There are different types of keys. In this chapter we will explain the differences between these, how to get them, and what to do with them.

3.5.1. Types and Terminology

Evaluation Key - provided when an evaluation copy of a Premium Edition is requested from the website. These keys place a time-limit and functionality-limit on the application usage.

Serial Number - provided for the Community Edition and purchased copies of Premium Editions. The Serial Number is a unique identifier and is used to register the user with the specific copy of the application in order to receive the License Key.

License Key - provided for the Community Edition and purchased copies of Premium Editions. These keys are made available after the registration data has been received by Gentleware. Once a License Key is in place, the registration process is complete. These keys need no further attention, unless the copy is moved to another machine or is upgraded to another version.

Concurrent Licenses - licenses granted as part of a license-sharing scheme. Each license is granted for a limited period of time by a License Server. When the license is released by the user or called back by the Server, the license becomes available for use by another user. This type of licensing is available for the Professional, Enterprise, and Embedded Enterprise Editions.

3.5.2. Community Edition

The Community Edition may be used without export options immediately upon download. To access the export features such as saving and printing, a license must be rented from the Gentleware webshop (http://www.gentleware.com/shop0.html) . Rentals are available for a one-time fee or on an automatically renewing basis. After a license has been ordered, a serial number will be sent via email. This Serial Number must be registered online from within Poseidon or on the website in order to obtain the License Key required to access the export functions. The registration process is painless, and all information collected by Gentleware AG during the registration process is kept completely confidential. Our privacy policy is available for your perusal on the website.

To register a copy of the Community Edition:

1. Download and install a copy of the Community Edition.
2. After starting Poseidon for the first time, the License Manager will appear. Copy the serial number from the email, then click the 'Add' button.
3. Click on the Serial Number, then click the 'Register' button. A dialog will appear. Complete the form and click 'Next'.

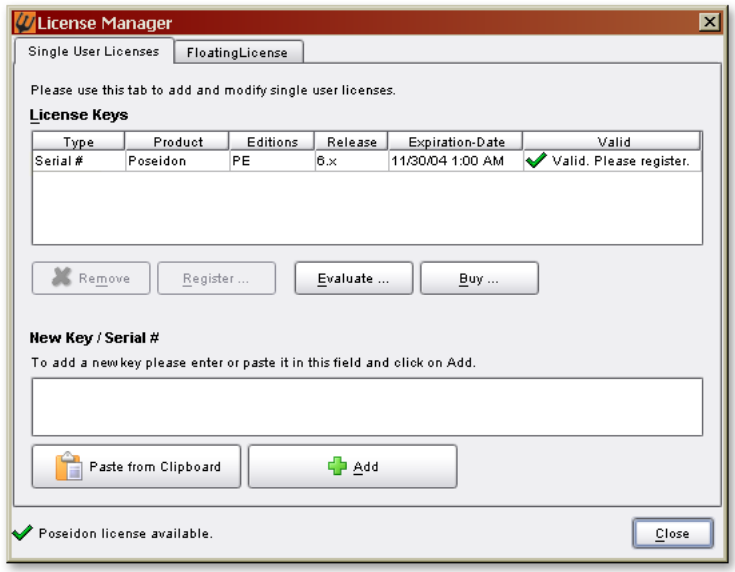
4. To finish the registration process, select either the online option or the web option (ideal for users who are behind a firewall).

3.5.3. Evaluation Copy

Premium Editions of Poseidon (Standard, Professional, Enterprise, or Embedded) can be evaluated free-of-charge, but be aware that some functionality is limited, e.g. saving is limited to eight diagrams. The evaluation key is valid for 30 days after registration. As with any registration with Gentleware, your information is kept strictly confidential.

To register an evaluation copy of a Premium Edition:

1. You will receive your Evaluation Key via email after filling out the evaluation request form and downloading the software from the website.
2. Enter the Evaluation Key in the 'New Key' box of the License Manager.



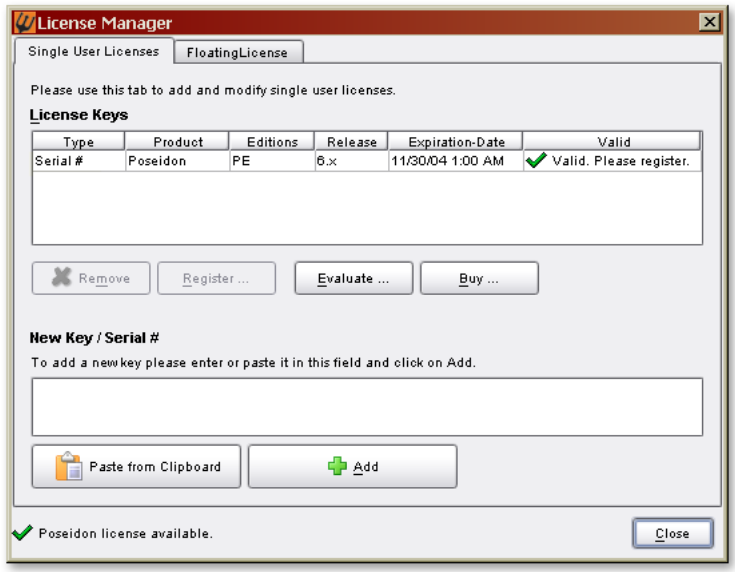
3. Click 'Add'.

3.5.4. Premium Version Purchase

Registering a purchased copy of a Premium Edition follows a similar process as an evaluation copy.

To register a Premium Edition:

1. You will receive your Serial Number via email after downloading the software.
2. Enter the Serial Number in the 'New Key' box of the License Manager.



3. Click 'Add'.
4. You will now need to register the Serial Number by clicking the 'Register' button and completing the registration dialogs that follow.

Note that if you try to register via the website, Poseidon searches for Netscape as the default browser. You can change the default browser from the Settings dialog in the Edit menu. Note that this setting does

not currently work for Macs.

3.5.5. Premium Version Upgrade/Update

To move from one edition to another within a version (e.g. from SE 6.0 to PE 6.0), you will need an Upgrade key. To move from one version to another within a single edition (e.g. from PE 5.0 to PE 6.0), you will need an Update key. Both of these are added to the License Manager as usual and will have to be registered. However, the registration process varies a bit from the standard registration process. You will be asked to provide access to you old key. There are three ways to do this:

1. Search old configuration and installation directories. Poseidon 3.x stores configuration in a new location, so the registration mechanism can search in old directories for the appropriate key.
2. Search a specified directory. You can tell Poseidon where you have stored the old key.
3. Paste the old key into the Registration dialog. If you have a copy of your old key in an email or have printed a copy, for example, you can type or paste it directly.

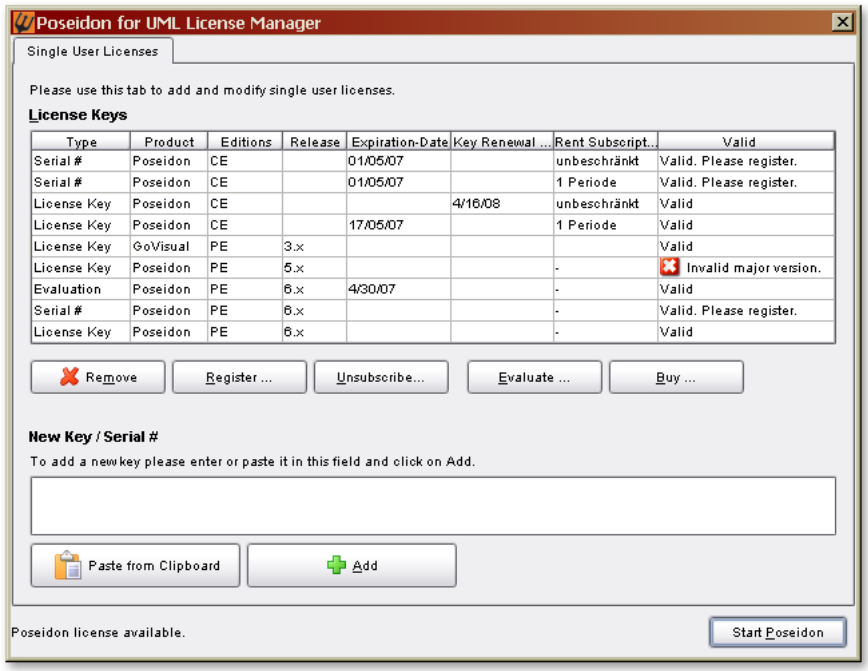
Once the old key has been successfully provided, the registration process proceeds as usual.

3.5.6. Keys for Plug-Ins

If you want to use additional plug-ins, you will need an additional key specific to that plug-in. Plug-ins that are free of charge or are in beta-release are delivered with a valid license key, similar to the Community Edition. For each commercial plug-in you purchase or want to evaluate, you are sent a Serial Number by email. You need to register these Serial Numbers to receive the corresponding License Key.

The Professional and Enterprise Editions come with four plug-ins. They do not need to be registered separately.

3.6. License Manager



Several licensing options are now available for Poseidon. Stand-alone licenses allow you to work with or without network connectivity for any length of time. These licenses are registered per machine and per user. Concurrent licenses allow a set number of people to use Poseidon at one time with users sharing licenses from a common license pool. It is possible to work offline with these licenses, but there is a time limit.

All editions can use stand-alone licensing. Concurrent licensing is available for the Professional, Enterprise, and Embedded Enterprise editions only.

For more information about installing licenses, please refer to the installation guide for your edition of Poseidon.

3.6.1. Key Descriptions

Each license contains information about the specific key.

- **License Key** - Describes the key type. In the Enterprise Edition, you may have a Serial Number that requires registration after the first three days, or you may have a Concurrent License. Concurrent Licenses are used for both the Server and Client applications.
  - `Evaluation` - Valid for a limited time.

  - `Serial #` - Used in the intermediate state of installation and is valid for a limited time. Must be registered in order to obtain a valid key.
  - `License Key` - Final, non-expiring key providing full access to all features of the Edition
  - `Concurrent License` - Keys for the Enterprise Edtion

- **Product** - Indicates to which product the key applies.
- **Editions** - Indicates the Editions for which the key is valid
  - `CE` - Community Edition
  - `SE` - Standard Edition
  - `EmbEd` - Embedded Edition
  - `PE` - Professional Edition

- **Release** - The key is valid for the stated release number. In the case of plug-ins, this number refers to the release number of the plug-in, not to the version of Poseidon.
- **Expiration Date** - Indicates when this key expires. In most cases, a Serial Number has a 3 day limit, and an Evaluation Key has a 30 day limit. Purchased copies do not have expiration dates; rented licenses expire based on the terms of the rental.
- **Valid** - Indicates the status of the key. You may have a key that is valid but needs to be registered, a limited key that has expired, or a valid key.
  - `Valid. Please register.` - Keys that display this message will only work for a limited amount of time. Once the key has expired, it is still possible to register the key, but it is not possible to use the key to operate Poseidon.
  - `Expired` - The key was valid for a limited amount of time, which has now passed. In the case of an expired Serial Number for a purchased edition, it is still possible to register the key and activate Poseidon.
  - `Valid` - The key is valid and operational.

3.6.2. Buttons

- **Remove** - Removes a key from the list of keys.
- **Register** - Registers a Serial Number with Gentleware.
- **Evaluate** - Opens the Gentleware web site in a browser so that you may select a program to download and evaluate
- **Buy** - Opens the Gentleware web site ina browser so that you may select a program to purchase and download.

3.6.3. Add a Key

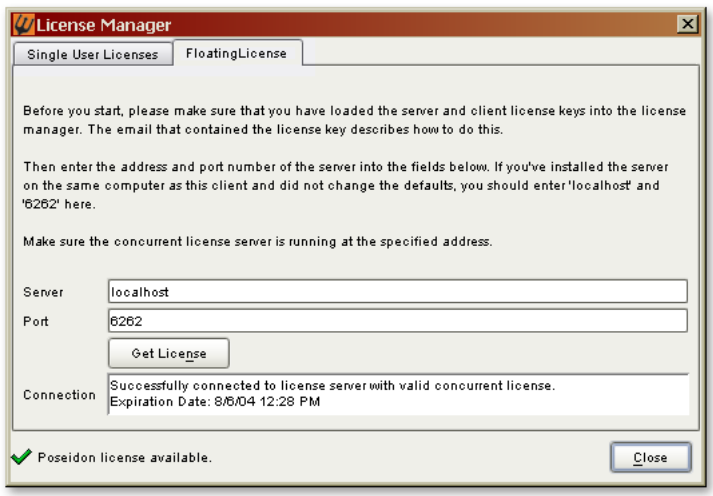
- **Paste From Clipboard** - Places the contents of the clipboard into the text box above. This is used to copy the text key from an email or from the Gentleware web site.
- **Add** - Adds the pasted key to the list of available keys.

3.6.4. Using Concurrent Licenses with the Professional and Embedded Editions

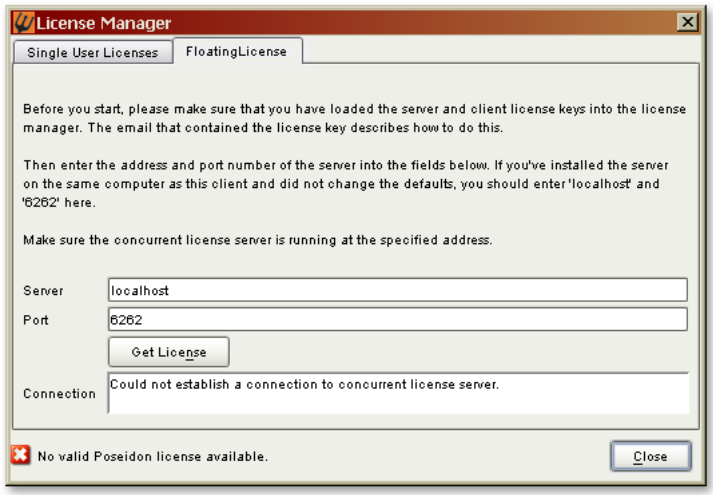
Complete information about installing and using concurrent licenses with the Professional and/or Embedded Edition is available from the Poseidon Installation Guide (http://www.gentleware.com/index.php?id=instguides) .

Poseidon will first check to see if a stand-alone license exists. If a license exists, Poseidon will use that license and leave the concurrent licenses for other users.

If a stand-alone license is not present, Poseidon will attempt to contact the license server for a concurrent license. To see where Poseidon is looking for the license server, open the License Manager from the Help menu. The Floating License tab contains the server address and port it is using to contact the license server. If concurrent licenses are available and Poseidon is able to contact the license server, a license will be automatically assigned and you may begin to work with Poseidon.

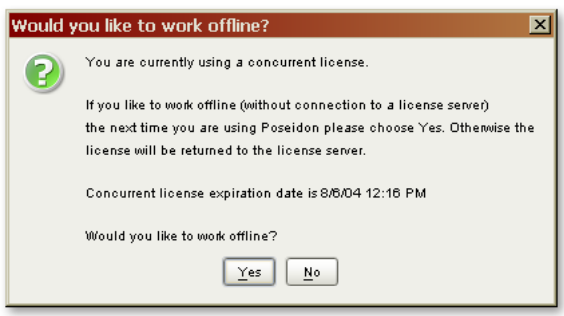


If the license manager can be contacted but no licenses are available, Poseidon will fail to start. If Poseidon is unable to connect to the license server, the License Manager will open with the message 'No Licenses Available'. If you are not sure where your license server is installed, contact your system administrator for this information.



Working Offline

Once you have a license, it is possible to disconnect from the license server, keep the concurrent license for a limited time, and work offline. When you close Poseidon, a dialog will ask you whether you would like to return the license or not. Selecting 'Yes' will allow you to keep your license until the expiration date stated. Within the License Manager, the status will read: 'Working offline with a valid concurrent license. Expiration Date: <your expiration date>'.



Chapter 4. A Short Tour of Poseidon for UML

This chapter introduces all basic concepts of Poseidon for UML by guiding you through an example project. On our tour, we will touch most features and a great variety of UML elements. However, this is not intended as a UML reference guide; thus, it will not explain all of the details of the modeling process. It will gradually teach you what you can do with Poseidon for UML and how you can use it for your own purposes.

4.1. Opening the Default Example

Let us start our tour through Poseidon for UML. The product is distributed with an example project, which we will be looking at during the guided tour. If you want to follow the tour on your own computer (highly recommended), do the following:

- Start Poseidon.
- From the main menu, select Help , then Open Default Example

This example is based on a car rental scenario in which a company wants a web store and needs to model its business processes and create a corresponding software system. This is a typical situation for the usage of a CASE tool, but UML as well as Poseidon for UML are not restricted to this kind of application design. As a general tool, Poseidon for UML can be used to model any kind of object-oriented software system, as well as a system that has nothing to do with software at all, such as a business-workflow system.

4.2. Introducing the Work Area

The work area of Poseidon is separated in five parts. At the top of the window, there is a main menu and

a toolbar that provide access to the main functions. Below this are four **panes:**

Figure 4-1. Poseidon for UML application work area.

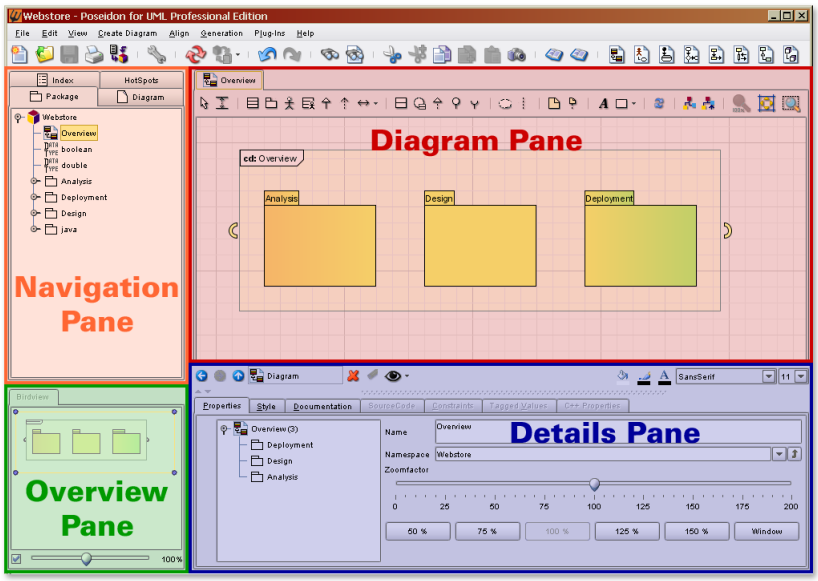


Diagram Pane

- Generally the largest pane
- Located in the top right-hand section of the screen
- Displays the various UML diagrams and is the main working screen

Navigation Pane

- Located in the top left-hand section of the screen
- Displays models and model elements based on the selected view
- Provides quick and intuitive movement through the diagrams

Overview Pane

- Located in the bottom left-hand section
- Bird's-eye view provides another means of navigation and display control

- Usually the smallest pane

Details Pane

- Located in the bottom right-hand section of the screen
- Displays all information about selected elements, some of which may not be available in the diagram
- Provides the means to add or change details of an element
- Yet another means of navigation

You can hide and redisplay panes by clicking on the small arrows that are located on the separation bars between panes, much in the same way you can manipulate panes in most other GUI applications. This allows you to gain extra room for drawing in the Diagram pane while the other panes are not needed. You can also resize the panes to best fit your needs by moving the separation bars with the mouse.

4.2.1. The Navigation Pane

The first pane we will explore is the Navigation pane in the upper left corner. It is used to access all of the main parts of a model by presenting the elements of the model in various tree structures. There are many different ways the model information could be organized into a tree structure; for example, the tree could be sorted alphabetically by element name, by diagram name, or by model element type. The classic way to organize them is by **packages** . Poseidon for UML uses the package structure as the default navigation tree, as do most UML tools. But, as we will see a little later, Poseidon provides additional tree structures called **views** . This is one of the strong points of Poseidon for UML, providing enormous flexibility for navigation. The default view is called the `Package` view.

The root node of the tree is the model itself, in our example it is called `Webstore` . The first level of the tree is open by default. In the Package Centric view, all first-level packages are shown, as well as all model elements that are not inside a specific package. As you can see, each element in the tree is preceded by a little icon. Element icons have one symbol, diagrams have several of these symbols combined into one icon. These icons are used consistently throughout the application.

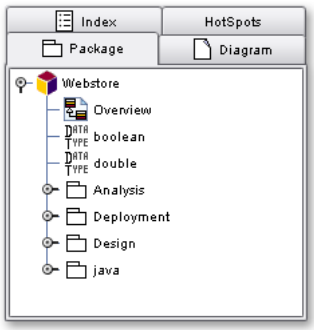
Some sample icons:

- **Model** - The model icon is a colored box, which is also used as a logo for UML
- **Package** - The package icon is a folder
- **Class Diagram** - The class diagram icon is a combination of two class icons

You can navigate through the tree by clicking on the icon in front of an element name, similar to many other applications. Any element you subsequently add to the model will automatically appear in the corresponding branch of the tree hierarchy, no matter how it is created.

Right now, your Navigation pane should look like this:

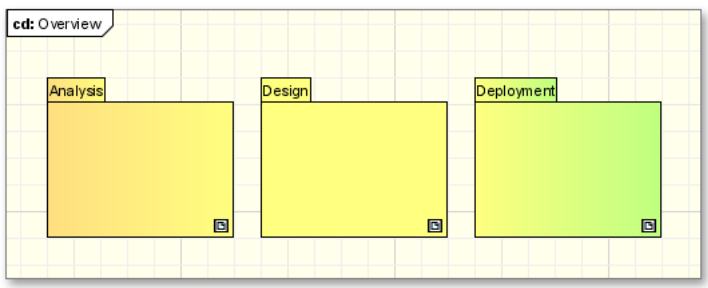
Figure 4-2. Navigation pane in the Webstore model.



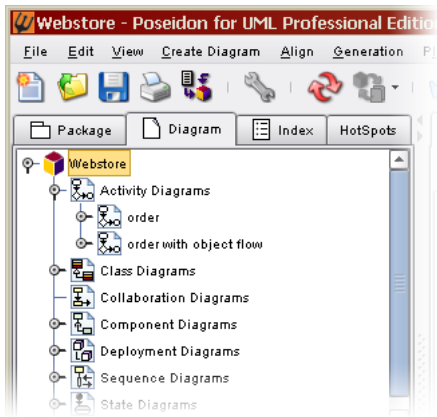
The model Webstore contains many packages (e.g. Analysis and Design) as well as diagrams (Overview).

Select the class diagram Overview by clicking on it in the Navigation pane. The selected diagram will then be displayed in the Diagram pane, which is located to the right of the Navigation Pane. The 'Container Class Analysis-Packages' diagram (Figure 6-3) visualizes the dependencies between the included packages: Analysis , Design , and Deployment .

Figure 4-3. Class diagram 'Overview'



Inside the packages you can find further diagrams, but to quickly browse through the existing diagrams you need not navigate through the packages themselves. You can find diagrams directly (and much more quickly) using the **diagram tree** .



This view sorts the model elements according to the diagrams in which they are included. Of course, this view includes only includes those model elements that are included in at least one diagram. The organization of this view has the advantage of quick navigation to any diagram or to the elements they contain. It logically follows that sometimes the Diagram Centric view and at other times the Package Centric view is more useful. Take a few moments now to look at the other available views.

We will now turn our attention to the diagrams themselves and how to edit them by looking at the Diagram Pane.

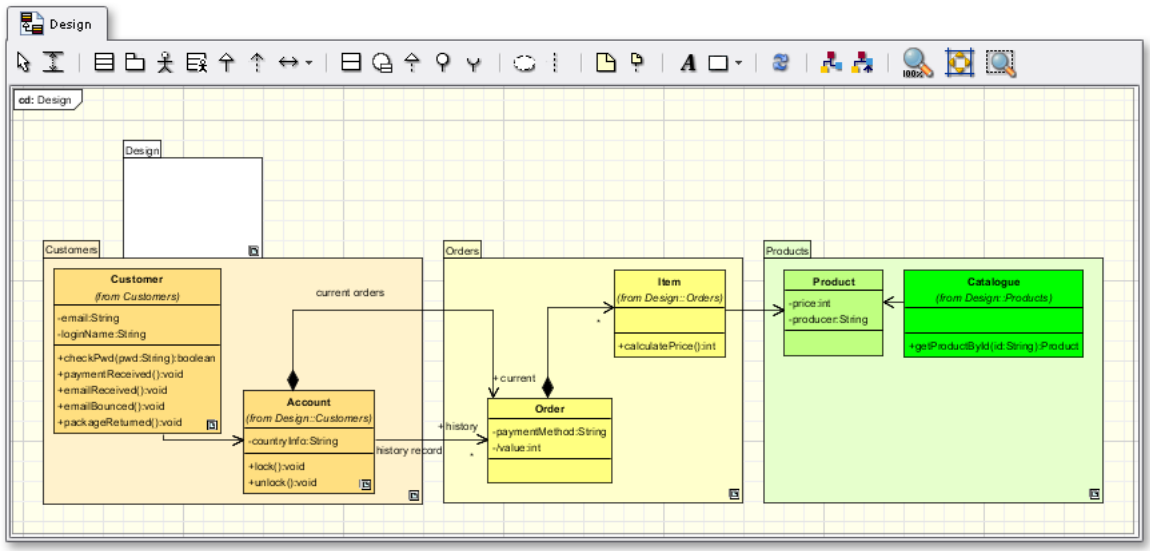
4.2.2. The Diagram Pane

As diagrams are the center of UML, naturally the Diagram pane is the main working space in Poseidon for UML. It is the primary place for constructing and editing the diagrams that compose all models. Just as the Navigation pane can display multiple views, the Diagram pane also makes use of tabs to open additional workspaces. Let's take a closer look at some of the functions available in the Diagram pane.

Open the diagram `Design` in the Diagram pane by clicking on its name under Class Diagrams in the Navigation pane. Expand the tree for this diagram by clicking the 'expand tree' icon that appears to the left of the diagram name.

The Diagram pane to the right should now look like this:

Figure 4-4. The Diagram pane displaying the diagram 'Design'.



This is an overview diagram that provides a high level view of the design of our example. The classes from this diagram happen to be located in different packages. You can see the package name in parentheses under the class name (e.g. *(from Design::Customers)*). For each package in this example, there is another diagram you can view that shows the classes of that package and how they relate to each other. In UML, model elements can be represented in different diagrams to highlight specific aspects in different contexts. Other diagrams covered later in this guide will give us another perspective.

This diagram shows important classes from our example model. It already tells you quite a bit about this example, such as:

- It models `Accounts` that are associated with `Customers` .
- `Accounts` can have `Orders` and a history is maintained
- We also see how the packages `Customer` , `Orders` , and `Products` relate to one another.

If you select one of these classes in the navigation tree, you will see that the corresponding class is also selected in the diagram. Similarly, if you select a class in the diagram it is also selected in the Navigation pane. This is true for all elements: your selection is synchronized between the different panes.

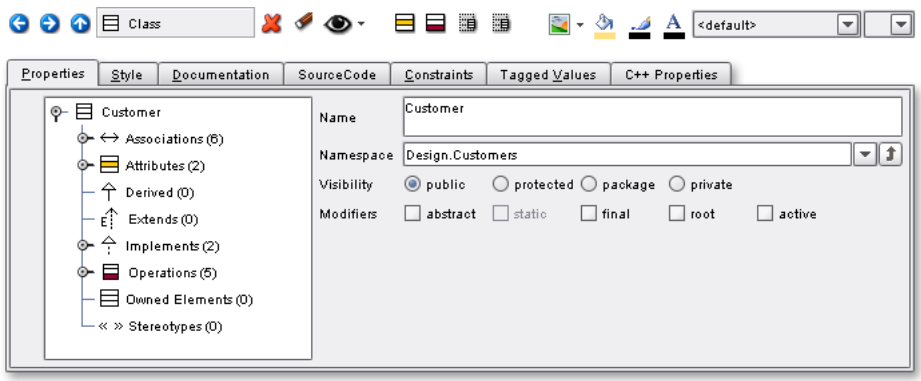
Try it for yourself by selecting one of the classes in this diagram from the Navigation pane. Notice how the class name is highlighted in the Navigation pane, while the Diagram pane displays the same class with its rapid buttons visible around it.

4.2.3. The Details Pane

So much more goes into a model than just the shapes representing elements and the connections between them. But if all of this information were displayed in the Diagram pane, the diagrams would quickly become cluttered and unreadable. The Details pane organizes and presents all of these important particulars via tabs.

So let's now take a closer look at the Details pane, located at the bottom of the application. Select the class `Customer` by either clicking on the class itself in the diagram or clicking on the class name in the Navigation pane.

Figure 4-5. The Details pane with class 'Customer' selected.



The Details pane is composed of seven tabs. These tabs (sometimes referred to as panels) display all of the detailed information about the element currently selected, allow changes to be made to these elements, add related elements, or delete the element all together. Properties can be changed, documentation can be written, the resulting code can be previewed and edited, and more. The tabs always reflect the currently selected model element and are enabled only if they make sense in the context of the selected element. The Details pane also serves as another mechanism to navigate through the model.

Tabs available in the Details pane:

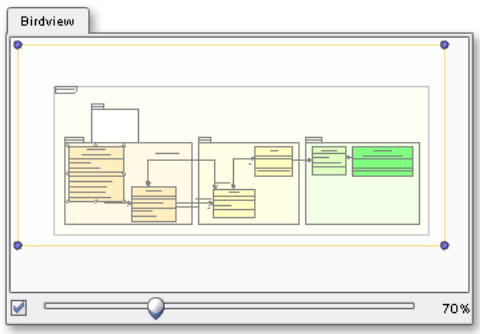
- Properties
- Style

- Documentation
- SourceCode
- Constraints
- Tabbed Values
- C++ Properties

4.2.4. The Overview Pane

The larger a diagram becomes, the harder it gets to keep track of all of the elements, especially once they are out of the immediate viewing area. The Overview pane allows you to keep track of the elements already in the diagram. The pane, located at the bottom left, provides access to the 'Birdview' tab, which displays a graphic summary of the diagram currently displayed in the Diagram pane. From this tab you can zoom and/or pan in either the Diagram pane or the Overview pane.

Figure 4-6. Class diagram as seen in the Birdview tab



To directly scale the section displayed in the main diagram area, enable the checkbox in the lower left-hand corner and use the slider bar to adjust the zoom factor. To pan and zoom the small diagram in the Birdview tab without disturbing the Diagram pane, disable this checkbox.

4.3. Navigation

A UML model can become quite complex as it expands to include more and more information. Different aspects of the model are important to a variety of people at particular times. Additionally, there is no one correct approach to viewing a model and the information it contains. A UML tool should provide

comprehesive yet simple-to-use mechanisms to access and change that information as each individual requires. Therefore, Poseidon for UML offers various ways of navigating between model elements to accomodate all of these needs. We will now take a closer look at some of the most important ones.

4.3.1. Navigating with the Navigation Pane

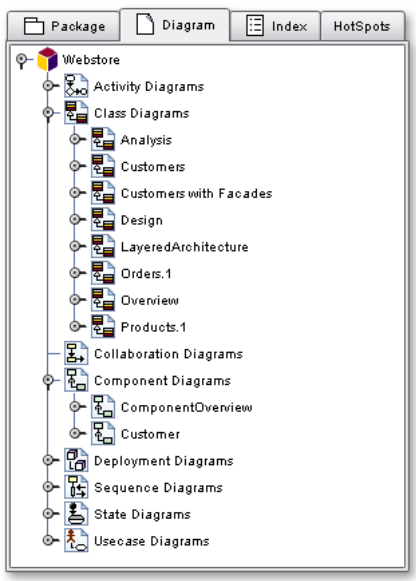
The central mechanism for moving through the models is the Navigation pane, mentioned above. It organizes the complete UML model into a tree view that provides access to almost all parts of that model via the opening and closing of subtrees. At the top of the Navigation pane you will find a drop-down menu where you can choose between a number of views.

Views available from the Navigation pane

- Diagram View
- Model Index
- Package View
- HotSpots

Each view organizes the tree structure with its own different focus. By default, the Package view is displayed. To change the view, simply select a different tab.

Figure 4-7. The Navigation pane in a Diagram view.



Verify that the current view is the Diagram view. From this view you can see all of the diagrams contained in the model at one glance. By clicking on one of the diagram names or icons, the corresponding diagram is shown in the Diagram pane. The elements contained in that diagram are displayed when the subtree is expanded.

The first two views (Package and Diagram) are the most commonly used views. The others are primarily used for more limited cases; for example, to find out the inheritance structure of the model or the structure of the navigation paths.

Remember that the Navigation pane displays the complete model, while a single diagram will only show you specific aspects of it. It is possible that there may be elements that are not contained in any diagram at all and are therefore only accessible from the Navigation pane.

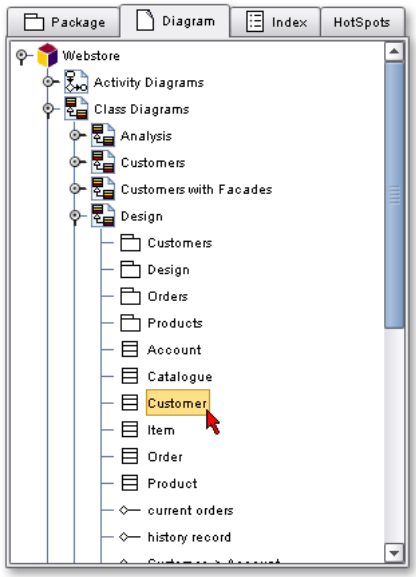
Take a look at the Model Index view by selecting Model Index from the drop-down menu. The Navigation pane will change to display an alphabetical list of all elements in the model. This illustrates yet another useful way to locate elements.

4.3.2. Navigating in the Properties Tab

The Properties tab in the Details pane provides a very convenient method of drill-down navigation. Navigating in such a way is very intuitive due to the relational nature of the elements and therefore of the navigation between them. It is easy to visualize moving from a class to a method of that class to a parameter of the method.

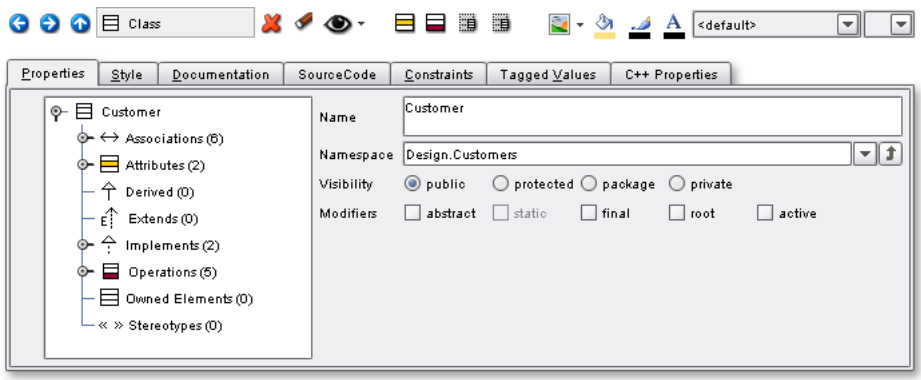
From the Diagram view in the Navigation Pane, open the diagram 'Design' subtree and select the class 'Customer'.

Figure 4-8. Select class 'Customer' from the Diagram view



Take a look at the right side of the Properties tab. Listed here are properties of the class itself which can be modified, such as name and visibility. To the left are components of the class, elements in and of themselves. These components have their own properties in their own properties tab.

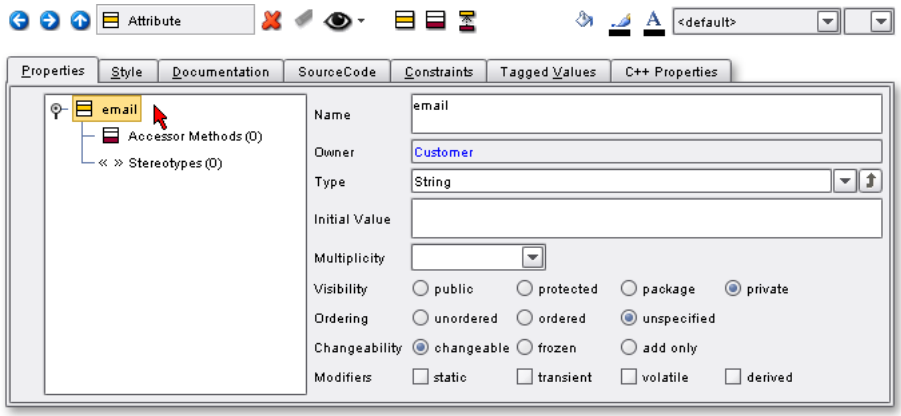
Figure 4-9. The Details pane with the class 'Customer' selected.



Open the 'Attributes' subtree and double-click on the attribute called 'email'. Notice that the Properties

tab has changed and now displays the properties of the attribute. Notice, also, that the fields present on the right side have changed to details which are useful for attributes instead of classes.

Figure 4-10. The Properties tab with the attribute 'email' selected.



4.4. Modify Elements

Once we have arrived at a desired element, we may need to make some modifications. Poseidon provides several ways of changing information relating to an element.

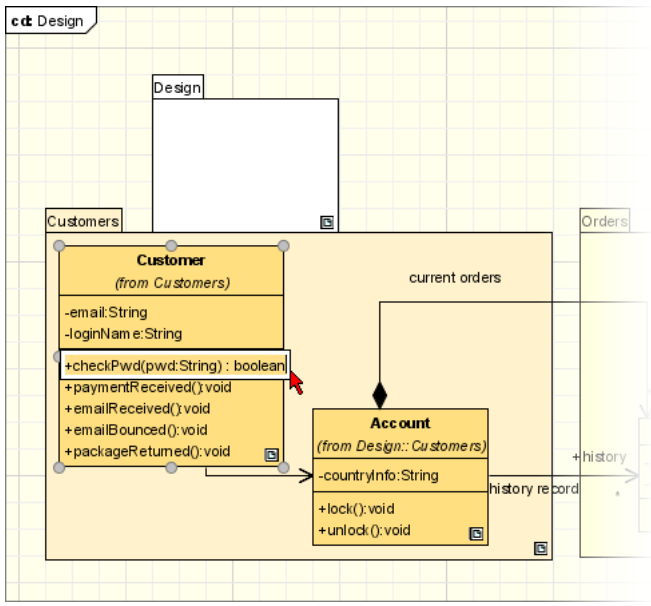
4.4.1. Change Elements

The quickest and easiest way to change information relevant to an element is to change it directly in the diagram. Be aware, however, that not all information can be changed in this way.

At this point in the tour, the Diagram pane should be displaying the class diagram titled, 'Design' and the Details pane should be displaying information about the attribute 'email'. We will now change the name of an operation from the same class, 'Customer'.

In the Diagram pane, double-click on the operation 'checkPwd()' in the class 'Customer'. The Details pane displays the information about this operation, and the text in the Diagram pane itself is now editable from a text box. Change the name of the operation to 'checkPassword()' and then press Ctrl-Return or click elsewhere in the diagram.

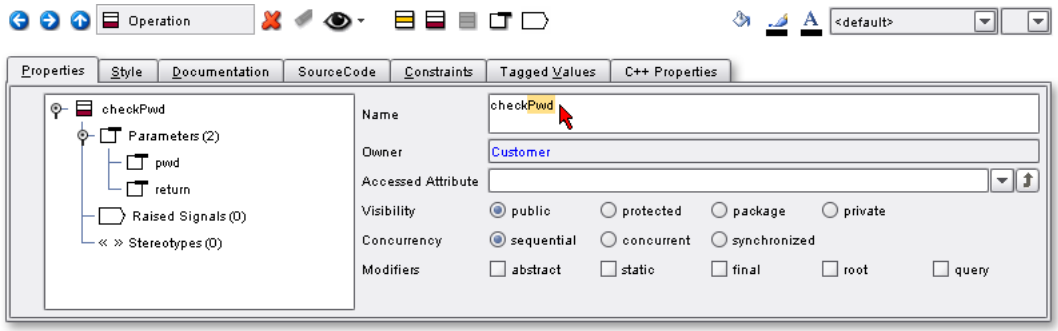
Figure 4-11. Change an operation name in a diagram



The name change will be propagated throughout the model, with ’Customer.checkPwd()’ replaced by ’Customer.checkPassword()’ in every instance.

Another method of changing information is via the Details pane. Select the operation ’checkPassword()’ again. Notice that the Details pane provides lots of information about this operation. In the ’name’ field, change the name from ’checkPassword()’ back to ’checkPwd()’ and press return or change the focus of the window by moving the mouse out of the Details pane. The change will now be reflected back in the diagram.

Figure 4-12. Change operation name from the Details pane



4.4.2. Create Elements

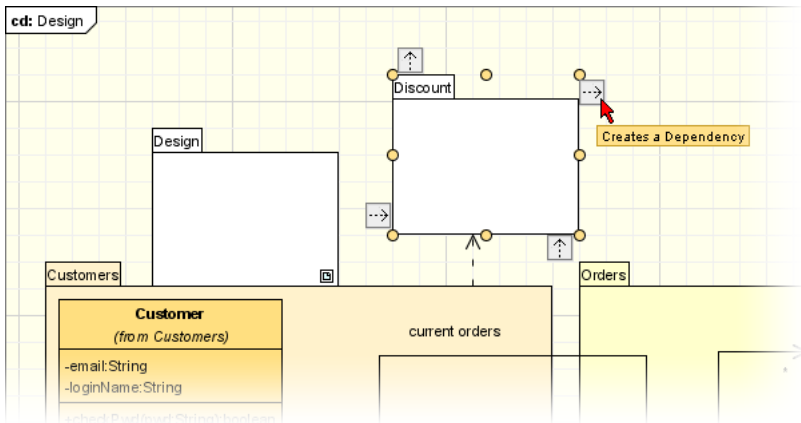
Creating new elements is just as simple as modifying existing ones. And just like changing elements, there are several ways to create new ones.

Perhaps we would like to associate a discount program with customers. Let's create a new package, Discount. In the 'Design' class diagram, click on the 'Package' button from the Diagram pane toolbar. A cross-hair appears. Click in the Diagram pane to place the new package into the diagram. Rename this class 'discount' using one of the methods outlined in the previous section.

Now that we have the package, we need to associate it with the package 'Customer'. We could do this by creating a new association through the toolbar and connecting the association ends to the classes, or we could speed things up and use the aptly-named 'Rapid Buttons'. Click on the new package 'discount'. Several buttons appear around the edges of the package. Click and hold the mouse button down on the left rapid button. Drag the crosshair that appears onto the package 'Customer' and release the mouse button. An association has now been created.

Now perhaps we need to make a connection between 'Discount' and a region because different discount schemes are offered in different regions. This will require the addition of another package and another association. One rapid button can take care of everything. Select 'Discount' in the diagram. Click (and this time do not hold) the mouse button on the right rapid button for the package 'Discount'. An new package and an association have been added to the diagram. Rename the new package 'Region'.

Figure 4-13. Add a package to a diagram with the rapid buttons

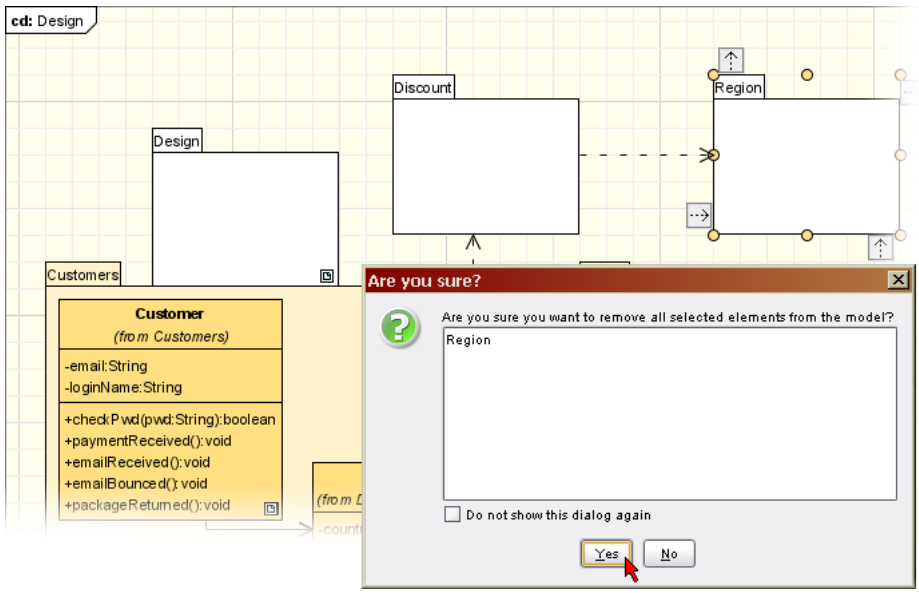


4.4.3. Delete Elements

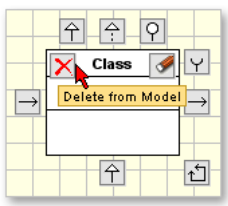
So after further review, we have decided that the package 'Discount' is a good idea, but not for this diagram. We have further decided that the 'Region' package is unnecessary and will not be used elsewhere in the model. Let's first delete 'Region' completely.

Select the package 'Region' in the Diagram pane. Now press the 'Del' key. A dialog box will prompt you before removing the class. Notice that the association has been deleted as well, as there is no point to an association with only one end.

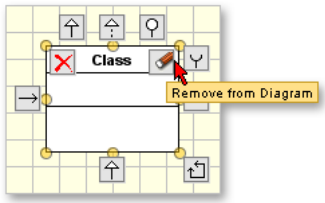
Figure 4-14. Delete an element from a model



To delete an element without encountering the dialog box, you can press Ctrl and mouseover the element. A delete rapid button will appear in the upper right corner.



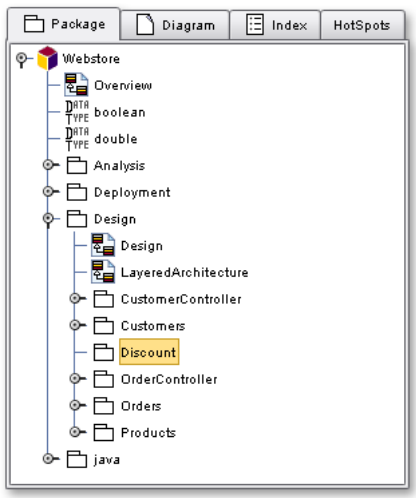
The package 'Discount' is a different story. We may need to use this again elsewhere, so we just want to remove it from this one diagram, not from the entire model. To remove it from this diagram, select the element in the Diagram pane and press the 'Remove from Diagram' button. A rapid button exists to do this as well; hold Ctrl and mouseover the element. Then use the 'Remove from Diagram' button.



Alternatively, you can cut an element from the diagram using either the Cut option from the main toolbar, Cut from the Edit menu, or the shortcut Ctrl-Shift-X. You will not encounter a warning here, but the package still exists within the model and can be viewed from the Navigation pane.

Notice that the element no longer appears in the Navigation pane under the class diagram. Change the Navigation pane to display the Model Index view and take a look at the packages listed there. You will see that, although it is not included in any current diagrams, the package 'Discount' still exists and is ready to be used in another diagram.

Figure 4-15. Remove an element from a diagram



Now that you have seen some of the basic ways of working with models in Poseidon, you should be ready to strike out on your own and see all that Poseidon has to offer.

Chapter 5. Interface

5.1. Toolbar

 **New Project**

 **Open Project**

 **Save**

 **Print**

 **Import Files**

 **Open Settings Dialog**

 **Roundtrip Enabled****

 **Roundtrip Disabled****

 **Synchronize Model and Code****

 **Start Roundtrip Import****

 **Start Code Generation****

 **Open Roundtrip Settings****

 **Undo**

 **Redo**

 **Jump to Element**

 **Jump to Diagram**

 **Cut**

 **Complete Cut***

 **Copy**

 **Complete Copy***

 **Paste**

 **Clone Diagram*** (not available for Sequence Diagrams)

 **Open Preview Document Panel**

 **New Class Diagram**

 **New Use Case Diagram**

 **New State Machine Diagram**

 **New Activity Diagram**

 **New Collaboration Diagram**

 **New Sequence Diagram**

 **New Object/Component Diagram**

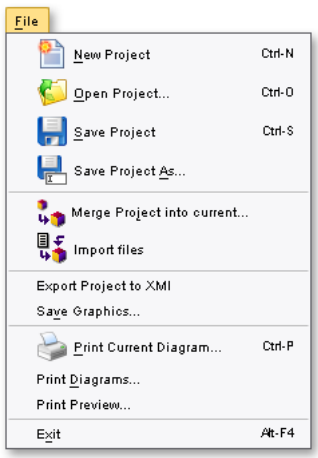
 **New Deployment Diagram**

* Not available in the Community Edition

** Available in the Professional and Enterprise Editions only

5.2. Menus

5.2.1. File



- **New Project** - Opens a new project with a blank class diagram. If a project is already open, it will prompt for a save.

  Quick Key - Ctrl-N

- **Open Project** - Opens a browser dialog to select a file to open.

  Quick Key - Ctrl-O

- **Save Project** - Saves the project to its last saved location.

  Quick Key - Ctrl-S

- **Save Project As...** - Opens a save dialog to save a project with a new name and/or location.
- **Merge Project into Current...** - Opens a browser dialog to select a project to combine with the project currently open in Poseidon.

  The file types that may be merged are: zuml, zargo, and xmi.

- **Import Files** - Opens a browser dialog to select files to incorporate into the project currently open in Poseidon.

  The file types that may be imported are: java, jar, mdl.

  > **Note:** Community Edition import is limited to java files; importation of jar and mdl files is available in the Premium Editions.

- **Export Project to XMI** - Opens a browser dialog to select the location to which to save the xmi file.
- **Save Graphics** - Opens a browser dialog to save the diagram as a graphic.

  The file types that may be used to save graphics are: ps, pdf, wmf, svg, png, eps, gif, and jpg.

- **Print Current Diagram...** - Opens the printer dialog.

  Quick Key - Ctrl-P

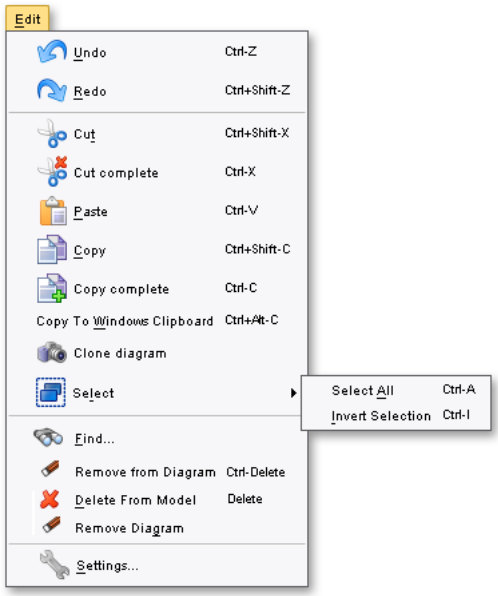
  > **Note:** Diagrams printed from the CE will automatically have a footer stating, "Created with Poseidon for UML Community Edition". CE versions prior to 3.x will have a watermark.

- **Print Diagrams...** - Opens a tree of the diagrams. Use Ctrl-click to select multiple diagrams to print at one time.

  > **Note:** Diagrams printed from the CE will automatically have a footer stating, "Created with Poseidon for UML Community Edition". CE versions prior to 3.x will have a watermark.

- **Print Preview...** - Opens the Print Preview dialog, allowing you to view the diagram before you print it.
- **Exit** - Exits Poseidon. If there are any edited projects, a save prompt will appear.

5.2.2. Edit



- **Undo** - Steps backwards in the edit history.

  Quick Key - Ctrl-Z

- **Redo** - Steps forwards in the edit history.

  Quick Key - Ctrl-Shift-Z

- **Cut** - Removes the selected element(s) to the Poseidon clipboard. The element will remain in the model.

  Quick Key - Ctrl-Shift-X

- **Cut Complete** - Removes the selected element(s) and the corresponding element representation from the diagram to the Poseidon clipboard.

  Quick Key - Ctrl-X

  Note: Complete cut is not available in the Community Edition.

- **Paste** - Places the element(s) currently in the Poseidon clipboard into the model.

  Quick Key - Ctrl-V

- **Copy** - Places the element(s) into the Poseidon clipboard without altering the element(s) selected.

  Quick Key - Ctrl-Shift-C

- **Copy Complete** - Places the element(s) and the corresponding element representation into the Poseidon clipboard without altering the element(s) selected.

  Quick Key - Ctrl-C

  Note: Complete copy is not available in the Community Edition.

- **Clone Diagram** - Duplicates the entire selected diagram, including element representations, without altering the selected diagram. Not available for sequence diagrams until version 3.2.

  Note: Diagram cloning is not available in the Community Edition.

- **Select** -
  - **Select All** - Selects every element in the current diagram.

    Quick Key - Ctrl-A

  - **Invert Selection** - Selects all unselected elements while de-selecting all selected elements.

    Quick Key - Ctrl-I

- **Copy to Windows Clipboard** - Places the currently displayed element(s) on the Windows clipboard.

  Quick Key - Ctrl-Alt-C

- **Find...** - Opens the Poseidon search dialog.

Quick Key - F3

- **Remove from Diagram** - Removes an instance of an element from a diagram without removing the element from the model.

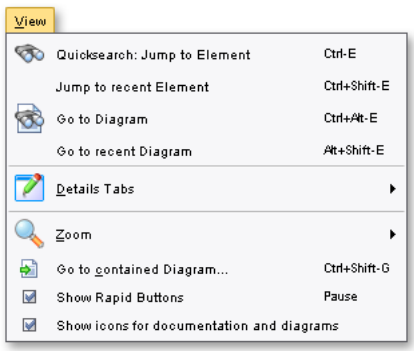
  Quick Key - Ctrl-Del

- **Delete from Model** - Removes the selected element(s) from the model completely. All instances of the element(s) will be removed from every diagram.

  Quick Key - Del

- **Remove Diagram** - Removes the selected diagram from the project without deleting the elements contained within that diagram.
- **Settings...** - Opens the Settings dialog, where you can change some of the default properties of Poseidon.

5.2.3. View



- **Quicksearch: Jump to Element** - Opens element navigation dialog.

  Quick Key - Ctrl-E

- **Jump to recent Element** - Opens element navigation dialog, with a dropdown list of the most recently accessed elements.

  Quick Key - Ctrl-Shift-E

- **Go to Diagram** - Opens the diagram navigation dialog.

  Quick Key - Ctrl-G

- **Go to recent Diagram** - Opens the diagram navigation dialog, with a dropdown list of themost recently accessed diagrams.

  Quick Key - Alt-Shift-E

- **Details Tabs** -
  - **Next Details Tab** - Changes the active tab to be the next tab to the right.
  - **Individual Tabs** - List of individual tabs for navigation.

- **Zoom** -
  - **Zoom Factors** - List of zoom factors to change the diagram display.

- **Go To Contained Diagram...** - Navigates to the sub-diagram of the current element. If the element has more than one sub-diagram, the diagram navigation dialog will open, filtered on the current element.

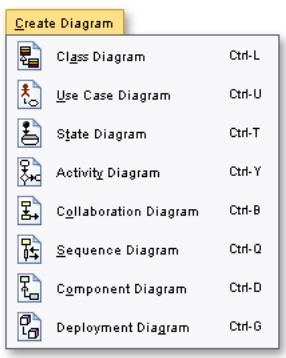
  Quick Key - Ctrl-Shift-G

- **Show Rapid Buttons** - Toggles rapid button display.

  Quick Key - Pause

- **Show icons for documentation and diagrams** - Toggles icon display.

5.2.4. Create Diagram



- **Class Diagram** - Creates a new blank diagram.

  Quick Key - Ctrl-L

- **Use Case Diagram** - Creates a new blank diagram.

  Quick Key - Ctrl-U

- **State Diagram** - Creates a new blank diagram.

  Quick Key - Ctrl-T

- **Activity Diagram** - Creates a new blank diagram.

  Quick Key - Ctrl-Y

- **Collaboration Diagram** - Creates a new blank diagram.

  Quick Key - Ctrl-B

- **Sequence Diagram** - Creates a new blank diagram.

  Quick Key - Ctrl-Q

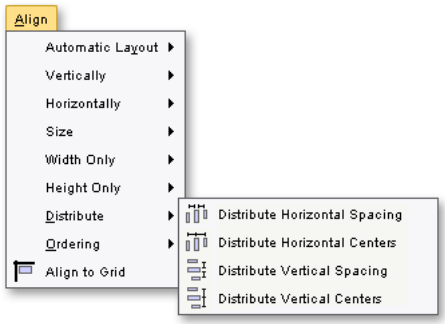
- **Component Diagram** - Creates a new blank diagram.

  Quick Key - Ctrl-D

- **Deployment** - Creates a new blank diagram.

  Quick Key - Ctrl-G

5.2.5. Align



- **Automatic Layout** -
  - **Do Layout** - Creates a new, optimal layout.
  - **Update Layout** - Adjusts the current layout.
  - **Settings** - Opens the layout settings dialog.

- **Vertically** -
  - **Align Tops** - Moves selected elements so that top edges are aligned. Horizontal placement is not affected.
  - **Bottoms** - Moves selected elements so that bottom edges are aligned. Horizontal placement is not affected.
  - **Center** - Moves selected elements so that center points are aligned. Horizontal placement is not affected.

- **Horizontally** -
  - **Align Lefts** - Moves selected elements so that left edges are aligned. Vertical placement is not affected.
  - **Align Rights** - Moves selected elements so that right edges are aligned. Vertical placement is not affected.
  - **Center** - Moves selected elements so that center points are aligned. Vertical placement is not affected.

- **Size** -
  - **Greatest Current Width and Height** - Uniformly resizes selected elements so that each is the size of the largest selected element.
  - **Smallest Current Width and Height** - Determines the size of the smallest element that can display the information of each of the selected elements and uniformly resizes the selected elements.

  - **Minimum Possible Width and Height** - Determines the smallest possible size for each of the selected elements and uniformly resizes them so that each is the size of the largest minimized element.

- **Width Only** -
  - **Greatest Current Width** - Uniformly resizes selected elements so that each is the width of the largest selected element.
  - **Smallest Current Width** - Determines the width of the smallest element that can display the information of each of the selected elements and uniformly resizes the selected elements.
  - **Minimum Possible Width** - Determines the smallest possible width for each of the selected elements and uniformly resizes them so that each is the width of the largest minimized element.

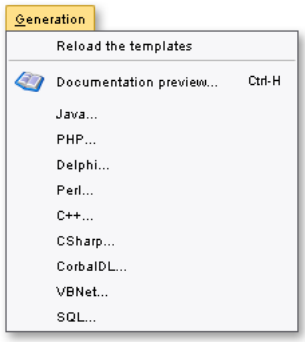
- **Height Only** -
  - **Greatest Current Height** - Uniformly resizes selected elements so that each is the height of the largest selected element.
  - **Smallest Current Height** - Determines the height of the smallest element that can display the information of each of the selected elements and uniformly resizes the selected elements.Note: this is not functional in the current version of Poseidon.
  - **Minimum Possible Height** - Determines the smallest possible height for each of the selected elements and uniformly resizes them so that each is the height of the largest minimized element.

- **Distribute** -
  - **Distribute Horizontal Spacing** - Places a uniform vertical gutter between elements. Vertical spacing and element size are not changed.
  - **Distribute Horizontal Centers** - Places a uniform amount of space between the center vertical axes of the selected elements. Vertical spacing and element size are not changed.
  - **Distribute Vertical Spacing** - Places a uniform horizontal gutter between elements. Horizontal spacing and element size are not changed.
  - **Distribute Vertical Centers** - Places a uniform amount of space between the center horizontal axes of the selected elements. Horizontal spacing and element size are not changed.

- **Ordering** -
  - **Bring To Back** - Places the selected element(s) on the bottom layer of the diagram display.
  - **Bring To Front** - Places the selected element(s) on top of the diagram display.
  - **Send Backward** - Moves the selected element(s) down one layer in the diagram display.
  - **Send Forward** - Moves the selected element(s) up one layer in the diagram display.

- **Align to Grid** - Snaps the top and left edges of an element to the grid. Element size is not changed.

5.2.6. Generation



- **Reload the Templates** - Refreshes the templates if they have been altered.

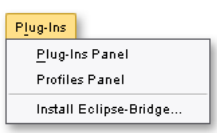
  Note: Available in the Professional and Enterprise Editions only

- **Documentation Preview...** - Opens the HTML documentation preview window.

  Quick Key - Ctrl-H

- **Language Listing** - Opens the generation dialog of the selected language.
  - Community Edition: Java
  - Standard Edition: HTML Documentation (includes UMLdoc), Java
  - Embedded Edition and Embedded Enterprise Editions: HTML Documentation (includes UMLdoc), Java, Embedded
  - Professional and Enterprise Editions: HTML Documentation (includes UMLdoc), Java, PHP, Delphi, Perl, C++, CSharp, CorbaIDL, VB.Net, SQL

5.2.7. Plug-Ins



- **Plug-Ins Panel** - Opens the Plug-Ins Manager, where you can add, remove, enable, and disable the plug-ins.

  Note: Plug-Ins are not available in the Community Edition

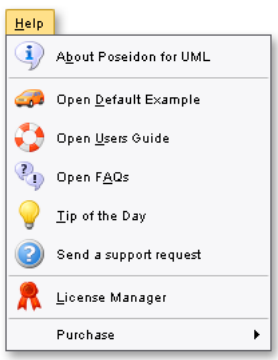
- **Profiles Panel** - Opens the Profile Manager, where you can enable and disable profiles.

  Note: Plug-Ins are not available in the Community Edition

- **Install Eclipse-Bridge** - Installs the necessary files in order to be able to use Poseidon from within the Eclipse IDE. See Section 17.1 for more details about this functionality.

  Note: Plug-Ins are not available in the Community Edition

5.2.8. Help



- **About Poseidon for UML** - Displays Poseidon version information and credits.
- **Open Default Example** - Opens the Webstore project.
- **Open Users Guide** - Opens the local html version of the Poseidon User Guide.
- **Open FAQs** - Opens the local html version of the FAQ.
- **Tip of the Day** - Opens the Tip of the Day dialog.

- **Send a Support Request** - Opens the Support Request web page at the Gentleware web site. Poseidon will launch a web browser, if necessary.

  Note: Support Requests are not available in the Community Edition

- **License Manager** - Opens the License Manager, where you can add and remove keys, serial numbers, and register your copy of Poseidon.

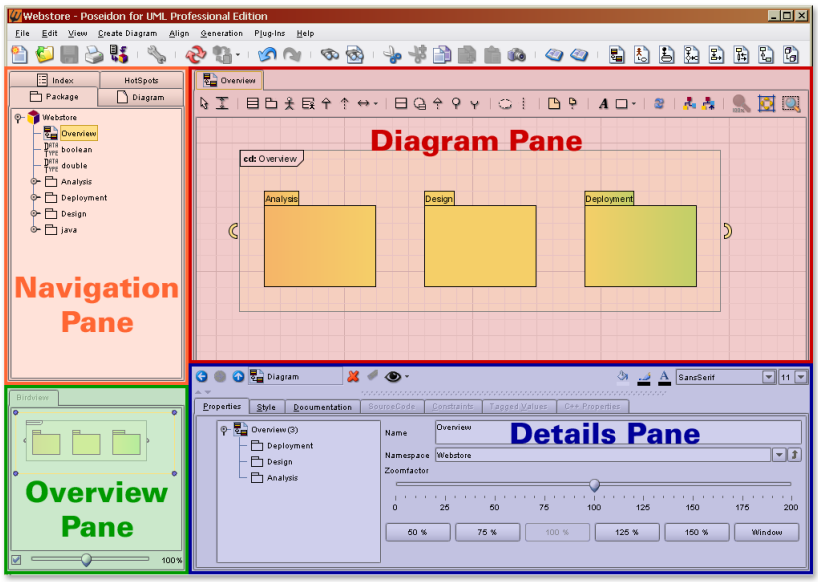
  Note: The License Manager is not available in the Community Edition

- **Purchase** - Provides links to the Gentleware Web Store. This is a separate menu in the Community Edition.
  - **Upgrade Poseidon**
  - **Upgrade Subscription**

Chapter 6. Panes

The panes in Poseidon for UML divide the application workspace into 4 sections, each with a specific purpose. This design eases the process of modeling by providing quick access to all parts of the project. The panes can be resized and hidden as you work with your models.

Figure 6-1. Panes in Poseidon



6.1. Navigation Pane

As a model grows, its complexity likewise increases. It becomes more and more necessary to have different organizations of the model to facilitate easy navigation. This is what the Navigation pane has been designed to do; present the elements of a model in different arrangements based on pre-determined criteria. Poseidon calls these arrangements 'views'.

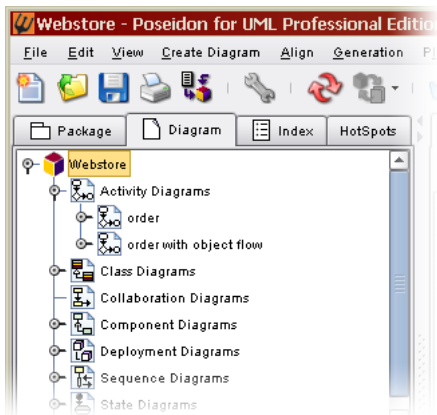
Each view positions the elements within the model hierarchy differently. Views are not required to display all of the elements of a model, only those which pertain to their organization schema. The one similarity between all of the views is the root node, which is always the model itself. In the case of the default example, this would be 'Webstore'.

The views offered by Poseidon are as follows:

- **Diagram Centric**
- **Model Index**
- **Package Centric**
- **HotSpots**

6.1.1. Navigation Tabs

The four views are always available by selecting a tab from the Navigation Pane..



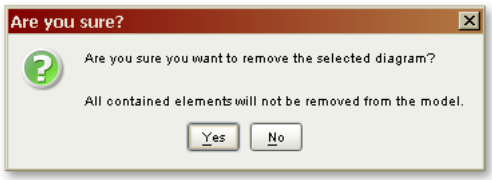
6.1.2. Delete a Diagram

Deleting a diagram in Poseidon removes the diagram itself completely from the model, but leaves the the elements contained within that model intact.

There are two ways to delete a diagram, through the Edit menu and through the context menu.

To delete a diagram using the Edit menu:

1. Select a diagram in the Navigation pane.
2. Select 'Remove Diagram' from the Edit menu. A dialog will appear to prevent unintended deletion of the diagram.



To delete a diagram through the context menu:

1. Select the diagram in the Navigation pane.
2. Right-click on the diagram name and select 'Remove Diagram'. The same dialog box mentioned above will appear.

6.2. Diagram Pane

The Diagram Pane is the area used to do most of the diagram creation and modification. It is generally the largest pane.

This section covers some of the functions available from the diagram pane, as well as changing the settings of this pane. Chapter 7, 'Working with Diagrams', provides a more extensive look at all of the functions available. Chapter 9, titled 'A Walk Through the Diagrams', contains detailed information about the diagrams themselves.

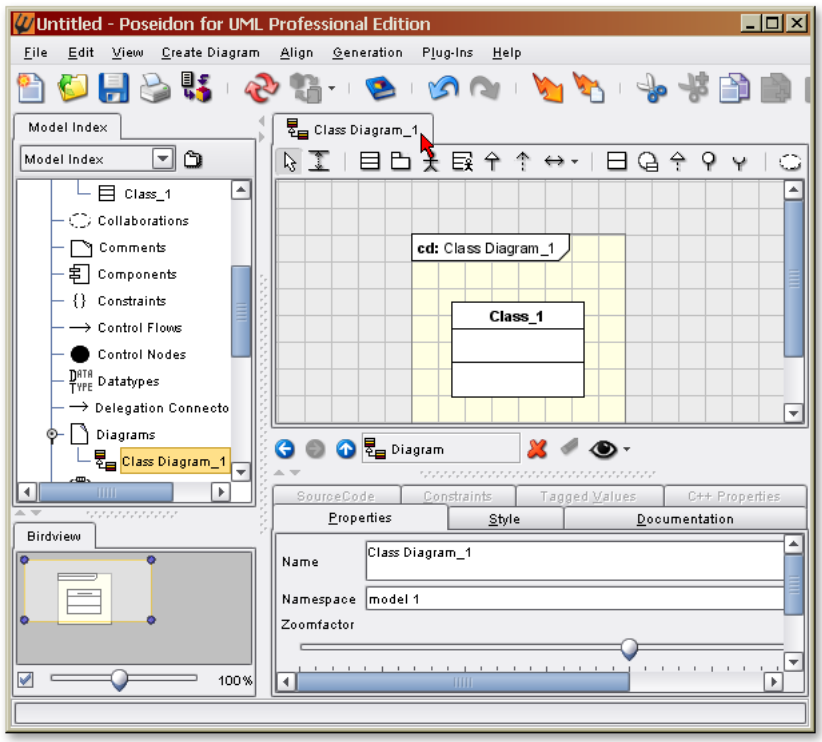
The graphical editor is embedded in the Diagram pane. This pane, as has been previously mentioned, is used to display and edit the diagrams of your model

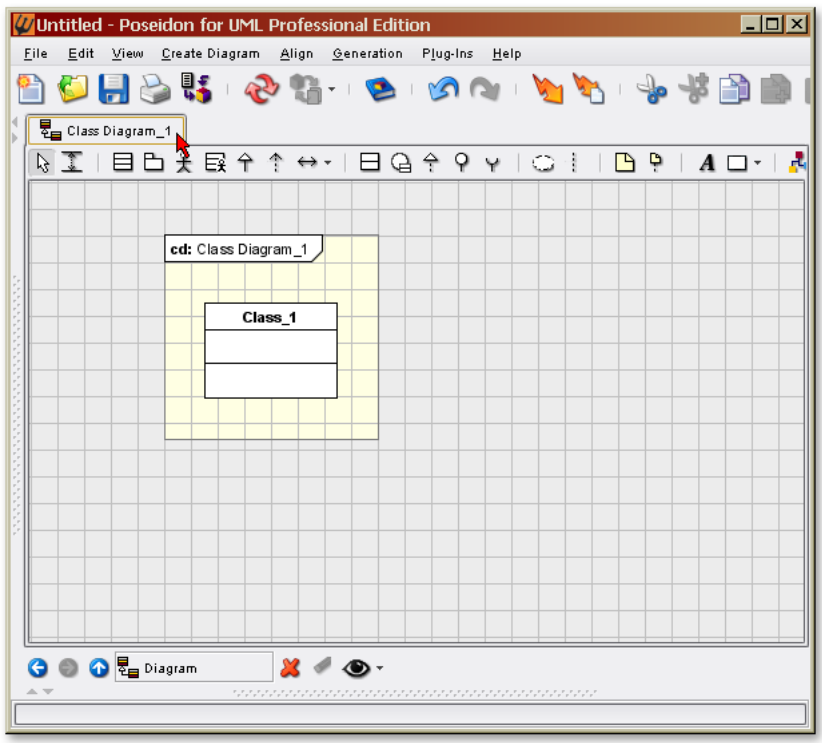
6.2.1. Diagram Display Modes

The standard mode in which to view the Diagram pane also displays the three other panes. This mode is quite useful for normal diagram editing and has many advantages, including drag and drop from the Navigation pane, zooming from the Overview pane, and in-depth editing from the Details pane.

However, sometimes it is much easier to work in full-screen mode, especially when working with layout. Beginning with version 4.2, diagrams can be viewed in full-screen mode by double-clicking on the diagram's tab in the Diagram pane. This mode temporarily hides the other three panes to maximize the space available to the diagram. Notice that the toolbar from the Details pane remains visible in full-screen mode to assist with diagram editing. Standard mode is restored by double-clicking the tab again.

Figure 6-2. Diagram pane in Standard and Full-Screen modes.





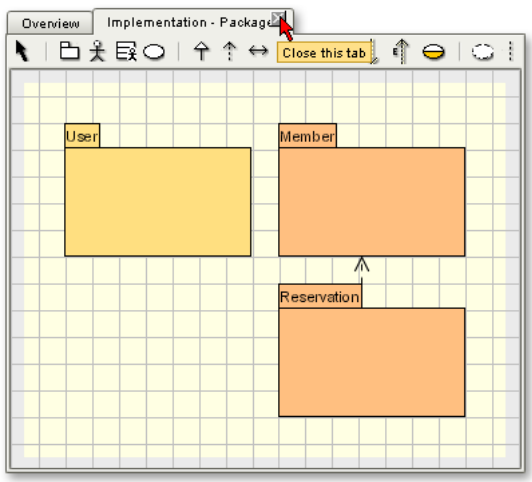
6.2.2. Diagram Pane Toolbar

Across the top of the Diagram pane, there is a toolbar that contains a number of tools you can use to create and modify your UML models. If you have already worked with a UML tool or a drawing tool capable of creating UML diagrams, you are probably familiar with the general idea. Each diagram type has a specialized set of tools in addition to the tools that are common to all diagram types. To display the name of each individual tool, position your mouse over it and wait a second or so, the name will appear in a box underneath.

In general, the Diagram pane toolbar changes according to the type of diagram currently displayed. There are, however, some tools which are available in all or nearly all of the diagrams:

6.2.3. Remove Tabs

To remove the a tab from the Diagram pane, move the mouse over the tab to be deleted. A 'delete' button with an 'X' will appear. Click the button and the tab will be removed from the pane. From Poseidon 3.0 on, all diagram tabs may be closed in this pane. Previous versions require at least one diagram to be open.



6.2.4. Change Properties of the Diagram Pane

6.2.4.1. Grid Settings

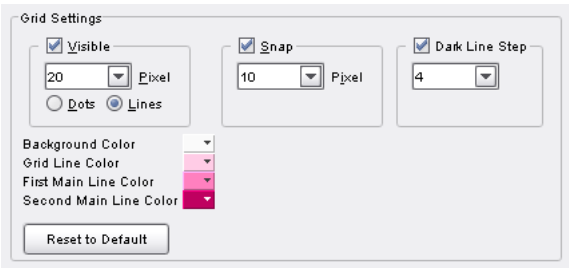
The first thing you may notice about the Diagram Pane is the grid that is drawn over the drawing area. By default, the drawing area displays this grid. The visible grid is only a collection of lines, they have no functions of their own.

A second grid, called the snap grid, is invisible to the user. When this option is enabled, diagram elements align themselves along the intersections of this grid which are closest to the element (in a process called snapping) to aid with element positioning.

To make elements snap to the visible grid, set the visible and snap grids to be the same size. The settings shown in Figure 10-1 will have the visible grid drawn every 20 pixels, and the elements will be able to snap to intermediate positions of the visible grid.

You can change the properties of both grids from the Grid Settings dialog in Edit -> Settings -> Appearance .

Figure 6-3. Grid Settings dialog



Grid Settings

- **Visible** - Determines whether the visible grid is drawn at all.

  Spacing and line appearance are also set for the visible grid here.

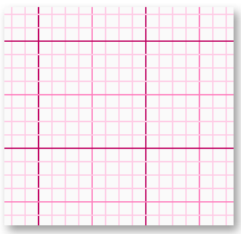
- **Snap** - Determines whether the elements placed in the diagram will be forced to align to a snap grid.

  The pixel dropdown sets the spacing of the snap grid.

- **Dark Line Step** - Indicates the spacing of darker grid lines.

  In the example above, the Dark Line Step is set to ’4’, which means that every fourth line will be a darker line.

- **Color Selectors** - Determines the colors of the grid lines.



  - Background Color - color of the page behind the diagram.
  - Grid Line Color - color of the normal grid lines
  - First Main Line Color - color of a darker line. This color alternates with the Second Main Line Color every other darker line - in the example above with Dark Line Step set to ’4’, this color appears every eighth line, and four lines after the Second Main Line.
  - Second Main Line Color - color of a darker line. This color alternates with the First Main Line Color every other darker line - in the example above with Dark Line Step set to ’4’, this color appears every eighth line, and four lines after the First Main Line.

- **Snap** - Resets the grid settings to the Gentleware defaults.

6.2.4.2. Other Settings

The grid is not the only setting that can be changed for the Diagram pane.

- **Display/Hide Tabs** - Hide or redisplay diagram tabs at the top of the pane with the Appearance Tab in the Settings Dialog.
- **Number of Tabs Displayed** - Set the maximum number of tabs with the Settings Dialog, Appearance Tab.
- **Display/Hide Information About Elements** - Hide or redisplay information such as operations or attributes from the Settings Dialog, Diagram Display Tab.
- **Resize the Drawing Area** - Drag the pane separation bars to the desired size. The arrows on the bars open and close the panes completely.
- **Enlarge/Reduce the Diagram** - Change the zoom factor in the Properties tab for the diagram or hold the Ctrl key while turning the mouse wheel.

6.3. Details Pane

The Details pane provides access to all of the aspect of the model elements. Within this pane, you can view and modify properties of the elements, define additional properties, and navigate between elements.

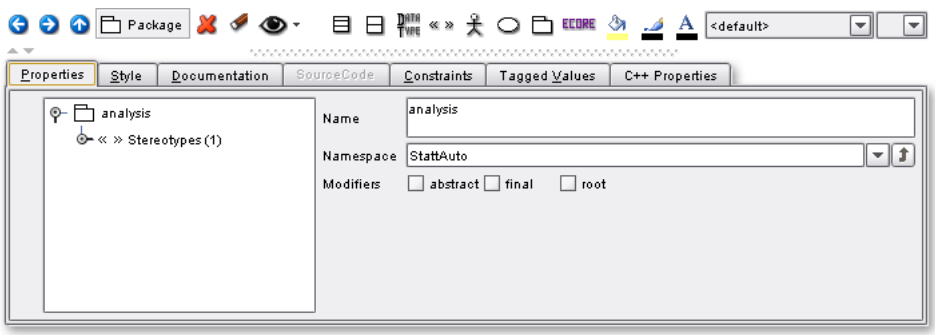
The pane is composed of eight tabs:

- Properties
- C++ Properties *(not available in CE)*
- Style
- Source Code
- Documentation
- Constraints
- Tagged Values

The following sections investigate these tabs in greater detail.

6.3.1. Selection Bar

Versions 2.4 and later now contain a selection and editing toolbar over the Details pane. This toolbar provides constant access to functions previously available only in the Properties tab. This toolbar remains visible even when the Diagram pane is viewed in full-screen mode. For more information on the viewing modes, see Section 6.2.1 .



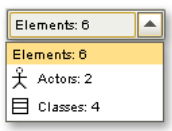
Navigation

The first set of buttons are the navigation buttons. The left and right buttons move you through the history of selected elements in a style very similar to a web browser. For example, if you select Package_1, then Class_1, then Class_2 and click the left arrow, you will arrive at Class_1. The up arrow moves you logically up to the next level of the model. Say Package_1 contains Class_2. Regardless of

whether Package_1 was previously selected or not, clicking the up arrow while Class_2 is selected will move you to Package_1.

Elements

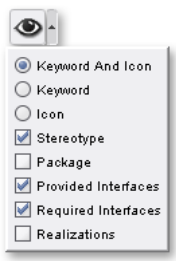
The next item displays the type of element currently selected. In the example above, this happens to be a package. When multiple elements are selected, this will display the generic message 'Elements' followed by the number of elements selected. At the same time, this becomes a dropdown list populated by the individual element types. You can select one of these element types from the list and your current selection will be refined to display only previously selected elements of the chosen type. For example, assume a diagram has 10 classes and lots of other elements. Say you select 4 classes and 2 actors. If you select 'Classes:4' from the dropdown, only those 4 classes remain selected. The other 6 unselected classes are irrelevant, and the 2 actors are de-selected.



The next two buttons delete items from the diagram, but each in a slightly different way. The delete icon is the red 'X', which completely removes the element from the project. All occurrences of this element are deleted, whether they appear in the current diagram or not. The second button is the eraser, which removes the element from the current diagram only. The element remains available in other diagrams and in the Navigation pane.

The visibility selector is comprised of two parts. The first is the visibility button itself, which hides and displays the frame that appears around the diagram, including the diagram name. The second is the dropdown that is accessible from the arrow next to the button. This dropdown allows you to select exactly which parts of the element will be displayed in the diagram. The options available from this dropdown will vary according to the element currently selected.

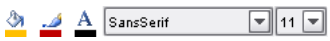
Figure 6-4. Visibility options for a Component



The final set of buttons allow you to add elements to the current attribute. The elements available depend on the element selected. For classes, your options include adding attributes, operations, inner classes, and inner interfaces. Associations and the like have a selector to determine how the edges should be displayed.

Style

New to Poseidon 5.x are the style options in the toolbar. The colors and font properties can be easily changed from the dropdowns, and these settings will 'stick' as the default settings for any new elements of the same type that are created. To change the default, create a new element and apply the desired properties. These settings will now remain as the default.

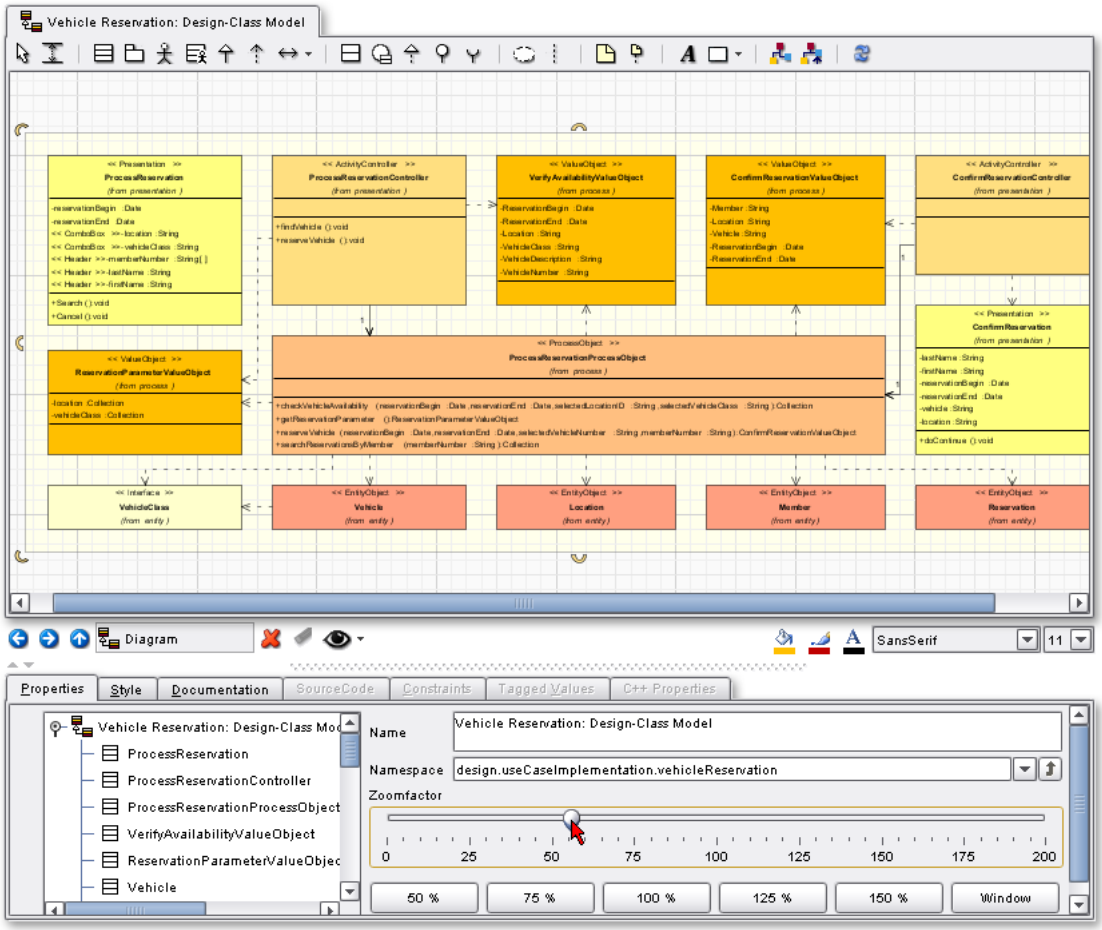


6.3.2. Properties Tab

The most important tab is the **Properties** tab, which is selected by default. The Properties tab looks a little different for each different type of model element. So far in this tour we have selected packages, diagrams and classes. All of these elements have only one common property, the property 'name'. It makes sense that this would be the only field in this tab which is duplicated for all of the elements.

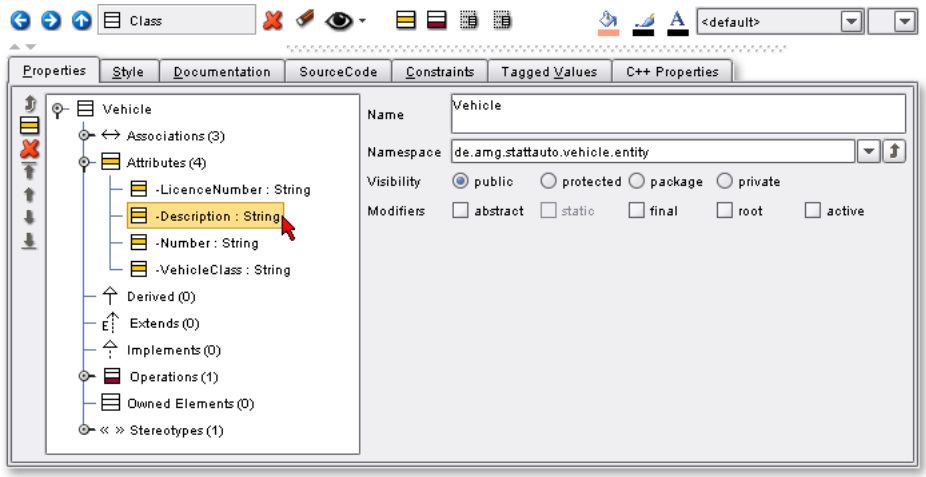
An important property, the zoom factor, becomes visible in the Details pane when a diagram name is selected in the Navigation pane. You can use the slider to change the zoom factor interactively or use the buttons to set it to pre-selected zoom factors (The range of zoom factors is limited in the Community Edition). To access this property, select a diagram in the Navigation pane or click on empty space in the Diagram pane.

Figure 6-5. Properties tab with zoom



But the real power and importance of the Properties tab becomes apparent for complex model elements like classes or methods. For these, the Properties tab becomes an important tool to view and change the model details. As a general rule, properties that can be changed are placed to the right. On the left, related model elements are displayed in a tree. By double-clicking on the related model elements, you can navigate to them and change their properties. This way, you can drill down from a package to a class to a method to its parameters and so forth.

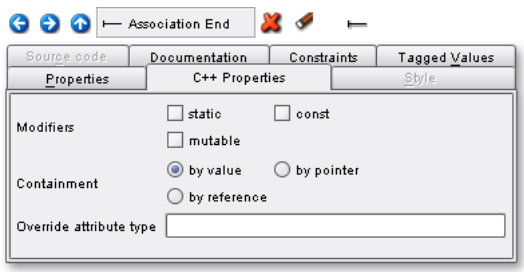
Figure 6-6. Drill-down navigation



6.3.3. C++ Properties

The C++ functionality has been ported from the Embedded Edition. The available properties are dependent upon the element currently selected.

Figure 6-7. C++ tab for an association end

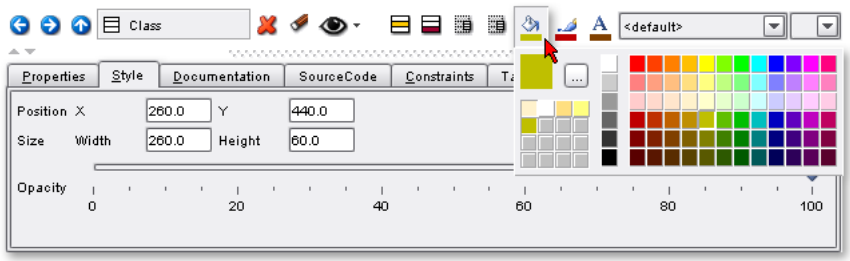


Note: Not available in the Community or Standard Editions

6.3.4. Style Tab

- Offers possibilities for defining colors and certain other display characteristics of selected elements
- Style can be changed for a single element, or for several selected elements at a time

Figure 6-8. Style tab for a class



The Style tab allows you to control the display of elements within a particular diagram. If, for example, you wanted all your classes in a diagram to have fill color green, you can select all the elements (using the mouse, or by pressing Ctrl-A) and then use the color chooser in the toolbar to change their color to green. You can also change the line color, text color, font, and font size. The display properties of an element exist only within that diagram, that is, if you give a font color to a class in one diagram, that font color will not be automatically applied to the other diagrams in which that element appears.

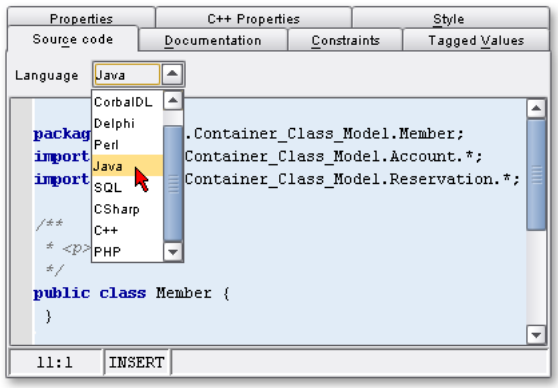
The display of compartments is also configurable from the Style tab. In general, the visibility of compartments (for those elements that have compartments, such as classes) is determined at the diagram level. But this can be altered for individual elements without affecting the rest of the elements in that diagram. Select the option "Specify individual compartment display" and enable or disable the appropriate checkboxes.

The color selectors have been moved to the toolbar in Poseidon 5.x and are available regardless of the tab selected. The currently selected colors for an element type will be applied to newly created elements.

6.3.5. Source Code Tab

- Available for different elements, based upon the target language selected
- Shows the code generated by Poseidon

Figure 6-9. Source code tab for a class



At the start this code just represents the skeleton that has to be filled with content. For example, method names and the corresponding parameters may already present and defined, but the method body might still be empty. With most of the target languages, you can use this editor to fill in the body. With round-trip engineering you can also use any other external editor or IDE. Note also that documentation entered in the Documentation tab is included in the generated code.

The editor in Poseidon will not allow you to change all of the code. The sections of code which are highlighted in blue are 'read-only' in the Poseidon editor. Text highlighted in white may be edited, deleted, and appended. This functionality originates from a NetBeans project and is the result of a plug-in.

New in version 2.1 is the ability to select the target language of the source code. The list of available languages is dependent upon the list of enabled plug-ins and profiles. Each language must have both the plug-in and profile specific to that language enabled.

The same diagrams may be used to generate code in different languages. Any code written in the 'your code here' sections is available only in the language selection in which it was written. For example, any code manually entered into the editable section of this tab while Java is the selected language will not be seen if the language is changed to C# or Perl.

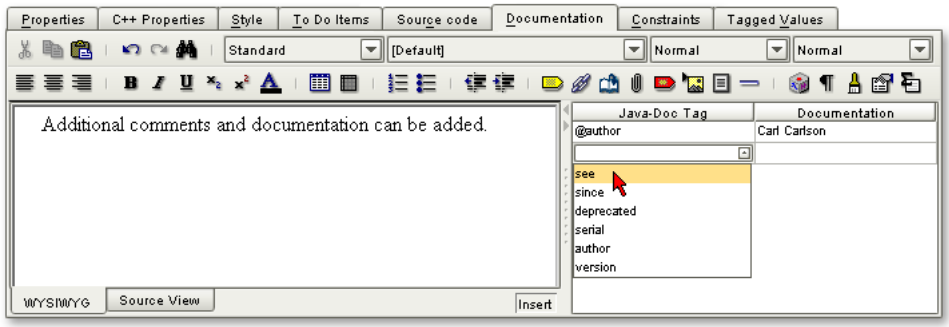
Should there be ambiguity, a second dropdown will appear next to the language selection dropdown in order to determine the correct option for the implementation.

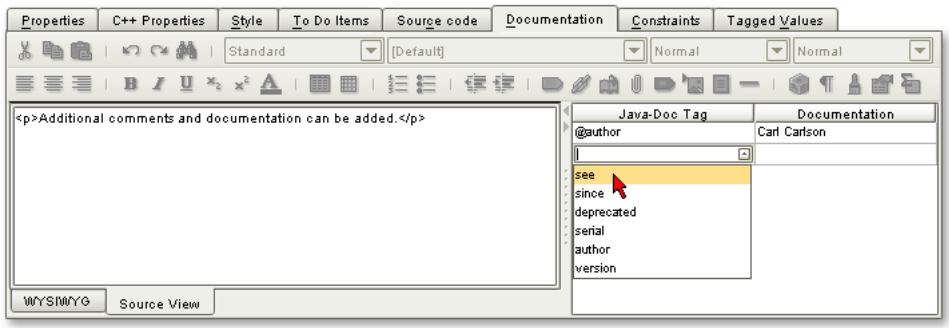
6.3.6. Documentation Tab

- Contains a WYSIWYG editor, where you can easily add your own documentation for model elements

- You can also use Javadoc tags, like `@author` , `@see` and others

Figure 6-10. Documentation tab for a class - WYSIWYG and source



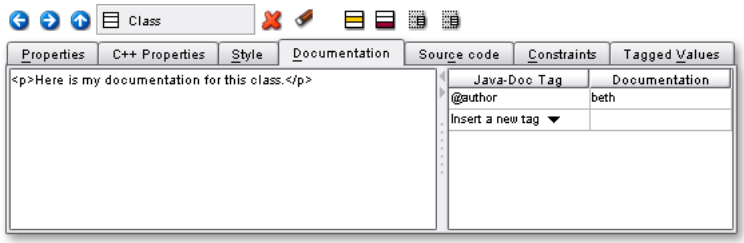


The Documentation tab provides a mechanism for adding your own freeform text as well as supported JavaDoc tags to the generated code.

There are a couple of choices for entering text - you can use the WYSIWYG editor and format the text using the documentation toolbar, or you can edit in plain text mode and add html formatting tags by hand. Only HTML tags that are legal within <body> tags can be entered into the source code view. Both methods result in html-formatted text; for instance, pressing 'Enter' while in the WYSIWYG tab generates a new paragraph with the   tags, and pressing 'Ctrl-Enter' generates the
 tag.

This information is stored as tagged values and can be previewed in the Java Source tab. Any entries made via the editor on the left side of the tab are placed in paragraph tags by default and are displayed before the JavaDoc entries.

While the WYSIWYG editor is a handy and convenient tool, it also slows the performance of Poseidon. The Settings | Appearance tab includes an option to turn off this editor, which removes the toolbar and editor selection tabs. The source editor remains in place.



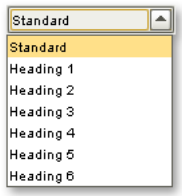
6.3.6.1. Toolbar

- Cut
- Copy
- Paste
- Undo
- Redo
- Search and Replace
- Align Left
- Align Center
- Align Right
- Bold
- Italic
- Underline
- Subscript
- Superscript
- Font Color
- Insert Default Table
- Insert Table
- Numbered List
- Bulleted List
- Decrease Indent
- Increase Indent
- Link to Model Element
- Insert Hyperlink
- Insert Bookmark
- Insert HTML
- Insert Span
- Insert Image
- Insert Text
- Insert Horizontal Line
- Insert Symbol
- Show All Formatting Characters

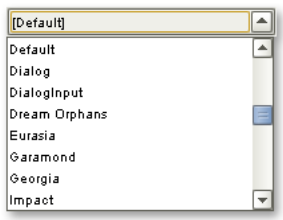
- Remove Formatting
- Open Style Properties Dialog
- Display Document Statistics

6.3.6.2. Dropdowns

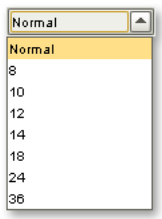
Paragraph Style



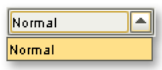
Font Type



Font Size



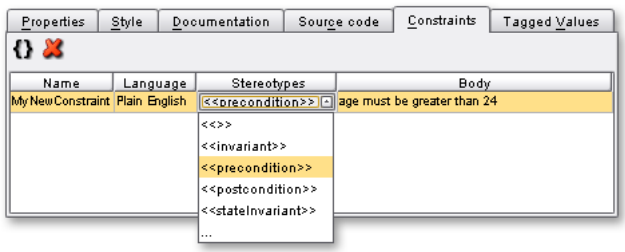
Style Sheet



6.3.7. Constraints Tab

- Holds the constraints for the given element

Figure 6-11. New constraint in the Constraints tab

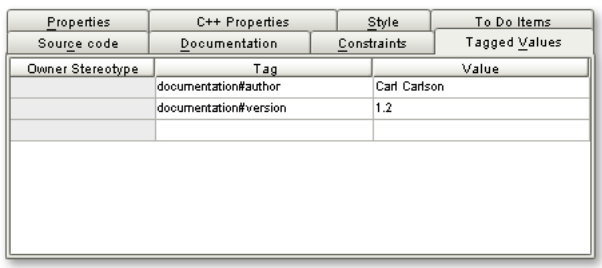


The UML does not include specifications regarding constraint language. Poseidon is now able to hold constraint information that is language independent, including plain English. Of course, you are still free to use OCL if you so choose.

6.3.8. Tagged Values Tab

- Edit different pairs of names and values that you might want to use in order to enhance your model with specific characteristics.
- This is a general mechanism of UML that can be extended for special purposes

Figure 6-12. Documentation stored in the Tagged Values tab



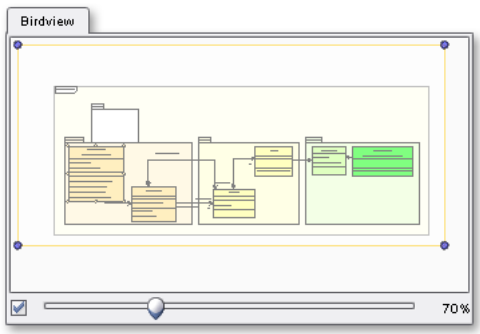
If, for example, you need special information for external processing of the model you can add this information here. This is also where Poseidon stores any documentation entered in the Documentation tab.

Tags may also originate from a stereotype. The tag is specified in the Properties tab of the stereotype. Whenever the stereotype is applied to an element, the tag and owner stereotype are populated in the Tagged Values tab.

6.4. Overview Pane

Especially when working with large models, the Overview pane is quite helpful for keeping track of the big picture of the model.

6.4.1. Birdview Tab



Screen space is limited, and it is often impractical to view an entire diagram at one time. Scrolling around or zooming in and out repeatedly is time-consuming, inefficient, and generally annoying. The Birdview tab resolves these issues by maintaining a snapshot of the entire diagram that can be quickly referenced while working on a diagram.

6.4.1.1. Redisplay a Section of a Diagram

The portion of the diagram visible in the Diagram pane is highlighted in the Birdview tab. You can change the display by dragging the highlighted portion within the birdview.

6.4.1.2. Zoom in Birdview Only

Perhaps you would like to keep track of a smaller section of the diagram, and then later decide to view the entire diagram again. This is easily done by adjusting the zoom factor in the Birdview tab.

To change the zoom factor of the Birdview tab:

1. Uncheck the box in the lower left corner of the pane.
2. Use the slider bar to change the zoom factor.
3. The scroll bars can be used to position the view as required.

6.4.1.3. Zoom in a Diagram

The Birdview tab provides the means to resize the diagram in the Diagram pane as well. The view in the Birdview tab remains unchanged while the diagram itself is enlarged or reduced.

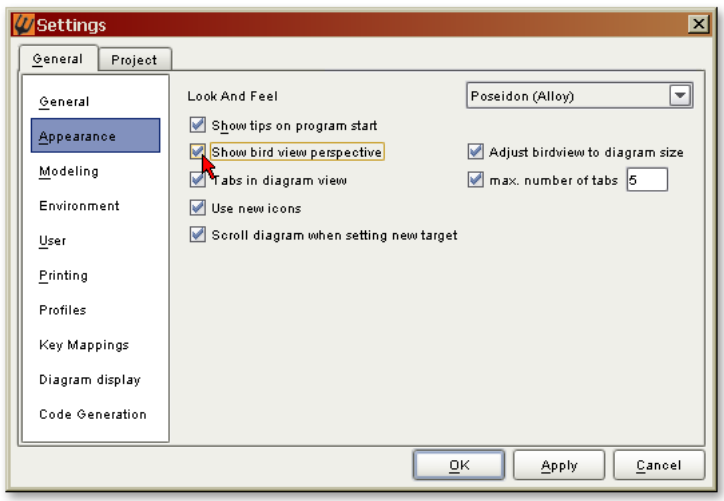
To change the zoom factor of the diagram in the Diagram pane:

1. Check the box in the lower left corner of the pane.
2. Use the slider bar to change the zoom factor.

Note that the zoom factor is set for each diagram individually. A zoom factor set for one diagram will not be applied to subsequently displayed diagrams.

6.4.1.4. Turn Off Birdview in Settings

The Birdview, while helpful, can slow down the performance of Poseidon. At times, it may be useful to turn off the Birdview option. This can be set in the Appearance Tab of the Settings dialog.



Chapter 7. Setting Properties

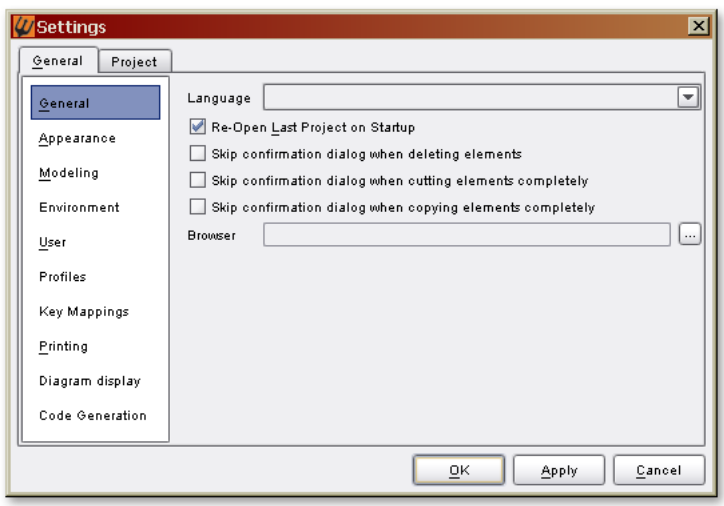
The behavior of Poseidon is defined by a number of properties. You can adjust the behavior of Poseidon to your personal needs by changing the corresponding properties using the settings dialog. Once the settings dialog is open (choose Settings from the Edit menu), you will see a number of tabs.

7.1. General Tab

The properties set in this tab are stored in the Poseidon.properties file, located in your user home directory under .poseidon/<version>. For Windows users, this is most likely ' *C:\Documents and Settings\<username>\poseidon\<version>* '.

7.1.1. General

Figure 7-1. The General settings tab.



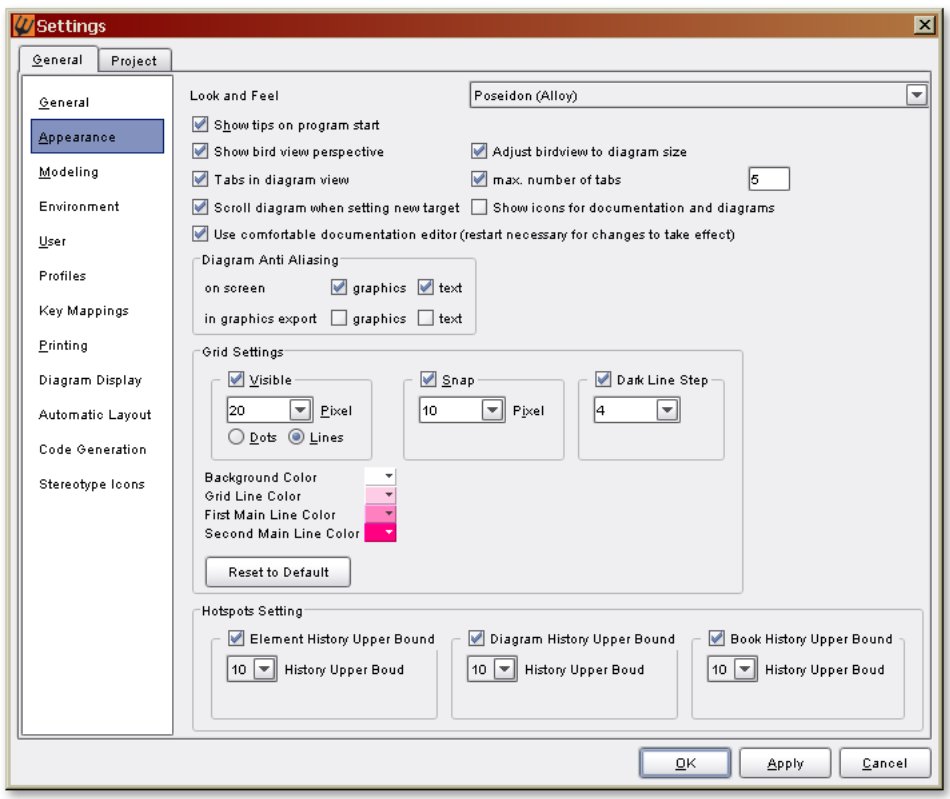
- **Language** - This is the language used for the Poseidon user interface. You can switch the interface to a different language by choosing your preferred language from this selection list. Poseidon currently supports English, German, French, Spanish, Italian and Chinese. By default, the system language is used - or English, if the system language is not available. In other words, if you start Poseidon on a

Spanish system, for example, the program will start in Spanish - but on a Swedish system the program will start in English, as Swedish is not currently supported.

- **Re-Open Last Project on Startup** - If checked, Poseidon opens the most recently used project on startup.
- **Skip confirmation dialog when deleting elements** - If checked, the dialog asking if you really want to delete selected elements will not appear.
- **Skip confirmation dialog when cutting elements completely** - If checked, the dialog asking if you really want to cut the element selected (not just its representation) will not appear.
- **Skip confirmation dialog when copying elements completely** - If checked, the dialog asking if you really want to copy the element selected (not just its representation) will not appear.
- **Browser** - This sets the default web browser used by Poseidon. To change the browser, click on the ellipse button and navigate to the location of your favorite web browser. This does not currently work on Macs.

7.1.2. Appearance

Figure 7-2. The Appearance settings tab.



- **Look and Feel** - Determines the look and feel of the Poseidon user interface. You can change the interface appearance by choosing an entry from this list. The list of options is determined by the operating system under which Poseidon is running.

  It may sometimes be necessary to change the Look and Feel from outside of the Settings dialog (as with MacOS X and Java combinations). To change this, use the following procedure:

  1. Start Poseidon, then shut it down again.
  2. Open the Poseidon.properties file in a text editor. In Windows, this is found in: `C:\Documents and Settings\<User>\.poseidon\<Poseidon Version>`
  3. Change the line

     ```
     poseidon.init.laf=...
     ```

     to

     ```
     poseidon.init.laf=com.incors.plaf.alloy.AlloyLookAndFeel
     ```
  4. Save this file and restart Poseidon.

- **Show tips on program start** - If checked, a `Tip of the Day` dialog appears when you start up the program.
- **Show birdview perspective** - If checked, the Overview pane shows the bird's-eye perspective. The speed of Poseidon increases when the bird's-eye perspective is disabled.
- **Adjust birdview to diagram size** - If checked, the bird's-eye view is scaled to show all elements of the currently selected diagram.
- **Tabs in diagram view** - If checked, the diagram view shows the most recently viewed diagrams as tabs over the Diagram pane. This allows for faster navigation between the diagrams.
- **Max. number of tabs** - If checked, the maximum number of tabs in the Diagram pane is limited to the specified value.
- **Scroll diagram when setting new target** - When selecting a different element within the same diagram, the diagram will pan smoothly if this option is enabled. Disabling this option can enhance the speed of Poseidon.
- **Use comfortable documentation editor** - Enables the WYSIWYG editor in the Documentation tab of the Details pane. Leaving this unchecked speeds up the performance of Poseidon.
- **Diagram Anti Aliasing** - Enables the anti-aliasing of the selected diagram components.
- **Grid Settings** -
  - **Visible** - Determines whether the visible grid is drawn at all.

Spacing and line appearance are also set for the visible grid here.

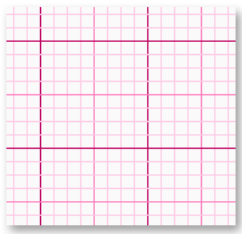
- **Snap** - Determines whether the elements placed in the diagram will be forced to align to a snap grid.

  The pixel dropdown sets the spacing of the snap grid.

- **Dark Line Step** - Indicates the spacing of darker grid lines.

  In the example above, the Dark Line Step is set to ’4’, which means that every fourth line will be a darker line.

- **Color Selectors** - Determines the colors of the grid lines.

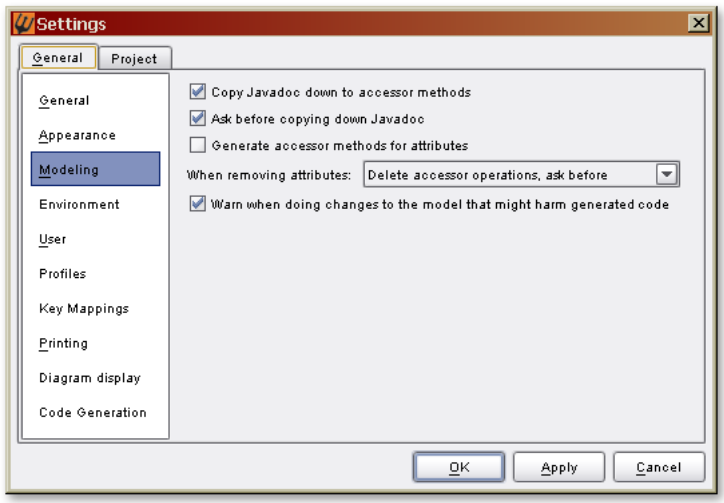


  - Background Color - color of the page behind the diagram.
  - Grid Line Color - color of the normal grid lines
  - First Main Line Color - color of a darker line. This color alternates with the Second Main Line Color every other darker line - in the example above with Dark Line Step set to ’4’, this color appears every eighth line, and four lines after the Second Main Line.
  - Second Main Line Color - color of a darker line. This color alternates with the First Main Line Color every other darker line - in the example above with Dark Line Step set to ’4’, this color appears every eighth line, and four lines after the First Main Line.

- **Snap** - Resets the grid settings to the Gentleware defaults.

7.1.3. Modeling

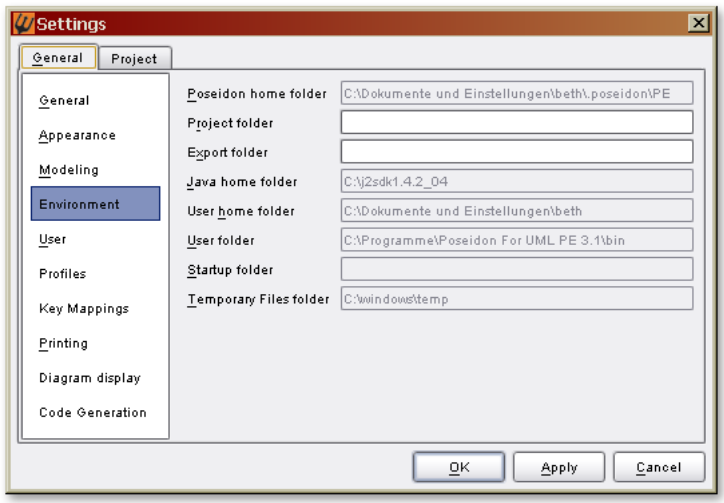
Figure 7-3. The Modeling settings tab.



- **Copy Javadoc down to accessor methods** - If checked, the documentation of the attribute is passed to its accessor methods.
- **Ask before copying down Javadoc** - If unchecked, Poseidon will prompt before overwriting existing documentation of an attribute with the documentation of its accessor methods.
- **Generate accessor methods for attributes** - If checked, accessor methods (get/set) will be automatically created when a new attribute is created.
- **When removing attributes** - Defines what to do with associated accessor methods when an attribute is removed from the model. The possible options are to keep the methods in place, to delete them but ask first (this is the default setting), or to delete them immediately.
- **Warn when doing changes to the model that might harm generated sourcecode** - If checked, a warning will appear to remind you that the change you are attempting could cause problems with code generation.

7.1.4. Environment

Figure 7-4. The Environment settings tab.

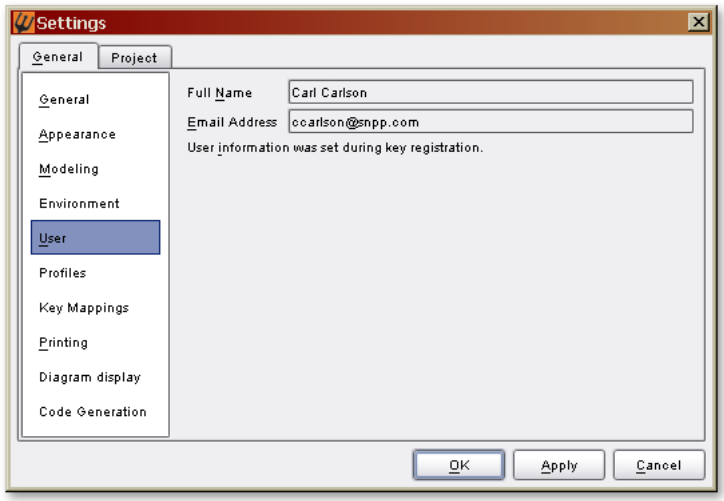


The Environment tab contains properties regarding the local environment and the directories used for loading and saving files.

- **Poseidon Home folder** - The folder Poseidon stores user-related information into, e.g. log files and the saved properties. This property cannot be changed.
- **Project folder** - Projects are loaded from and saved into this preferred folder.
- **Export folder** - Exported files (like graphics) are saved into this preferred folder.
- **Java Home folder** - The folder in which the currently-used version of Java is installed. This property usually points to the runtime part of the installation, even if the used Java is a SDK installation. This property cannot be changed.
- **User Home folder** - The folder your operating system uses as your personal folder. This property cannot be changed.
- **User folder** - The folder into which Poseidon is installed. This property cannot be changed.
- **Startup folder** - The folder your system points to at the startup of Poseidon. This property cannot be changed.
- **Temporary Files folder** - The folder used when any temporary files are created.

7.1.5. User

Figure 7-5. The User settings tab.



The User tab contains properties regarding information about the user. These properties cannot be changed from the settings dialog, because they are part of the product registration. They can be changed using the license manager, but any change would require a new registration of the product.

- **Full Name** - Full name of the user who registered this copy of Poseidon.
- **E-mail Address** - E-mail address of the user who registered this copy of Poseidon. The presented email address must be a real one, or registration of Poseidon will fail.

7.1.6. Profiles

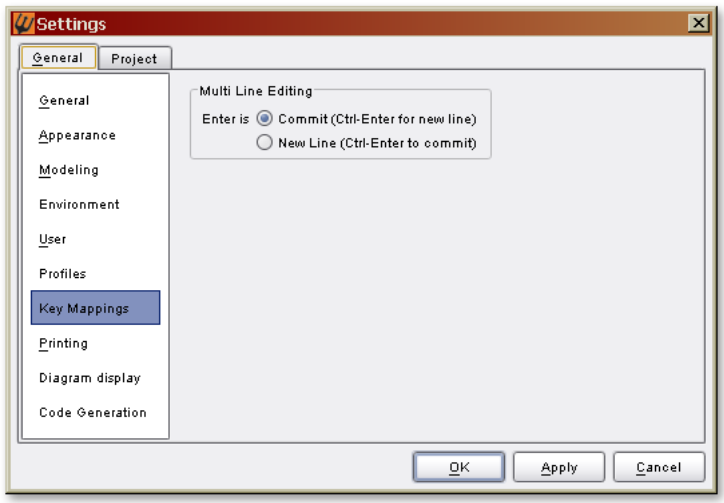
Figure 7-6. The Profile settings tab.

Profiles generically extend the UML through the use of stereotypes that are most often language-specific, provide a common graphical notation and vocabulary, and define a subset ot the UML metamodel (that could possibly be the entire UML metamodel).

- **Default Activated Profiles** - Indicates profiles which will be applied to all projects.

7.1.7. Key Mappings

Figure 7-7. The Key Mappings settings tab.



The Key Mappings tab allows you to determine the behavior of particular keys.

- **Multiline Editing** - This option tells Poseidon whether to use the 'Enter' key to commit an edit or create a new line while entering the name of an element or any other field that accepts multiline text.

7.1.8. Printing

7.1.8.1. Page Setup

Figure 7-8. The Printing Page Setup settings tab

- **Size** - List of paper sizes.
- **Source** - Paper drawer or other source Poseidon should use for paper.
- **Orientation** - Position of the graphic on the page.
- **Measurement Units** - The units to be used for all specified measurements.

- **Margins** - Size of the margins on the paper.
- **Alignment** - Determines the placement of the graphic on the page.
- **Gaps of footer and header** - The amount of space between the header or footer and the diagram graphics.

7.1.8.2. Scaling

Figure 7-9. The Printing Scale settings tab

- **Fit to Page** - Scales the current diagram to fit on exactly one page.
- **Scale to Pages** - Scales the diagram to fit on the specified number of pages.
- **Scale** - Scales the diagram to the specified percentage.
  - Fit to Page - Determines the scale for the entire project based on the 'fit to page' scale of the current diagram. For example, if the current diagram fits one page when scaled to 74%, all diagrams will be printed at 74%.

7.1.8.3. Appearance

Figure 7-10. The Printing Appearance settings tab

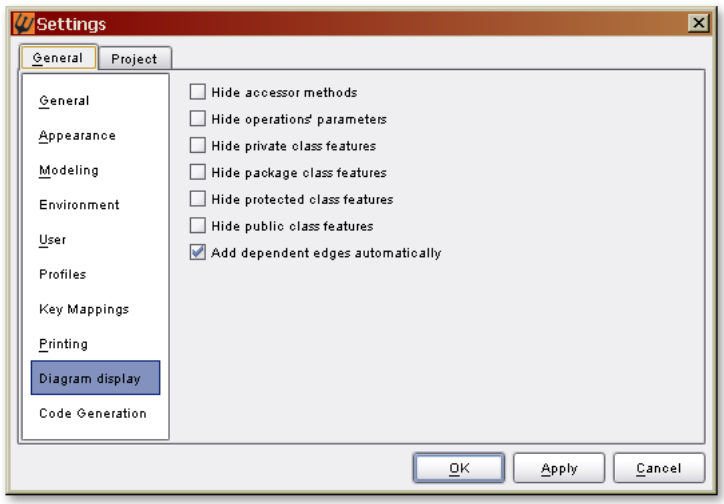
- **Color Appearance** - Sets the print to color or black and white.
- **Quality** - Caliber of printing to be used. In general, lower quality results in faster printing. Three options are available: Draft, Normal, and High.
- **Sides** - Sets single or double-sided printing
  - Simplex - single-sided
  - Tumble - double-sided, flip on short edge
  - Duplex - double-sided

- **Job Attributes**
  - Banner Page - When checked, the print job will be preceded by a page with the job information, such as date, time, and machine name.
  - Priority - Sets the priorty of the print job in the print queue.
  - Job Name - Identifies the job.
  - User Name - Identifies who initiated the print job.

- **Watermark** - Uses the indicated graphic file as a background for the printed diagrams. The graphic is stretched to fit the background, and is partitioned if the diagram is printed on multiple pages.
  - Transparency - Determines the transparency of the diagrams. A value of 0 indicates that the diagram is completely opaque and the background is not visible. A value of 1 indicates that the diagram is invisible and only the background can be seen.

7.1.9. Diagram Display

Figure 7-11. The Diagram display settings tab.



The Diagram Display tab contains properties regarding the display of information within the diagrams. Currently, most of the properties refer to Class Diagrams only.

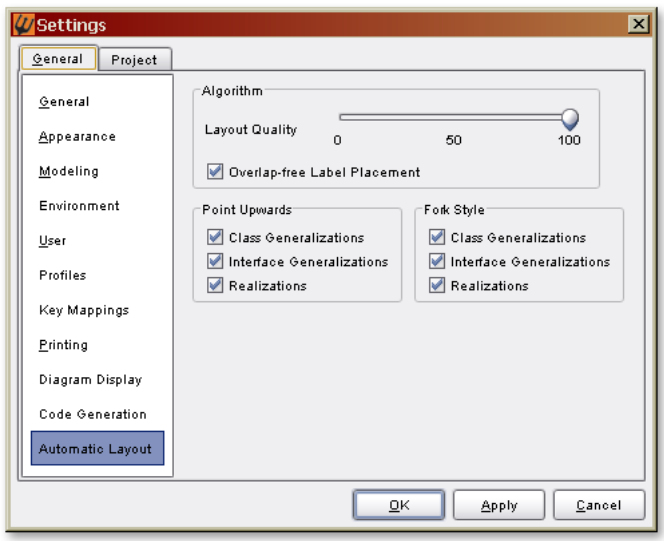
- **Hide accessor methods** - If checked, accessor methods will not be displayed in the operation compartment.
- **Hide operation's parameters** - If checked, parameters will not be displayed. Operations with parameters will then be displayed like `operation1(...)`.
- **Hide private class features** - If checked, private attributes and operations will not be displayed.
- **Hide package class features** - If checked, package attributes and operations will not be displayed.

- **Hide protected class features** - If checked, protected attributes and operations will not be displayed.
- **Hide public class features** - If checked, public attributes and operations will not be displayed.
- **Add dependent edges automatically** - If checked, dependent edges are added to a node which has been created via cut-and-paste or drag-and-drop. No dialog or warning is used.

  Helpful hint: When using Component diagrams extensively or using multiple representations of an element within a single diagram, it is often better to turn off this option.

7.1.10. Automatic Layout

Figure 7-12. The Automatic Layout settings tab.



The Automatic Layout tab contains properties affecting the behavior of the auto-layout function.

- **Algorithm**
  - **Overlap-free Label Placement** - Deselect the checkbox to allow labels to overlap.

- **Point Upwards** - Determines the arrow head direction of the selected element.

- **Fork Style** - Allows relationships to appear forked, sharing a line. When deselected, each relationship appears with it's own line.

7.1.11. Code Generation

Previous to version 2.6, generation settings were contained within their own tabs, accessible from the Generation dialog. Versions 2.6 and later have collected these settings in the central settings dialog, although they are still accessible from the Generation dialog.

7.1.11.1. Documentation Generation

Figure 7-13. The Documentation Generation settings tab

Common Generation Settings

- **Clear destination folder** - When checked, the document target directory is cleared before the generation begins.
- **Parse tagged values and constraints for escape sequences** -
- **Repaint all diagrams** - Redraw diagrams before generation begins.

HTML Generation

- **Image Export Format** - Determines the format of all images, including the diagrams.
- **Generate Authors Doc** - With this setting enabled, @author tags are included in the UMLdoc.
- **Generate External Links** - With this option enabled, @link destinations that are external will be activated within the document.
  - External Link Base - The site noted here will be used as the base link for all external links within the document. The 'Set Default' button will reset the external link base to Sun's Java site (http://java.sun.com/j2se/1.4.2/docs/api (http://java.sun.com/j2se/1.4.2/docs/api)).
  - Copyright for UMLdoc - The copyright notice in HTML format. This information appears at the bottom of every UMLdoc page. The 'Set Default' button resets the copyright to the original Gentleware message.

7.1.11.2. Java

Figure 7-14. The Java Code Generation settings tab

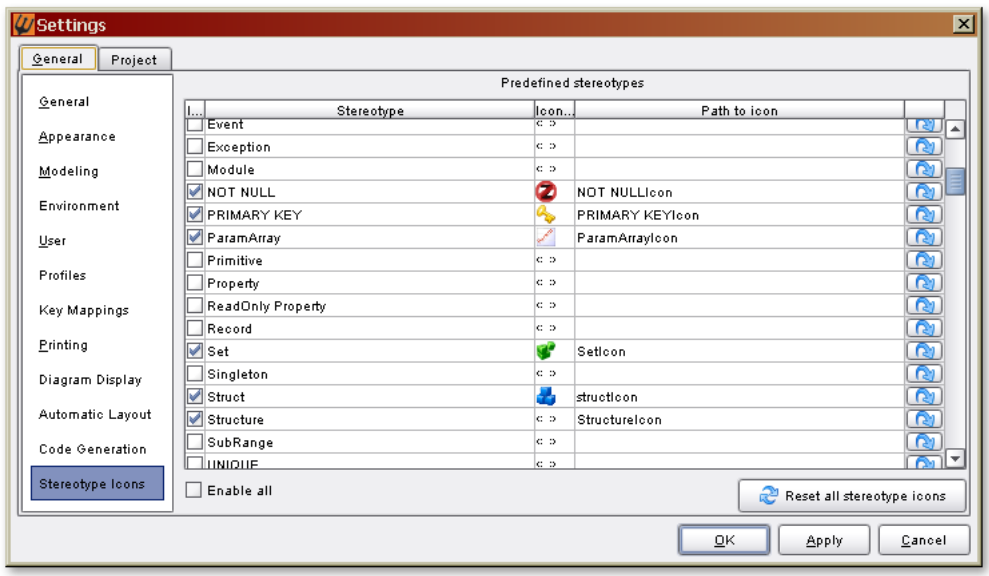
- **Generate accessor methods for associations** - Creates accessor methods for every attribute that is created. If the attribute has a multiplicity of 1..1 or 0..1, simple getAttribute() and setAttribute() methods are created. For attributes with a finite multiplicity, an array is generated and the accessor methods include addAttribute() and setAttribute(). For an unbounded multiplicity, a Collection is generated, and the appropriate access methods like addAttribute() and removeAttribute() are produced.

  You should unset the check box Generate accessor methods after you have generated accessors once. Otherwise, they would be generated again, and would clutter up your classes. The preferred way to create set/get methods is by adding them in an attribute's Properties tab, and by checking Create accessor methods for new attributes in the dialog Edit-Settings.

- **Compiler** - Indicates the path to the desired compiler
- **Type for unordered associations** - Determines how unordered associations will be represented
- **Type for ordered associations** - Determines how ordered associations will be represented

7.1.12. Stereotype Icons

Figure 7-15. The Stereotype Icons settings tab.



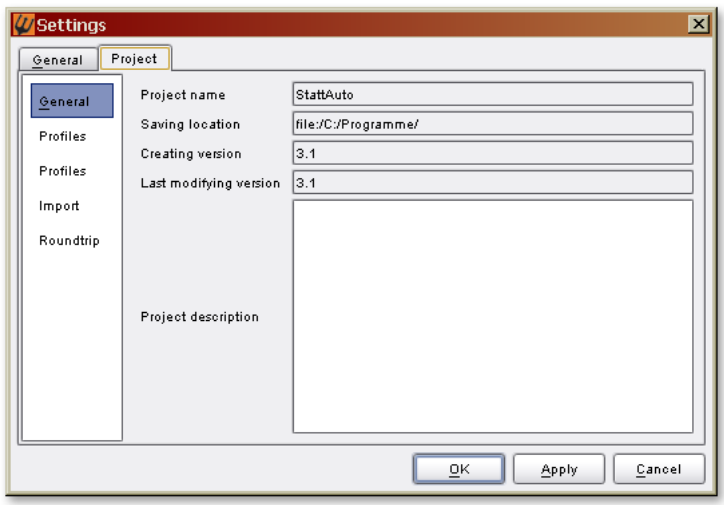
The Stereotype Icons settings dialog sets the graphic representations for stereotypes.

7.2. Project Tab

The properties set in this tab are stored within the project itself in the .proj file within the .zuml bundle.

7.2.1. General

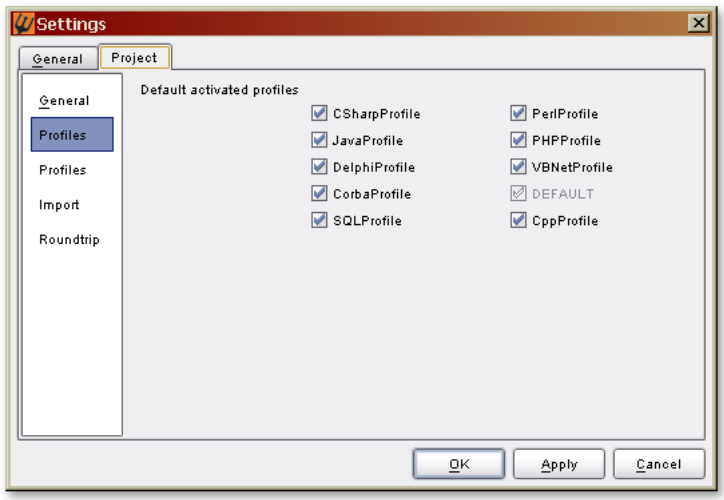
Figure 7-16. The General Project settings tab.



- **Project Name** - Name under which the current project has been saved. If the project has not yet been saved, this will be 'Untitled'.
- **Saving Location** - Path to the location of the saved project. This will be blank if the project has not yet been saved.
- **Creating Version** - The version of Poseidon which was used to create the project originally.
- **Last Modifying Version** - The version of Poseidon which was last used to edit the project.
- **Project Description** - A short, user-defined description of the current project.

7.2.2. Profiles

Figure 7-17. The Project Profile settings tab.

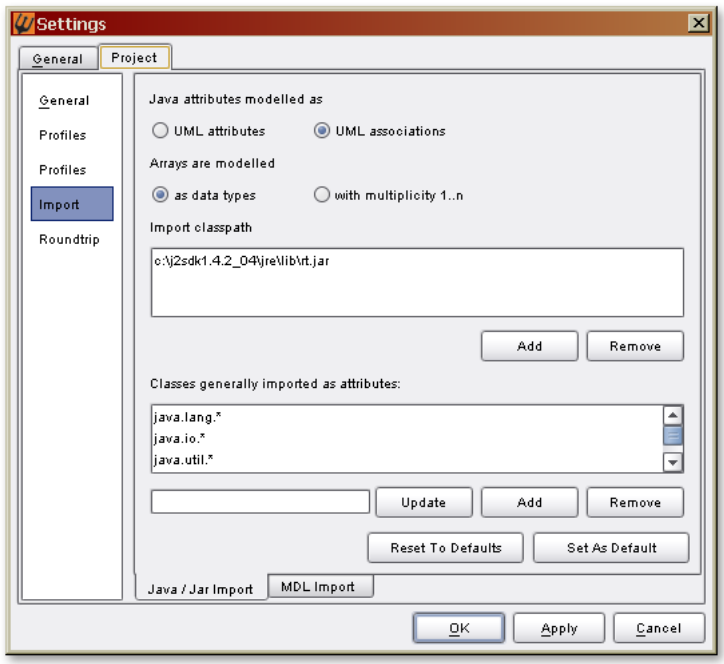


- **Activated Profiles** - Checked profiles are available to the project.

7.2.3. Import

7.2.3.1. Java/Jar Import

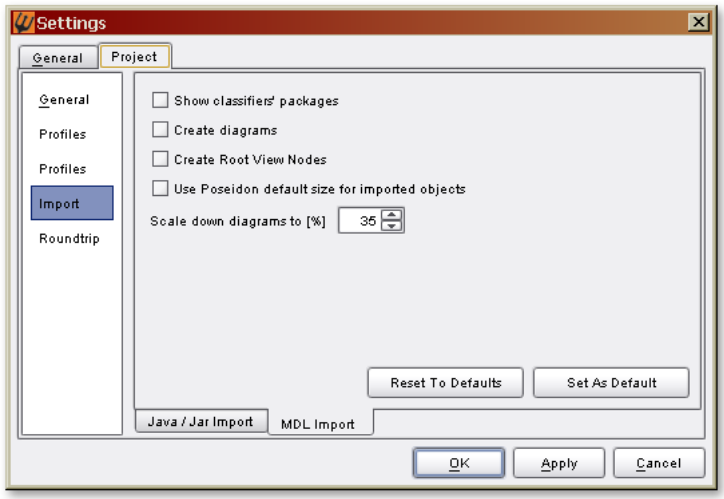
Figure 7-18. The Project Java/Jar Import settings tab.



- **Java attributes modelled as** - Select either attributes or associations. If you select associations, be aware that primitive types and types from the packages java.net, java.io, java.util, and java.lang are always imported as attributes.
- **Arrays are modelled** - Select either as data types or with a multiplicity of 1..n
- **Import classpath** - Add classpaths by clicking the 'Add' button and navigating to the classpath in the browser that opens automatically.
- **Classes generally imported as attributes** - Classes from this list will be imported as attributes, regardless of the setting of 'Java attributes modelled as'.

7.2.3.2. MDL Import

The options available in the import tab depend on the type of project to be imported. Below is an example of the import settings when an mdl file is being imported.



- **Show classifiers' packages** - Hides or displays the package of a classifier
- **Create diagrams** - When unchecked, only the elements will be imported. When checked, all corresponding diagrams will be created.
- **Create Root View Nodes** - When checked, nodes are created for top level Rose views.
- **Use Poseidon default size for imported objects** - When checked, an imported element will be of the same size as a new element of the same type created in Poseidon.
- **Scale down diagrams to [%]** - Automatically resizes diagrams to the specified percentage for easier readability

7.2.4. Roundtrip

7.2.4.1. Roundtrip Settings

Figure 7-19. The Roundtrip settings tab

- **Roundtrip is enabled** - This checkbox toggles roundtripping on and off. This is the same as using the roundtrip button on the toolbar.
- **Roundtrip imports are undoable** - When selected, actions in history before and including a roundtrip import may be undone with the undo command. A dialog will open to inform you that the model and sourcecode will be inconsistent after the undo. When deselected, only those actions occuring in history after the roundtrip import may be reverted with the undo button.
- **Show information about roundtrip settings while trying to enable roundtrip** - When checked, an informational dialog will appear to display the current roundtrip settings.

- **Automatically hide code generation log dialog after generation** - When checked, the code generation status dialog will close when generation is complete.
- **Project Root** - Specifies the root location of the project.
- **Roundtrip sourcepaths** - Indicates where Poseidon will generate the source code. Projects may have more than one sourcepath, and these are assigned to classifiers in the Section 7.2.4.2 tab. Source directories are relative to the project root and must initially be empty.
  - Add folder
  - Change folder
  - Deactivate folder
  - Browse files

7.2.4.2. Classifier Mapping

Figure 7-20. The Roundtrip Classifier Mapping settings tab

From this tab you can assign a sourcepath for each of the classifiers appearing in the project. Assignations are made by first selecting the classifier from the list in the upper field, then double-clicking on the desired sourcepath in the lower field. When a classifier has been assigned, it will change from red to black. The list of available sourcepaths is editable from the Section 7.2.4.1 tab.

7.2.4.3. Import and Code Generation Settings

These statements are merely informational and let you know where roundtrip is getting certain parameters. The buttons next to each statement open the appropriate tab in order to edit these parameters.

Figure 7-21. The Roundtrip Import and Code Generation settings tab

- **Roundtrip is re-using all Java Import settings, e.g. classpath** - Click the 'Import Settings...' button to open the 'Settings | Project | Import | Java/Jar' tab to make changes.
- **Roundtrip is re-using Java code generation settings, except the compiler** - Click the 'Generation Settings...' button to open the 'Settings | General | Code Generation | Java Settings' tab to make changes.

Chapter 8. Model Reference

A UML model can completely describe a variety of systems using both syntactic element definition and semantic diagram rendering. Although semantics are not included in the UML specification, they are important to a UML model in that they clarify meaning and enhance understanding for human readers. It is also possible to enhance diagrams with non-UML elements such as shapes and text objects. These items will not appear in any generated code, but they will be available for viewing within the diagrams.

8.1. Views

Several views of the elements are available within Poseidon. These are available from the tabs in the Navigation pane, and include:

- Diagram Centric
- Model Index
- Package Centric
- HotSpots

Each of these present the elements in a tree view with a different focus in mind. But one thing they all have in common is the root node, which is the model itself. In every UML model, it is the top level namespace.

8.2. Default Naming

The default diagram names were changed in Poseidon 5.x to reflect the namespace to which the diagram belongs.

Chapter 9. Using Models

In the tour of Poseidon, you learned how to navigate an existing model, how to work with existing diagrams and how to edit elements. Now let's take it to the next level and look at what you can do with whole models in Poseidon for UML.

9.1. Creating New Models

Creating new models is very simple. At startup, Poseidon for UML opens with an empty model. This model contains one Class Diagram that is immediately displayed in the Diagram pane. You can start working on this model right away.

To create a new model:

- **Main Toolbar** - Click the  'New Project' button on the main toolbar.
- **Main Menu** - Select 'New Project' from the File menu.

9.2. Navigation

Poseidon was designed to accelerate and simplify the modeling process; therefore it offers many ways to move around within a model. Some users prefer using a mouse, others work more quickly with a keyboard. Some users spend more time in the Details pane, while others spend more time in the diagrams themselves. Regardless, all users need different ways to efficiently find what they are looking for.

9.2.1. Navigation Pane

The Navigation pane in the upper left corner presents the elements of the model in various views. Each view contains a different organizational structure, represented as a tree of model elements. The root node of the tree is always the model itself.

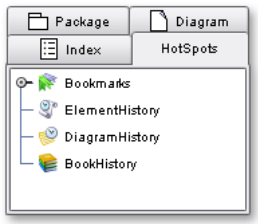
Model elements are available for closer inspection by clicking on the name or symbol of the element. The properties of the selected element are then displayed in the Details pane, located in the lower right section of the application. If the element appears in the diagram that is currently displayed in the Diagram pane, the element will also become the currently selected element within the diagram. Should a diagram be the selected element, that diagram will open in the Diagram pane.

This selection/activation interaction goes both ways. That is, if an element is selected from within the Diagram pane, it is likewise selected in the Navigation pane.

You can also move from a selected element to diagrams belonging to that element with the quick-key Ctrl-Shift-G. If the selected element has more than one diagram, the 'Go To Diagram...' dialog appears. You can then use the mouse or cursor keys and Enter button to select the desired diagram.

9.2.1.1. HotSpots

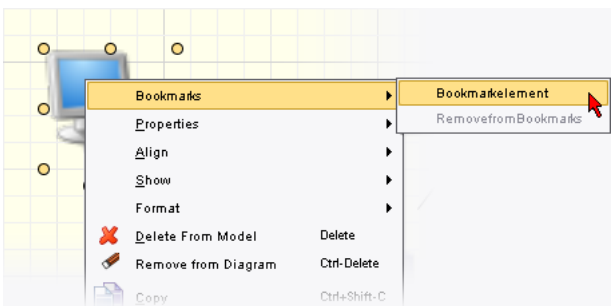
The HotSpots tab, made up of four subtrees, allows users to set bookmarks for easy navigation and maintains histories of various aspects of the model. Bookmarks and history items are displayed in the order in which they were added, with the newest appearing at the top of the list.



- **Bookmarks** - Create bookmarks to elements, diagrams, etc for quick access and navigation to various parts of the model.
- **Element History** - Maintains a list of the most recently accessed elements.
- **Diagram History** - Maintains a list of the most recently accessed diagrams.
- **Book History** - Maintains a list of themost recently accessed books for documentation generation.

Bookmark an Element

Simply right-click on the element and select 'Bookmark Element' from the context menu.



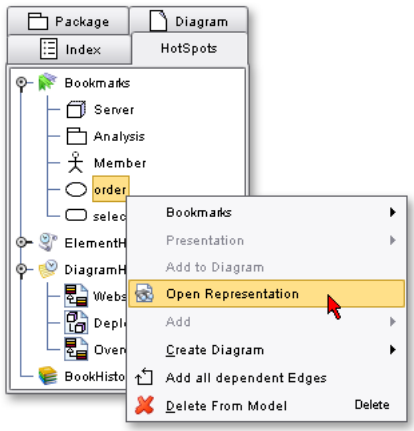
The same method is used to remove an element from the bookmarks list.

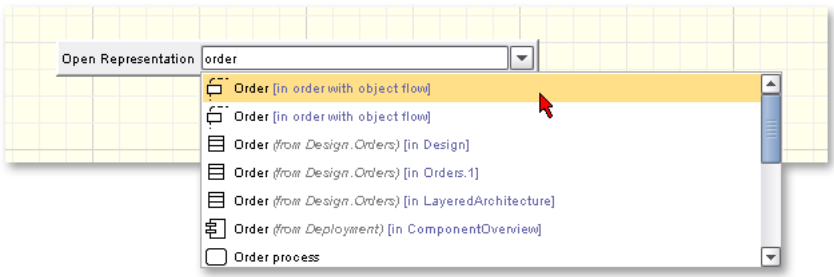
Navigate to an Element

Select the desired element from the list. The properties for the element will be opened in the Details pane at the bottom of the screen.

Open an Element in a Diagram

Right-click the element in the Bookmarks tree. As an element can have multiple representations, a dropdown selector will appear. Select the desired representation from the list, and the appropriate diagram will be opened.





9.2.2. Details Pane

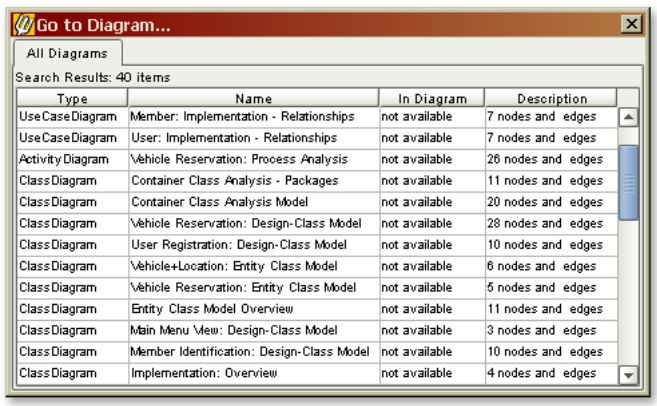
The Details pane contains numerous tabs, and it is the Properties tab that provides the means for drill-down navigation. Each element contains or is a part of other elements, as an operation is both a part of a class and contains parameters, for example. Moving between these elements is as simple as double-clicking on the name of the element in the tab.

Also available are the up, down, and back buttons, which move you through the hierarchy of the elements.

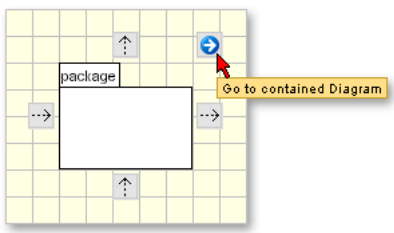
9.2.3. Diagram Pane

The easiest way to navigate between elements within in the Diagram pane is to simply click on any element.

But what about moving between the diagrams themselves? Easy. Open the 'Go To Diagram...' dialog by pressing Ctrl-G. In the dialog, you can use the arrow keys to move around. The space bar switches the diagram that appears in the background, and Enter selects the diagram and exits the dialog.



If an element contains diagrams (a package for example), a rapid button takes you to the elements sub-diagram(s).



The quick-key Ctrl-Shift-G will also do this, and if an element has multiple diagrams, it will open a dialog with the list of available diagrams.

9.3. Saving and Loading Models

Saving the models you created in Poseidon for UML is routine, but there are a few things that should be mentioned about saving and the format used.

Open Standards Support

Poseidon for UML supports open standards extensively, and this is also true for the saving format. UML is standardized by the Object Management Group (OMG). Part of the official UML specification by the OMG (http://www.omg.org) is a mechanism for the exchange of models between different tools. This mechanism is based on XML and has special extensions and rules to better represent object-oriented structures as well as metadata. The OMG has specified a concrete application of XML for this purpose that is called the XML Metadata Interchange, or XMI for short. Poseidon for UML makes use of this format. In fact, while most other tools can only import or export XMI, Poseidon for UML uses XMI, as specified by the Diagram Interchange standard, as the default saving and loading mechanism.

Introducing the .zuml File

The previous version of UML had no standards for storing the graphical information of a diagram. This made exchanging a model between tools very difficult. UML 2.0 has solved this issue with the inclusion of the **Diagram Interchange standard** , which specifies exactly how graphics are to be stored and rendered. Gentleware was at the forefront of the development of this standard and is therefore uniquely able to implement it in Poseidon.

Now diagrams are written in the XMI 1.2 format, the same format used to store the model itself in both UML 1.x and UML 2.0. Poseidon creates a project file with a '.zuml' extension, which is a .zip file containing a .proj file with project information, and an .xmi file with the model and layout (Diagram Interchange) information. This method of storage is supplementary to the previous method, meaning that projects created with previous versions of Poseidon (.zargo files) or other tools will open in Poseidon version 2.0, but diagram information will be converted before it is opened.

To Save a Model:

- **Main Menu** - Select 'Save Project' or 'Save Project As...' from the main menu
- **Quick-Key** - Use the quick-key Ctrl-S to save a project

To Load a Model

- **Main Menu** - Select 'Open Project' from the main menu
- **Quick-Key** - Use the quick-key Ctrl-O to open a project

You can also import XMI that was created by other UML tools.

Components Of A .zargo File

Poseidon 1.x saves projects with a .zargo extension. These files can be opened in Poseidon 2.0, but new projects created with Poseidon 2.0 cannot be saved in this format. The following section briefly explores these files.

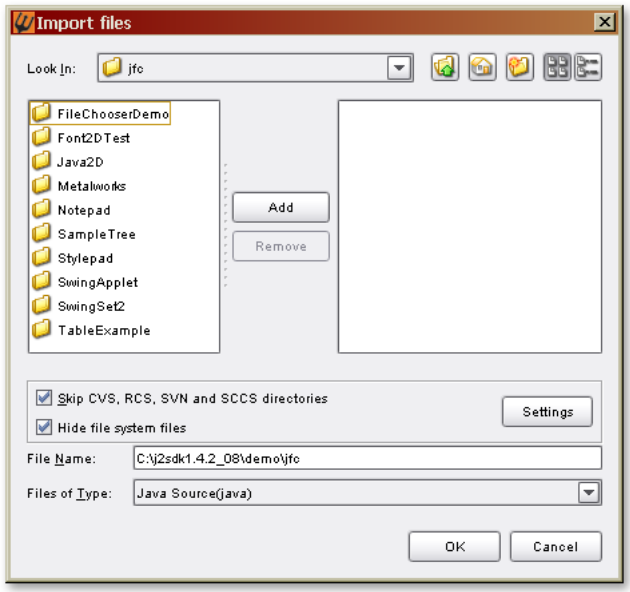
The current version of XMI is, by itself, not sufficient to save all aspects of a UML model. It can be used to transport the names and properties of all model elements, but diagram information (layout, colors, etc.) is not included, therefore this information has to be stored in a different format. Poseidon 1.x uses another XML application, called PGML, which is a predecessor to SVG, the Scalable Vector Graphics format, standardized by the W3C.

Finally, some internal information about the model needs to be stored. This is done in yet another XML-based format with the ending `.argo`. All of the files mentioned are zipped together into just one compressed file with the ending `.zargo`. This is actually just a regular ZIP file; you can decompress it using any ZIP tool or the Java JAR tool. Usually you don't have to worry about all this. But sometimes if, for example, you want to access the XMI file to exchange it with other tools, you may need to unzip this file and have a closer look inside.

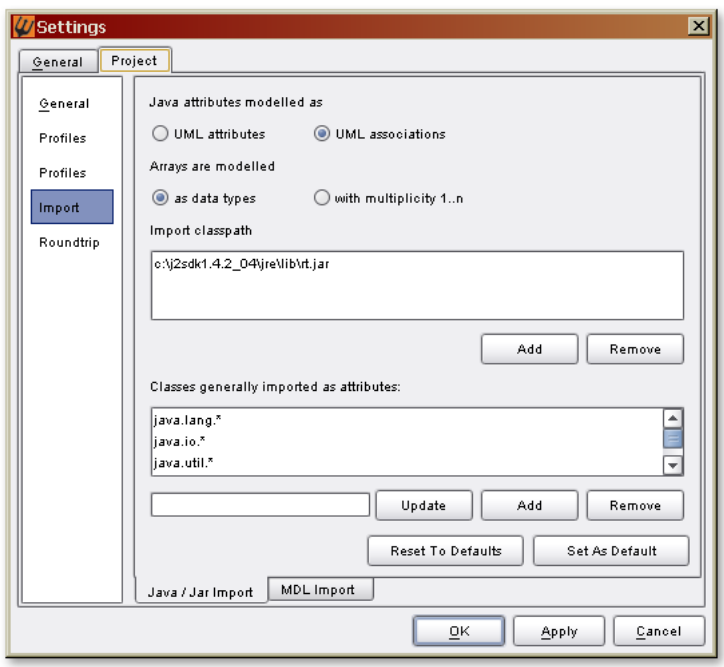
Poseidon 1.5 and 1.6 use a different XMI format than previous Poseidon versions (up to version 1.4.1). Support for XMI 1.1 and 1.2 as well as UML 1.4 was added, and all .zargo files that were created with older Poseidon versions are converted when necessary, so there is no need to care at all about the different versions. Gentleware also works on better import functionality so that the XMI generated by other CASE tools can be imported smoothly.

9.4. Importing Files

Poseidon for UML Professional and Embedded editions provide a convenient dialog to assist with the importation of source code. Select 'Import files' from the file menu, navigate to the desired file, and click the 'Open' button.

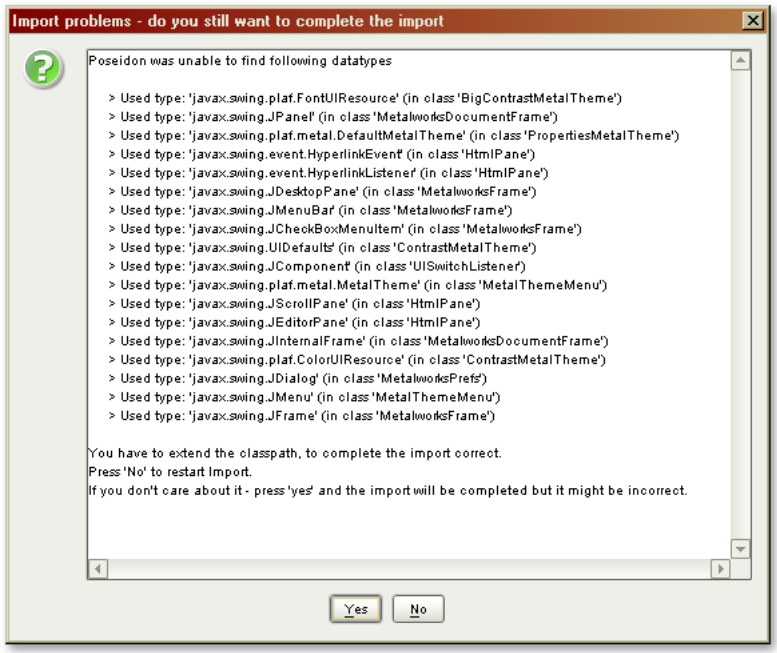


The 'Settings' button will open the settings dialog. Import options, such as adding a classpath, are available from the Project tab under 'Import'. More information about this tab is contained in the section Section 7.2.3 .



Should the import fail because needed files have not been located, an error dialog will appear. If you would like to continue without adding to the classpath and create dummy classes instead, click 'Yes'. To

return to the previous dialog and add the necessary files to the classpath, click 'No'.



Once the import has begun, a dialog will appear asking if you would like to generate diagrams. A tree of the project is also presented, so that you can choose which packages and classes you would like to include in the diagrams. By default, any packages that contain new classes are automatically selected.

9.5. Importing Models

Models from other tools can be imported directly into Poseidon models. This allows for the merging of two or more sub-models into one or the importing of models from different formats. For example, Poseidon can import files stored in MDL format - the file format used by Rational Rose. This feature is available in the Professional, Enterprise, and Embedded Enterprise editions.

To Import Sub-Models:

- **Main Menu** - Select 'Import Files' from the main menu

Keep in mind that importing an XMI file means that no layout, color, or style information is included, as the XMI format simply does not contain this kind of data. You will have to create your own diagrams by dragging elements from the Navigator to the Diagram pane.

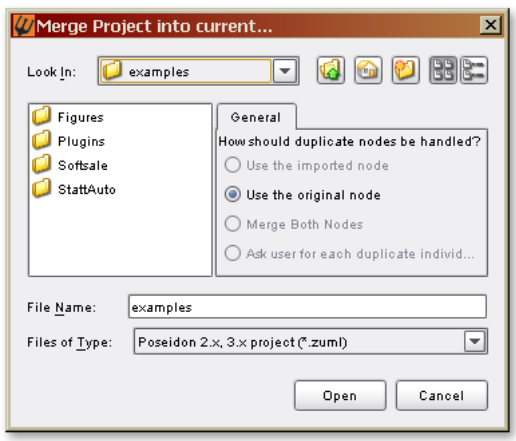
To Import XMI Files Created By a Different Tool:

- **Main Menu** - Select 'Open Project', change the file chooser to XMI, then select the XMI file

9.6. Merging Models

While the import option will bring .java, .jar, and .mdl files into a Poseidon project, the merge option allows you to add .xmi, .zargo, and .zuml files to the open Poseidon project.

From the File menu, select 'Merge Project Into Current...' A dialog will appear where you can specify the file to merge.



Note: The only option available at the moment for duplicate node handling is to use the original node. The rest of the options have not been implemented and are displayed greyed out.

9.7. Exporting Models

The XMI File Type

For the interoperability of different UML tools, it is important to be able to export models from a proprietary to a common format. UML defines a standard exchange format for UML models called XMI, which Poseidon for UML uses as the default saving format. This means that every time you save a model

it is stored in an XMI file. However, since XMI is a quite wordy xml format and lacks layout information, Poseidon compresses this file, and zips it together with other project information. This file has the name of your project and the ending .zuml. To get to the XMI file, unzip this file with the compression software of your choice.

From version 2.1 on, it is possible to include diagram data in an XMI file by selecting this option during the export process.

Advantages of XMI

Such a standard interchange format has a number of applications. It not only makes sense to be able to replace one tool by another or to exchange models with people using other tools, it also makes sense for chains of tools. The following example well illustrates this value addition:

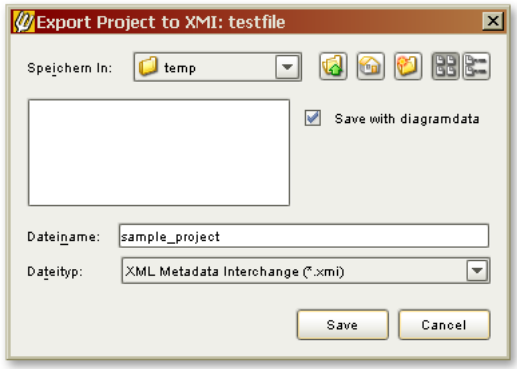
Some tools are especially well prepared for capturing models designed in cooperative sessions on the white-board. The model is sketched on the white-board, the gestures are tracked and transformed to UML model elements, and a laptop in conjunction with a projector places the newly-created diagram back onto the white-board. This demonstrates and facilitates a creative and cooperative style of working on a model as a team.

Different tools from other vendors are specialized on generating code. They might not even have a graphical user interface, they simply read in a model and produce code for a specific type of application or platform. Such tools can be connected to Poseidon using the XMI format. However, the XMI standard is not implemented equally well among different tools. Poseidon for UML is known to produce one of the cleanest XMI files for its models, and many tools have chosen to support our variant of XMI. However, the interchange might not work with all other tools. The Diagram Interchange standard should alleviate some of these issues once other tools implement the recognized standard.

To Export a Project

1. Open the File menu and select ’Export Project to XMI’.
2. The Export Project dialog will open. To the right, select or deselect ’Save with diagramdata’.
3. Select a location and file name, then click ’Save’.

Figure 9-1. Export a project to XMI



9.8. Exporting Graphics and Printing

Another option that you will find useful is the export of diagrams as graphics. Whether you want to use your diagrams in other documents, in a report, a web site, or a slide show, you can export them in a range of different formats depending on your needs.

Formats

The currently available formats are Joint Photographic Experts Group (JPG), CompuServe Graphics Interchange (GIF), Portable Networks Graphics (PNG), Portable Document Format (PDF), Postscript (PS), Encapsulated Postscript (EPS), Scalable Vector Graphics (SVG), and Windows Meta File (WMF).

The first six are well known for their respective areas of usage, but for our purposes the most promising format is SVG. There are not many applications that support it yet, but in the near future this is likely to change to be the standard format of choice for web content as well as for text documents. If you want to try to exporting and viewing diagrams in SVG, there is a browser plug-in (for the Internet Explorer) available from Adobe. There also is an appropriate graphics tool called Batik, available from the Apache project.

Export a Diagram to a Graphic File

- **Main Menu** - Select 'Save Graphics...' from the File menu.

In the Community Edition of Poseidon for UML version 2.x, graphics generated contain a watermark that appears in the background of the exported graphic file but does not affect any of the diagram

information. This watermark has been removed for version 3.0 and is replaced by a default footer that states, "Created with Poseidon for UML Community Edition".

Figure 9-2. Watermarked Community Edition diagram graphic

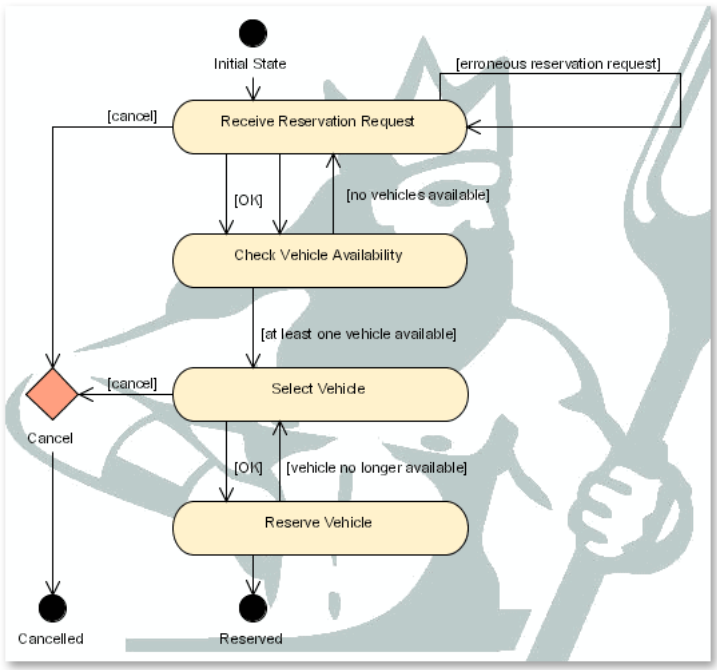
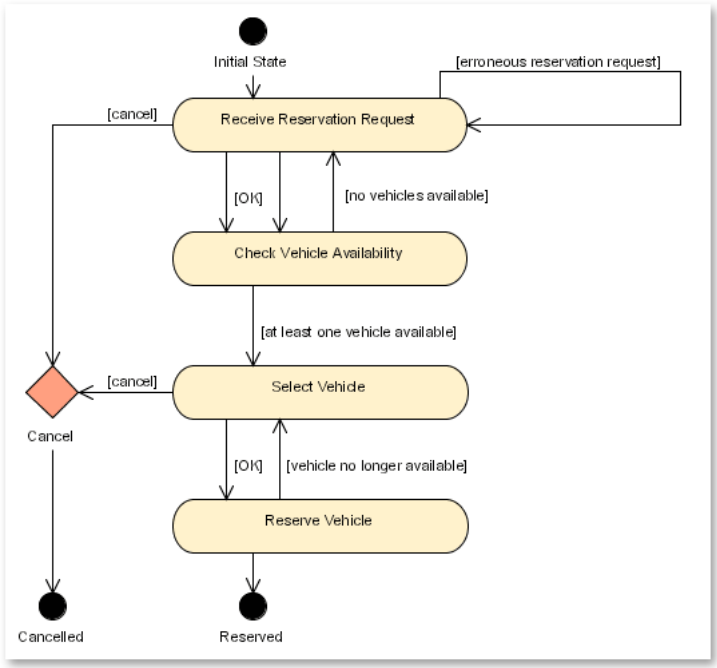


Figure 9-3. Premium Edition diagram graphic without watermark



Printing

You can also directly print diagrams to a printer. The Print function (Ctrl-P) prints the current diagram; be aware that the diagram must be the currently selected element or the print function will not be available. Many new printing features like model-wide scaling and custom headers and footers are available in versions 2.6 and later. See Section 7.1.8 for details about how to set these options.

The Print Diagrams function calls up a window for you to select which diagrams to print. You can navigate through the diagram tree and select any number of diagrams by pressing the Ctrl key and clicking the relevant entries.

Anti Aliasing

Anti-aliasing smooths out the lines and text in a diagram, making them appear less pixilated. At times this may cause the graphics and text to appear fuzzy, particularly when printed. From the Settings dialog, you can set anti-aliasing for text and diagrams separately for printing and screen viewing. More information is available from Section 7.1.2 .

Chapter 10. Diagram Reference

There is a lot to say about when to use which diagram type when developing a design, and what the role of it should be. The different answers to this are referred to as the design process or design method. This document is not intended to describe a concrete design process. Poseidon for UML can be used for any such process. Instead, in this chapter we will look at the various diagram types and how the corresponding model elements are created or edited in Poseidon. For many of these diagrams, a short example has already been given in the default model `Stattauto` , which we looked at in Chapter 4.

10.1. Structural Diagrams

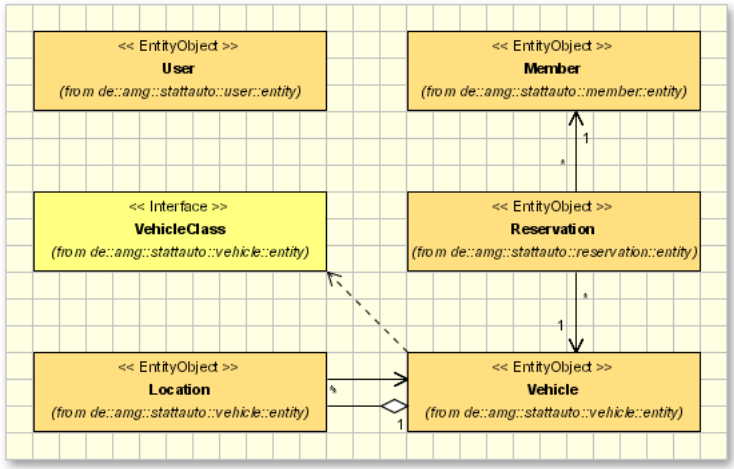
Structural diagrams are used to show the organization of the system and include Class, Component, Deployment, and Object diagrams.

10.2. Class Diagram

Class diagrams are probably the *most important diagrams* of UML. They can be used for various purposes and at different times in the development life cycle. Class diagrams are often applied to analyze the application domain and to pin down the terminology to be used. In this stage they are usually taken as a basis for discussing things with the domain experts, who cannot be expected to have any programming nor computer background at all; therefore, they remain relatively simple like this typical example, the `Entity Class Model Overview` Class Diagram.

Please note that graphical elements have been added to this diagram simply to highlight different regions.

Figure 10-1. A Class diagram.



Once the domain has been established, the overall architecture needs to be developed. Class Diagrams are used again, but now implementation-specific classes are expressed in addition to the terms of the domain.

If a class is shown in a diagram of a different package, the text *(from package.subpackage)* is displayed just under the class name in the diagram. You can turn it off with the Context menu of the class. Move the mouse over the class, right-click, and select Display - Hide Package display.

10.2.1. Diagram Elements

- **Packages** - Packages are used to structure the model. Placed into Class Diagrams, they illustrate the hierarchy explicitly. Classes can then be nested inside them, or they can be used exclusively to express the interdependencies of the packages. These diagrams are sometimes referred to as package diagrams, but in Poseidon you do not need to make a difference here and can combine them at will.
- **Dependencies** - Exist between packages, and express that classes within one package use classes from the package on which it depends.
- **Collaborations** - Exist between objects. Additionally you have to associate a Classifier Role to this collaboration to illustrate what role a special element plays in that collaboration.
- **Interfaces** - Restricted to contain operations only, no attributes. Operations are abstract and have no implementation from within the interface. The class that implements the interface is also responsible for implementing the operations. Interfaces can also be represented with lollipop (or ball) n
- **Classes** - Classes are the most important concept in object-oriented software development, and in UML as well. Classes hold operations and attributes and are related to other classes via association or

inheritance relations. A class has a few properties of its own, such as name, stereotype and visibility, but the more important aspect is its relation to other classes.

- **Inheritance relations** - Relations between interfaces or between classes. They are not allowed between an interface and a class.
- **Implementation relations** - Relations which exist only between interfaces and classes.
- **Association Relations** - Relations between classes.

10.2.2. Toolbar

 Select
 Add or remove space between elements
 Class
 Package
 Actor
 Actor as Classifier
 Generalization
 Dependency
 Association
 Directed Association
 Aggregation
 Composition
 Association Class
 Interface
 Interface as Circle
 Realization
 Lollipop
 Socket
 Collaboration
 Classifier Role
 Attribute
 Operation
 Comment
 Connect Comment to Element
 Text
 Circle
 Rectangle
 Polygon
 Polyline
 Repaint the diagram
 Do layout

 Update layout

 Zoom to 100%

 Zoom to Fit

 Zoom to Selection

10.3. Object Diagram

Object diagrams show classes at the instance level.

Since objects are not on the same conceptual level as classes, although very closely related, they are expressed in separate diagrams. On the other hand, objects are on the same conceptual level as instances of components and instances of nodes. That's why Poseidon for UML combines the functionality for creating Object diagrams, Component diagrams and Deployment diagrams into a single editor; therefore, to create an Object diagram, use the editor for the Component diagram

This may not seem very intuitive at first, but we found it to be very useful. Objects, component instances and node instances can thus be used conjunctively. You can still restrict yourself to use only objects and their links in a deployment diagram.

The diagram elements and toobar options are provided here for quick reference. A much more comprehensive look at the editor is provided in the section on Component diagrams.

10.3.1. Diagram Elements

- **Nodes** - Nodes represent the hardware elements of the deployment system.
- **Components** - Components stand for software elements that are deployed to the hardware system.
- **Links** - Links are used to connect instances of nodes or objects.
- **Dependencies** - Dependencies exist between components and can be specified by utilizing predefined or user-defined stereotypes.
- **Associations** - Associations are used to display communication relations between nodes. They can be specified by utilizing predefined or user-defined stereotypes.
- **Instance Specifications, Classes, Interfaces** - Components and nodes can include objects, classes or interfaces.

10.3.2. Toolbar


Select


Add or remove space between elements


Component


Instance Specification


Node


Realization


Assembly Connector


Delegation Connector


Link between two Instance Specifications


Dependency


Association


Directed Association


Aggregation


Composition


Association Class


Class


Package


Interface As Circle


Collaboration


Comment


Connect Comment to Element


Text


Circle


Rectangle


Polygon


Polyline


Repaint


Do layout


Update layout


Zoom to 100%


Zoom to Fit


Zoom to Selection

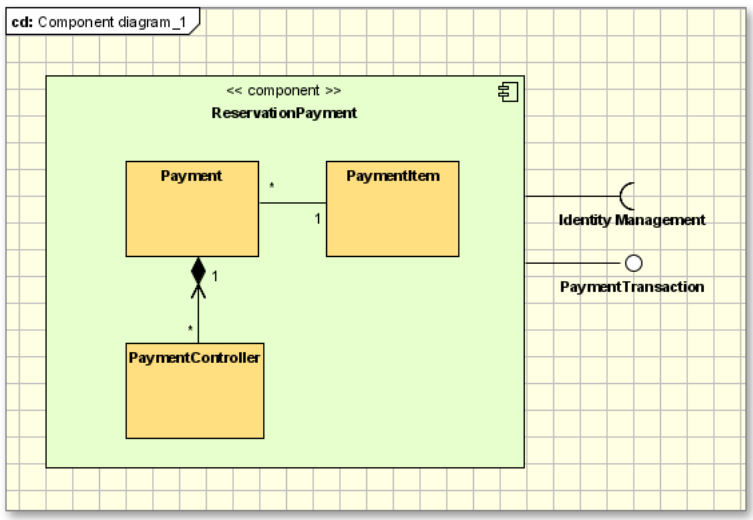
10.4. Component Diagrams

After a while, clusters of classes that strongly interact and form a unit will start to peel out from the architecture. To express this, the corresponding clusters can be represented as components. If taken far

enough, this can lead to a highly reusable component architecture. But such an architecture is hard to design from scratch and usually evolves over time.

Component diagrams are often used to model high-level software components and how they interact. The interfaces between these components become clearer as the model grows, which provides a much clearer delineation of duties of each component.

Figure 10-2. A Component diagram.



When working extensively with Component diagrams, it is a good idea to turn off automatic edge rendering from `Edit -> Settings -> Diagram display -> Add dependent edges automatically` . This will prevent potentially confusing edges to appear when working with multiple representations of the same element within a single diagram.

Components

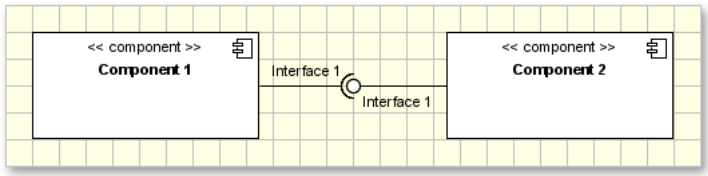
The update from UML 1.x to 2.0 where the actual meaning of the term 'component' was altered necessitated some changes to the Component Diagram. Previously, a component referred to an item to be implemented, such as an executable. As of UML 2.0, components are treated as encapsulated entities that serve as a black box to hide behavior and provide interfaces to the component. This transforms them into more logical elements that can be easily replaced as required by the design.

Required and Provided Interfaces

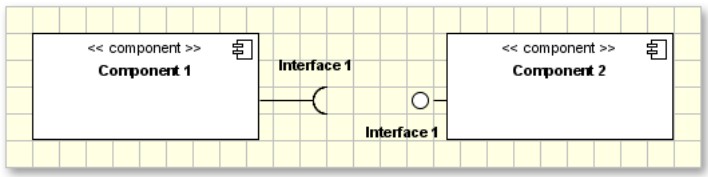
Components hide the inner workings of a component; each component is a general logical unit. Truly

essential to Component diagrams are the provided and required interfaces that determine how the components are wired together and how they interact with one another.

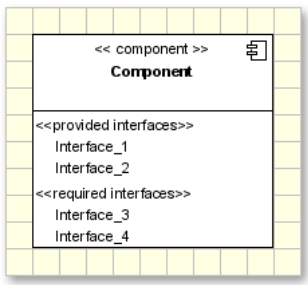
Interfaces can be represented with 'ball and socket' notation: sockets indicate required interfaces and balls (or lollipops) indicate provided interfaces. They can be shown together as a single interface:



Or they may be shown shown separately. The 'Split Assembly Connector' rapid button will disconnect them and display the same interface in two places.



Interfaces may also be shown within the components themselves, regardless of whether the ball and socket notation is being used simultaneously. To do this, select the 'Provided Interfaces' and 'Required Interfaces' checkboxes from the Visibility dropdown menu in the Details Pane toolbar.



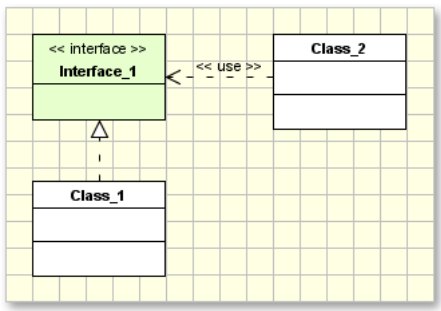
Add an Interface to a Component

To simply add an interface to a component, you can create a new component in a Component diagram, then create the new interface with either the required interface or the provided interface rapid buttons. See Section 12.3 for more information on working with interfaces in Poseidon.

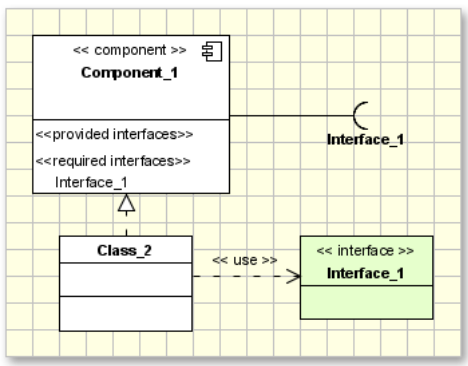
The method mentioned above will certainly create the interfaces, but they are lacking the semantic

information that makes UML so useful. If we take a look at the class diagram below, we can see behind the scenes. We have an interface called Interface_1 that is implemented by Class_1 (Class_1 has Interface_1 as a Provided Interface), and that is used by Class_2 (Class_2 has Interface_1 as a Required Interface).

It is in these classes and interfaces that the true behaviors of the component are defined.

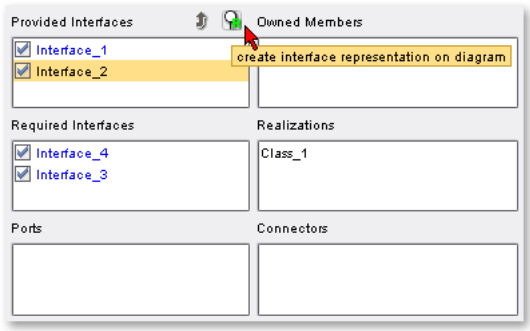


When seen from the Component diagram, we see that Component 1 is realized by Class_2. Logically, Component_1 has the same interfaces as Class_2, which we see here in three ways.



First, we see the use relationship between Class_2 and Interface_1 along with the realization between Class_2 and Component_1. Next, we see the list of required interfaces within Component_1. And finally, we see Interface_1 in socket notation. Any one of these methods is sufficent, we have simply chosen to use all three for this demonstration.

To display the representation of an interface that already is listed by a component, select an interface from the Properties tab, then click the 'Add Representation' button from the toolbar above the interfaces list.



Try it Yourself - *Add a Provided Interface to a Component*

1. Open a Class diagram.

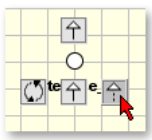
2. Create a new interface by clicking the 'New Interface' button on the toolbar, then click in the diagram to create 'Interface_1'.

3. Create a new Component diagram by clicking the 'New Component Diagram' button.

4. Add a component to the diagram by clicking the 'New Component' button, then clicking in the diagram.

5. From the Navigation pane, drag Interface_1 into the Component diagram.

6. Click and drag from Interface_1's 'Realization' rapid button to Component_1. Component_1 now has Interface_1 as a Provided Interface.



Try it Yourself - *Add a Required Interface to a Component through a Class Realization.*

1. Open a Class diagram.

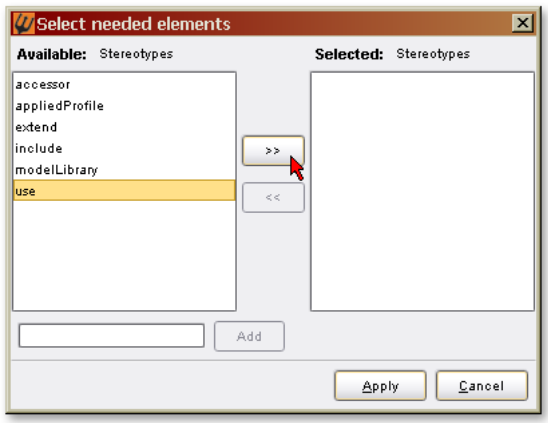
2. Create a new class by clicking the 'New Class' button on the toolbar, then click anywhere in the class diagram. By default, it will be called 'Class_1'.

3. Create a new interface by clicking the 'New Interface' button on the toolbar, then click in the diagram to create 'Interface_1'.

4. Click on the 'Dependency' button in the toolbar, then click and drag from 'Class_1' to 'Interface_1'.

5. In the Properties tab of the dependency, click the Ellipse button to open the Stereotype dialog.

6. Click 'use' from the left column, then click the arrow pointing to the right, then click 'Apply' to add the 'uses' stereotype to the dependency.



7. Create a new Component diagram by clicking the 'New Component Diagram' button.

8. Add a component to the diagram by clicking the 'New Component' button, then clicking in the diagram.

9. From the Navigation pane, drag Class_1 into the Component diagram.

10. Click and drag from Class_1's 'Realization' rapid button to Component_1. Component_1 now has Interface_1 as a Required Interface.

Extra hint: Class_1 can be taken out of the diagram by selecting the class, then clicking the 'Remove from Diagram' button in the Properties pane.

Ports

Along with interfaces, ports may be added to components. For full details about how to work with ports, please see Section 12.3.3 .

Instance Specifications

Objects from UML 1.4 have been redefined in UML 2.0 as instances using Instance Specifications.

Assembly and Delegation Connectors

An assembly connector is a connector between two components that defines that one component provides the services that another component requires. An assembly connector is a connector that is defined from a required interface or port to a provided interface or port. - *OMG UML 2.0 Superstructure Specification p. 150*

In Poseidon, an assembly connector between two components with compatible interfaces will automatically add these interfaces to the component. Their display can be controlled from the Properties panel.

A delegation connector is a connector that links the external contract of a component (as specified by its ports) to the internal realization of that behavior by the component's parts. It represents the forwarding of signals (operation requests and events): a signal that arrives at a port that has a delegation connector to a part or to another port will be passed on to that target for handling. - *OMG UML 2.0 Superstructure Specification p. 150*

10.4.1. Diagram Elements

- **Nodes** - Nodes represent the hardware elements of the deployment system.
- **Components** - Components stand for logical elements that are deployed within the system.
- **Links** - Links are used to connect instances of nodes or objects.
- **Dependencies** - Dependencies exist between components and can be specified by utilizing predefined or user-defined stereotypes.
- **Associations** - Associations are used to display communication relations between nodes. They can be specified by utilizing predefined or user-defined stereotypes.
- **Instance Specifications,** **Classes,** **Interfaces** - Components and nodes can include objects, classes or interfaces.

10.4.2. Toolbar

 Select

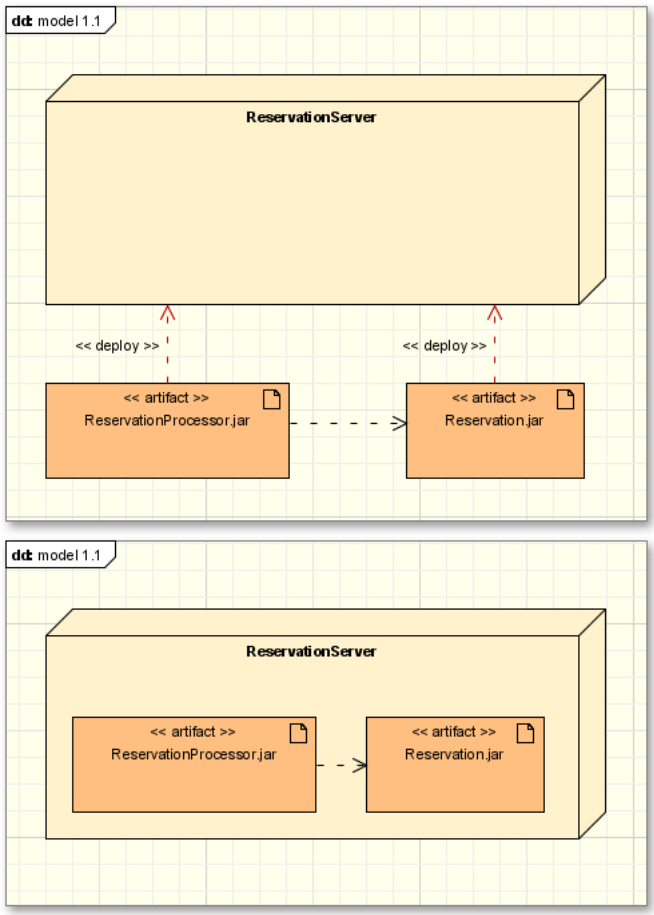
 Add or remove space between elements

 Component
 Instance Specification
 Node
 Realization
 Assembly Connector
 Delegation Connector
 Link between two Instance Specifications
 Dependency
 Association
 Directed Association
 Aggregation
 Composition
 Association Class
 Class
 Package
 Interface As Circle
 Collaboration
 Comment
 Connect Comment to Element
 Text
 Circle
 Rectangle
 Polygon
 Polyline
 Repaint
 Do layout
 Update layout
 Zoom to 100%
 Zoom to Fit
 Zoom to Selection

10.5. Deployment Diagrams

Finally, the way the individual hardware and software components and artifacts are deployed within a system can be described using a Deployment diagram. The specifications for Deployment diagrams changed in the transition from UML 1.x to UML 2.x. Beginning with Poseidon 5.0, the UML 2.1 specifications are used.

Figure 10-3. Deployment diagram, two representations.



A deployment diagram can cover a wide variety of levels within a single model. For instance, one diagram might cover how the application layers are connected, another might indicate where the source code is located within the system, and another might show how an executable is used across several nodes.

Node

A node can be anything that performs work in a system. It is denoted in UML as a 3-dimensional box. In UML 2.x, nodes may now contain other nodes, and further can be sub-classed into Devices and Execution Environments. To create a node in a deployment diagram, click on the Node button, then click in the diagram.

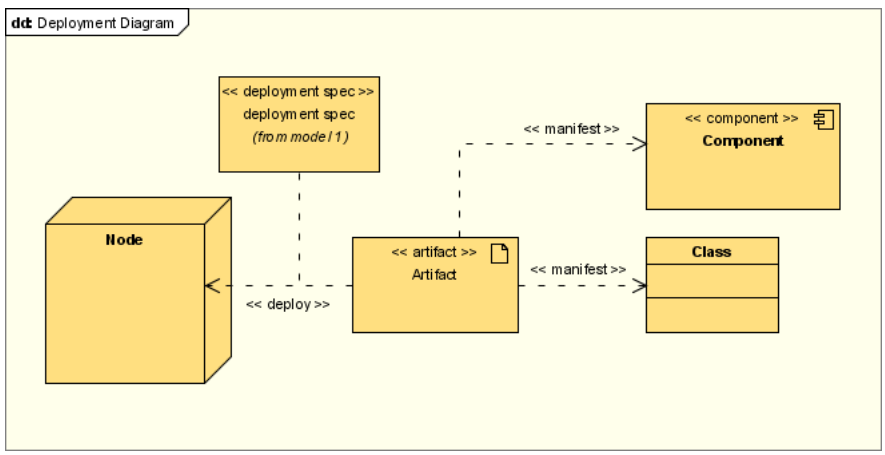
Component

Components in a UML 2.x Deployment diagram are not directly used by nodes. The type-level depiction of components occurs in the Component diagram, and the instances of these components are used by the Deployment diagram. This is most often done with the use of artifacts.

Artifacts

Artifacts are a physical pieces of information, such as files, models, or tables. Artifacts are said to be manifested from an element abstraction. In the context of a Deployment diagram, this could be, for example, a use case, a class, or a component. This relationship is represented with the 'manifest' relationship.

Artifacts are nested in nodes on a Deployment diagram as the implementation of a component. This may be represented in two ways: the artifact may be placed directly in the node, or it may be connected to the node via a 'deploy' relationship.



10.5.1. Diagram Elements

- **Nodes** - Represent the hardware elements of the deployment system, anything that performs work in the system.
- **Components** - Define requirements for software elements that are deployed to the hardware system.
- **Artifacts** - A physical piece of information.
- **Links** - Used to connect instances of nodes or objects.
- **Dependencies** - Exist between components and can be specified by utilizing predefined or user-defined stereotypes.
- **Associations** - Used to display communication relations between nodes. They can be specified by utilizing predefined or user-defined stereotypes.

10.5.2. Toolbar

 Select

 Add or remove space between elements

 Node

 Device

 Execution Environment

 Artifact

 Deployment

 Manifestation

 Deployment Specification

 Instance Specification

 Link between two Instance Specifications

 Component

 Dependency

 Dependency with ’use’ Stereotype

 Association

 Directed Association

 Directed Aggregation

 Aggregation

 Directed Composition

 Composition

 Association Class

 Comment

 Connect Comment to Element

 Text

 Circle

 Rectangle

 Polygon

 Polyline

 Repaint

 Do layout

 Update layout

 Zoom to 100%

 Zoom to Fit

 Zoom to Selection

10.6. Behavioral Diagrams

Behavioral diagrams are used to show the functionality of the system and include Use Case, Sequence, Collaboration, State Machine, and Activity diagrams.

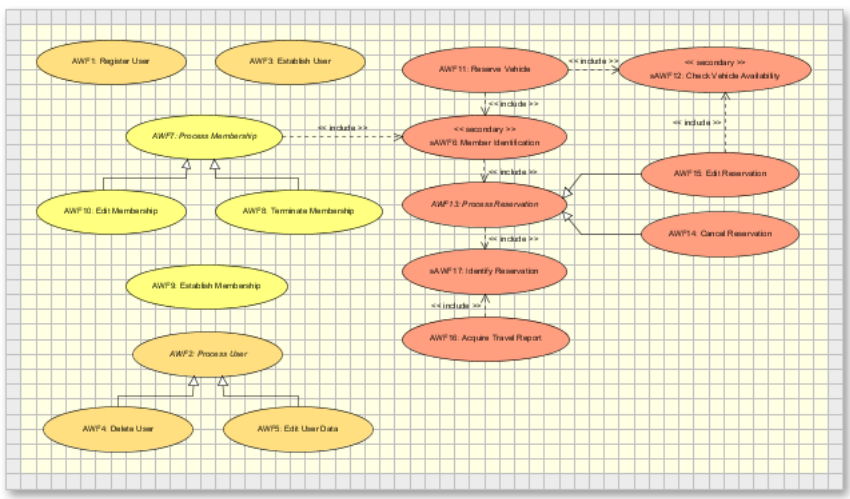
Poseidon limits the inclusion of certain elements to only one instance in one behavioral diagram. As a result of this policy, the context menu option 'Add to Diagram' may be greyed out when a behavioral diagram is active and the selected element has already been represented in one of these diagrams.

10.7. Use Case Diagrams

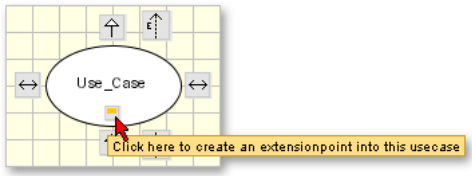
The first diagram to look at is the Use Case diagram. The main ingredients for this type of diagram are *use cases* and *actors* , together they define the *roles* that users can play within a system. They are associated to the tasks, or *use cases* , they are involved in. It is often used in early stages of the design process to collect the intention requirements of a project.

If you are not well-acquainted with UML yet, remember that a use case is not just a bubble noted in the diagram. Along with this bubble, there should be a description of the use case, a typical scenario, exceptional cases, preconditions etc. These can either be expressed in external texts using regular text processing tools, requirements tools or file cards. It can be and is often refined using other diagrams like a sequence diagram or an activity diagram that explain its main scenarios. The basic description of a use case can also be inserted in the Documentation tab of the Details pane.

Figure 10-4. A Use Case diagram.



Extension points can be added to a use case via the extension point button in the Details pane or via the extension point rapid button.



10.7.1. Diagram Elements

- **Actors** - Also referred to as Roles. Name and stereotype of an actor can be changed in its Properties tab.
- **Inheritance** - Refinement relations between actors. This relation can carry a name and a stereotype.
- **Use cases** - These can have Extension Points.
- **Extension Points** - This defines a location where an extension can be added.
- **Associations** - Between roles and use cases. It is useful to give associations speaking names.
- **Dependencies** - Between use cases. Dependencies often have a stereotype to better define the role of the dependency. To select a stereotype, select the dependency from the diagram or the Navigation pane, then change the stereotype in the Properties tab. There are two special kinds of dependencies: <<extend>> and <<include>>, for which Poseidon offers own buttons (see below).
- **Extend relationship** - A uni-directional relationship between two use cases. An extend relationship between use case B and use case A means that the behavior of B *can be* included in A.
- **Include relationship** - A uni-directional relationship between two use cases. Such a relationship between use cases A and B means, that the behavior of B *is always* included in A.
- **System border** - The system border is actually not implemented as model element in Poseidon for UML. You can simply draw a rectangle, send it to the background and use it as system border by putting all corresponding use cases inside the rectangle.

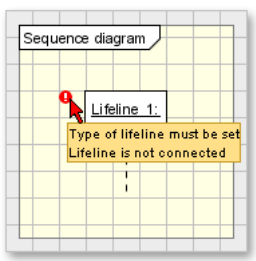
10.7.2. Toolbar

- Select
- Add or remove space between elements
- Package
- Subsystem
- Actor
- Actor as Classifier
- Use Case

- Generalization
- Dependency
- Association
- Directed Association
- Include
- Extend
- Collaboration
- Classifier Role
- Comment
- Connect Comment to Element
- Text
- Ellipse
- Rectangle
- Polygon
- Polyline
- Repaint
- Do layout
- Update layout
- Zoom to 100%
- Zoom to Fit
- Zoom to Selection

10.8. Sequence Diagrams

A sequence diagram is an easily comprehensible visualization of single scenarios or examples of business processes with regard to their behavior in time. It focuses on *when* the individual objects interact with each other during execution. It is particularly useful for modeling usage scenarios such as the logic of methods and the logic of services. Poseidon now makes it even easier to model with the addition of inspections. Whenever Poseidon has a suggestion for your diagram, a red exclamation point will appear near the questionable item. Mouse over the exclamation point to reveal the critique.

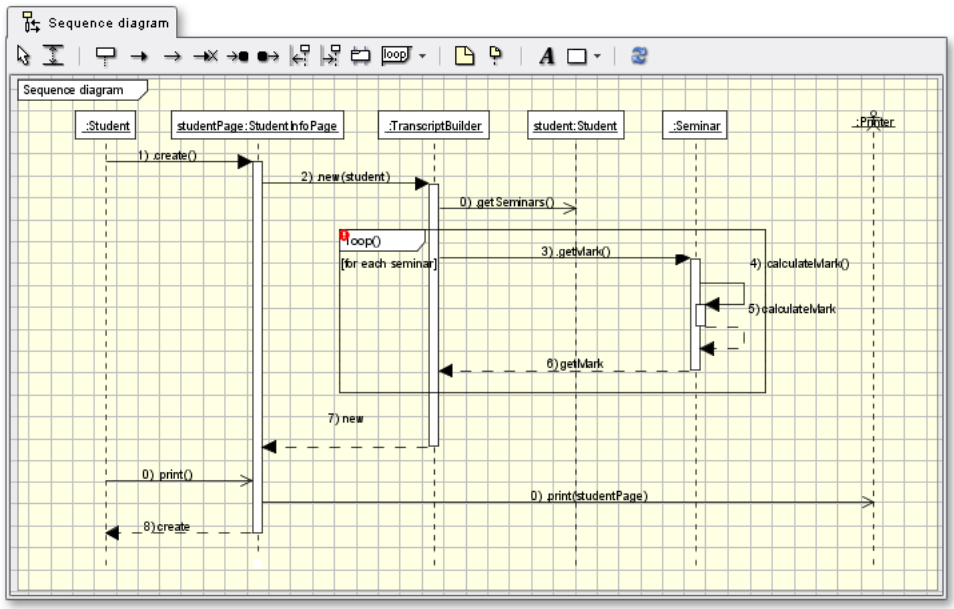


The diagram essentially includes a timeline that flows from the top to the bottom of the diagram and is displayed as a dotted line. The interaction between objects is described by specifying the different kinds of messages sent between them. Messages are called stimuli. They are displayed as arrows; the diverse arrowheads stand for different kinds of messages (see below).

New in UML 2 is the frame surrounding the the diagram with a label in the upper left corner. These frames are also used for interaction occurrences and combined fragments within the diagram itself, such as with a loop or switch statement (called 'alt' in UML).

The following diagram shows a typical example:

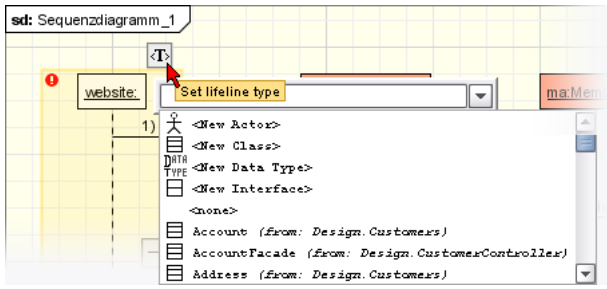
Figure 10-5. A Sequence diagram.



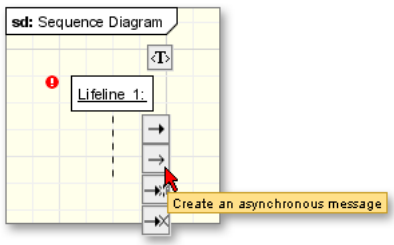
Full-Screen Editing

Many operations necessary to create and edit meaningful Sequence diagrams are available from within the diagram itself. The advantage is that the diagram may be edited in full-screen mode (see Section 11.4.4), providing a much greater diagram viewing area for easier editing.

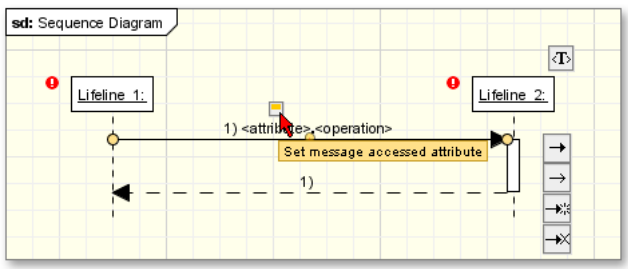
For example, the type of a lifeline can be set:

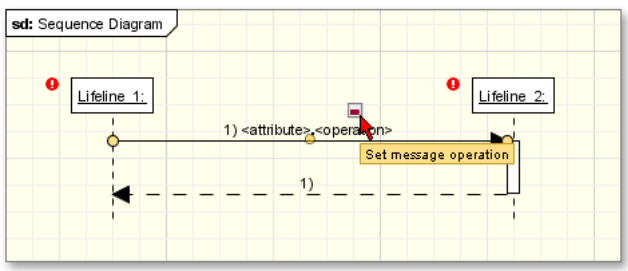


Or messages created:



Or the attributes and operations of messages edited:



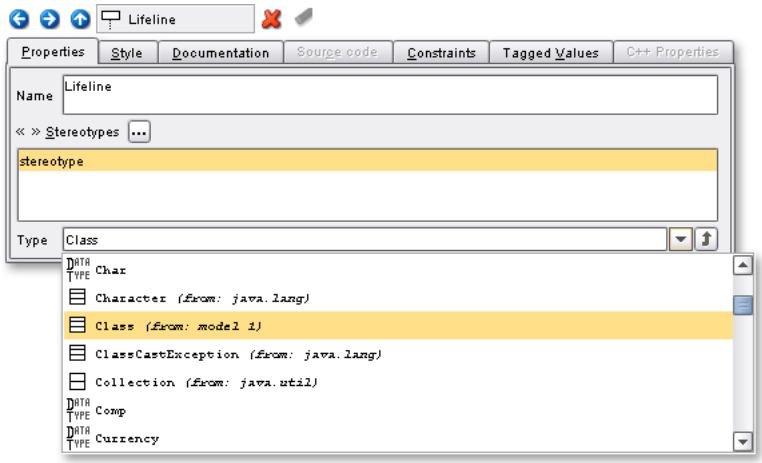


Objects and Lifelines

After creating or changing objects, they are automatically arranged in the Diagram pane. Focus of

control is also automatically rendered by Poseidon. Each object can have a lifeline, which is represented by a dashed line that extends downward from the object. The lifeline is a time axis for the diagram, with time passing as one moves down the lifeline. Focus of control is displayed along this lifeline as a thin rectangle.

An object receives a message and then begins an activity resulting in an execution occurrence. The object now has the focus of control. An activated object is either executing its own code or is waiting for the return of another object to which it has sent a message.

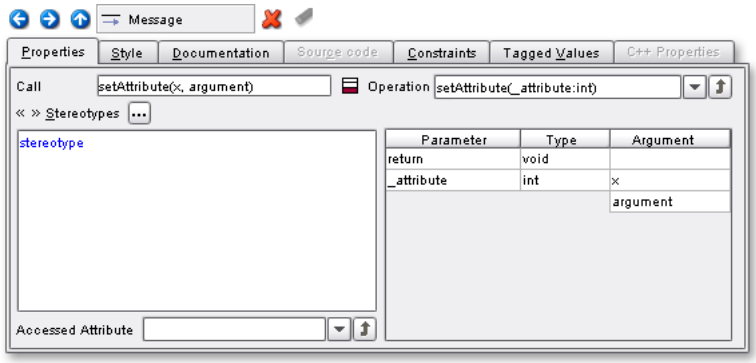


Self messages

In Poseidon, an object can send a message to itself, called a self-message. In the case of a message to itself, the arrow starts and finishes on the object's lifeline. A self stimulus is created by dropping the stimulus target point on the source object.

Selecting an operation

A message can be regarded as a procedure call and can be connected with any operation provided by the receiving object, depending on the type of the receiving object. This is achieved by connecting the message with an action that will cause the class operation to be called.



Try it Yourself - *Assign an operation to a message*

1. Open a class diagram.

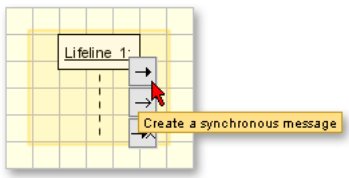
2. Create a new class by double-clicking the 'New Class' button on the toolbar, then click anywhere in the class diagram.

3. With the new class selected, add an attribute by clicking the 'Add Operation' button in the Properties tab toolbar (below the diagram).

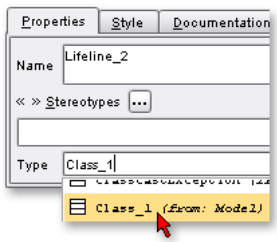
4. Open a sequence diagram.

5. Add an object to the diagram by clicking the 'New Lifeline' button, then click anywhere in the diagram.

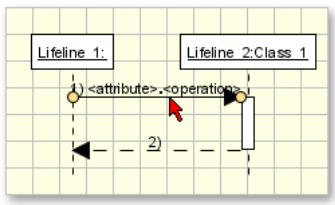
6. Add a message and new object by clicking the 'New Synchronous Message' rapid button.



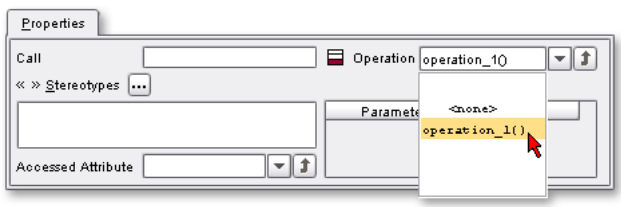
7. Select the new lifeline, then set the type of the lifeline from the Type dropdown by selecting the class created in step 2.



8. Select the message in the diagram.



9. Finally, select 'operation_1' (created in step 3) from the Operation dropdown menu of the message properties tab.



Similarly, a message can access an attribute in the originating object.

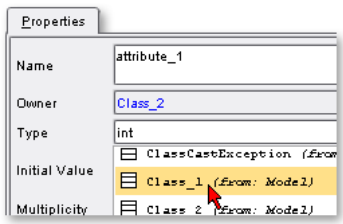
Try it Yourself - *Add an Accessed Attribute to a Message*

1. Open a class diagram.

2. Create two new classes by double-clicking the 'New Class' button on the toolbar, then click twice anywhere in the class diagram. By default, these will be called 'Class_1' and 'Class_2'.

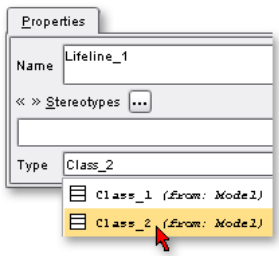
3. With 'Class_2' selected, add an attribute by clicking the 'Add Attribute' button in the Properties tab toolbar (below the diagram).

4. From the resulting attribute properties tab, select 'Class_1' from the Type dropdown.

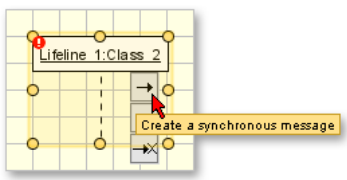


5. Open a sequence diagram.

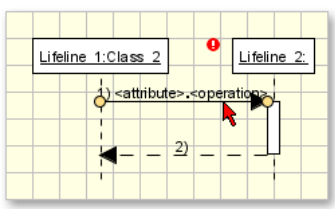
6. Add an object to the diagram by clicking the 'New Lifeline' button, then click anywhere in the diagram. Set the type of this new lifeline by selecting 'Class_2' from the Type dropdown of the lifeline properties tab.



7. Add a message and new object by clicking the 'New Synchronous Message' rapid button.



8. Select the message in the diagram.

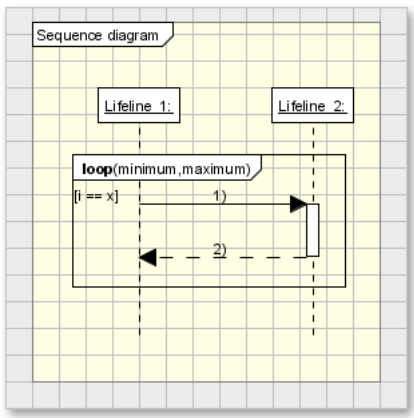


9. Finally, select 'attribute_1' from the Accessed Attribute dropdown menu of the message properties tab.

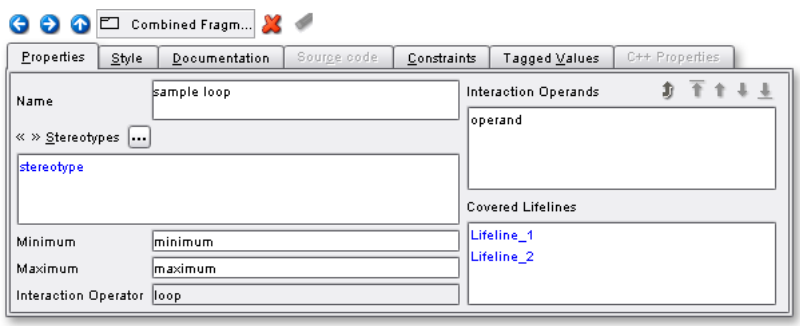
Combined fragments

A fragment is simply a piece of a sequence diagram. These fragments can be combined to form logical units surrounded by a frame, each with one or more interaction operands and an interaction operator. The operators are indicated by the label given in the frame surrouding the fragment. Operands are contained within this frame, and can be separated into different regions with a dashed horizontal line. Frames are created by first selecting the desired operator in the toolbar, then dragging a box around the lifelines and messages to be included within the frame.

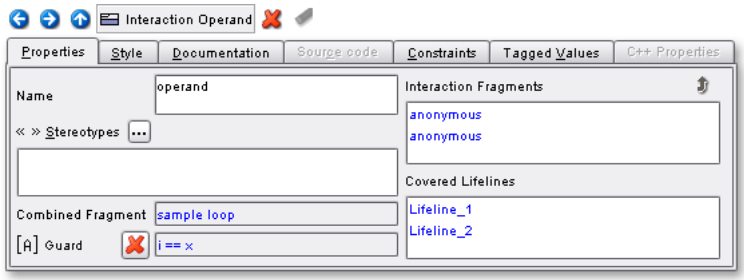
Take, for instance, the loop operator. The loop will execute one fragment a specified number of times, and therefore does not have separate regions. The conditional for the loop can either be a number or a boolean expression.



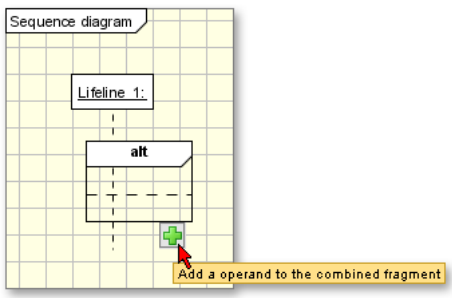
For this combined fragment, the minimum and maximum iterations can be added directly in the properties panel. These are reflected in the fragment label.



Constraints can also be added by navigating to the operand (click on the operand name) and then adding a guard condition.



The 'alt' operator functions like a switch statement, where the conditional (guard) for the first region is evaluated, then the second, and so on through the regions so that at most one of the operands will execute. The final region should contain the 'else' region. In Poseidon, additional regions can be created (and deleted) directly in the diagram with rapid buttons.



Activations

An activation shows the period of time during which an object will perform an action, either directly or through a subordinate procedure. It is represented as a tall thin rectangle with the top aligned with its point of initiation and the bottom aligned with its point of completion.

Now, let's consider how Poseidon deals with starting and terminating activations. When an object receives a stimulus, an activation is created that starts at the tip of the incoming arrow. When an object sends a stimulus, an existing activation is terminated at the tail of the outgoing arrow. There are two exceptions: First, an outgoing send stimulus does not terminate an existing activation, because it represents an asynchronous message. Second, if an object has explicitly set the focus of control, its activation will continue during the whole lifetime.

10.8.1. Diagram Elements

- **Objects** - Elements responsible for sending and receiving messages.

- → **Synchronous Message** - The message sender waits for a response before continuing on.
- → **Asynchronous Message** - Illustrates an asynchronous message, which means that it is regarded as a signal. As such, the sender doesn't wait for an answer from the receiver.

10.8.2. Toolbar


Select


Add or remove space between elements


Add new lifeline


Create synchronous message


Create asynchronous message


Create destroy message


Create lost message


Create found message


Create asynchronous message from lifeline to gate


Create asynchronous message from gate to lifeline


Create an interaction occurrence

Create a combined fragment


Comment


Connect Comment to Element


Add Shape


Repaint


Do layout


Update layout


Zoom to 100%


Zoom to Fit


Zoom to Selection

10.9. Collaboration Diagrams

Collaboration diagrams are another means for representing the interactions and relationships between objects. Unlike sequence diagrams, however, they do not focus on the timeline of interaction, but on the structural connections between collaborating objects. Of central interest are the messages and their intent, when creating a collaboration diagram. The chronological order of messages is represented by numbers preceding each message.

10.9.1. Diagram Elements

- **Objects** - In collaborations, objects represent different roles - these are specified as Classifier Roles in Poseidon for UML.
- **Associations** - Associations illustrate the connections between collaborating objects. Messages are then placed along them.
- **Messages** - Just like in sequence diagrams, messages are used to describe the interaction between objects. The numbers in front of the given names represent the chronological order of messages. Using the corresponding buttons in the toolbar of the Properties tab, you can specify an action for the message, and you can change the direction of the message.

10.9.2. Toolbar

 Select

 Add or remove space between elements

 Object

 Link

 Call Stimulus

 Create Stimulus

 Destroy Stimulus

 Send Stimulus

 Return Stimulus

 Comment

 Connect Comment to Element

 Text

 Circle

 Rectangle

 Polygon

 Polyline

 Repaint

 Do layout

 Update layout

 Zoom to 100%

 Zoom to Fit

 Zoom to Selection

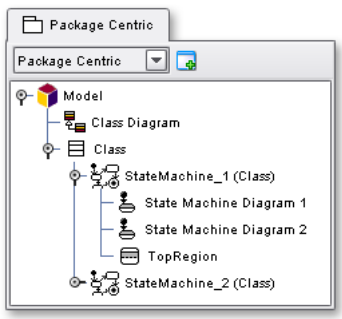
10.10. State Machine Diagrams

Business process models do not lend themselves to implementation in an object-oriented way. If you go the UML way, you will break down the business process and express it in terms of states for each object involved in the process. Numerous other systems, real-time systems for example, also have great need to plan the states of objects. State Machine diagrams (also referred to as State Diagrams) are used to illustrate the possible states of an object and the causes and effects of those state changes within the object.

In UML 2.0, a BehavioredClassifier such as a Class, UseCase, Collaboration, Component or Node may have 0..n state machines. These state machines are displayed in the Package Centric and State Centric views as subnodes of the owning BehavioredClassifier.

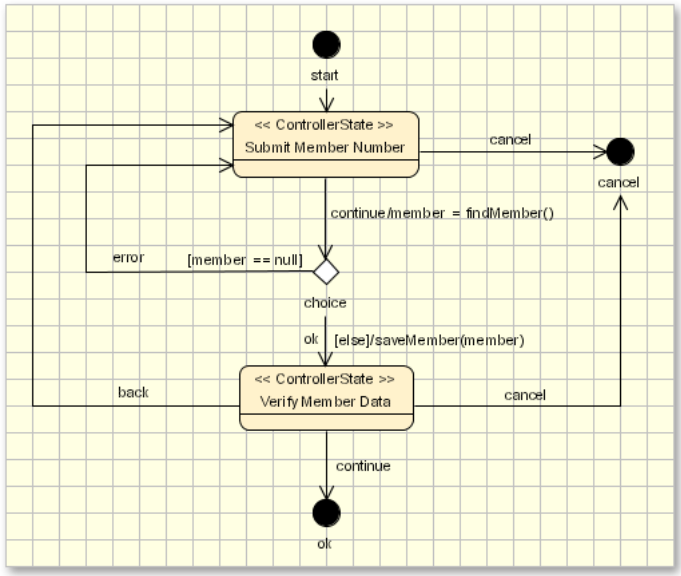
Each state machine may have 0..n state machine diagrams that can depict all or part of the state machine. State machines imported from XMI without related diagram information are added to the model and are accesible from the model navigation tree. State machine diagrams exist in their state machine's namespace and are therefore displayed as subnodes of the state machine.

Figure 10-6. Class with two state machines, State machine with two state machine diagrams.



A state machine in Poseidon has exactly one region, which is referred to as the top region. With UML 2.0, this region replaces the UML 1.4 top-level state. All of the states and transitions of the state machine reside in this top region, regardless of the diagram in which they appear.

Figure 10-7. A State Machine diagram



10.10.1. States

In a state machine diagram, each state has at least two compartments, the top one always keeping the name of the state. The name usually is an adjective describing the recent object.

The states' properties are a lot more meaningful and complex than they are in the activity diagrams. Not only does a state have ingoing and outgoing transitions, but also different activities that are to be taken with it.

Let's take a look at the states themselves. In the diagram toolbar you'll find four different symbols. State types can be distinguished by their regions and connection to submachines:

- **Simple State**

  Simple state has no regions and is not associated to a state machine. The region list and submachine list in the property panel is always empty.

- **Composite State**

  Composite States make visual use of the second compartment that encloses refinements of the given state. Enclosed states don't have to have an initial state. Ingoing as well as outgoing transitions might

be connected directly to one of them. When the corresponding object is in the composite state, it is exactly in one of the sub-states (OR relation).

In a composite state, the second compartment of the state has exactly one region

- **Orthogonal State**

  Orthogonal States are, like the above, refinements; therefore, they are focused in the second compartment. When the corresponding object enters the concurrent state, all initial sub-states all are enabled at once (AND relation).

  An orthogonal state has more than one region in the second compartment of the state. The regions are divided with dashed lines.

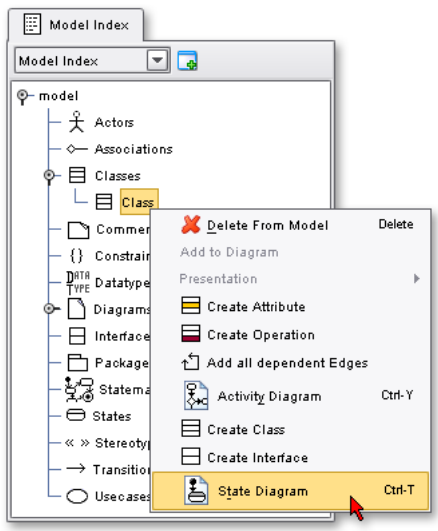
- **Submachine State**

  A submachine state is similar to a simple state in that it has no regions, but it is associated to another state machine.

10.10.2. Creating Diagrams

State Machine diagrams can be created in three ways. The way the state diagram is created and where it is placed in the model depends on the model element currently selected.

1. **With the context menu** - Right click on a BehavioredClassifier, then select 'State Diagram'.

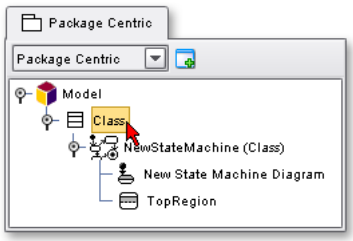


This context menu item is available only for BehavioredClassifiers in the navigation tree and in the diagrams.

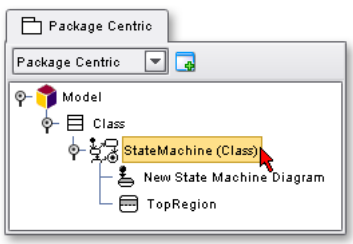
2. **From the toolbar** - Click the 'State Machine Diagram'  button in the toolbar. This option is available regardless of the selected element.
3. **With a quick-key** - Use 'Ctrl-T' to create a new diagram. This option is available regardless of the selected element.

The selected element determines how the state machine diagram is added:

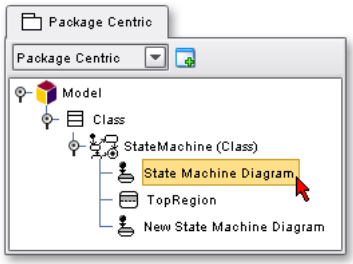
- **BehavioredClassifier (Class, UseCase, Collaboration, Component or Node) selected** - a new StateMachine is added as a subnode to the selected element, and a new diagram is created as a subnode of the new state machine.



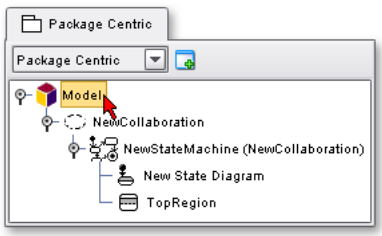
- **State Machine selected** - A new state machine diagram is added as a subnode of the selected StateMachine.



- **State Machine Diagram selected** - A new state machine diagram is added as a subnode to the owning StateMachine of the selected state machine diagram.



- **Any other model element selected** - A new Collaboration is created in the currently selected namespace, a new StateMachine is added as a subnode of the new Collaboration, and a new state diagram is added as a subnode of the new StateMachine.



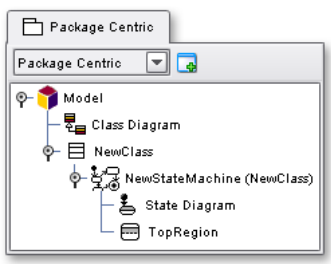
Try it Yourself - *Create A State Machine for a Class*

1. Open a class diagram.

2. Create a new class by clicking the 'New Class' button on the toolbar, then click anywhere in the class diagram.

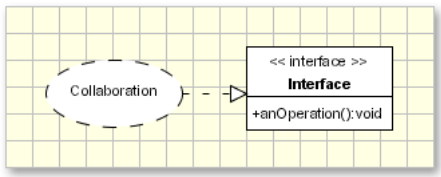
3. With the new class still the selected element, click the 'New State Machine Diagram' button. A new state machine is created for the class, and a state machine diagram is created for the state machine.

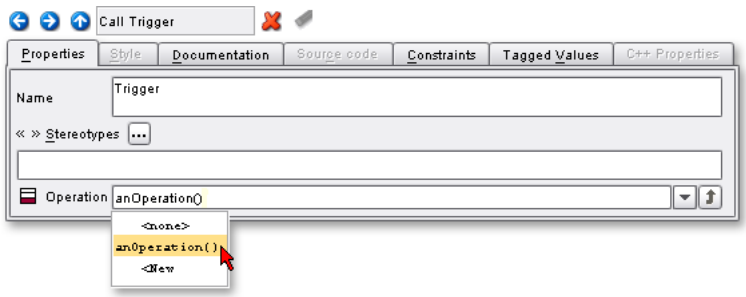
The new state machine is best seen in the 'Package Centric' or 'State Centric' view of the Navigator pane:



State Machines for Elements other than BehavioredClassifiers

Use cases and collaborations are among the BehavioredClassifiers and can thus have their own state machines. But what about other elements? Interfaces are not BehavioredClassifiers in UML 2.0, for example; if an interface needs a state machine, it must first be implemented in a collaboration or class, which can then have state machines. In the following example, the operations of the implemented interface are then available to the triggers of the collaboration's state machine.





Conversion to UML 2.0

The statechart diagrams of UML 1.4 have notations and organizations that are different from the UML 2.0 state machine diagrams. Poseidon 3.1 is the first version of Poseidon for UML to implement the state machine diagram, and it will automatically convert the older statechart diagrams to the new state machine diagrams.

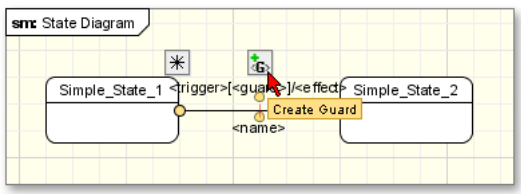
For example, interfaces were allowed to have state machines in UML 1.4, but they are not BehavioredClassifiers and so they are no longer permitted in UML 2.0. In this case, a new collaboration is created and the state machine of the interface is added to this collaboration. The collaboration will get all of the operations specified in the triggers in the state machine. The same conversion is done for Classifier Role, Exception, Subsystem, Signal, and DataType.

10.10.3. Editing Diagrams

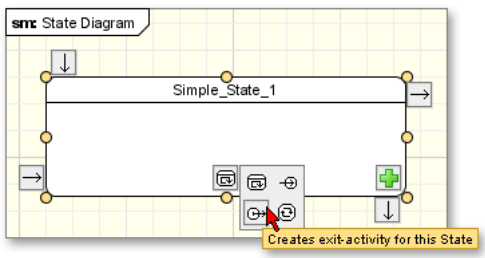
Full-Screen Editing

Rapid buttons and inline editing make it possible to quickly and easily edit diagrams from within the Diagram pane only. The other panes can be hidden to provide a greater work area (see Section 11.4.4).

For example, guards and triggers can be added with a rapid button (effects are edited inline by double-clicking on the effect):



Some rapid buttons make multiple options available through a context menu. In State diagrams, this can be seen in the activities rapid button:

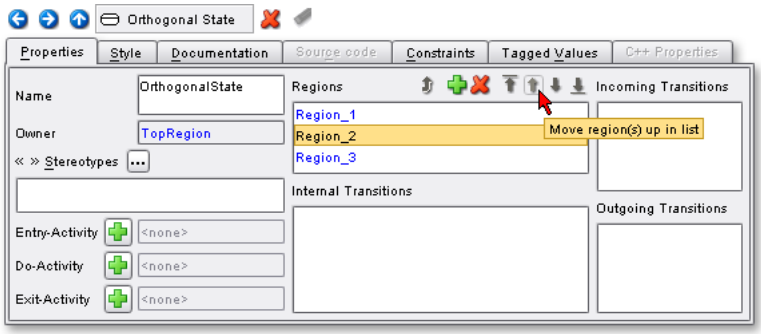


States

For more information about the specific fields, see the Element section Section 12.4 .

- **Region** - A region is a part of a composite state, orthogonal state, or state machine that contains states and transitions.

  In orthogonal states, they are used to indicate concurrency. Reordering regions in orthogonal states in the Property panel reorders them in the diagram display as well.



As of Poseidon 3.1, regions can be added and removed from states using the 'Add' and 'Delete' buttons in the Property panel or rapid buttons. The number of regions determines the type of state, so removing a region from a composite state transforms it into a simple state, and so on.

- **Activities** - Activities are elements in themselves, just like any other part of the model. As such, editing them is done in the same way as any other element - just like an attribute is a part of a class.

  Activities are added from within the state's Property tab using the 'Add' button. Whenever you add an activity, Poseidon will automatically navigate to the new element. Upon return to the parent element, the 'Add' button will no longer be visible, rather it will be the 'Delete' button, indicating that the element has been successfully added.

  To remove an existing element, use the 'Delete' button. Note that this will delete the element entirely from the model.

  In Poseidon 3.1, the 'Asynchronous' option in Actions (now renamed Activities) has been replaced by the tagged value 'isAsynchronous'.

- **Submachines** - A submachine is used to hide or extract details of a state machine. The submachine state is similar to a simple state in that it has no regions, but unlike a simple state, it is associated to a state machine that belongs to the same BehavioredClassifier.

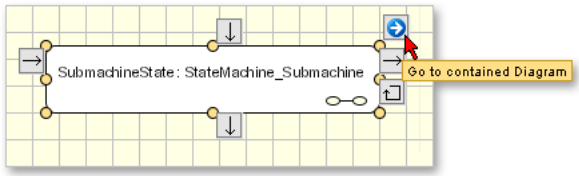
  References are used to indicate the entry and exit points of the submachine, which can optionally be displayed in a state machine diagram. Deleting the reference does not delete the entry and exit points of the submachine, it only removes their display in the upper-level diagram.

  To directly navigate from a submachine to the diagram(s) for the submachine, use the 'Go To Contained Diagram' rapid button.

Try it Yourself - *Create A Submachine with Entry and Exit Points*

1. In a state machine diagram, click the 'New Submachine State' button, then click anywhere in the diagram. A new state machine is created for the submachine and a diagram is created for the new state machine.

2. Navigate to the new diagram by hovering over the submachine state to display the rapid buttons, then click 'Go To Contained Diagram'.



3. Click the 'New Simple State' button in the toolbar, then click anywhere in the submachine state machine diagram to add a state.

4. Click the 'New Entry Point' button in the toolbar, then click anywhere in the submachine state machine diagram.

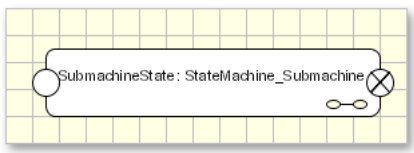
5. Hover over the entry point to activate the rapid buttons, then click and drag the 'Outgoing Transition' button to the simple state.

6. Click the 'New Exit Point' button in the toolbar, then click anywhere in the submachine state machine diagram.

7. Return to the first state machine diagram by clicking its tab at the top of the diagram pane.

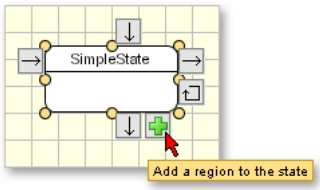
8. Select the submachine state. Connect the entry and exit points by clicking their checkboxes in the property pane of the submachine state.

The submachine state now displays references to entry and exit points:

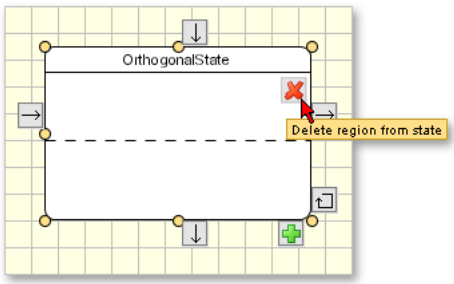


Converting From One State to Another

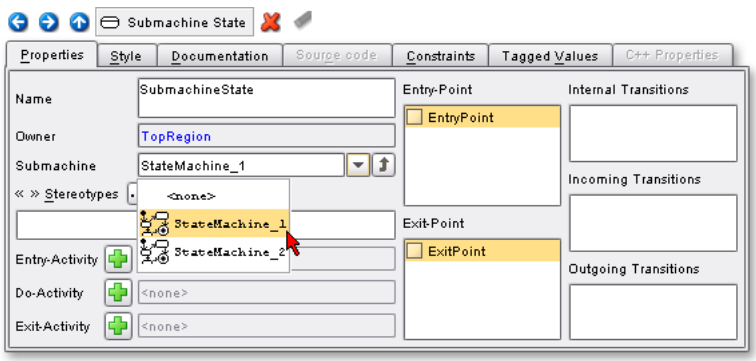
As of Poseidon 3.1, it is possible to convert one type of state to another, e.g. from a simple state to a composite state.



Delete regions to change, for example, an orthogonal state to a composite state or a composite state to a simple state.



Simple states can be converted to submachine states - simply select a submachine from the list of available submachines in the Property panel. Likewise, a submachine state can be converted to a simple state by deleting the association to the state machine. This is done by selecting <none> from the list of state machines.



Transitions

- **Triggers** - If present, a trigger initiates a state change from the source to the target of the transition. A trigger can be a signal, an event, a change of condition, or the passage of time.

  The Property panel of the trigger contains a dropdown list with the available operations for the trigger. These are the operations of the owning BehavioredClassifier, the inherited operations of any

superclasses, or operations of an implemented interface.

- **Guards** - Guards, if present, must be satisfied in order for the transition to occur.

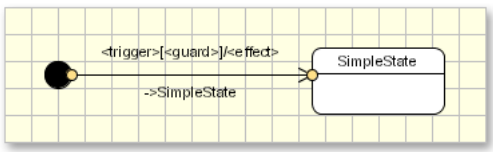
  If the name and expression are empty, nothing will be displayed in the guard portion of the transition label. If the name is present, the name will be displayed, regardless of whether or not there is an expression. If no name is present but the expression is present, the expression will be displayed.

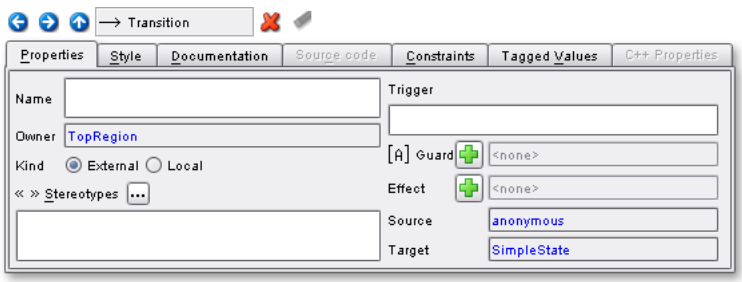
  In UML 2.0, the guard element has been replaced by the constraint element. The guard association remains, however. This change only affects the appearance of the property panel of the guard itself, which now displays a property panel for a constraint.

- **Effects** - An effect is an Activity that will be invoked as a result of the transition.

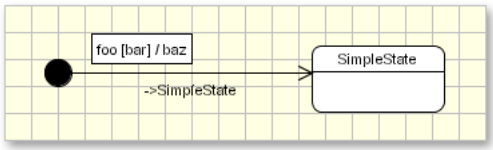
  The effect activity is the same as an entry, do, or exit activity, except that it must be triggered.

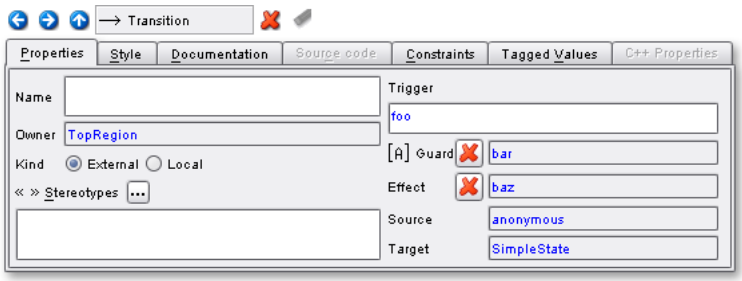
- **Inline Editing** - Transitions can be edited inline as with other diagram elements. When the transition is selected but does not have a trigger, guard, or effect, the diagram and property panel are displayed as such:





Double-click the text above the trigger to enable inline editing. You may then enter any combination of triggers, guards, and events. Text appearing before the square brackets [ ] will be entered as the trigger, text between the brackets as the guard, and text after the slash / as the effect, and will also be available in the Property panel.





Drag and Drop, Copy and Paste

While editing a state machine diagram is similar to editing any other diagram in Poseidon, there are a few exceptions you should be aware of.

- The drag and drop of elements from the navigation panel to the diagram panel is available only for elements of the same state machine.
- Complete state machines can be cut and/or copied to another BehavioredClassifier. However, all operations of the trigger of the state machine and its states are deleted if they do not belong to the new BehavioredClassifier.
- An operation can belong to the new BehavioredClassifier if the new BehavioredClassifier is a parent of the old one.

10.10.4. Diagram Elements

- ● **Initial States** and ◉ **Final States** - Indicate the beginning and end of the observed process.
- ▭ **Action States** - Specific activities that comprise the process. They must be executed in a specified chronological order. Sometimes you may want to split the sequence. Therefore, you have two different possibilities: Branches (choice) and Forks (concurrency).
- ⫪ **Forks** and ⫫ **Joins** - Forks divide the sequence into concurrent sub-sequences. Joins merge the sub-sequences.

- → **Transitions** - The ingredient that keep states active and the model elements together. Each transition can be given *guards* [A], *triggers* ✳, and *activities* A as properties to describe its behavioral details.
- ◇ **Choices** and ● **Junctions** - Both elements are used in sequential systems to define decision points. The difference between them is that choices are dynamic and junctions are static.
- Ⓗ **Shallow History** and Ⓗ* **Deep History** - History states are used to memorize past active states so that you can return to a marked point and don't have to start again from the beginning. A deep history allows you to return from any sub-state, whereas a shallow one only remembers the initial state of a composite state.

10.10.5. Toolbar

 Select

 Add or remove space between elements

 Simple State

 Composite State

 Orthogonal State

 Submachine State

 Transition

 Initial State

 Final State

 Deep History

 Shallow History

 Choice

 Junction

 Entry Point

 Exit Point

 Terminate Point

 Fork

 Join

 Comment

 Connect Comment to Element

 Text

 Circle

 Rectangle

 Polygon

 Polyline

 Repaint

 Do layout

 Update layout

 Zoom to 100%


Zoom to Fit


Zoom to Selection

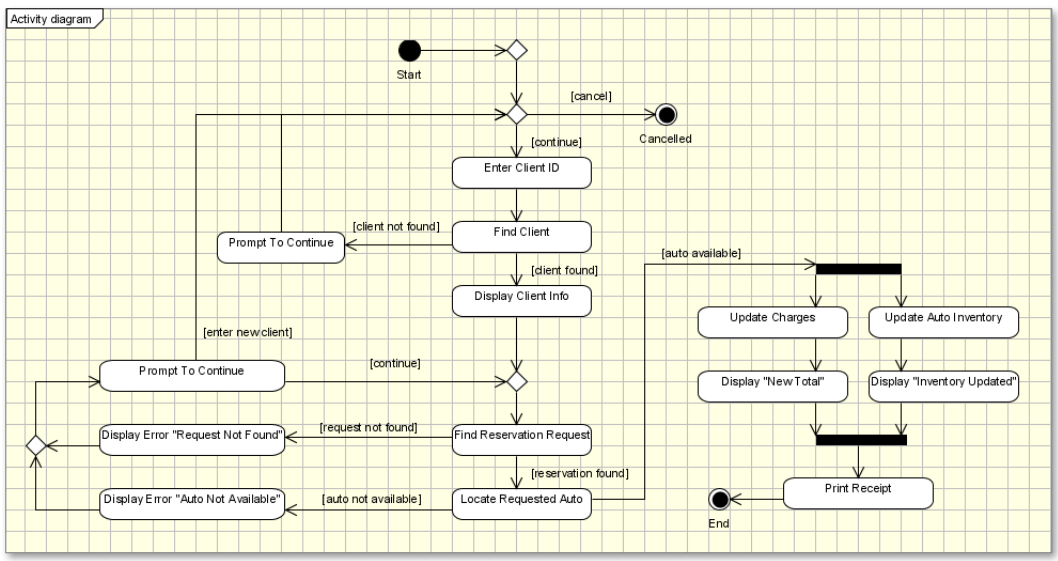
10.11. Activity Diagrams

Activity diagrams are often used to model business processes. They simply and quite plainly show *how things work* , and so function as a good aid to discussions of aspects of the workflow with the domain experts. These are less abstract than the often used object-oriented state machine diagrams.

Activity diagrams are an essential part of the modeling process. They are used to clarify complicated use cases, illustrate control among objects, or to show the logic of an algorithm. The UML 2.0 metamodel substantially refined and improved Activity diagrams; whereas the UML 1.4 Activity diagram was a subclass of State Machine, UML 2.0 defined a separate metamodel, making the diagram much more suited to the job of describing behavior.

The following example shows an activity diagram that depicts the rules and the process of reserving a car. In the following example, `StattAuto` will not accept a reservation if vehicles are not available, etc. Take a closer look for yourself in order to become more familiar with the notation.

Figure 10-8. An Activity diagram.



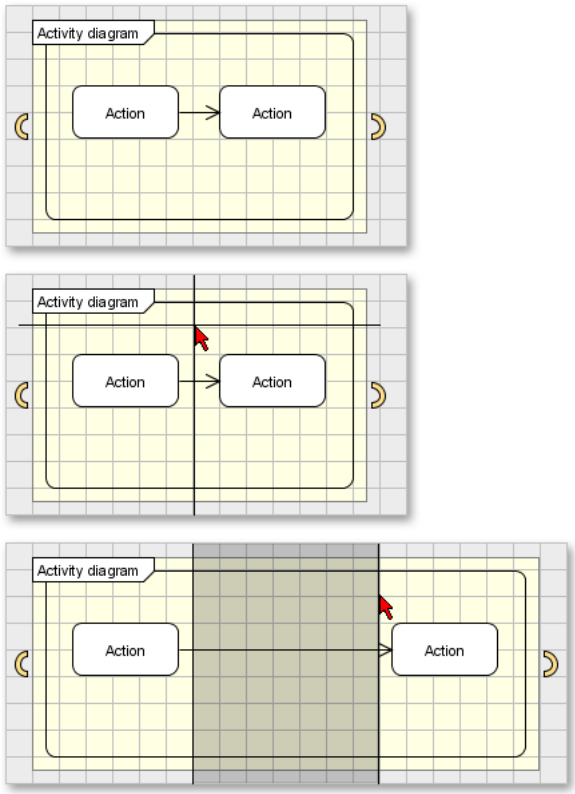
10.11.1. Creating Activity Diagrams

An Activity can have multiple diagrams. An activity diagram belongs to exactly one activity. The activities and diagrams are created differently based upon the element selected:

1. **BehavioredClassifier Selected** - When a BehavioredClassifier such as a class, a collaboration, or a use case is selected, a new activity is created with the BehavioredClassifier as its owner. A new Activity Diagram is then created with the new activity as its parent.
2. **Activity Selected** - When an existing activity is selected, a new Activity Diagram is created with the selected activity as its parent.
3. **CallAction Selected** - When a CallAction is selected, a new activity is created with the same parent as the selected CallAction, and a new Activity Diagram is created for the new activity.
4. **For Any Other Selection** - A new collaboration is created in the namespace of the currently selected element, an activity is created within this collaboration, and an Activity Diagram is created for this activity.

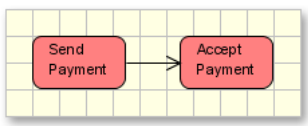
10.11.1.1. Using the Spacer Feature

The spacer increases or decreases the amount of space between elements along a single axis. To use this handy feature, click on the Spacer button in the toolbar, then click and drag where you would like to create whitespace. All elements on the motion side of the cursor will be nudged in the direction of the cursor.

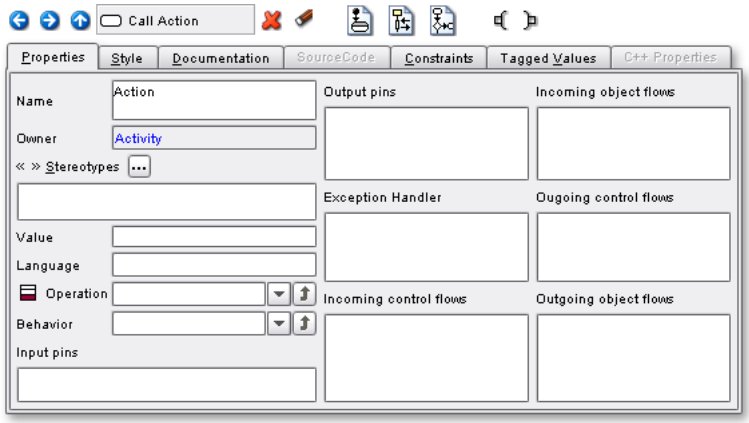


10.11.2. Actions

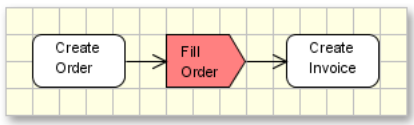
10.11.2.1. Call Action



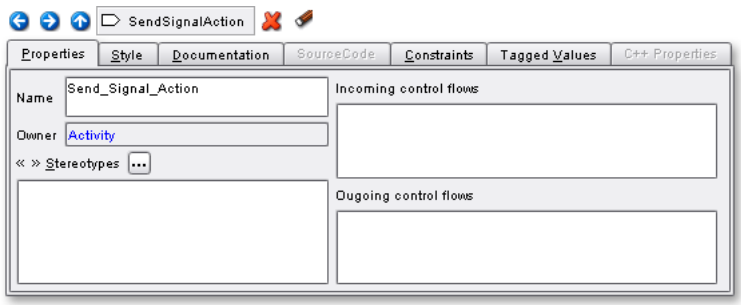
A CallAction is an abstract class for actions that invoke behavior and receive return values. - *OMG UML 2.0 Superstructure Specification p. 259*



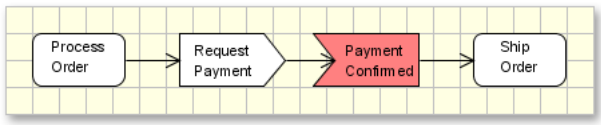
10.11.2.2. Send Signal Action



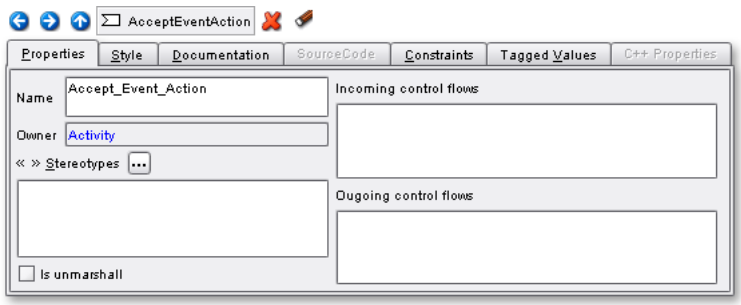
SendSignalAction is an action that creates a signal instance from its inputs, and transmits it to the target object, where it may cause the firing of a state machine transition or the execution of an activity. - *OMG UML 2.0 Superstructure Specification p. 302*



10.11.2.3. Accept Event Action

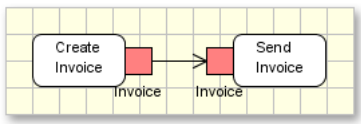


An AcceptEventAction is an action that waits for the occurrence of an event meeting specified conditions. - *OMG UML 2.0 Superstructure Specification p. 249*



10.11.3. Nodes

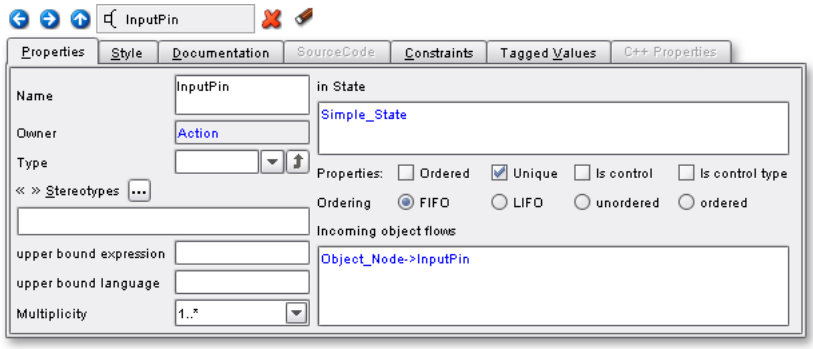
10.11.3.1. Input and Output Pins



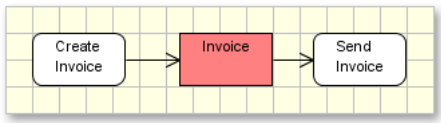
A pin is a typed element and multiplicity element that provides values to actions and accepts result values from them. - *OMG UML 2.0 Superstructure Specification p. 282*

An input pin is a pin that holds input values to be consumed by an action. - *OMG UML 2.0 Superstructure Specification p. 273*

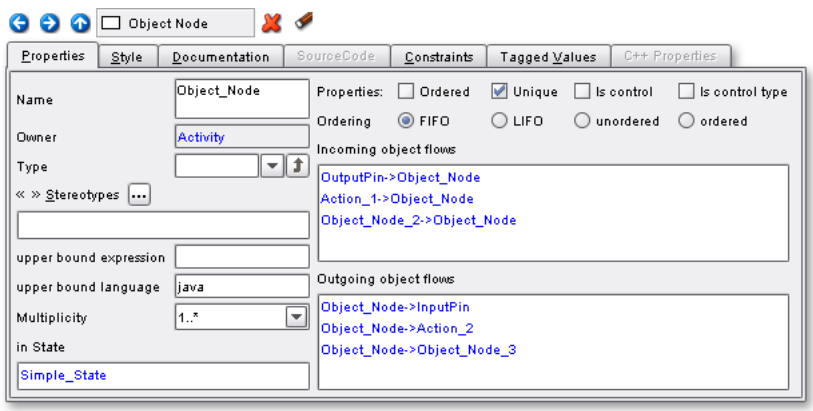
An output pin is a pin that holds output values produced by an action. - *OMG UML 2.0 Superstructure Specification p. 281*



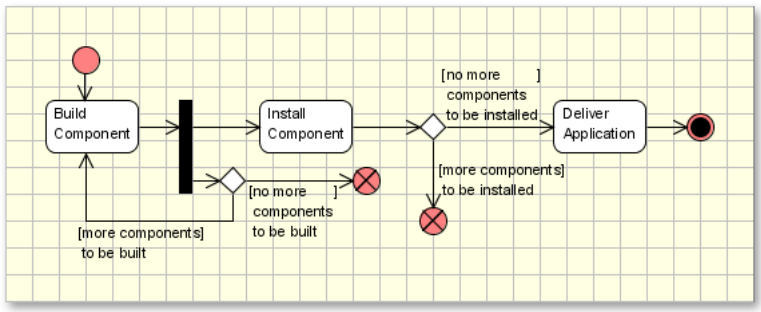
10.11.3.2. Object Node



An object node is an activity node that indicates an instance of a particular classifier, possibly in a particular state, may be available at a particular point in the activity. Object nodes can be used in a variety of ways, depending on where objects are flowing from and to, as described in the semantics section. - *OMG UML 2.0 Superstructure Specification p. 422*



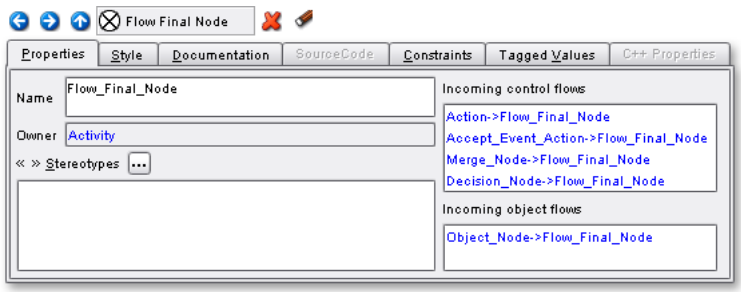
10.11.3.3. Initial, Final Activity, and Final Flow Nodes



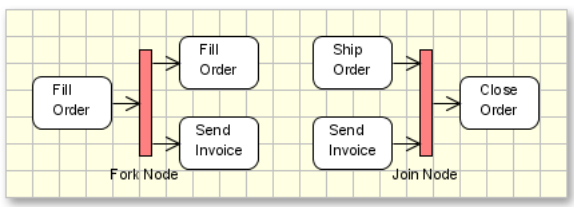
An initial node is a control node at which flow starts when the activity is invoked. - *OMG UML 2.0 Superstructure Specification p. 406*

An activity final node is a final node that stops all flows in an activity. - *OMG UML 2.0 Superstructure Specification p. 356*

A flow final node is a final node that terminates a flow. - *OMG UML 2.0 Superstructure Specification p. 403*

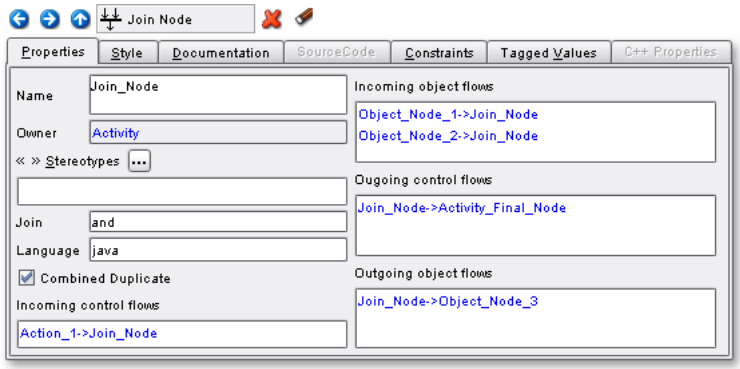


10.11.3.4. Fork and Join Nodes

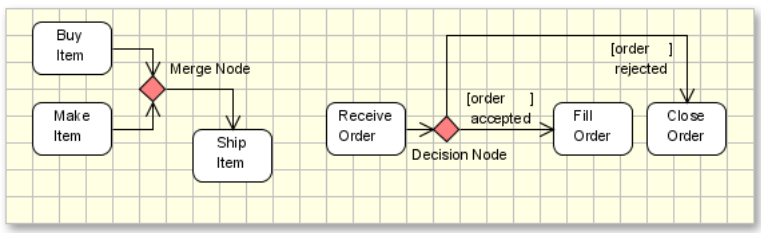


A fork node is a control node that splits a flow into multiple concurrent flows. - *OMG UML 2.0 Superstructure Specification p. 404*

A join node is a control node that synchronizes multiple flows. - *OMG UML 2.0 Superstructure Specification p. 411*

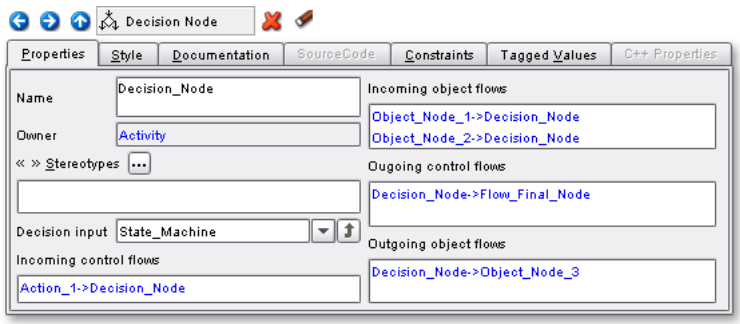


10.11.3.5. Decision and Merge Nodes



A decision node is a control node that chooses between outgoing flows. - *OMG UML 2.0 Superstructure Specification p. 387*

A merge node is a control node that brings together multiple alternate flows. It is not used to synchronize concurrent flows but to accept one among several alternate flows. - *OMG UML 2.0 Superstructure Specification p. 416*



Activity Parameter Node

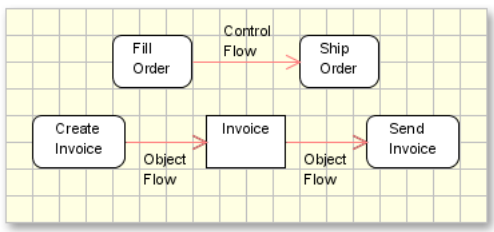
10.11.3.6. Activity Parameter Node

An activity parameter node is an object node for inputs and outputs to activities. - *OMG UML 2.0 Superstructure Specification p. 363*

To create an Activity Parameter Node, simply click the 'Activity Parameter Node' button in the toolbar, then click anywhere in the Activity diagram. The new node will be placed along the edge of the activity, and can be dragged along to the perimeter to the desired location.

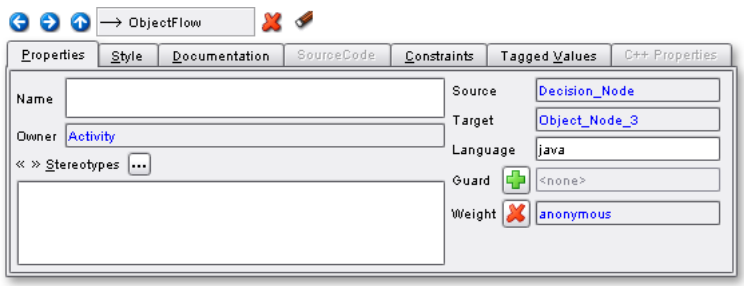
10.11.4. Flow

10.11.4.1. Object and Control Flow



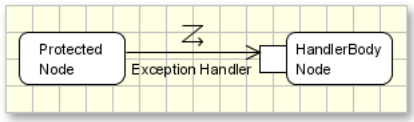
An object flow is an activity edge that can have objects or data passing along it. - *OMG UML 2.0 Superstructure Specification p. 418*

A control flow is an edge that starts an activity node after the previous one is finished. - *OMG UML 2.0 Superstructure Specification p. 382*



10.11.4.2. Exception Handler

An exception handler is an element that specifies a body to execute in case the specified exception occurs during the execution of the protected node. - *OMG UML 2.0 Superstructure Specification p. 390*



To create an exception handler

1. Click the Exception Handler button  in the toolbar
2. Click and drag from an action to another action or an input pin.

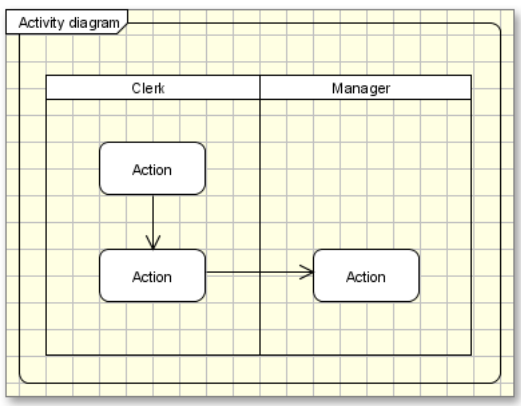
   Alternatively, a new HandlerBody node with input pin can be created by clicking and dragging from the action to wherever the new node should be created.

   In both cases, the resulting exception handler will have the same type as the input pin.

10.11.5. Activity Groups

10.11.5.1. Activity Partitions

An activity partition is a kind of activity group for identifying actions that have some characteristic in common. They often correspond to organizational units in a business model. They may be used to allocate characteristics or resources among the nodes of an activity. - *OMG UML 2.0 Superstructure Specification p. 367*

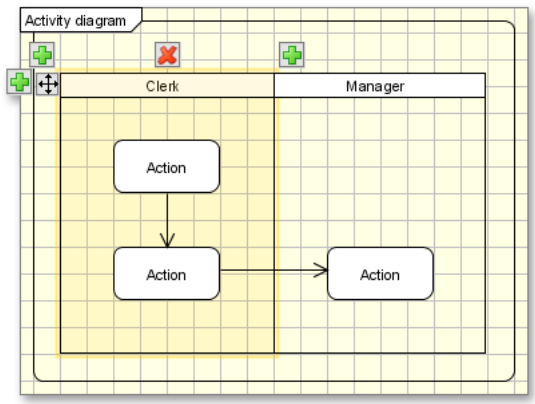


Partitions are the UML 2.0 version of the swimlane functionality that divides and clarifies an activity diagram according to certain characteristics. Quite often, this is used to delineate responsibility between roles.

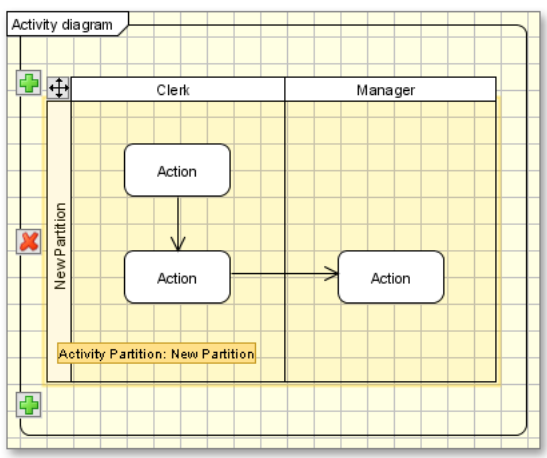
Partitions are not limited in number or orientation; that is, you may have any number of partitions in both the X and Y directions. The above example shows two vertical partitions.

Creating an Activity Partition

1. Click on the Activity Partition button in the toolbar, then click anywhere in the activity diagram. Any existing elements can now be dragged into the appropriate partitions.
2. Partitions can be added or deleted by clicking the associated rapid button.



In this case, clicking the rightmost plus sign will create a new vertical partition between the Clerk and Manager Partitions. The left button will create a vertical partition to the left of Clerk, and the button below will create a horizontal partition:



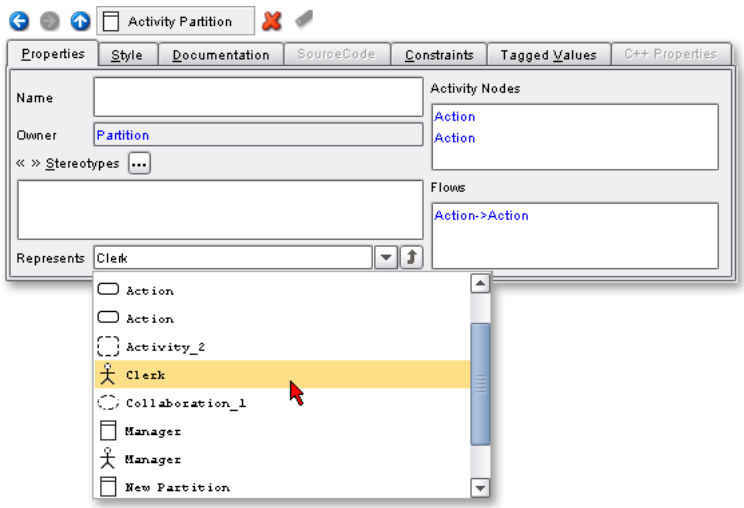
Activity Partition Display

Partitions can be resized just as any other element. Select the partition and use the handles to adjust.

A partition can represent a theoretical category that is not in the model, or it can represent a concrete idea that has already been modeled. In the former case, the name of the partition is important to clearly convey exactly what is being presented in the model. To name a partition, simply enter the desired name in the Properties tab or double-click the name in the diagram to edit inline.

But it will more often be the case that the partition will represent something that is already in the model, such as an actor from a use case diagram. It is more robust and clearer within the model if the display name of the partition is the result of an actual link to that element.

In the Properties tab of the partition, select the appropriate element from the 'Represents' dropdown list. If the name of the partition is empty, the name of this selected element will display. If the name of the partition exists, then the explicit name will display.



Two Dimensional Activity Partitions

Activity Partitions may also be created in two dimensions - that is, both horizontal and vertical partitions may be used at the same time.

10.11.5.2. Interruptable Activity Regions

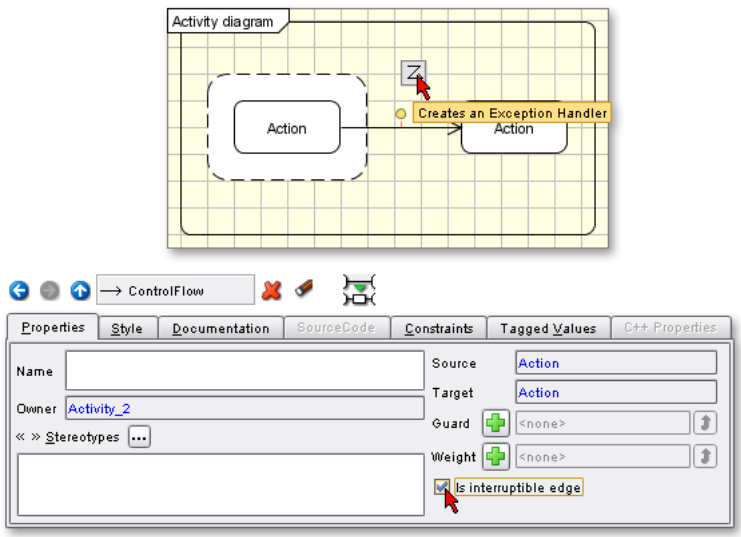
An interruptable region contains activity nodes. When a token leaves an interruptable region via edges designated by the region as interrupting edges, all tokens and behaviors in the region are terminated. - *OMG UML 2.0 Superstructure Specification p. 367*

An interruptable region is denoted with a broken line, just like other activity groups. To create the region, click the 'Interruptable Region' button in the toolbar, then click and drag in the diagram to make the region the appropriate size.

Edges

Two types of edges may cross the boundaries of the interruptable region: 'normal' activity edges that do not terminate other token flows, and 'interruptable' edges that are marked with with an icon similar to the exception handler icon. (This is potentially confusing, but UML 2.0 specifies this notation.)

To mark and edge as interruptable, either use the rapid button that appears when hovering over the edge, or select the 'Is interruptable edge' checkbox in the Properties tab.

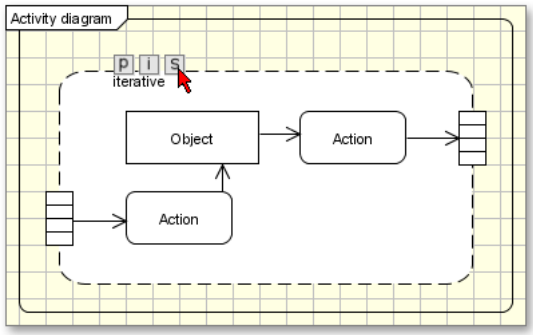


10.11.5.3. Expansion Regions

An expansion region is a structured activity region that executes multiple times corresponding to elements of an input collection. - *OMG UML 2.0 Superstructure Specification p. 395*

To create an expansion region, select the 'Expansion Region' button from the toolbar, then click in the diagram. The size of the expansion region can be adjusted with the handles.

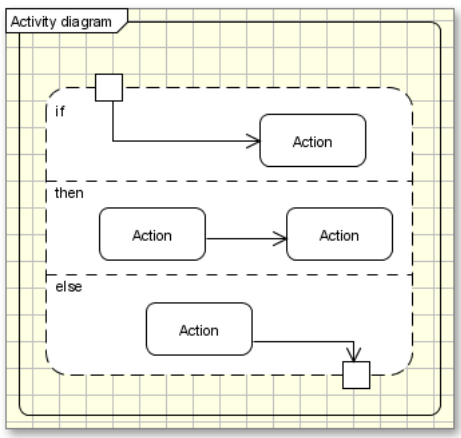
Each expansion region has a mode, which can be changed using the rapid buttons or by adjusting the radio buttons in the Properties tab. Rapid buttons also exist to create the expansion nodes, which are essentially specialized input and output pins.



10.11.5.4. Conditional Nodes

A conditional node is a structured activity that represents an exclusive choice among some number of alternatives. - *OMG UML 2.0 Superstructure Specification p. 380*

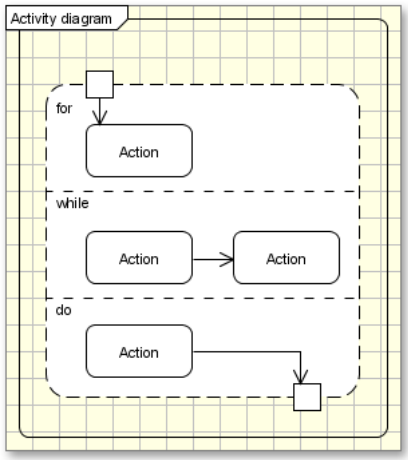
Conditional nodes convey 'if - then - else' logic. To create a conditional node, select the 'Conditional Node' button from the toolbar, then click in the diagram. The size of the conditional node can be adjusted with the handles.



10.11.5.5. Loop Node

A loop node is a structured activity that represents a loop with setup, test, and body sections. - *OMG UML 2.0 Superstructure Specification p. 414*

The loop node models a 'for - while - do' loop. To create a loop node, select the 'Loop Node' button from the toolbar, then click in the diagram. The size of the loop node can be adjusted with the handles.



10.11.6. Diagram Elements

- **Activities** - Specification of a parameterized sequence of behavior.
- **Initial States** and **Final States** - Indicate the beginning and end of the observed process.
- **Action States** - Specific activities which comprise the process. They must be executed in a specified chronological order. Sometimes you may want to split the sequence; therefore, you have two different possibilities: Branches (choice) and Forks (concurrency).
- **Branches** - These divide the sequence into several alternatives specified by different conditions (guards).
- **Forks** and **Joins** - Forks divide the sequence into concurrent sub-sequences. Joins merge the sub-sequences.
- **Transitions** - The ingredient that keep states active and the model elements together. Each transition can be given *guards* [A], *triggers* ✳, and *actions* as properties to describe its behavioral details.
- **Object Flow States** - Objects are inputs or outputs of activities and are accordingly connected by transitions to them.
- **Dependencies** - Always possible between any model elements.

10.11.7. Toolbar

- Select
- Add or remove space between elements
- Call Action
- Send Signal Action
- Accept Event Action

 Object Node

 Activity Edge

 Exception Handler

 Initial Node

 Activity Final Node

 Flow Final Node

 Fork Node

 Join Node

 Decision Node

 Merge Node

 Activity Parameter Node

 Vertical Activity Partition

 Horizontal Activity Partition

 Matrix Activity Partition

 Conditional Node

 Loop Node

 Structured Activity Node

 Expansion Region

 Interruptable Activity Region

 Comment

 Connect Comment to Element

 Text

 Shape

 Repaint

 Do layout

 Update layout

 Zoom to 100%

 Zoom to Fit

 Zoom to Selection

Chapter 11. Using Diagrams

UML is a graphical language; therefore, from a user's perspective at least, the most important part of a UML tool is the graphical editor. This chapter introduces the general features of the diagram editor that are available for all or most of the diagram types, then takes a detailed look at the graphical editor and explains Poseidon's most important functionalities for editing diagrams.

11.1. Creating New Diagrams

Creating new diagrams is the core of creating new models. After all, it is the diagrams that communicate the design. With Poseidon for UML, generating new diagrams is a very simple process.

Diagrams are considered model elements themselves; therefore, you must decide where the diagram will fit into the hierarchy of the model before you create the diagram. The Package Centric view of the Navigation pane displays the distinct hierarchy. New diagrams are created in the topmost package of this hierarchy by default, but you can also create new diagrams for a specific package. If you select a specific package and then create a new diagram, the diagram will be created within that package. If anything else is selected in the Navigation pane, the new diagram will be created in the topmost package.

There are two ways to create a new diagram. The first is through the main toolbar. Simply click one of the create diagram buttons. The new diagram will be placed in the navigation tree to the left. Where it is placed depends on what was selected in the Navigation pane prior to the creation of the new diagram. By default, new diagrams are placed in the top level of the model, which can be easily seen in the package centric view. A diagram can be created elsewhere by first selecting the package in which it should be placed, then clicking the create button.

Some diagrams are specific to certain model elements. State and Activity diagrams, for example, are used to design the details of a class or a use case. Such a diagram needs to be associated with a class or a use case. To do so, you need to select the class or use case prior creating the new state or activity diagram. Notice that this association is fixed and cannot be changed later.

New diagrams can be created in several ways:

- **Main Toolbar** - Click the appropriate button for the corresponding diagram type.
- **Main Menu** - Select the diagram type from the 'Create Diagram' menu in the main menu.
- **Quick-Key Combinations** - Use these shortcuts to create a new diagram:

  - Class Diagram - *Ctrl-L*
  - Collaboration Diagram - *Ctrl-B*
  - Deployment / Object / Component Diagram - *Ctrl-D*

- Sequence Diagram - *Ctrl-Q*
- State Machine Diagram - *Ctrl-T*
- Activity Diagram - *Ctrl-Y*
- Use Case Diagram - *Ctrl-U*

11.1.1. Cloning Diagrams

Beginning with Poseidon for UML version 3.1, entire diagrams can be cloned. This is extremely useful when a diagram gets very busy or cluttered and you would like to break down the diagram into several diagrams. You can then clone the big diagram and remove (but not delete) elements that are not essential to the smaller diagram. Cloning creates new representations of all of the elements in the diagram, it does not duplicate the elements themselves.

Take, for example, a class diagram with a single class. When this diagram is cloned, there are now two representations of a single class element, each in a different diagram. If the name of the class is changed in one diagram, it is also automatically changed in the other, as well as in the navigation tree.

This should not be confused with the complete copy function, which makes a duplicate of the element itself, resulting in two separate and unconnected elements. Complete copy is only available for elements, not for entire diagrams. It is possible, however, to select all of the elements in a single diagram, complete copy them, and then paste them into a new diagram. In this case, changing the name of an element in the new diagram will not affect the element in the original diagram.

To clone a diagram:

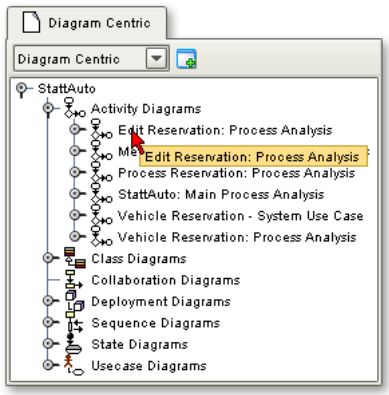
To clone a diagram, select the diagram and then click the 'Clone Diagram'  button in the toolbar. A new diagram will be created with the name 'Copy of <diagram name>'.

Sequence Diagrams

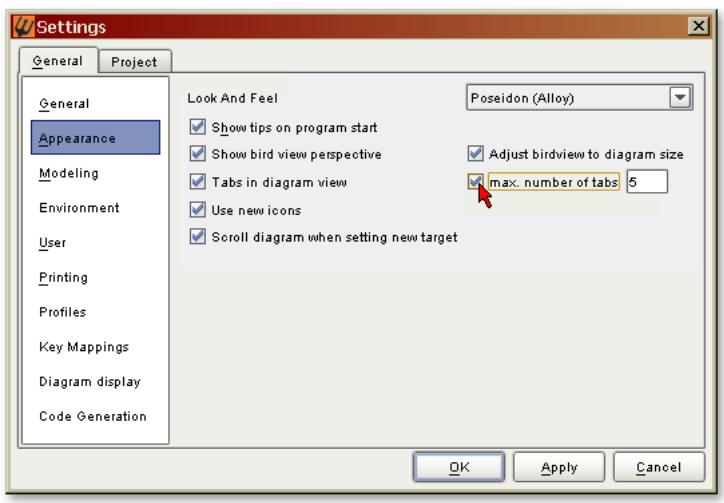
Cloning diagrams is not available for sequence diagrams in version 3.1. However, it is still possible to effectively clone a sequence diagram. Select the containing collaboration of the sequence diagram you would like to clone, then click the 'Copy Complete'  button. Select any other diagram in the model and click the 'Paste'  button. A clone of the sequence diagram will be added to the collaboration. If the model is selected when you click 'Paste', a representation copy of the entire collaboration and all of its elements and diagrams will be added to the model.

11.2. Opening Diagrams

All existing diagrams are listed in the Navigation pane. To open one of these diagrams, simply click on the name of the diagram. The diagram will open in its own tab in the diagram pane to the right.



The number of diagrams which can be open at one time is set to 5 by default. This number can be changed in the Appearance tab of the Settings dialog. Unchecking the 'max number of tabs' box removes any limits to the number of tabs.



11.3. Viewing Diagrams

Viewing a single diagram is easy. You simply select the diagraLm you wish to view from the Navigation

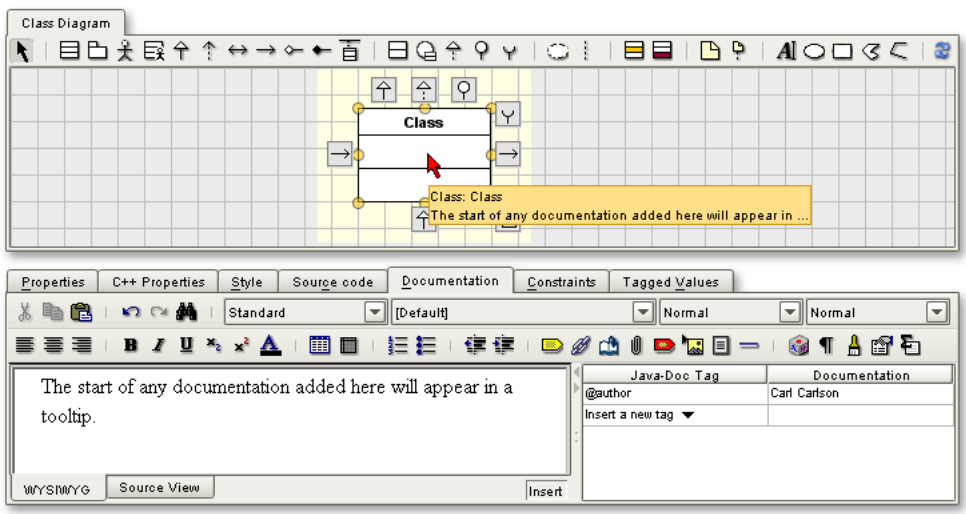
pane and the chosen diagram is then displayed in the Diagram pane. Much more interesting are the relationships between elements and how specific elements are represented in different diagrams. Each element contributes to the overall picture of the model. It may occur in only one diagram, or it may be repeated throughout many diagrams, or perhaps it does not appear in any diagrams at all. The element remains constant throughout the model, with the same characteristics and properties. The only differences it may have from one diagram to another are in the way it is rendered, such as color and compartment visibility.

The yellow field behind the diagram elements indicates the actual size of the diagram. This is important when printing or exporting graphics. The size and shape of this field will change automatically when moving or adding elements.

Select individual classes or associations in a diagram by single-clicking on them. Note how they are simultaneously selected in the Navigation and Details panes.

Hovering with the mouse over any element in the Diagram pane will display the beginning of any documentation that has been entered for that element.

Figure 11-1. Tooltip displaying documentation



The model can be changed directly from the diagram. For example, double-click on the class name of any class in a class diagram. The text field now slightly changes its look and becomes editable. Changing the class name here perpetuates the name in the model everywhere this model element is used. Most

name fields accept multiline text; therefore, the Enter key will add a newline. To commit a change to such a field, use Ctrl-Enter in place of Enter, or simply click elsewhere in the application.

You can also select and change attributes or operations. You need to be aware, though, that in this case you are not simply editing an ordinary text field, you are editing text rendered from a number of model elements. As such, your changes will be propagated throughout the model. Poseidon for UML provides quite powerful parsers that allow you to change these directly by changing the text lines. This is referred to as in-place editing. If you are familiar with the notation used in UML, you can edit almost all of these directly in place.

Though most textual elements can be edited directly in place, another option for elements that are not so easily edited in the diagram is to use context-sensitive menus, which you call up by means of a right-click. In associations, for example, most elements are changeable through Context menus.

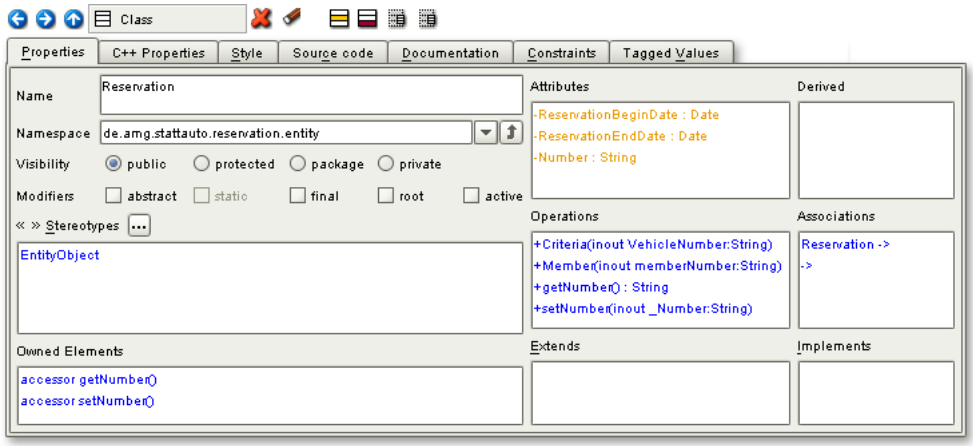
There are, however, some details that can conceptually not be changed in place or where it makes more sense to provide a special graphical user interface. It is for these purposes that the Details pane is used.

11.3.1. Details Pane

To explore the facets of the diagram, you can select elements with the select tool and delve deeper into them. Each time you make a new selection in a diagram, the Details pane (bottom right) is updated and shows specific information for the selected element, as has been previously mentioned. Within this pane, the Properties tab will be selected by default. It contains all relevant details of the selected model element and also displays links to other directly related model elements.

The Properties tab of Poseidon for UML has some similarities with an internet browser. And in a way, a UML model is very similar to hypertext. It is highly connected and navigation between the connected elements is important. All relations to other model elements function as a link to the corresponding Properties tab. Like a browser, this navigation has a history that can be accessed using the ⇨ forward and ⇦ back buttons. Since a model is also hierarchical, there is an ⇧ up button to access the element at the next higher level. For a class this could be the package or namespace to which it belongs, for example.

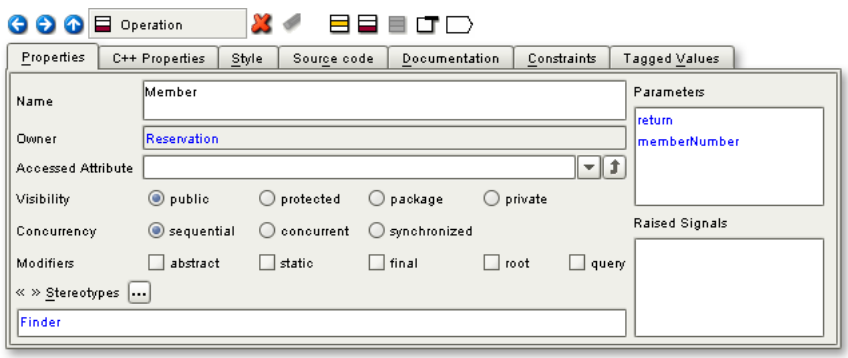
Figure 11-2. Properties tab displaying class 'Reservation'



Open the diagram 'Component Model Overview' and select the class 'Reservation'. Take a look at some of the fields, like `Associations` , `Operations` and `Attributes` . All entries in these text fields work like links in hypertext, which means that clicking on these links allows you to navigate to the related model elements. You can navigate from one class to its associations, operations or attributes and easily access their properties too. Of course, this kind of navigation works in both directions: e.g. from a class to its operations and back.

Now let's move to one of the operations of this class. Click the `Member` operation and have a look at its properties.

Figure 11-3. Properties tab with operation 'Member' selected.



Take an even closer look at the parameters of `Member` . The parameters have properties themselve; therefore, they have their own Properties tab, too.

Click on the parameter `return` . The UML specification treats return types as special parameters; thus, every operation has a return parameter that set to `void` by default. This type can be changed to any other type.

You should now be able to comfortably navigate through the model with the up, back, and forward buttons of the Properties tab toolbar, which is again similar to a hypertext browser.

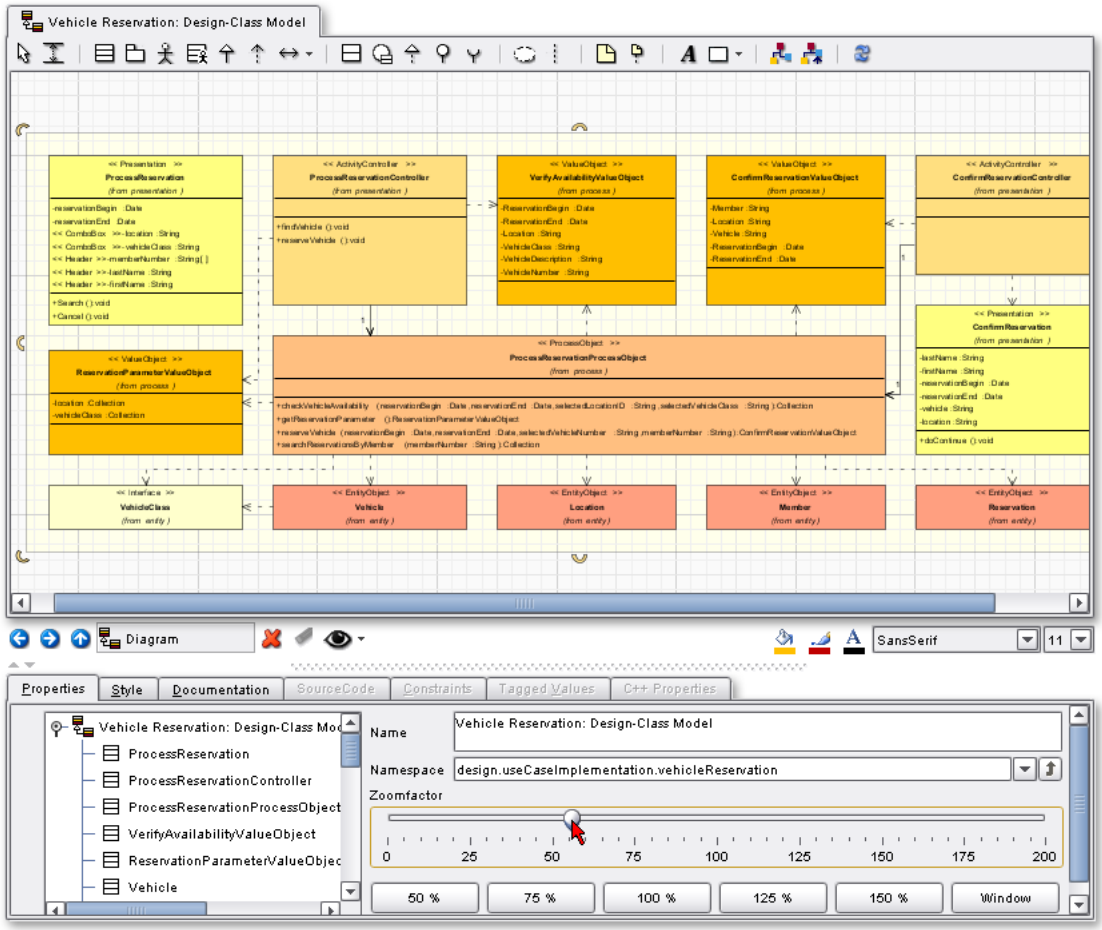
11.3.2. Zooming

The zoom factor is a property of the diagram. Your diagrams might get too large to fit completely into the visible part of the screen. In this case you will want to zoom out to get a better overview. Or you might want to zoom in on some specific part of a model to increase readability, for example during a presentation using a projector.

There are several ways to zoom in and out:

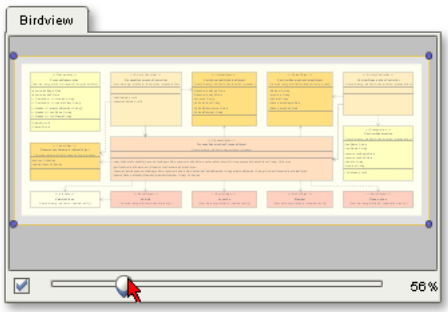
- Change the zoom factor of a diagram by clicking on an empty space in the diagram and using the slider (or the buttons with predefined zoom values) on the Properties tab in the Details pane.

Figure 11-4. Zooming by changing the properties of a diagram.



- Use the slider in the Birdview tab of the Overview pane. The checkbox indicates which pane is affected - unchecked means the birdview can be zoomed, checked means the diagram in the Diagram pane can be zoomed.

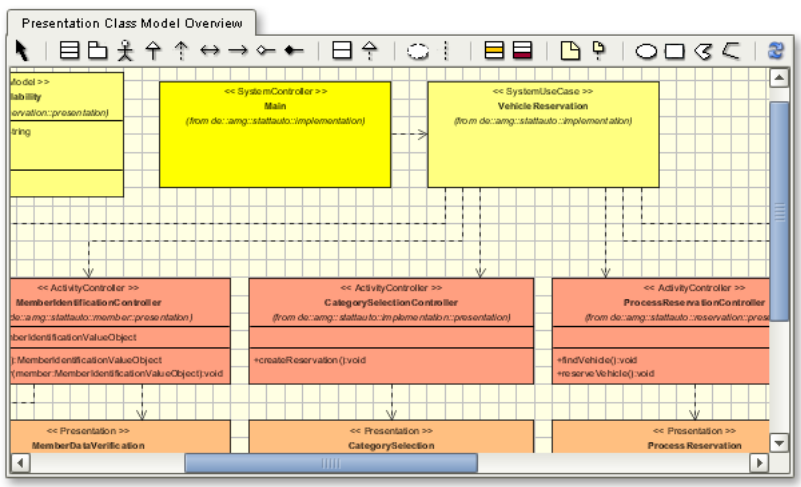
Figure 11-5. Zooming from the Birdview tab



- Hold down the Ctrl key and use the mouse wheel to zoom in and out.
- Choose a zoom factor from the menu (View | Zoom).

11.3.3. Scrolling

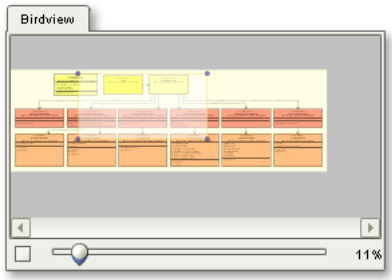
As diagrams get bigger and bigger, the scroll bars become more and more useful.



You can use Shift and the mousewheel to scroll horizontally through a diagram as well.

11.3.4. Birdview Tab

The Birdview tab displays an overview of the entire diagram. The portion of the diagram visible in the Diagram pane is highlighted and has blue handles around the edges. You can redisplay parts of the diagram by dragging the highlight box over different areas of the diagram.



11.4. Editing Diagrams

11.4.1. Adding Elements

There are two methods for placing new elements within a diagram: through the Diagram pane toolbar and through the rapid buttons. The toolbar contains miniature representations of all of the elements available in that particular diagram. Adding elements to a diagram in this manner is very straightforward, simply click on the element in the toolbar and then click in the diagram workspace. Creating elements through the rapid buttons is not only quick (as the name implies), but also has the advantage of creating a relationship to the new element from this one step.

11.4.2. Editing Elements

Perhaps the simplest way to edit an element is to edit it directly in the diagram. This is known as Inline Editing. Double-click on the aspect of the element that you would like to change, and the characteristic will be editable in a text box.

You can also edit an element in the Diagram pane through the context menus. Right-click on the element or characteristic to display the context menu to see what is editable from this menu for the particular element.

Some characteristics, however, are available for editing only from the Details pane. Open the Details pane for an element by selecting it from the Diagram pane or the Navigation pane. Navigate to the desired characteristic (such as a return type for a class operation) by double-clicking on the characteristic

in the left side of the Properties tab. Some of the characteristics may require navigating through several layers of characteristics. The Properties tab also provides navigation buttons which function similar to a web browser.

11.4.3. Deleting Elements

The are two ways to delete items from a diagram, but each works in a slightly different way.

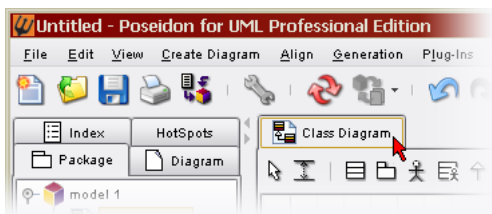
The delete function completely removes the element from the project. All occurrences of this element are deleted, whether they appear in the current diagram or not.

The remove function removes the element from the current diagram only. The element remains available in other diagrams and in the Navigation pane.

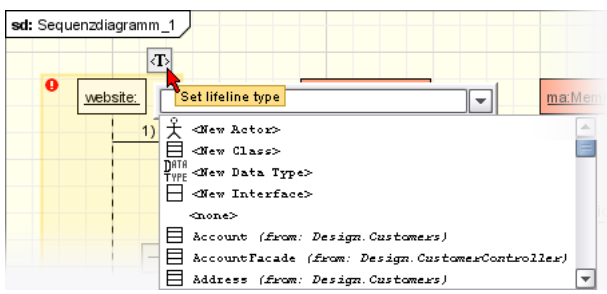
11.4.4. Full-Screen Editing

Many operations necessary to create and edit meaningful diagrams are available from within the diagram itself. The advantage is that the diagram may be edited in full-screen mode, providing a much greater diagram viewing area for easier editing.

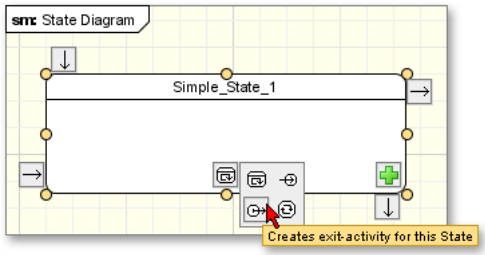
To quickly switch between full-screen editing with only the Diagram pane visible, double-click the diagram tab. You can always return to normal editing with all four panes by double-clicking again.



In a Sequence diagram, for example, the type of a lifeline can be set:



Some rapid buttons also have multiple options, which can be accessed through the context menu of the rapid button. For example, a variety of activities can be added to a state in State diagrams:



11.4.5. Drag and Drop

Some diagrams will be created solely from new elements. But sometimes you will want to use elements that already exist in the model. You just want to present them in a different context and show other specific aspects of its role in the overall architecture.

To do this, you can drag existing elements from the Navigation pane and drop them in the diagram. These elements will appear with all currently known associations to other elements already present in the diagram. Notice that this function will copy the representation of the element to the new diagram, it does not copy the element itself. See Section 13.2.7 for more information about the 'copy representation' and 'complete copy' functions.

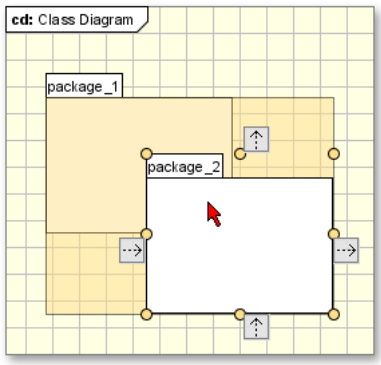
Another possibility is to select elements in a different diagram, copy them by hitting Ctrl-Shift-C and paste them into your new diagram by hitting Ctrl-V. To cut elements from a diagram, use Ctrl-Shift-X. Of course, you can also use these features via the Edit menu or the Context menu.

Note that Drag and Drop is not currently available in all diagrams.

11.4.5.1. Drag and Drop Elements within other Elements

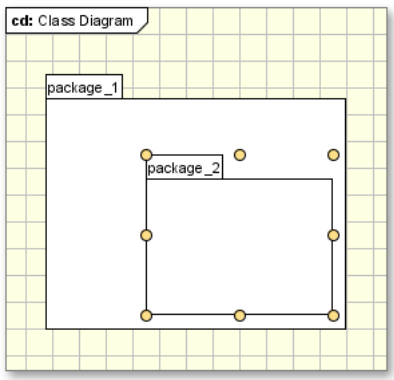
Drag and Drop can also be used to change element namespaces. This can be accomplished by selecting an element in the diagram and dragging it inside the parent element. When the cursor has crossed into the parent element, a preview of the resize is displayed and the element may now be dropped into the parent element.

Figure 11-6. Drag and Drop with preview



After the element has been dropped, the parent element will automatically resize to accomodate the new element, and the properties and navigation tree are immediately updated.

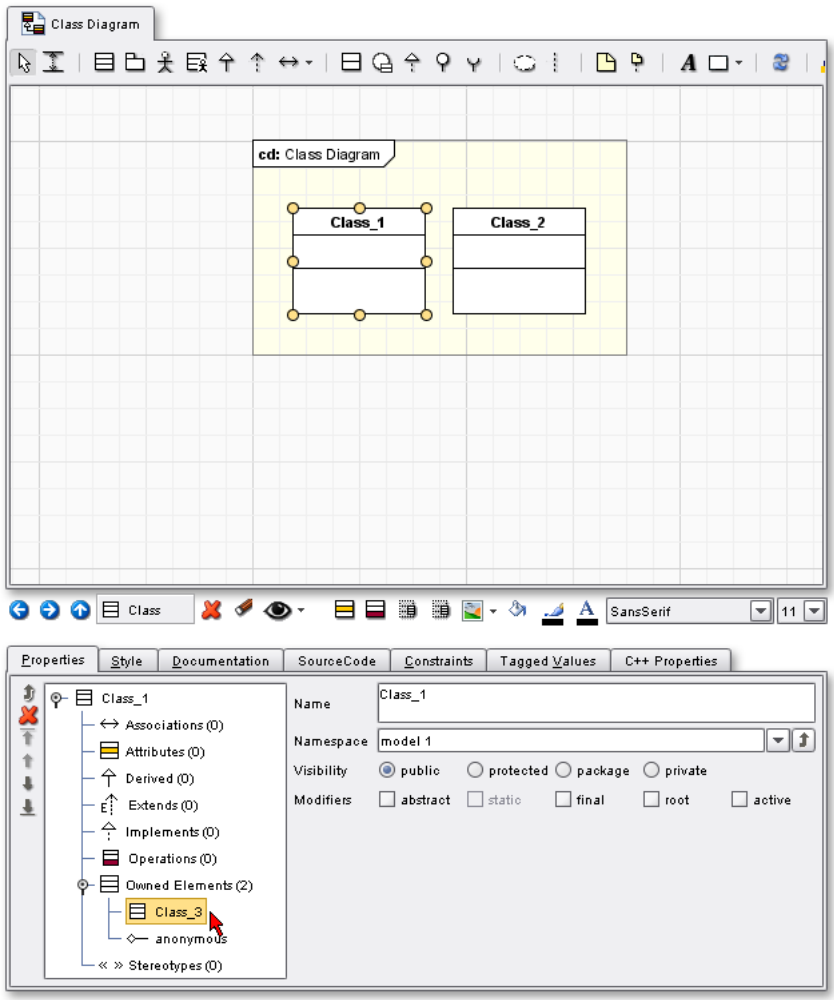
Figure 11-7. Drag and Drop completed



11.4.5.2. Drag and Drop from the Properties Tab

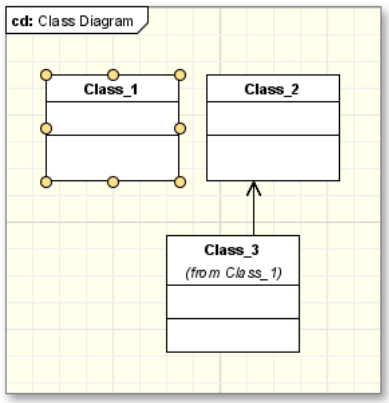
Elements can be added to a diagram from subtrees in the Properties tab of the Details pane. Any corresponding associations etc. are automatically added as well.

Figure 11-8. Drag and Drop from Properties tab



Here we see a Class diagram with `Class_1` and `Class_2` . `Class_3` is not currently in the diagram, but is a subclass of `Class_1` and has an association to `Class_2` . When `Class_3` is dragged to the diagram from the subtree of `Class_1` , the association to `Class_2` is automatically rendered as well.

Figure 11-9. Drag and Drop completed



11.4.6. Changing Namespaces

As your model evolves and grows bigger, you might want to restructure your model organization. Drag-and-Drop and Copy/Cut/Paste functions are surely one way of doing this. But there is a deeper concept behind the structure of models that you should be aware of.

UML has the notion of **namespaces** that define a structure for a model. This structure is not necessarily the same as the structure of your diagrams. Remember that model elements can be represented in several diagrams but can only have one namespace. And since diagrams can be created at very different points in the model structure (that is in different namespaces), model elements do not always share the namespace of that diagram.

A namespace is an abstraction of model elements that can contain further model elements. A typical example for a namespace is a package. Classes as well as diagrams are usually contained in a package, or to put it differently, their namespace is the package they are included in. Any model element that is not directly owned by another model element (like an operation that is owned by a class) has such a namespace.

To find out what namespace a model is in, look at the Properties tab in the Details pane. Any element either has a namespace or an owner. You can change the namespace by clicking on the little button to the right of the text field. This opens a drop-down menu with all namespaces you can move it to. For example, if you decide a class should not belong to the package you created it in, you can simply change its namespace to be a different package from the Properties tab.

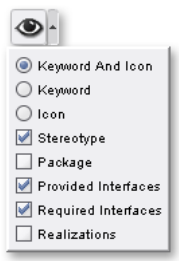
In some cases, changing the namespace for one element does not only effect this element but others as well. This is a convenience feature that was intentionally built in, believing that this is what the user intends to do in most cases. But this might not always be the case. If you change the namespace of a diagram, then all model elements in that diagram are assigned the new namespace as well. Also, if you change the namespace of a package, all included elements will likewise be moved to the new namespace.

Since packages are the most important type of namespace, there is another convenience feature for it. You can change a model element's namespace by dragging it with the mouse onto the figure of a package within a diagram.

11.4.7. Visibilities

As of Poseidon for UML version 4.2, it is possible to specify visibility options on a per-element basis from the Details pane. The visibility selector is comprised of two parts. The first is the visibility button itself, which hides and displays the frame that appears around the diagram, including the diagram name. The second is the dropdown that is accessible from the arrow next to the button. This dropdown allows you to select exactly which parts of the element will be displayed in the diagram. The options available from this dropdown will vary according to the element currently selected.

Figure 11-10. Visibility options for a Component



The final set of buttons allow you to add elements to the current attribute. The elements available depend on the element selected. For classes, your options include adding attributes, operations, inner classes, and inner interfaces. Associations and the like have a selector to determine how the edges should be displayed.

Additionally, the Properties tab contains checkboxes next to items that can be displayed or hidden, such as individual interfaces within a component.

11.4.8. Layout Functions

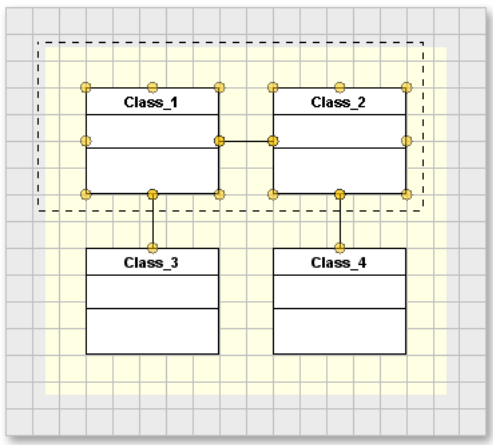
You already know that you can layout your diagram by using the select tool. But there are a number of other ways to rearrange your diagrams.

Select and Move Elements

A selected class can be moved not only by using the mouse, but also by means of the arrow keys. The elements get nudged in the direction of the selected arrow key. Holding down the SHIFT key while pressing the arrows causes the elements to move in larger increments.

You can easily select several elements by holding down the Shift key while you select further elements, or by clicking somewhere in the empty space of the drawing area and dragging the mouse over elements. A dashed line appears and all elements that are partially or wholly enclosed in it will be selected.

Figure 11-11. Selecting multiple elements with the mouse.



Movements always apply only to the selected elements. If you want to select all elements in a diagram, use the quick-key Ctrl-A.

Elements can be moved along invisible 'rails' by holding the Ctrl key while dragging the elements. This limits the movement to the X and Y axes of the original element. If multiple elements are selected, the center of the selected elements is used as the origin of the rails. This means that an element may or may not reside at the origin.

Arrange Elements

Another set of useful options that are accessible from the main menu are the arrangement options. These are a powerful set of tools to assist with the layout of a diagram.

The **Align Tools** include:

- **Align Tops** - Aligns the tops of the selected elements along the same horizontal axis
- **Align Bottoms** - Aligns the bottoms of the selected elements along the same horizontal axis
- **Align Lefts** - Aligns the left sides of the selected elements along the same vertical axis
- **Align Rights** - Aligns the right sides of the selected elements along the same vertical axis
- **Align Horizontal Centers** - Aligns the centers of the elements along the same vertical axis
- **Align Vertical Centers** - Aligns the centers of the elements along the same horizontal axis
- **Align to Grid** - Aligns the top-left corner of the element with the snap grid

The **Distribute Tools** include:

- **Distribute Horizontal Spacing** - Distributes elements so that there is the same amount of white space between the vertical edges of the selected elements
- **Distribute Horizontal Centers** - Distributes elements to that there is the same amount of space between the centers of elements along a horizontal axis
- **Distribute Vertical Spacing** - Distributes elements so that there is the same amount of white space between the horizontal edges of the selected elements
- **Distribute Vertical Centers** - Distributes elements so that there is the same amount of space between the centers of elements along a vertical axis

The **Size Tools** include:

- **Greatest Current Width and Height** - Uniformly resizes selected elements so that each is the size of the largest selected element.
- **Smallest Current Width and Height** - Determines the size of the smallest element that can display the information of each of the selected elements and uniformly resizes the selected elements.
- **Minimum Possible Width and Height** - Determines the smallest possible size for each of the selected elements and uniformly resizes them so that each is the size of the largest minimized element.
- **Greatest Current Width** - Uniformly resizes selected elements so that each is the width of the largest selected element.
- **Smallest Current Width** - Determines the width of the smallest element that can display the information of each of the selected elements and uniformly resizes the selected elements.
- **Minimum Possible Width** - Determines the smallest possible width for each of the selected elements and uniformly resizes them so that each is the width of the largest minimized element.
- **Greatest Current Height** - Uniformly resizes selected elements so that each is the height of the largest selected element.

- **Smallest Current Height** - Determines the height of the smallest element that can display the information of each of the selected elements and uniformly resizes the selected elements.Note: this is not functional in the current version of Poseidon.
- **Minimum Possible Height** - Determines the smallest possible height for each of the selected elements and uniformly resizes them so that each is the height of the largest minimized element.

The **Ordering Tools** include:

- **Bring To Back** - Places the selected element(s) on the bottom layer of the diagram display.
- **Bring To Front** - Places the selected element(s) on top of the diagram display.
- **Send Backward** - Moves the selected element(s) down one layer in the diagram display.
- **Send Forward** - Moves the selected element(s) up one layer in the diagram display.

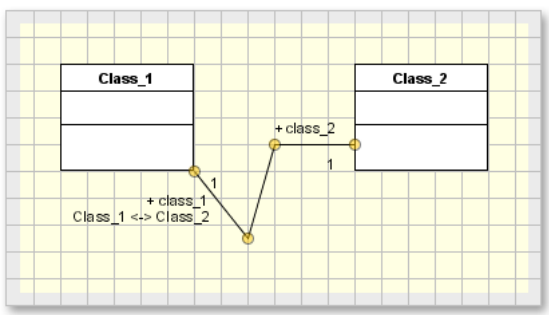
These groups of tools may be used alone or in conjunction with another tool of a different type.

The layout process is supported by a grid. It you want a finer or a coarser grid than the default, or if you want the grid to be displayed in a different manner, you can change this in the View menu.

Changing the Shape of Relationships

You can also change the layout of the edges. By default, Poseidon for UML always tries to draw a straight line without bends but you can easily add waypoints: Select an edge and move the mouse perpendicular to the edge. At first the edge simply moves, too. But as soon as a straight edge is no longer possible, a waypoint is automatically added. You can add several waypoints by clicking on the edge so that you can wire your diagrams as you prefer. To remove a waypoint, just move it over another waypoint or an endpoint and it disappears.

Figure 11-12. Adding waypoints.

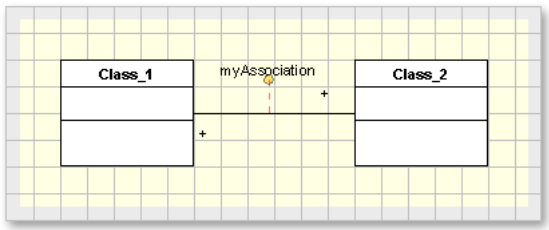


In Poseidon for UML version 2.0, waypoints have changed from blue boxes to yellow circles. Despite the change in appearance, they function in the same way.

Moving Adornments

You can also move adornments as you can move elements. Simply select the adornment and drag it around. You will notice a little dotted red line that indicates to which association this adornment belongs. Roles and multiplicities are attached to the association ends in the same manner.

Figure 11-13. Moving adornments.



In version 2.0, the adornments move in a slightly different (and more intelligent) manner. Previously, an adornment might obscure an edge. Adornments now 'hop' over edges, automatically providing a cleaner look to the diagrams.

11.4.9. Undo/Redo

Poseidon maintains a history of changes made to the model. The undo and redo buttons step forwards and backwards through this history. Beginning with version 2.6, the undo and redo history can include actions before an import, provided that the Settings dialog option has been enabled. See Section 7.2.4.1 for more information about the 'Roundtrip imports are undoable' option.

11.4.10. Non-UML Additions

Some of the items available in the toolbar exist to clarify and enhance models, even though they are not a part of the UML specification. These items do not affect any code generation, but increase the understandability of a project for human readers.

11.4.10.1. Select

The first tool in the toolbar is called the 'select' tool, and is the default active tool. It is used to select, move, and scale diagram elements, as well as modify the element directly from the diagram. When an element has been selected and is now the current active element, it will appear with yellow circles (called 'handles') surrounding it.

A brief list of functions:

- **Select an element** - Click on the desired element.
- **Move an element** - Click and hold the mouse button inside the element, then drag the element to its new location.
- **Resize an element** - Click and hold the mouse button on an element handle, then drag the handle.
- **Edit an element inline** - Double-click on a text element to activate the text edit box.

Try it Yourself - *Resize an Element*

1. Select the ⊟ `Client` class from a diagram.
2. Small round yellow handles appear on the corners of the element.
3. Click and hold the mouse button on one of these handles and drag it around the diagram to resize the class.

11.4.10.2. Comments

Sometimes a diagram requires a bit of extra explanation. This information is not a part of the final code, yet it helps the viewer better understand the diagram. This information can be included in a comment element. Comments are extra notes that are included and displayed in a diagram. These comments can be added to almost any element including other comments, or they can stand alone in the diagram. Comments cannot be added to relationships, transitions, or shapes created with drawing tools in any diagram, and objects in sequence diagrams.

Comments are ignored by the code generator; therefore they are never seen in the code output. They are likewise never seen in the Navigation pane.

To add a comment to a diagram:

1. Click the Comment button in the diagram toolbar.
2. Position the crosshairs in the diagram and click to place the comment in the diagram. At this point it is a freestanding comment that is not connected to any element.

To connect a comment to an element using the toolbar buttons:

1. Click the Connect Comment button from the toolbar.
2. Place the crosshairs over the comment to be connected. Click and hold the mouse button.
3. Drag the crosshairs to the element to be connected. Release the mouse button.

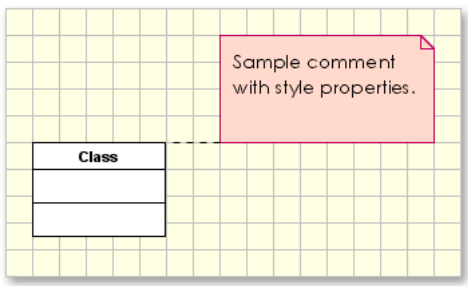
To connect a comment to an element using the rapid buttons:

1. Click one of the rapid buttons around the comment to be attached. Hold the mouse button down.
2. Drag the crosshairs to the element to be connected.
3. Release the mouse button.

You can either use the 'select' tool to make the comment the current active element and then begin typing, or double-click to open the editable text field.

Just as with any other element, notes can be resized with its handles and the color can be changed through the style panel of the Details pane. This makes it easy to introduce a color-coding scheme to diagram notations.

Figure 11-14. A new comment



11.4.10.3. Drawing Tools

The set of tools which appears at the end of the toolbar are for general drawing purposes. With these tools you can add other graphical elements such as shapes to your diagram. You should keep in mind that, although useful sometimes, these graphics are not part of UML; therefore, they don't show up in the model tree in the Navigation pane.

The Drawing Tools:

- **Text** - Click in the diagram area and begin typing to create a text object.
- **Circle** - Click in the diagram area and drag the mouse to create an ellipse.
- **Rectangle** - Click in the diagram area and drag the mouse to create a rectangle.
- **Polygon** - Click once everywhere the polygon is to have a corner. Double-click the last corner to close and render the polygon.
- **Polyline** - Click in the diagram area and to create a waypoint. Click again to create another waypoint and a line between them. A connected line can be added by lclicking to add a third waypoint. Double-click the last waypoint to cease the addition of lines.

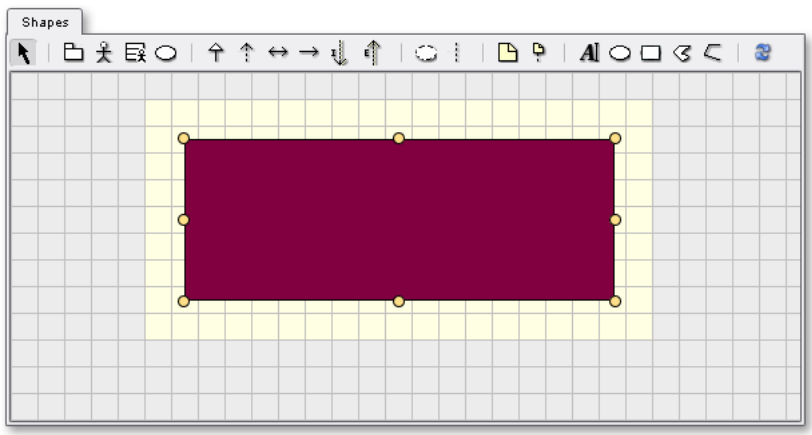
11.4.10.4. Toggle Between Editing Modes

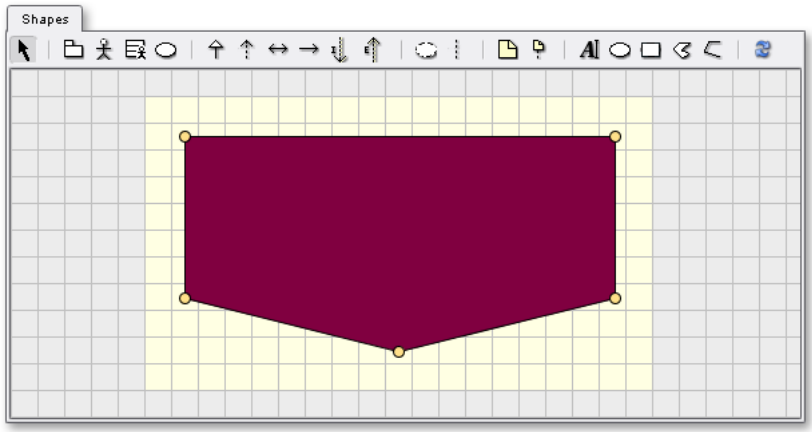
Two modes of editing are available for modifying shapes. You can switch between modes by double-clicking on a shape.

The first is a resize mode, which allows you to change the size of the shape by dragging the handles (gold circles) that surround the shape. Dragging one of the corner handles enlarges and shrinks the shape without changing its proportions. Dragging the side handles expand and compress the shape.

The second editing mode is available for all shapes except circles. It allows you to add, remove, and move waypoints to change the shape of the element. For example, you can create a rectangle, double-click on it, and then add a waypoint to create a new polygon.

Figure 11-15. Add a waypoint to a rectangle

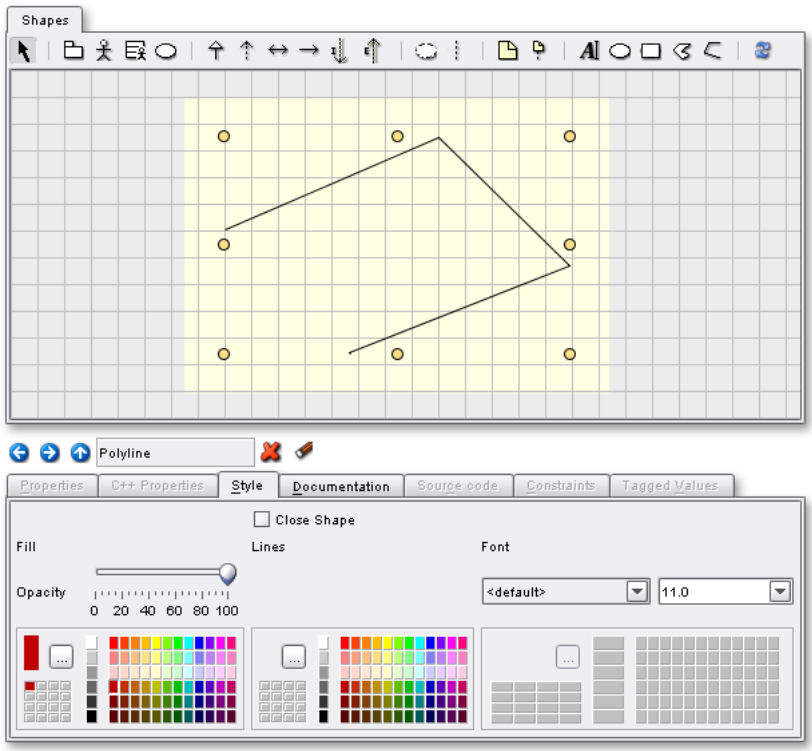


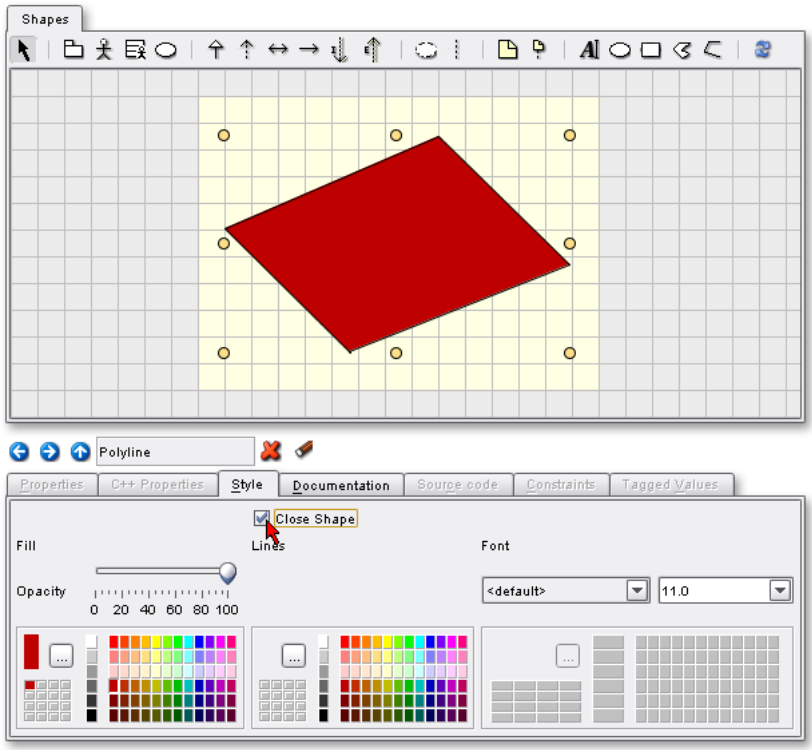


11.4.10.5. Close Shape

Once a shape has been drawn with the line tool, it is possible to close the shape automatically to create a polygon. Select the shape and open the 'style' tab of the Details pane. Check the box titled, 'Close Shape'. The shape can be reopened by unchecking the same box.

Figure 11-16. Open and closed lines

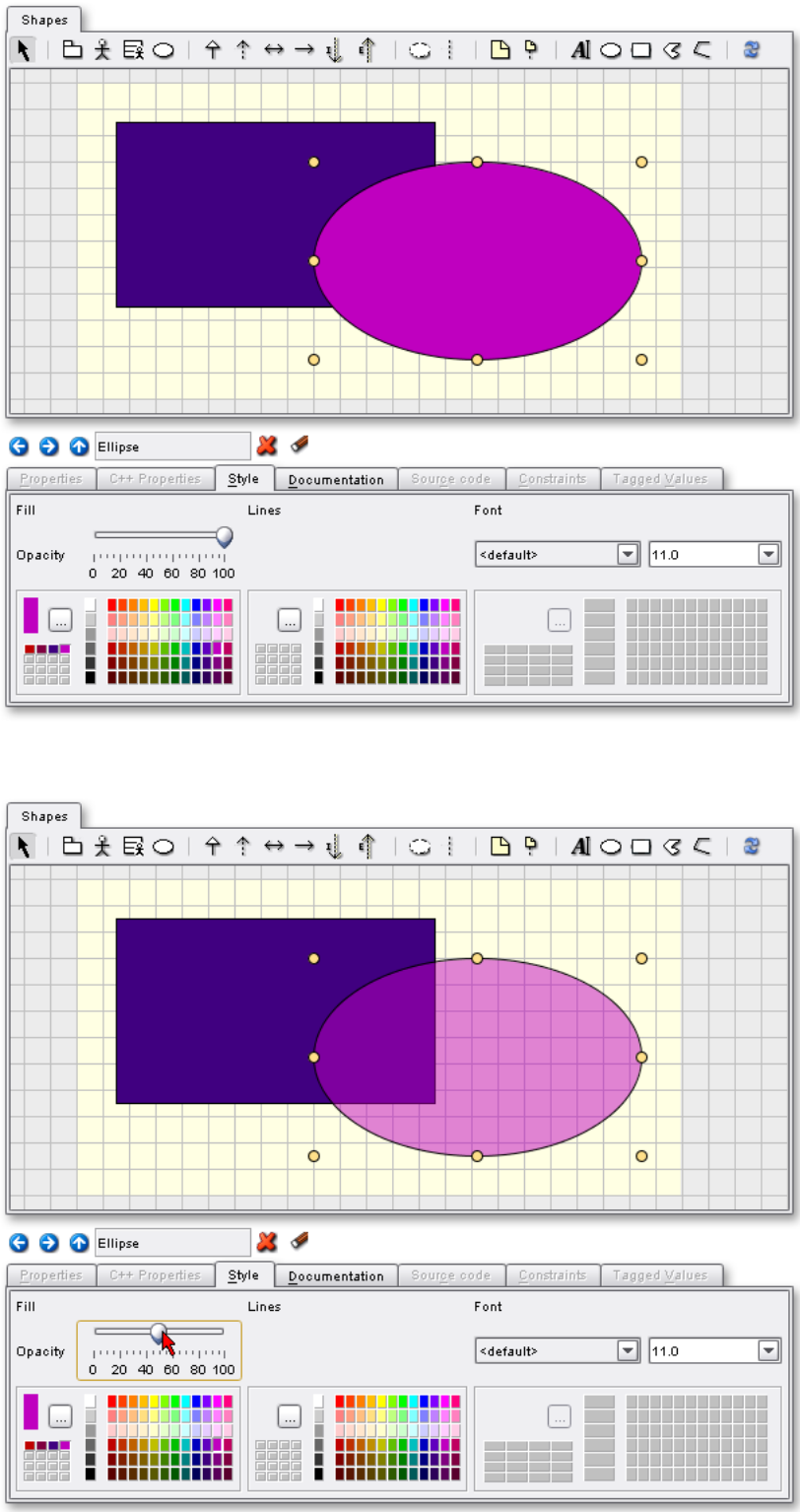




11.4.10.6. Opacity

Fill colors can be applied from the Style tab of the Details pane. It may be advantageous to change the opacity of this fill at times. Fortunately, this is very easily accomplished. Simply select the figure that will have a different opacity and use the slider bar within the 'style' tab of the Details pane.

Figure 11-17. Changing opacity



11.4.10.7. Waypoints

Once a line or polygon has been created, the shape can be altered by creating and moving a waypoint, much in the same way that connections between elements can be edited. Click on the perimeter and move the resulting gold circle to create an 'elbow'. In this same vein, waypoints can be deleted by selecting and dragging them over an existing waypoint or endpoint.

11.4.10.8. Diagram-specific Tools

The rest of the tools in the toolbar are specific to the current diagram type. They allow the creation of diagram elements and operate similarly to a stamp. With a single click on the icon you get a handle to create one corresponding diagram element. If you double-click, the tool stays selected and you can create a number of diagram elements, one after the other. The cursor changes to a hair cross with which you can select the position of the new element. To disable this feature just click on the 'select' tool.

Some tools are only available in a certain context. In Class Diagrams, the tools to create a new attribute or a new operation are only available when a class is selected. Select the desired class and click on the appropriate button to create a new attribute or operation for your class.

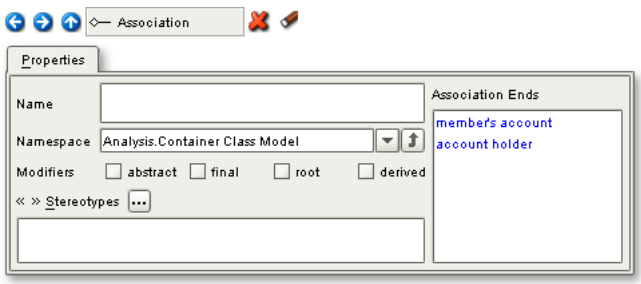
The individual tools are covered in detail in the chapter titled, 'A Walk Through the Diagrams'.

Chapter 12. Element Reference

12.1. Relationships

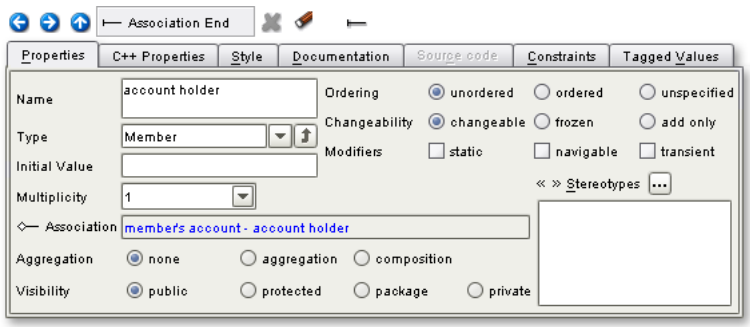
Relationships are very important in UML. They have properties in and of themselves, as well as consisting of other model elements. Every association has two ⊢ association ends that are model elements in their own right, as defined in the UML specification. Figure 9-4 shows the Properties tab for an association, in this case between Account and Member. Notice that there is no stereotype or name for this association, but they could conceivably exist. Also note that the association is part of the `Design.Use Case - Implementation.User Registration` namespace.

Figure 12-1. Properties tab for an association.



An association end can also be given a name, and like an association it doesn't require one. If an association end does not have its own name, the class name at that end of the association is displayed. Look to the left hand side of Figure 9-4. In this case, both association ends have been named. Like hypertext, they link to the association end properties, not to the class properties. Beginning with Poseidon version 3.0, association ends may now have an initial value entered in the Properties tab.

Figure 12-2. Properties tab for an association end.



Associations can be specialized to an ◇— aggregation or a ◆— composition. To do this, navigate to one of the association ends and change the aggregation type from none to either aggregation or composition. They can also be created directly from the toolbar, using the 'Create Aggregation' button or the 'Create Composition' button.

12.1.1. Types of Relationships

- Generalization
- Dependency
- Association
- Directed Association
- Aggregation
- Composition

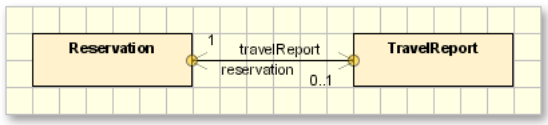
12.1.2. Navigability

The navigability of associations is similarly changed, using the association ends properties. The check box titled 'navigable', when checked, means *towards the class that this association end points to* . This is a bit counter-intuitive at first, so further explanation is warranted:

Associations can be modeled as navigable in both directions, navigable in only in one direction, or without any navigability. In most cases, navigability is indicated by arrows in the diagrams. The one exception is the default association, an association which is navigable in both directions. In this case arrows are omitted. The navigability of an association occurs at the beginning of the arrow, not at the end. You can easily navigate to the opposite association end using the navigation button in the Properties tab.

When you first create an association, it is navigable in both directions. The UML standard requires that both arrows are hidden in this case, so it looks just the same as an association with no arrows at all. To distinguish these two cases, the arrows of both its ends show up in grey, if necessary, when you select an association.

Figure 12-3. Highlight hints for associations.



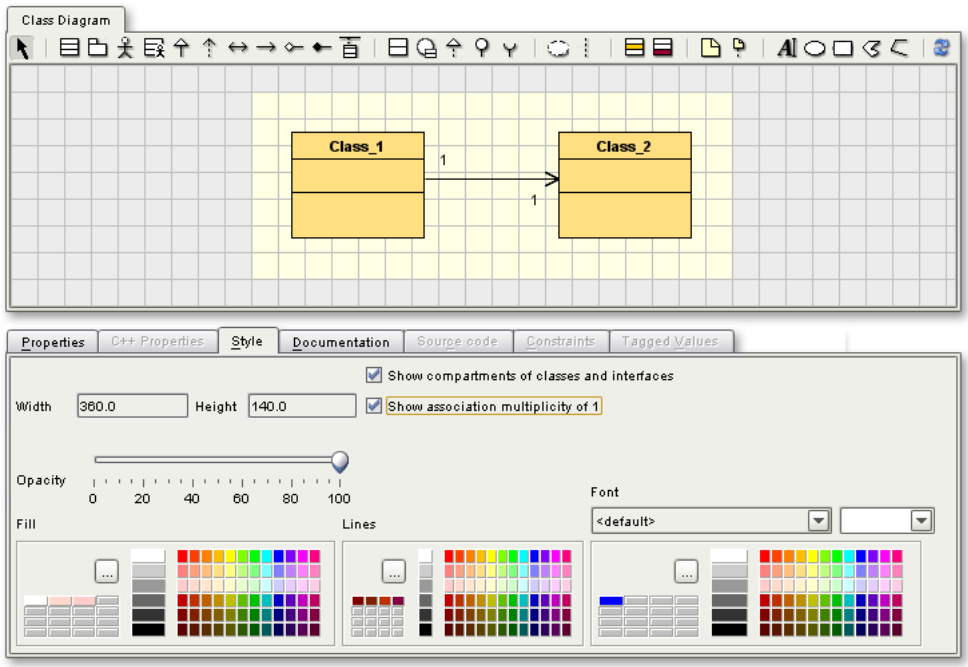
12.1.3. Hiding and Displaying Multiplicity of 1

When a multiplicity of 1 is set, some UML authors recommend hiding the 1, whereas others like to show the 1. To suit your needs, you can set the single multiplicity to be displayed or hidden. This can only be set diagram-wide in order to avoid confusion.

To change the display setting for single multiplicity:

1. Select the diagram where you want to change the setting.
2. Go to the Style tab.
3. Activate or deactivate the 'Show association multiplicity of 1' check box.

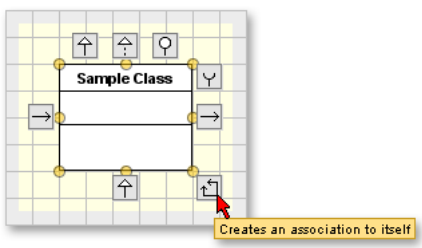
Figure 12-4. Style tab with multiplicity set

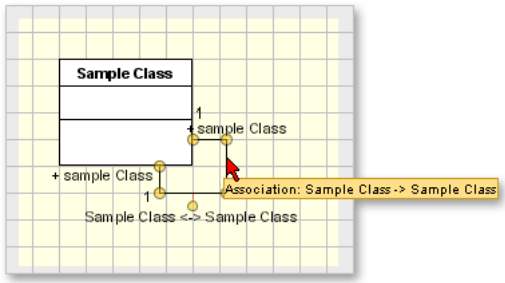


12.1.4. Self-Associations

Associations usually connect two different classes. But they can also be drawn from one class to itself. Simply use the rapid button in the lower right corner of the class.

Figure 12-5. The rapid button for self-associations





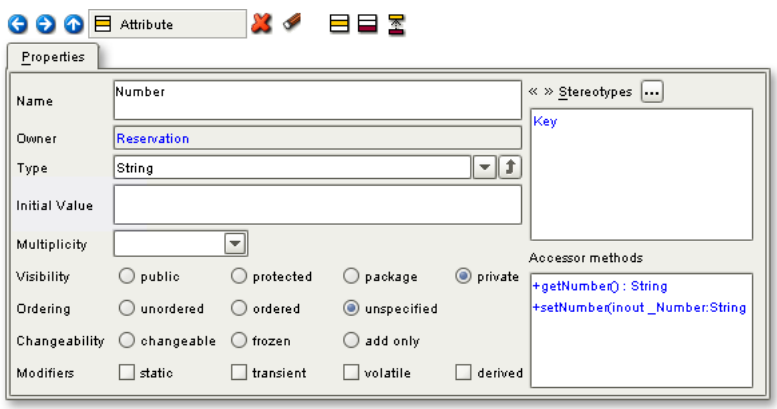
12.2. Classes

Classes are the core elements in an object-oriented model. They are represented by rectangles divided into three sections that display class information in the topmost portion, attribute information in the second, and operation information on the bottom. These compartments can be hidden via the context menu, as can package and stereotype information.

12.2.1. Attributes

Every class can have attributes that express some of its properties. In UML, every attribute has a name and a type. The type can be any other DataType, Class or Interface that is in the model. You can select the type from a combo box of all available types. If the type you need is not in the list, you can create it elsewhere, and then select it from the list.

Figure 12-6. Properties of an attribute.



Attribute Properties

- **Multiplicity** - The multiplicity field determines how many references the class has to this attribute.
- **Ordering** - Can be set to ordered, unordered, or unspecified (default).
- **Owner** - Specifies to which element the attribute belongs.
- **Visibility** - The visibility of an attribute expresses which other classes can access it. The options are:

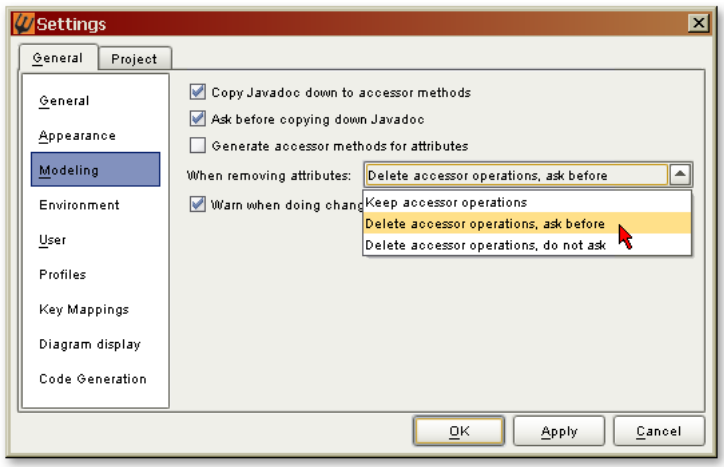
Public	+	Accessible to all objects
Protected	#	Accessible to instances of the implementing class and its subclasses.
Private	-	Accessible to instances of the implementing class.
Package	~	Accessible to instances of classes within the same package.

- **Modifiers** - You can also set whether the attribute is write-only by checking the **final** check box. An attribute can also be **static** (class scope instead of instance scope) or **transient** or **volatile** . An initial value can be given as a Java expression.
- **Stereotype** - Stereotypes can be added an removed via a dialog that is accessed by right-clicking the stereotype field and clicking 'Open' or clicking the ellipsis button  .
- **Type** - The Type dropdown sets the type of the attribute.
- **Initial Value** - This optional field gives an initial value to the attribute. Multi-line entries are accepted here; to create a new line, use Ctrl-Enter. (Note that this functionality is set in the Settings tab. More information is availalbe at Section 7.1.7).
- **Accessor Methods** - You can create the appropriate accessor methods for this attribute with a simple click. Just hit the button  'Add Accessors' in the Properties tab of the Details pane, and in the list below you will see a list of methods. This list depends on the multiplicity and the state of the final check box. If the multiplicity is 0..1 or 1..1, one `setAttribute` and one `getAttribute` method are created. If final is checked, it is only the `getAttribute` method that is created. If you chose a multiplicity that has a numerical upper bound (and not 1), array access methods are displayed. If you give a multiplicity with unlimited upper bound (also known as ..* or ..n), accessors for a `java.util.Collection` are created.

  When you create a new attribute, these methods are created automatically if you checked 'Create accessors for new attributes' in Edit-Settings. Every time you change the name or the multiplicity of the attribute, the access methods will change accordingly.

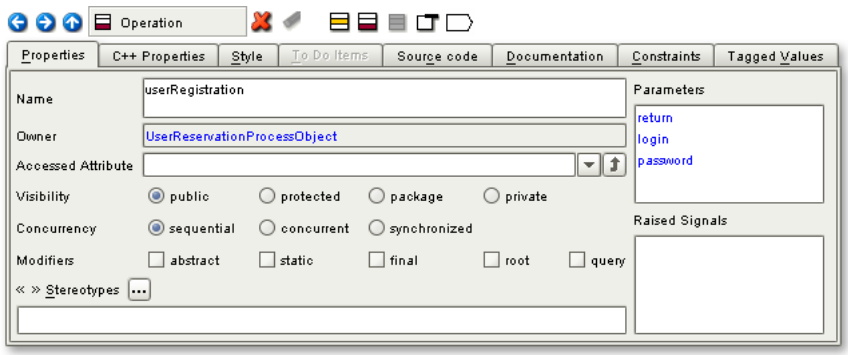
If you prefer to have only some accessor methods, right-click on one of the entries in the list 'Accessor Methods'. Then, select 'Delete'. The operation will be removed from the list; thus, it will not be marked as accessor any more. You will be asked if you want to remove this relation only or if you want to delete the operation completely. Whether this dialog appears or not is determined by your settings, which can be accessed in Edit-Settings. The 'Modeling' tab is displayed in Figure 9-9. Notice the 'When removing attributes' drop-down list. The default option is to ask before deleting the accessor operation. You can also choose to always keep the operations or to delete them directly without asking. This depends on your preference and your style of working with accessor methods.

Figure 12-7. 'Remove Attributes' setting



12.2.2. Operations

Figure 12-8. Properties of an operation.



For every operation, you can set several UML properties. Among them are visibility, scope (class or instance), and concurrency (with designators like sequential, synchronized, and concurrent). You can set an operation to be final, be a query, abstract, or a root method (with no parent).

The two lists on the right of the Properties tab are used to refine the operation's signature. In the list of parameters, the first parameter return is always there. This defines a return type for this operation. Similarly, you can add parameters that may be given a name, a type and the modifier 'final'. The final modifier is a special case that we introduced to handle Java.

You can define a constructor by setting the stereotype `<<create>>`. The code generation generates a constructor signature.

Operation Properties

- **Name** - This field gives a name to the operation. This field can also be changed directly in the diagram.
- **Owner** - The 'Owner' field specifies the element to which the operation belongs.
- **Accessed Attribute** - In the field 'Accessed Attribute', you can define whether this operation should be marked as an accessor method for an attribute. The primary use of accessor methods is to modify the attribute internally and to control access to the modifications externally. You can choose a modified attribute (if the operation is an accessor method created by Poseidon, an attribute is already selected) or select none if you want to decouple an accessor method from its attribute.
- **Visibility** - Can be set to public, protected, package, or private.
- **Modifiers** - One or more modifiers are set by means of the checkboxes.

- **Concurrency** - Set the concurrency to sequential, synchronized, or concurrent
- **Stereotypes** - Stereotypes can be added an removed via a dialog that is accessed by right-clicking the stereotype field and clicking 'Open' or clicking the ellipsis button ... .
- **Parameters** - Lists parameters of the operation. In Poseidon, return types are considered a special type of parameter.
- **Raised Signals** - The last list, 'Raised signals', is used to define whether this operation throws exceptions.

  Select 'Add...' from the context menu to insert a new exception.

  To edit this exception, select Open from the context menu, then enter a name and select a type for the thrown exception. As you know, only the type of the exception (not the name) is relevant for code generation. If you need more exception types, simply create the corresponding class in your model (e.g., `MailException` in your package `javax.mail`). Your exception type must end in `...Exception` in order to be visible in the type list of exceptions.

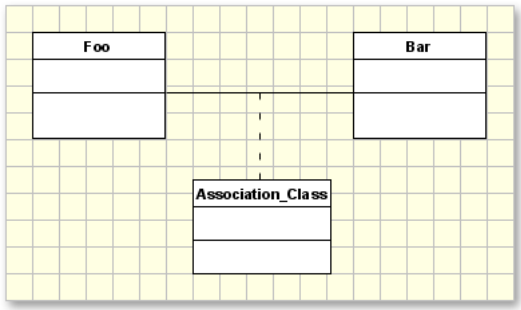
  You may also assign the <<exception>> stereotype to an element and use that element as a raised signal. First, apply the stereotype using the stereotype dialog. Second, add an exception to the desired operation through the 'raised signal' context menu within the properties tab. Third, select the exception-stereotyped element from the dropdown 'Type' list.

12.2.3. Association Classes

Poseidon now supports association classes.

To create an association class:

1. Create the two classes to be associated.
2. Select 'Association Class' from the toolbar.
3. Click on one of the original two classes and drag the mouse to the other class.
4. An association class will be created between the original two classes.



12.3. Interfaces

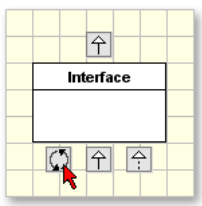
Interfaces are present in Class, Component, Deployment, and Object diagrams. Poseidon allows you to choose a variety of ways to represent interfaces, the standard being the traditional box notation, which looks like a class with only two compartments (as attributes are not allowed in interfaces). They can also be drawn using lollipop (or ball) notation, and you can toggle back and forth between these two.

12.3.1. Box Notation

The box notation is just what is sounds like - the Interface is shown as a box with two compartments. The uppermost compartment contains the name, stereotype, and package information, while the bottom compartment displays operations that belong to the interface. The options available to an interface (such as hiding compartments) are identical to those available to classes.

To create a new interface with box notation, select the Interface button from the toolbar and click in the diagram.

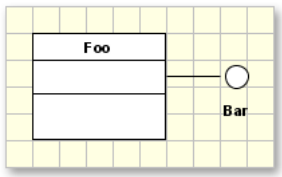
You can toggle between box and lollipop notation using the representation rapid button.



12.3.2. Lollipop Notation

Lollipop notation is a condensed way to represent an interface. The interface itself is drawn as a circle with a solid line connecting it to another element. This indicates that the element to which it is connected 'offers' the interface.

In this example, the class Foo offers Bar as an interface.



You can create the class first and then add the interface using the lollipop rapid button.

Alternatively, you can create the interface with the 'Interface as Circle' toolbar button and connect it to the element by either dragging the lollipop rapid button from the element to the interface, or you can select the lollipop button from the toolbar and drag it from the element to the interface.

The order in which the elements are connected is important to the meaning of the diagram. If you drag the relationship from the interface to the element, you will create a 'realize' relationship, indicating that the interface realizes the element.

Using Interfaces with Ports

Interface elements are used slightly differently when they are presented in conjunction with ports; for instance, when an interface is connected to a port, it is not possible to directly open the interface properties by clicking on it in a diagram. You may only adjust the physical representation of the interface (move it around, resize it, etc.)

There are several ways to access the properties of the interface. You can select the port to which it is attached, and then double-click on the name of the interface in the Properties tab of the port, or you can select the interface from the Model Index view in the Navigation pane.

For details on how to add interfaces to ports and then how to delete them, see Section 12.3.3.2 and Section 12.3.3.3 .

12.3.2.1. Sockets

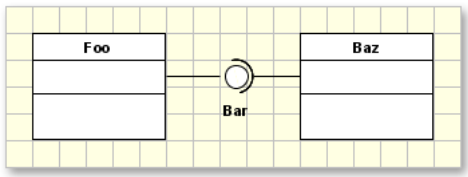
In addition to the compact representations, lollipop notation has an advantage in the clarity of a diagram.

This is because not only can 'provided' interfaces be modeled, but 'required' ones as well. These are called 'sockets' and are drawn with a semi-circle and line to the element requiring the interface.

To create a standalone socket that does not connect to a provided interface, you can either click the socket rapid button from an element, or you can click the socket button in the toolbar and drag it from an element to a blank space in the diagram.

To connect a socket to a provided interface, first create the provided interface as a circle. Then drag the socket rapid button (or socket toolbar button) from the element requiring the interface to the provided interface.

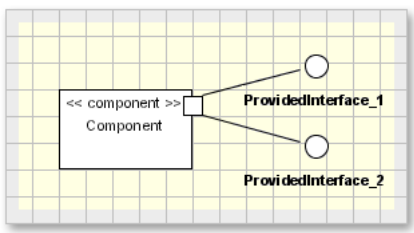
Here we see that class Foo offers the interface (Bar) that class Baz requires.



12.3.3. Ports

Ports provide a way to organize interactions between components and their environment within Component, Object, and Deployment diagrams. They can be uni-directional (only required or only provided interfaces connect to the port), or they can be bi-directional (both required and provided interfaces occur at one port).

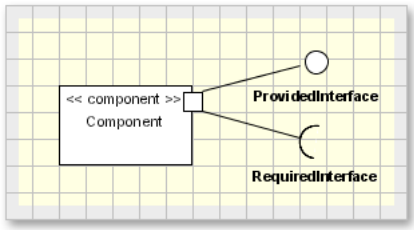
Figure 12-9. Uni-Directional Port

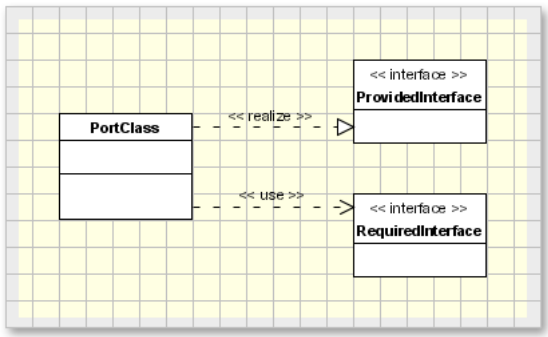


By default, each port has a class as its type. The manner in which a port and its interfaces are rendered within Deployment, Object, or Component diagrams is derived directly from the relationship between

the originating class and its interfaces. A realization will result in a provided interface; a dependency with the <<use>> stereotype indicates a required interface.

Below is an illustration of the same pair of interfaces in two different diagrams.





When a port is created in a diagram, a new class is automatically created semantically to function as the type for this port, although it is not added to the diagram. When interfaces are attached to this port, the interfaces are also semantically attached to the type class.

Try it Yourself - *Add Ports and Interfaces*

1. Create a new Component diagram by clicking the 'New Component Diagram' button.
2. Add a new Component by clicking the 'New Component' button and then clicking anywhere in the diagram.
3. Add a port to this component by hovering over the component, then click the port rapid button.
4. Click the 'Required Interface' button from the Details pane (below the Diagram pane).
5. Verify that the selection 'Add Dependent Edges Automatically' checkbox is activated in Edit | Settings | Diagram Display.
6. Create a new Class diagram by clicking the 'New Class Diagram' button.
7. Drag the class called 'Port_Class' and the new interface into the diagram.
8. Add a new interface and association by clicking hovering over the class, then clicking and dragging the realization icon to an empty space in the diagram.
9. Return to the Component diagram. Notice that a new provided interface has been added here as well.

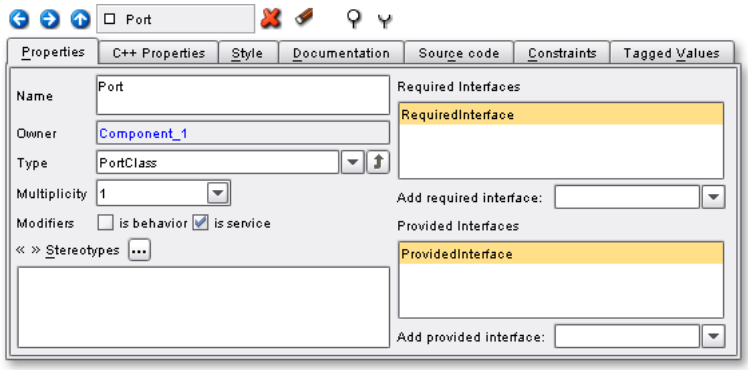
The properties of ports are very similar to those of other elements. The 'Port Properties' entry, however, only appears for ports. These checkboxes describe how the information entering the port will be handled.

- **isService** - When checked, this port provides access to the published functionality of a classifier; when unchecked, this port implements the classifier but is not part of the functionality of the classifier, meaning that it can be altered or deleted.

  The default value is checked.

- **isBehavior** - When checked, signal requests arriving at this port are received by the state machine of the object, rather than by any parts that this object contains.

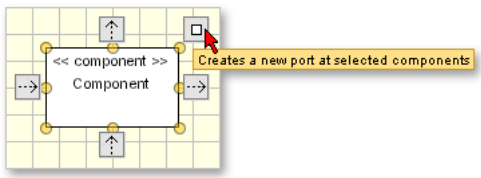
  The default value is unchecked.



12.3.3.1. Adding a Port

To create a port on a component, first select the component in the Diagram pane, then click the ❑ port toolbar button in the Details pane (located under the Diagram pane). This is a bit different than with other element types. You cannot first select the tool from the toolbar and then apply it to the component.

A rapid button can also be used to create a port. Hover over the target component, then click the Port rapid button.



12.3.3.2. Adding an Interface to a Port

There are three basic ways to add an interface to a port, either through the main toolbar or through the Selection Bar (located between the Diagram and Details panes).

The simplest method is via the Selection Bar:

1. Select the target port by clicking on it.
2. In the Details pane (below the Diagram pane), click on either the ♀ 'Provided Interface' or ♈ 'Required Interface' button.

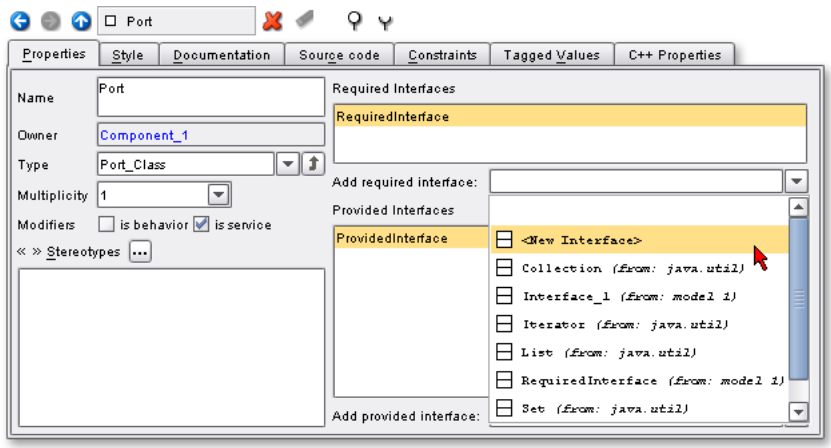
It is also possible to add an interface via the main toolbar:

1. From the main toolbar, click on either the ♀ 'Provided Interface' or ♈ 'Required Interface' button.

2. Click on the target port, then drag the mouse to an empty space in the diagram.

Existing and new interfaces can be added to a port from its Properties tab:

1. To add an existing interface, open the dropdown list from either the 'Add Required Interface' or 'Add Provided Interface' text box and select the desired interface.
2. To add a new interface, either type the new interface name in the text box or select '<New Interface>' from the dropdown list for an interface with a default name.



There is a fourth method using a Class diagram, which is more complicated and therefore not recommended in most situations. However, in limited cases it may prove useful:

1. Select the target port by clicking on it. In the Properties tab, note the class indicated as the port type.
2. If this class exists in an appropriate diagram, open that diagram. If not, create a new Class diagram and add the port type class.
3. Add the new interface to the Class diagram by clicking the Interface button from the main toolbar, then clicking in the diagram.
4. To make this interface a required interface, click on the ⇡ 'Realization' button in the toolbar (or use the class rapid button) and drag from the port type class to the interface. The socket rapid button on the class may also be used to create the realization relationship - click on the socket rapid button from the class and drag it to the interface.

   To make this interface a provided interface, click on the ⇡ 'Dependency' button in the toolbar and drag it from the class to the interface. Select this new dependency and click the ... ellipsis button to open the Stereotype dialog. Click on the stereotype 'use', click the '>>' button, then click 'Apply'. The lollipop rapid button on the class may also be used to create the dependency relationship - click on the lollipop rapid button from the class and drag it to the interface.

12.3.3.3. Deleting an Interface from a Port

Interfaces can be deleted from a port in a number of ways.

Delete from the Details pane:

1. Select the port to which the interface is attached.
2. Select the interface from the Properties tab of the port.
3. Click the 'Delete' button from the toolbar that appears directly over the interface list.

   Alternatively, you can right-click on the interface name in the Properties tab and select 'Delete' from the context menu.

Delete from the Navigation pane:

1. Locate the interface in the Navigation pane.
2. Either right-click and select 'Delete' from the context menu, or click the 'Delete' button in the Selection Bar.

As noted in the previous section, the interfaces can also be represented in a Class diagram. Deleting the interface from the Class diagram will likewise remove it from any other diagrams in which it occurs.

You can also simply remove the element from the diagram without completely deleting the interface from the model:

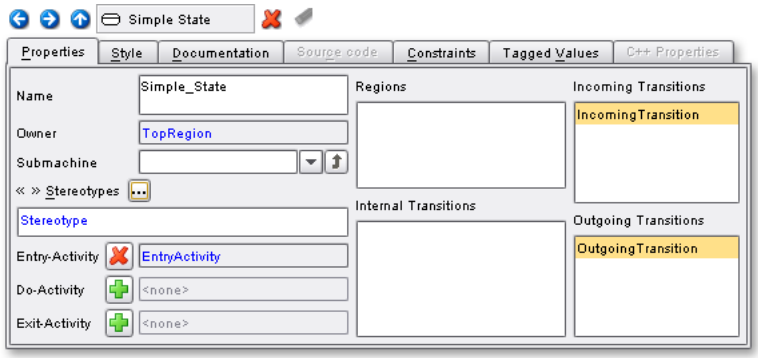
1. Select the interface from the Navigation pane to open the properties of the interface.
2. From the Selection Bar, click the 'Remove from Diagram' button.

12.4. States

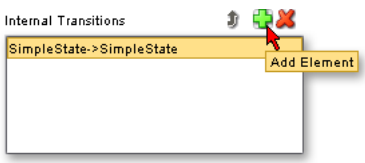
A state is a condition or situation in the life of an object during which it satisfies a condition, performs an activity, or waits for an event. They are rendered in State Machine diagrams.

12.4.1. Types

12.4.1.1. Simple States

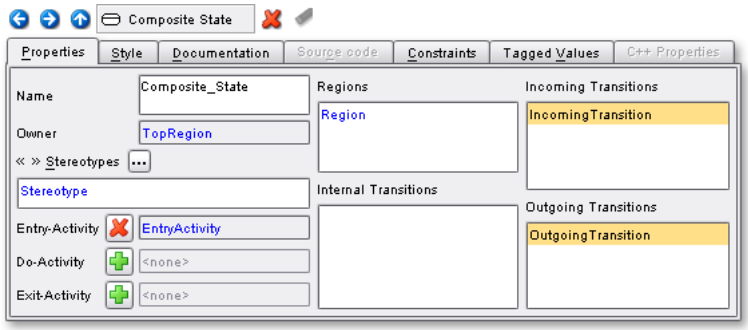


- **Name** - Name of the state.
- **Owner** - Name of the element logically one level higher. For a state, this is always a region.
- **Submachine** - A simple state cannot have submachines. Selecting a submachine from this list will convert the simple state to a submachine state.
- **Stereotypes** - List of stereotypes applied to the state.
- **Entry-Activity** - Activity to be invoked upon entry to the state.
- **Do-Activity** - Activity to be invoked after the Entry Activity has completed.
- **Exit-Activity** - Activity to be invoked before the state is exited.
- **Regions** - A simple state cannot have regions. Adding a region to a simple state will convert the simple state to a composite state.
- **Internal Transitions** - Activities that do not change the state of the object and therefore do not invoke the entry, do, or exit activities. Use the 'Add' and 'Delete' buttons to create and delete new internal transitions.

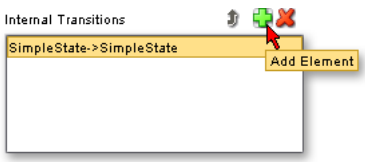


- **Incoming Transitions** - Transitions that allow entrance to this state.
- **Outgoing Transitions** - Transitions that exit this state.

12.4.1.2. Composite States

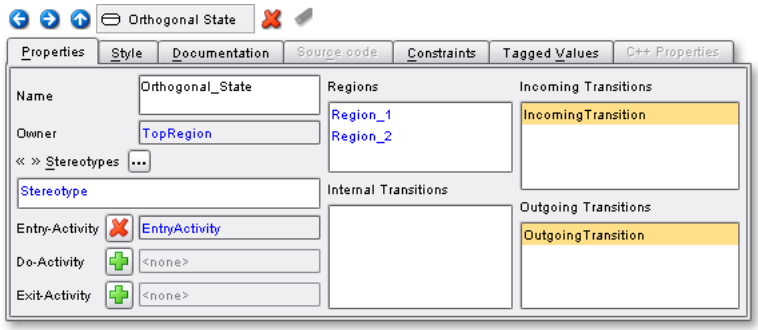


- **Name** - Name of the state.
- **Owner** - Name of the element logically one level higher. For a state, this is always a region.
- **Stereotypes** - List of stereotypes applied to the state.
- **Entry-Activity** - Activity to be invoked upon entry the state.
- **Do-Activity** - Activity to be invoked after the Entry Activity has completed.
- **Exit-Activity** - Activity to be invoked before the state is exited.
- **Regions** - Name of the region. A composite state has exactly one region - removing the region will convert the state to a simple state; adding a region will convert the state to an orthogonal state.
- **Internal Transitions** - Activities that do not change the state of the object and therefore do not invoke the entry, do, or exit activities. Use the 'Add' and 'Delete' buttons to create and delete new internal transitions.

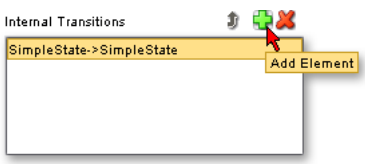


- **Incoming Transitions** - Transitions that allow entrance to this state.
- **Outgoing Transitions** - Transitions that exit this state.

12.4.1.3. Orthogonal States

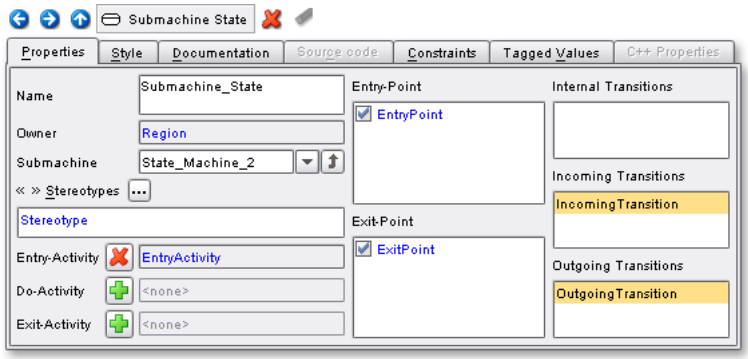


- **Name** - Name of the state.
- **Owner** - The name of the element logically one level higher. For a state, this is always a region.
- **Stereotypes** - List of stereotypes applied to the state.
- **Entry-Activity** - Activity to be invoked upon entry the state.
- **Do-Activity** - Activity to be invoked after the Entry Activity has completed.
- **Exit-Activity** - Activity to be invoked before the state is exited.
- **Regions** - The names of the regions for this state. Orthogonal states must have more than one region - removing regions until there is only one region remaining will convert the orthogonal state to a composite state.
- **Internal Transitions** - Activities that do not change the state of the object and therefore do not invoke the entry, do, or exit activities. Use the 'Add' and 'Delete' buttons to create and delete new internal transitions.

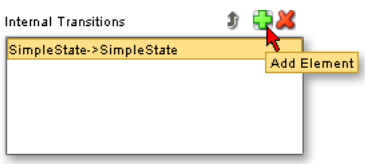


- **Incoming Transitions** - Transitions that allow entrance to this state.
- **Outgoing Transitions** - Transitions that exit this state.

12.4.1.4. Submachine States

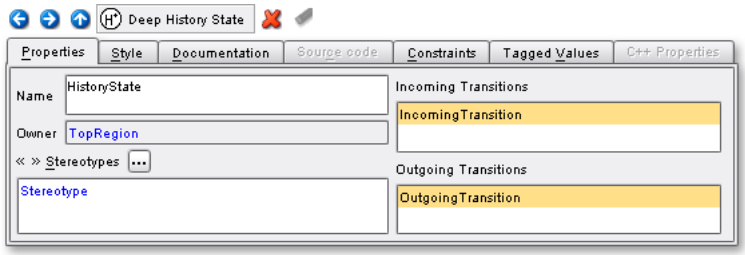


- **Name** - Name of the state.
- **Owner** - The name of the element logically one level higher. For a state, this is always a region.
- **Submachine** - The name of the associated state machine. A submachine state must have an associated state machine - if the option <none> is selected from the dropdown list, the connection to the state machine will be broken, and the submachine state will convert to a simple state.
- **Stereotypes** - List of stereotypes applied to the state.
- **Entry-Activity** - Activity to be invoked upon entry the state.
- **Do-Activity** - Activity to be invoked after the Entry Activity has completed.
- **Exit-Activity** - Activity to be invoked before the state is exited.
- **Entry-Point** - Name(s) of any entry point state(s) in the associated state machine. With the checkbox enabled, a reference to the entry point is displayed on the edge of the submachine state.
- **Exit-Point** - Name(s) of any exit point state(s) in the associated state machine. With the checkbox enabled, a reference to the exit point is displayed on the edge of the submachine state.
- **Internal Transitions** - Activities that do not change the state of the object and therefore do not invoke the entry, do, or exit activities. Use the 'Add' and 'Delete' buttons to create and delete new internal transitions.



- **Incoming Transitions** - Transitions that allow entrance to this state.
- **Outgoing Transitions** - Transitions that exit this state.

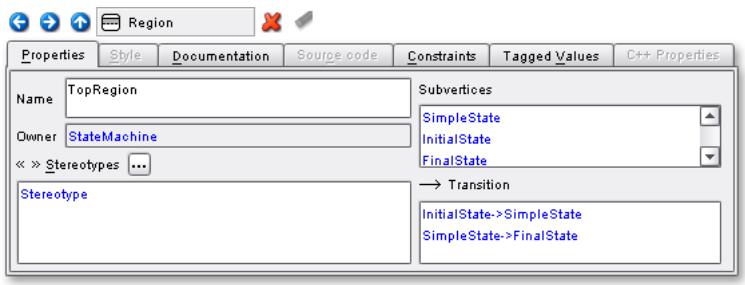
12.4.1.5. Additional States (Pseudostates)



This category of states includes Initial, Final, Terminate, Choice, Junction, Fork, Join, and History pseudostates.

- **Name** - Name of the state.
- **Owner** - Name of the element logically one level higher. For a state, this is always a region.
- **Stereotypes** - List of stereotypes applied to the state.
- **Incoming Transitions** - Transitions that allow entrance to this state. Initial states cannot have incoming transitions.
- **Outgoing Transitions** - Transitions that exit this state. Final and Terminate states cannot have outgoing transitions.

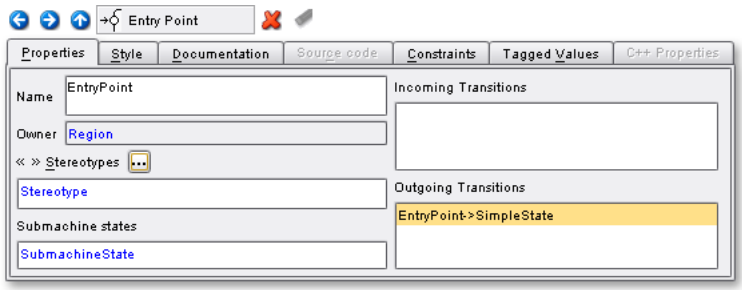
12.4.2. Regions



- **Name** - Name of the region. Regions themselves are not displayed in diagrams, so this name will only appear in the navigation panel.
- **Owner** - Name of the element logically one level higher. For regions, this is the name of either a state or a state machine.
- **Stereotypes** - List of stereotypes applied to the region.
- **Subvertices** - List of all states, including pseudo-states, contained in the region.
- **Transitions** - List of all transitions between elements in the region.

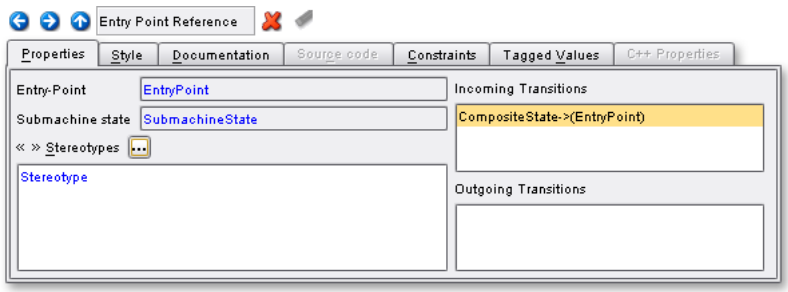
12.4.3. Entry and Exit Points

12.4.3.1. Entry and Exit Points



- **Name** - Name of the state.
- **Owner** - Name of the element logically one level higher. For a state, this should always be a region.
- **Stereotypes** - List of stereotypes applied to the state.
- **Submachine States** - List of submachine states that have enabled the entry and/or exit point reference display checkbox for this entry/exit point. See and .
- **Incoming Transitions** - Transitions that allow entrance to this state. Exit points can only have incoming transitions.
- **Outgoing Transitions** - Transitions that exit this state. Entry points can only have outgoing transitions.

12.4.3.2. Entry and Exit Point References

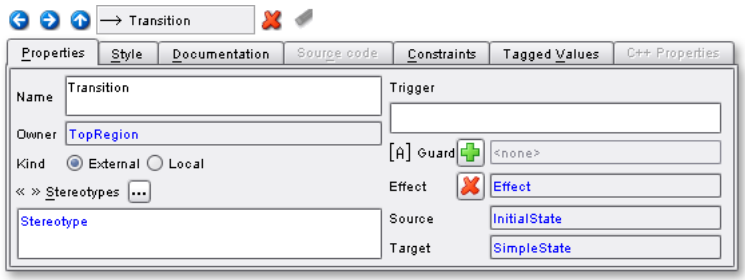


Entry and exit point references are notations in a submachine state that correspond to the actual entry and exit points of the associated state machine.

- **Entry-Point/Exit-Point** - Name of the entry/exit point in the associated state machine.
- **Submachine State** - Name of the associated state machine.

- **Stereotypes** - Stereotypes applied to this reference. Stereotypes applied here do not apply to the entry/exit point itself.
- **Incoming Transitions** - Transitions that allow entrance to this state. Entry point references can only have incoming transitions.
- **Outgoing Transitions** - Transitions that exit this state. Exit point references can only have outgoing transitions.

12.4.4. Transitions

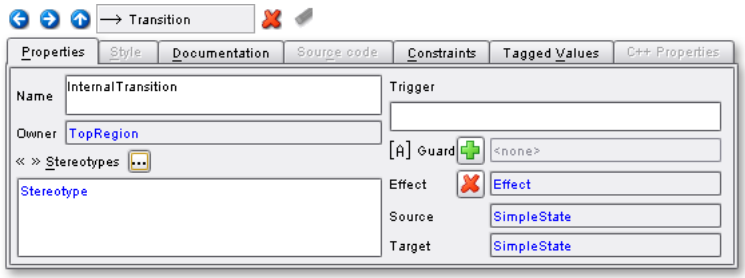


12.4.4.1. Between States

- **Name** - Name of the transition.
- **Owner** - Name of the element logically one level higher. For a transition, this is always a region.
- **Kind** - Indicates whether a transition is external or local. This option does not appear for internal transitions, which can only be added via the target state's Property panel.
- **Stereotypes** - List of stereotypes applied to the transition.
- **Trigger** - Name of the event whose occurrence makes a transition eligible to fire.
- **Guard** - Name of the boolean expression that must be satisfied in order for the transition to fire.
- **Effect** - Name of an optional activity to be performed during the transition.
- **Source** - The originating state of the transition.
- **Target** - The destination state of the transition.

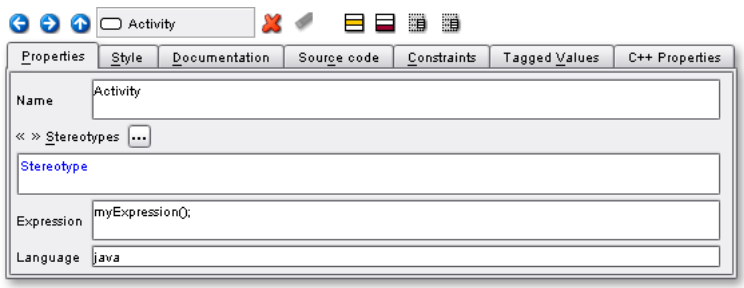
12.4.4.2. Internal Transitions

Internal transition activities do not change the state of the object and therefore do not invoke the entry, do, or exit activities. They are only accessible from the owning state, including for their creation and deletion. See Section 12.4 .



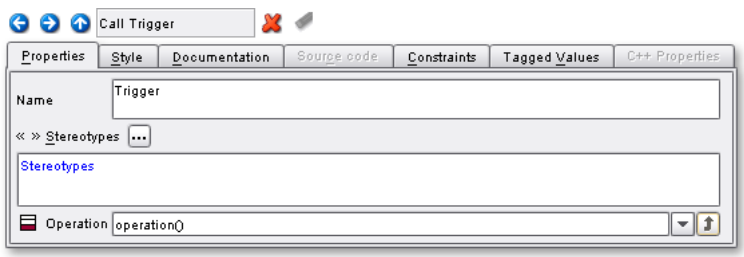
12.4.5. Activities, Triggers, and Guards

12.4.5.1. Activities



- **Name** - Name of the action.
- **Stereotypes** - List of stereotypes applied to the action.
- **Expression** - Syntax that will be completed during the action.
- **Language** - Indicates in which language the expression has been written. This can be anything, including plain English.

12.4.5.2. Trigger



- **Name** - Name of the trigger.
- **Stereotypes** - List of stereotypes applied to the trigger.
- **Operation** - Dropdown list with the available operations for the trigger. These are the operations of the owning BehavioredClassifier, the inherited operations of any superclasses, or operations of an implemented interface.

12.4.5.3. Guard

Note that entry and exit actions may not have guard conditions as they are invoked implicitly, not explicitly.

- **Name** - Name of the guard.
- **Stereotypes** - List of stereotypes applied to the guard.
- **Expression** - A procedural expression that is executed if and when the transition fires
- **Language** - Indicates in which language the expression has been written. This can be anything, including plain English.

Chapter 13. Using Elements

13.1. Creating New Elements

A new diagram, of course, requires elements in order for it to have significance. There are several ways to add elements to a diagram, as you will see in the next couple of sections.

13.1.1. Diagram Pane Toolbar

The diagram pane toolbar contains buttons to create elements that are specific to that diagram. For example, the button to create an initial state will not appear in a class diagram toolbar. This reduces the amount of buttons that you must deal with at one time.

The create buttons for most elements act as stamps, so that the element is placed wherever you click within a diagram. The exceptions are associations and ports.

Ports are created by first selecting the component that will have a port, then selecting the port tool from the toolbar. At this point the port will be automatically added, you need not click on the component to add the port.

Any sort of relationship needs to exist between two model elements; therefore, both elements of the association must be included in the creation process instead of just stamping a line anywhere. To create a new association element with the toolbar, select the type of association and place the cursor over the first element in the relationship. Click and hold the mouse button, then drag it to the second element in the relationship. Note that for some of the association types, the order in which the elements are connected affects the definition of the association.

Try it Yourself - *Create new elements with the toolbar*

1. Open the class diagram `User Registration: Design-Class Model` .
2. Select the 'create class' button from the toolbar. The mouse should now appear as a crosshair.

3. Place the crosshair to the right of the class 'User' and click the mouse button to create the new class.

4. Select the 'generalization' button from the toolbar. The mouse will again appear as a crosshair.

5. Place the crosshair in the new class, press and hold the mouse button, then drag it to 'User'.

** Note that the order in which they are connected determines the direction of the inheritance.*

6. You are now ready to incorporate the new class into the model. Look through the rest of this guide to learn how to change the name of the class, color-code it, add elements and operations, and more.

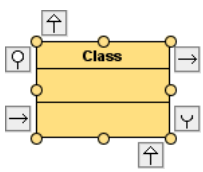
13.1.2. The Rapid Buttons

The toolbar is not the only way to create new diagram elements or associations. Poseidon for UML provides an intelligent shortcut that can speed up the development of a diagram. Select a class and wiggle your mouse near the edge of the class and several additional buttons will appear. They are called Rapid Buttons and are only available if an element is selected.

These rapid buttons can be used in several ways. You can either click on it to create and associate a new corresponding model element with appropriate connection in one step, or keep the mouse button pressed and drag it to an existing model element to create a new association without creating a new class.

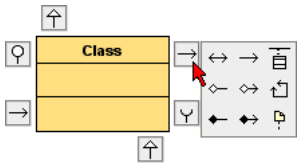
Rapid buttons are available for many diagram elements in each of the editors. Here is a class example:

Figure 13-1. Rapid buttons for a class element.



New in Poseidon 5.x, the rapid buttons have been cleaned up and organized. Right-click on a rapid button to see all available related rapid buttons. Once a command has been selected from the expanded selection, this command 'sticks' as the default for that rapid button location.

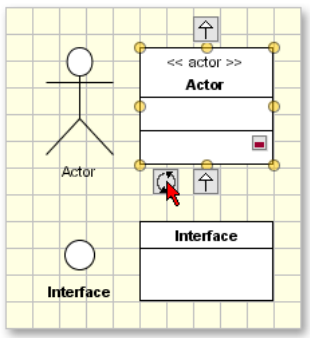
Figure 13-2. Expanded rapid buttons for a class element.



Try to click on the rapid button underneath your new class and you will see that a new subclass appears close to it. If you click and hold the button, you can move the mouse and place the new element where you want it to be. Or if you click, hold, and move the cursor over an existing element, only a connection between these elements is created.

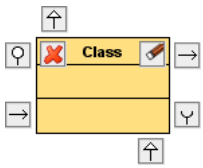
Rapid buttons for some elements change the way the element is presented. For interface elements, you can toggle between lollipop (or ball) representation and box representation. Similarly, for actor elements you can toggle between stickman and box representation.

Figure 13-3. Toggled representations of actors and interfaces



The rapid buttons displayed are some of the more commonly used buttons. To display additional buttons as rapid buttons, hold down the 'shift' button while rolling over the element with the mouse.

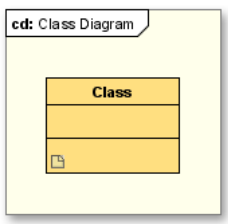
Figure 13-4. Additional rapid buttons for a class element.



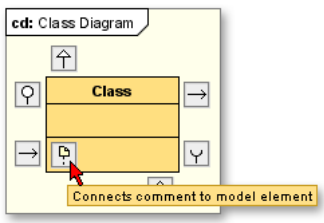
- **Attributes and Operations** - You can create new attributes and operations by clicking the rapid button that appears when you hover over the attribute or operation compartment of a class.
- **Delete** - Press Ctrl while hovering over an element to display a rapid button that will not open a confirmation dialog.
- **Go to Sub-Diagrams** - If an element contains diagrams, an additional rapid button will appear in the top right-hand corner that will take you to the sub-diagram.

Documentation Rapid Button

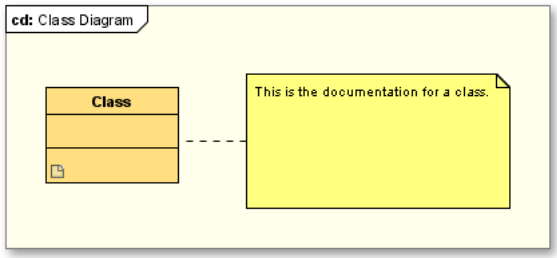
Poseidon 5.x also introduces a documentation rapid button. An element with documentation entered in the documentation properties tab will have a small, faint icon in the lower-left corner.



When the element is moused-over, the icon will change into a comment rapid button.



Clicking this rapid button will attach a comment to the element, and the documentation text from the documentation tab will be entered into the comment.



13.2. Editing Elements

Now that new elements have been created, they must be modified in order to be meaningful to the model.

13.2.1. Inline Editing Text Values

The diagram drawing area in the Diagram pane not only allows for creating, deleting and moving graphical elements; it is also possible to enter values, such as names, directly into the elements without using a different pane. Exactly which element properties can be modified depends upon the specific element. Most of the elements allow editing of the name of the element at a minimum. For example, selecting a state from within a state machine diagram and then typing will immediately open a small text editor. When editing is finished, the typed text will replace the previous text in the navigation tree and Properties tab, as well as in the diagram of the selected state.

Classes and interfaces offer far more options for editing values than just editing their names. Both of them are constructed of different parts called **compartments** . The first compartment holds the values for the name, the stereotype and the package of the class or interface. You can edit the name of the class as described above; however, stereotypes and packages can only be changed using the Properties tab. The second and third compartments hold the attributes and operations defined for the class or interface (in UML, interfaces can have operations only). Inline editing works the same way here. Select the attribute or operation you want to change and start typing (or double-click on it to open the inline editor). Press Ctrl-Return on the keyboard or click elsewhere in the application to end the editing. Note that in some cases, interfaces may be rendered as a circle in lollipop notation, and therefore will not display compartments.

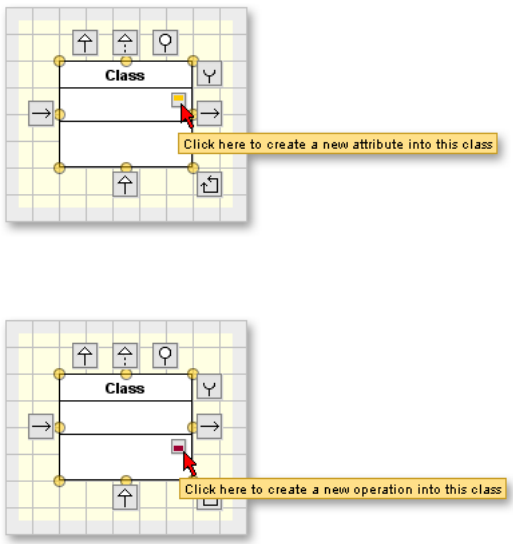
After editing an attribute or operation, you can directly add another attribute or operation without leaving the element by hitting 'return' on the keyboard instead of Ctrl-Return after editing the first attribute or operation.

You can also create a new attribute or operation with a rapid button by moving the mouse to the right side of the compartment and then clicking on the 'create' button that appears. As above, Ctrl-Return will end

the editing and add the new attribute/operation to the class or interface, and 'return' on its own will end the editing and create a new attribute or operation..

The attributes and operations compartments in the diagram can be set to invisible for the current diagram via the Context menu, or for the entire model via the 'Settings' dialog from the 'Edit' menu.

Figure 13-5. Add a new attribute or operation to a class inline



13.2.2. Editing Via the Details Pane

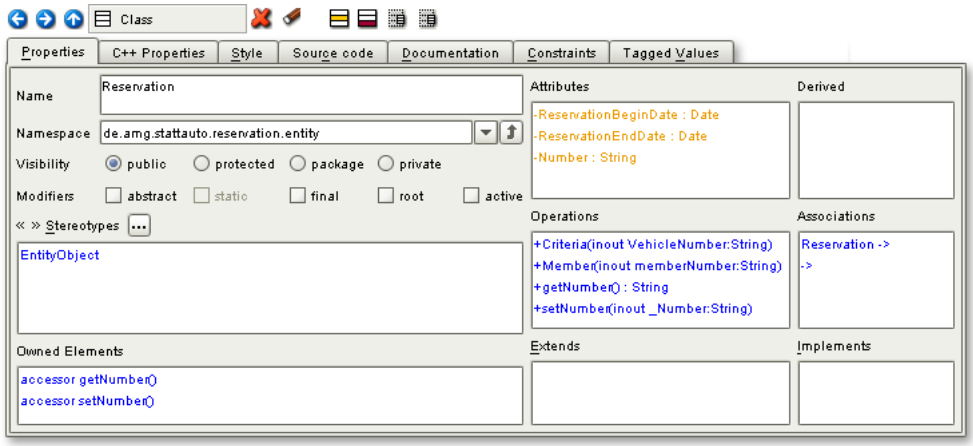
The first tab you will see in the Details pane is the Properties tab.

13.2.2.1. The Properties Tab

There are many modifications that can be made to elements from the Details pane. You can add attributes and operations, rename elements, change namespaces and stereotypes, add colors and borders, and much more. This section will outline some of the most important modifications that can be made. Many of these procedures can be extrapolated to other editing procedures.

Let's look at a class element, as these are very frequently used elements.

Figure 13-6. Properties tab for a class



The toolbar across the top of the tab contains buttons for navigation between elements, creation buttons, and a delete button. These buttons will change depending on the type of element selected as the current active element.

Below this toolbar are the editable characteristics of the class. The name of the element can be typed directly into the name field with no restrictions. Likewise, Visibility and Modifiers can be directly modified from their checkboxes. Note, however, that these two properties are not displayed in the diagram itself; thus, the changes made will be visible only from the Properties tab (the modifier 'abstract' is the exception to this). The Namespace must be selected from the list of available options. Stereotypes can be applied through the Stereotypes dialog, accessed by right-clicking the Stereotypes field and selecting 'Edit' from the context menu that appears or clicking the ellipsis button ... . The Owned Elements section is automatically populated.

All changes made to the class are propagated throughout the model. For instance, when a namespace is changed, the navigation tree is updated and the class is moved from the original package to the new one that was just selected. This change is also reflected in the Diagram pane: the top compartment of the class will display *(from new_namespace)* in place of *(from old_namespace)* , where old_namespace and new_namespace refer to the original namespace and most recently selected namespace. This easy and convenient mechanism for changing namespaces is provided for nearly all of the elements.

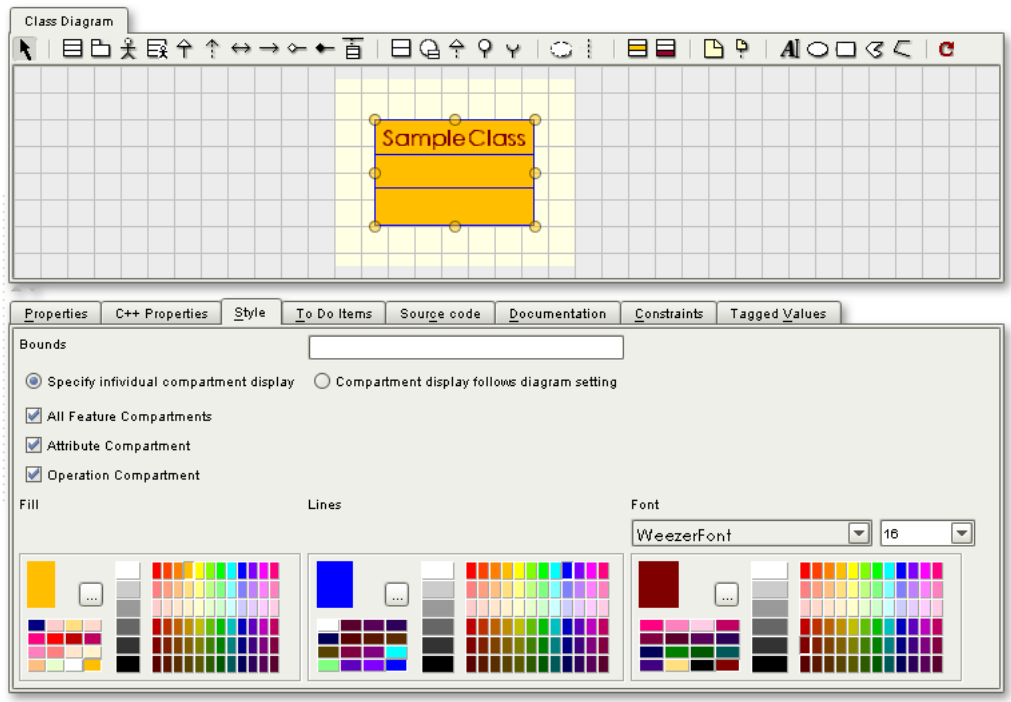
To the left of the editable characteristics are elements which are affiliated with the selected element. In UML, operations and attributes are considered both an elements in their own right as well as a characteristics of a class. As they are elements, they have their own Properties tabs; therefore, to edit the name or any other properties of an operation, for example, we must go to the Properties tab of that operation. That is why it is not editable here. The remaining fields are: `Extends` , `Implements` , `Associations` , and `Derived` . These properties show different relations between the focused class and

other model elements.

13.2.2.2. The Style Tab

Next we can look at the Style tab, which determines how the element is rendered in the diagram.

Figure 13-7. Style tab for a class



The Style tab indicates which colors and fonts will be used to display the element. This is very useful when color-coding diagrams or highlighting aspects of the diagram. It is also possible to override diagram-level specifications for compartment visibility. As with the properties tab, not all of the options make sense for every element; therefore, only the appropriate style options are available.

Options for the Style tab:

- **Fill** - Determines the background fill color of the element
- **Lines** - Determines the border color of the element

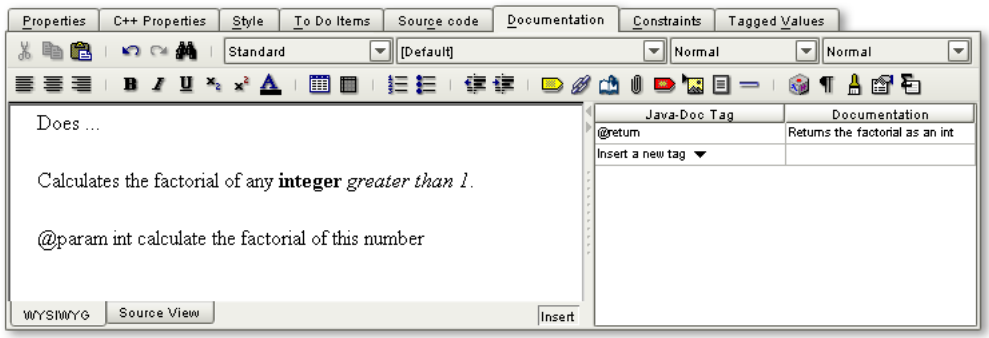
- **Font** - Determines the color and font of the text.
- **Visibility** - Radio buttons and checkboxes determine the visibility of element compartments and association multiplicities of 1.
- **Shape Characteristics** - Sets properties such as opacity for non-elemental items included in a diagram.

Whereas changes made to an element in the Properties tab are propagated throughout the model, changes made to the style of an element apply to the current diagram only.

13.2.2.3. The Documentation Tab

To add documentation to a model element, select the documentation tab in the Details pane. When you have imported Java source code, the Javadoc contained in the source code is likewise imported and viewed in the documentation tab. When working with text based IDEs, you put your Javadoc in doc comments (`/** */`). When using Poseidon's HTML editor, this is not necessary. The doc comments are added automatically to your source code when you generate it.

Figure 13-8. Editing a method documentation.



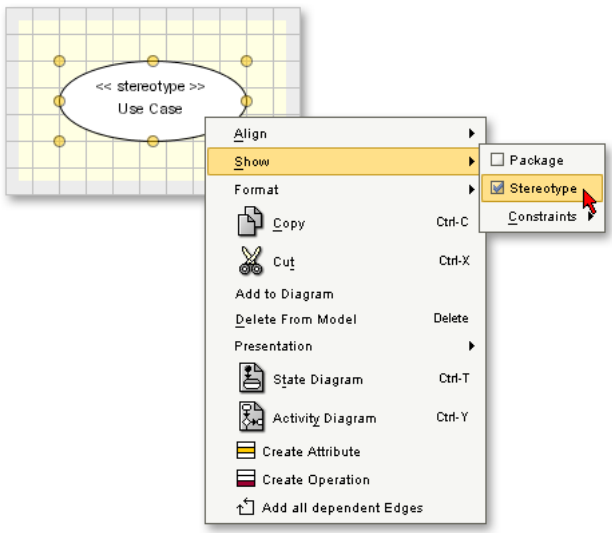
13.2.3. Editing Via the Context Menu

The Context menu can be accessed by right-clicking on an element in a diagram. Entries relevant to the selected element are displayed. Remember that things like attributes and operations are considered elements in their own right; therefore, the context menu for an attribute will be different than that for the class in which it occurs. If you do not see what you expect, be sure that you have selected the proper element to be the active element.

The Show option displays all checked items in the diagram. In the case of a class element, this includes stereotype, package, and compartment options. Unchecked items remain hidden from view.

It is also possible to create things like attributes, operations, and dependent edges when appropriate. These items are listed towards the bottom of the context menu and, once created, are available for editing in the Properties tab.

Figure 13-9. Context menu options for a Use Case



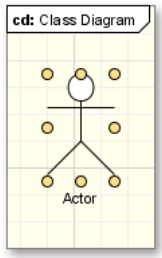
13.2.4. Graphic Representations

Elements may be represented by imported graphics instead of the default UML representation. This can be particularly useful in diagrams such as the Use Case diagram, where actors can be displayed with various photographs rather than the usual 'stickman', or in Deployment diagrams where various nodes may be better represented by graphics rather than the standard cube.

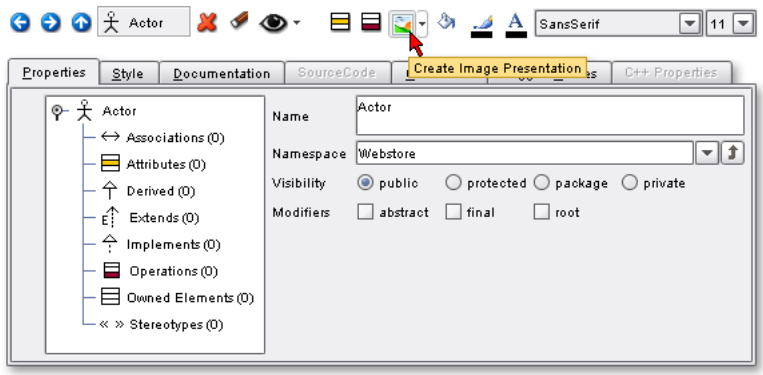
Elements that may be represented by graphics: Class, Package, Actor, Interface, Collaboration, Usecase, Object, Lifeline, Component, Artifact, Instance specification, and Node.

To add an image representation:

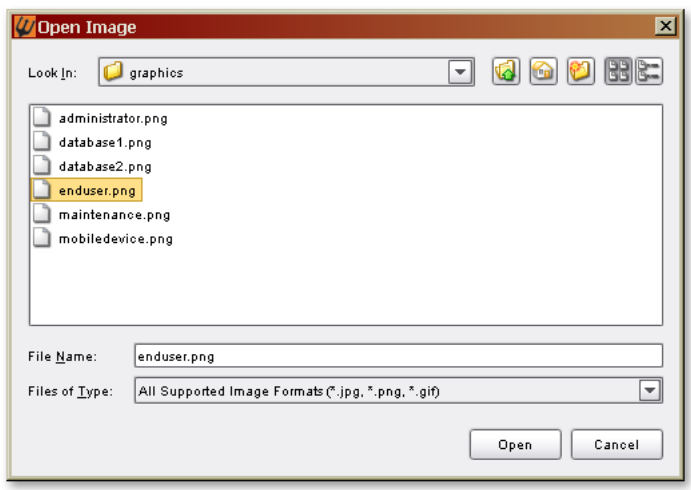
1. Select the desired element in the diagram.



2. Click the 'Create Image Representation' button from the Properties tab of the Details pane.



3. Select the desired image file.



4. The selected image will be placed in the diagram in place of the default representation.

The following file extensions are supported: .jpg, .jpeg, .eps, .epsi, .epst, .wbpmp, .wbm, .bmp, .svg, .ps, .pdf, .gif, .png.

13.2.5. Undo/Redo

Sometimes when working with your models, you might have done something you did not really intend to do. If this happens, the ability to revert your work can be very valuable. Poseidon for UML offers such an undo mechanism. The Undo function is not limited to the last change you made - you can undo all the steps you took prior to that, and you can even redo the things you just undid.

To Undo or Redo actions :

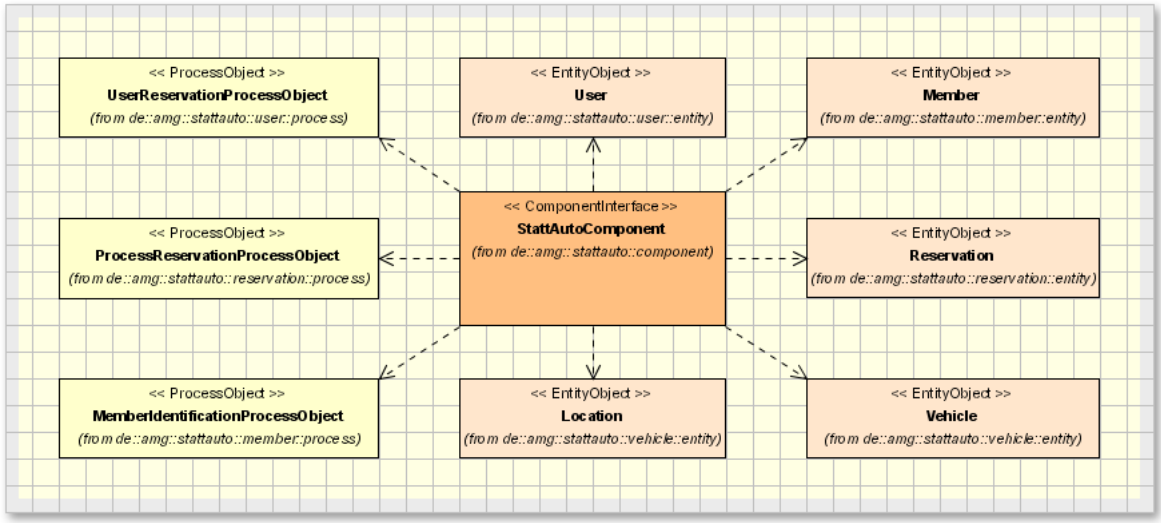
- **Main Menu** - Select Undo or Redo from the Edit menu
- **Main Toolbar** - Click the Undo or Redo button on the main toolbar
- **Quick-Keys** - Use the quick-key Ctrl-Z for undo or Ctrl-Shift-Z for redo

13.2.6. Stereotypes

One of the general patterns of an architecture is the **Model-View-Controller-Pattern** , or the ***Boundary-Control-Entity-Schema*** , as it is often rephrased in the UML community. According to this, an architecture is constructed in three layers.

First, the **Boundary** is responsible for representing information to the user and receiving his interactions. Users of the system interact with this layer only. The next layer, **Control** , contains the rules on how to combine information and how to deal with interaction. It is responsible for transferring control based on the input received from the Boundary layer. And finally, the **Entity** layer holds the data and is responsible for its persistence. To which layer a class belongs is expressed using corresponding stereotypes. You obtain these in the Properties tab of each class. An example for the usage of stereotypes is shown below.

Figure 13-10. A Class diagram using stereotypes.



The code generation functionality of Poseidon for UML can distinguish between different stereotypes for the same element type. In this way it can select the appropriate template for generation based on both of these factors. Stereotypes can be displayed for nearly every element type.

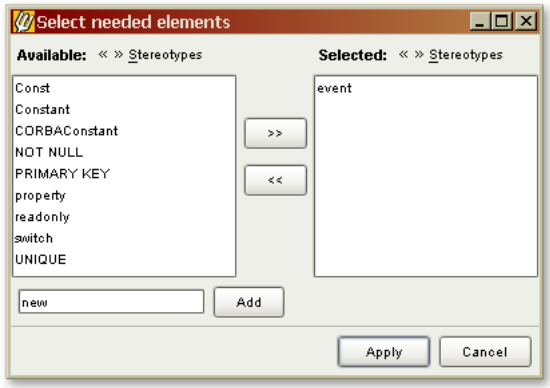
Poseidon supports multiple stereotypes for single elements. Adding, editing, and removing these stereotypes is accomplished via a dialog that is accessible from the Details pane.

To access the Stereotype dialog:

1. Select the element the stereotype applies to from the diagram, Details pane, or Navigation pane.
2. Open the Properties tab for this element in the Details pane.
3. Click the ellipsis button [...] to open the stereotype dialog.

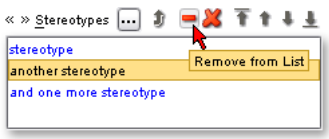
   Note that the previous method to access the dialog (right-click in the stereotype field and select 'Edit' from the menu) still works.

Figure 13-11. Stereotype dialog

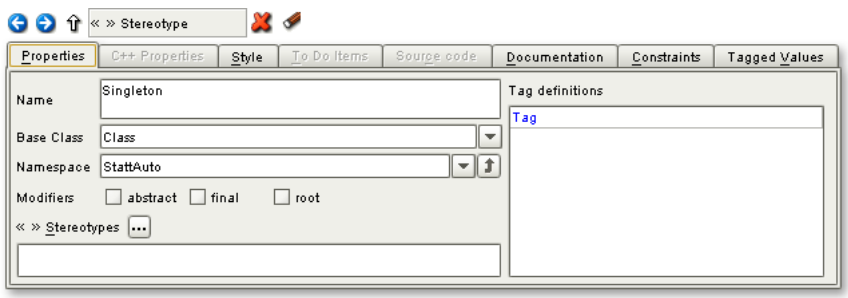


Once this dialog is open, altering and applying stereotypes is quite simple. The buttons with the arrows allow you to add and remove stereotypes from the element. The 'Add' box below the list of stereotypes will create new stereotypes, but will not automatically add them to the element. Removal of stereotypes from an element is only possible through this dialog.

Selecting a stereotype and clicking the delete button will remove the stereotype from the model completely, not just from the selected element. To remove a stereotype from the element to which it is applied but leave the stereotype in the model, click the 'minus' button  .

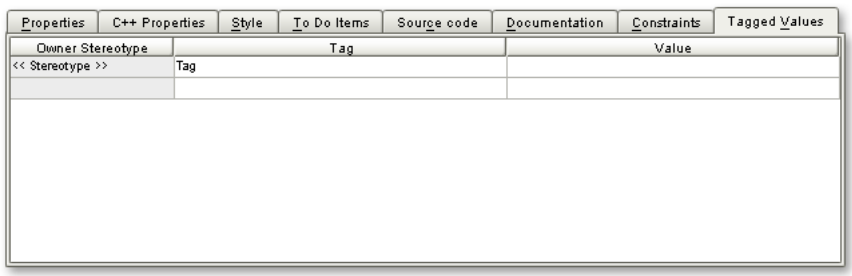


13.2.6.1. Stereotype Properties



- **Name** - Name of the stereotype
- **Base Class** - The stereotype is intended for this type of element
- **Namespace** - Namespace of the stereotype
- **Extends** - Indicates a parent stereotype, if necessary
- **Modifiers** - Additional characteristics of the stereotype
- **Tag Definitions** - Tags defined here will be included in any element to which the stereotype is applied

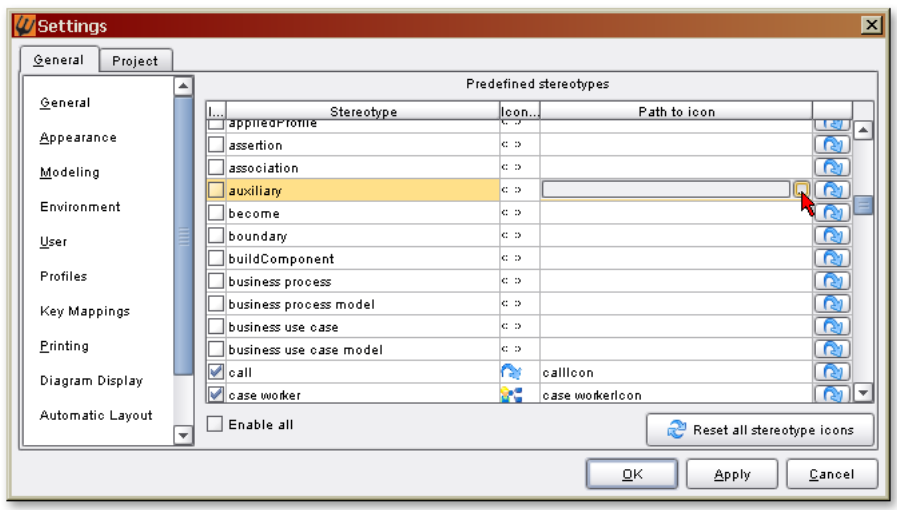
Below illustrates a Tagged Values tab for a class with the above pictured 'Stereotype' applied.



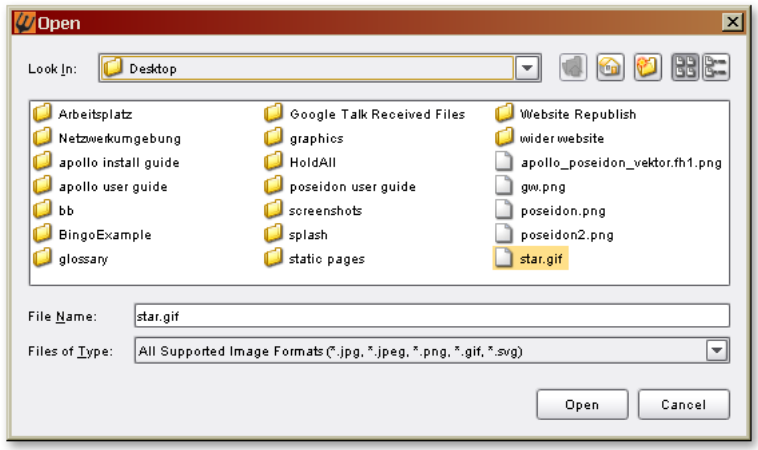
13.2.6.2. Graphical Representations

When stereotypes are displayed in an element, they may also include a graphic. The graphics are assigned to the stereotypes from Settings dialog (Edit -> Settings -> Stereotype Icons).

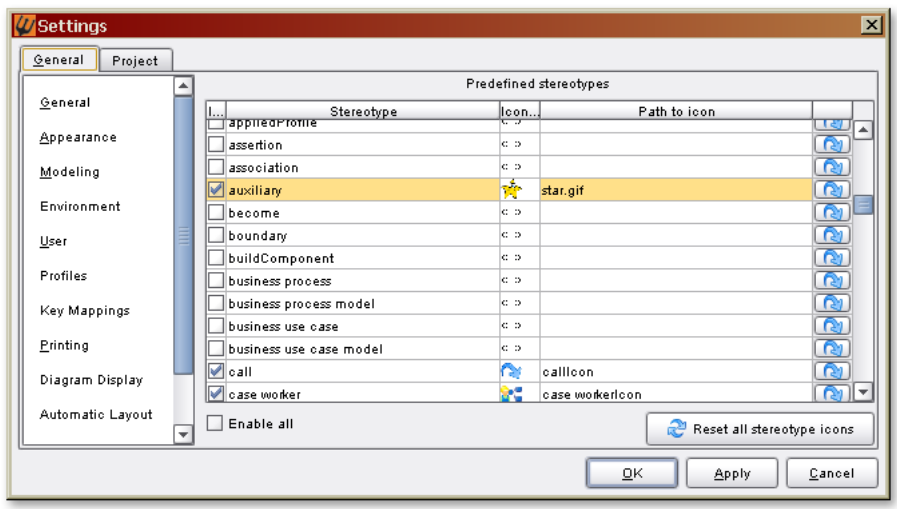
1. Open the file chooser from the 'Path to Icon' column.



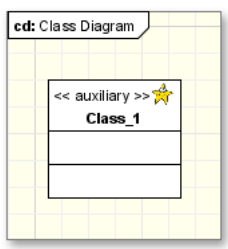
2. Navigate to the desired file and click 'Open'.



3. The Settings dialog will display a preview of the icon.



4. The stereotype icon will be displayed along with the name of the stereotype.



13.2.7. Copying, Cutting, and Pasting Elements

Copying a class from one diagram to another is simple, right? Select the class, copy, paste... but what happens to the class if the name of the class is changed, or an attribute is added, or any of a host of other possible edits are made? Beginning with Poseidon 3.1 the answer is, 'That depends.'

Prior to 3.1 (and in the Community Edition), the copy function merely copied the *representation* of that element. There still existed only one of that element in the model. To illustrate, a model is created with a single class diagram consisting of a single class. The Model Index view and Diagram Centric view each show one class in the model. A new, empty class diagram is created, and the class is copied to the new diagram. Now the Diagram Centric view will show two classes - one in each diagram. But the Model Index view will still show only one class - this single class is represented in two different diagrams. Any changes made to this single element, regardless of the diagram in which it appears, will be reflected in all representations of the element.

Poseidon 3.1 introduces the concept of 'complete copy', where the element itself is copied. Following the previous example, if the class were complete copied and then pasted to a new diagram, the Model Index would now show two classes. At this stage, the classes are identical, but because they are completely separate entities, changes to one class will not affect the other.

'Cut Representation' and 'Complete Cut' follow the same scheme as the copy functions.

> **Note:** Complete Copy, Complete Cut, and Clone Model are not available in the Community Edition.

13.2.7.1. Elements

To cut a model element:

- The 'Complete Cut'  button
- The quick-key combination 'Ctrl-X'
- The context menu option 'Cut Complete' for the selected element
- The Edit menu option 'Cut Complete'.

> **Note:** Complete Cut is not available in the Community Edition.

To copy a model element:

- The 'Complete Copy'  button
- The quick-key combination 'Ctrl-C'
- The context menu option 'Copy Complete' for the selected element
- The Edit menu option 'Copy Complete'.

Note: Complete Copy is not available in the Community Edition.

13.2.7.2. Element Representations

To cut *only the representation* of an element:

- The 'Cut'  button
- The quick-key combination 'Ctrl-Shift-X'
- The context menu option 'Cut' for the selected element
- The Edit menu option 'Cut'.

To copy *only the representation* of an element:

- The 'Copy'  button
- The quick-key combination 'Ctrl-Shift-C'
- The context menu option 'Copy' for the selected element
- The Edit menu option 'Copy'.

Premium editions of Poseidon for UML also allow you to clone diagrams with one click. See Section 11.1.1 .

13.2.8. Removing and Deleting Elements

With drawing tools like Visio or Powerpoint, deleting an element from a diagram simply removes the figure from that single location. With full-blown UML modeling tools this is different. You are always working on a single, consistent model. The different diagrams and the elements contained within them are just components of views rendered from this single model, even if the diagrams are constantly used as a means to change the model. The consequence of this is that modifications to any element within a diagram are applied to the element, not to the diagram. As such, a change made to the element will be seen throughout the entire model.

It then follows that selecting an element and then pressing delete means that the element itself is deleted, meaning that it no longer exists within the model and is removed from all aspects of the model, including other diagrams. Additionally, all connections to other elements, such as associations or inheritances, are completely removed. Note that there is a big difference between deleting an element from a model and removing an element from a diagram.

This leads us to use different terminology with different meanings: You can **delete an element from the model** , which means that the element is removed entirely and is no longer available in the Navigation pane or in any of the diagrams, or you can just **remove its figure from the current diagram** you are working with, leaving the element available to the rest of the model. These are very different things, and different commands are used to achieve them.

To completely remove an element from the model:

- Use the delete button in the Properties tab
- Select an element or part of an element in the diagram and hit 'delete' on the keyboard
- Select 'Delete from Model' in the Context menu
- Use the delete rapid button

To remove an element's representation from the current diagram:

- Select an element or part of an element in the diagram and use Ctrl-X to cut the item
- Select 'Cut' from the Context menu
- Hold the Ctrl button while mousing over the element, then use the 'Remove from Diagram' rapid button .

The element, as part of the model, remains untouched in other diagrams and it also remains in the tree in the Navigation pane. For elements that are connected to other elements through, for example, an association or inheritance, removing the first element (e.g. a class) means that the association is no longer valid; therefore, the second element (e.g. the association) is also removed from the diagram, but is likewise still accessible from the navigation pane or other diagrams.

If you want to remove an element but not the connections it has to other elements, you can detach it by selecting the connection and dragging the handle at the end of it to another element *before* you remove the element.

Chapter 14. Generation

UML wouldn't be worth all the sophisticated work if all it came down to was pretty vector graphics. When analyzing and designing a software system, your final goal will be to generate well-implemented code.

Poseidon for UML provides a very powerful and flexible code generation framework, based on a template mechanism. It is currently used to generate a variety of code types including Java and HTML, but it is flexible enough to generate any kind of programming language, or other output, such as XML.

14.1. Code Generation

Java code generation is usually based on the classes of a model and other information displayed in the respective Class Diagrams. Additionally, Poseidon can generate setter and getter methods for the fields of each class.

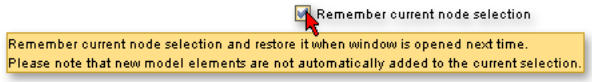
By default, associations between classes in UML are bi-directional; that is, associations allow navigation between classes in both directions. For the common object-oriented programming languages, these need to be transformed into separate uni-directional associations. If one of these is set, the other should be set accordingly. The code for managing bidirectional as well as unidirectional associations is also generated automatically.

14.1.1. Generation Settings

Code and documentation generation are both invoked from the Generation menu.

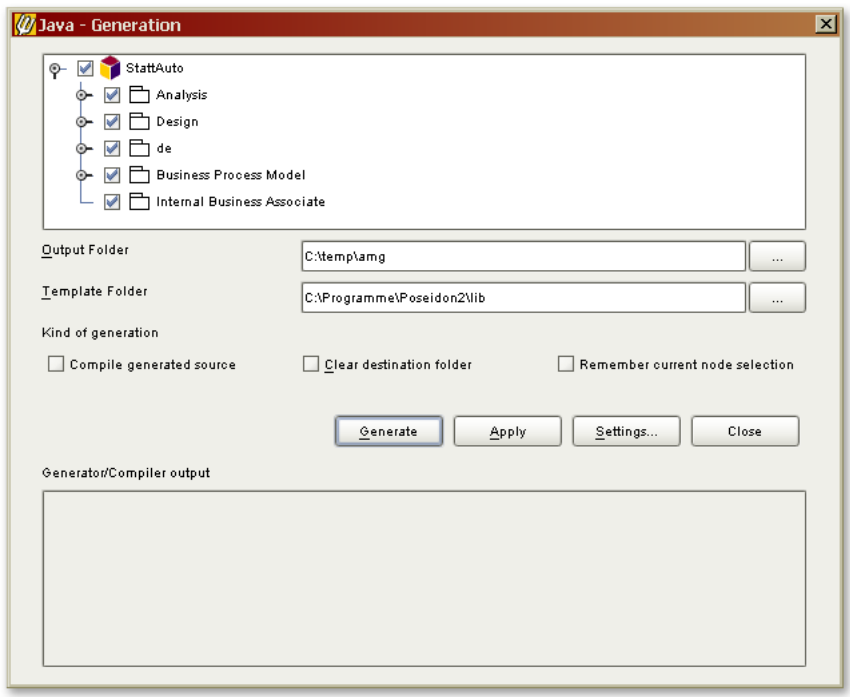
Figure 14-1. Generation menu

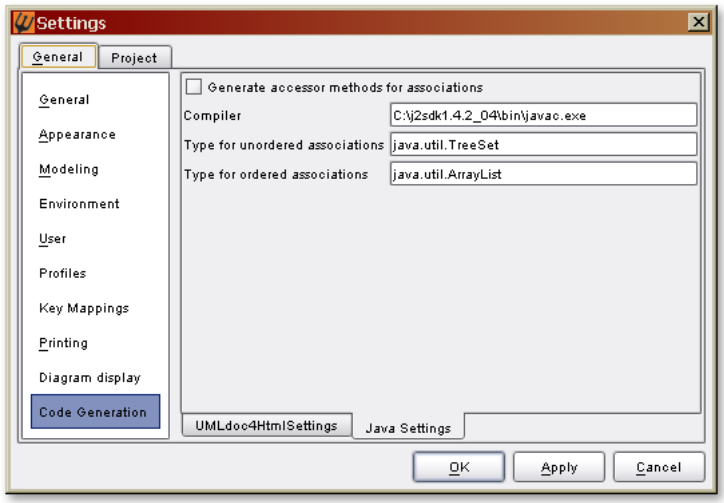
Select the type of generation you would like from the Generation menu and a dialog will appear. Here you can select or deselect model elements from the tree, specify an output and a template folder, and indicate if the destination folder should be cleared. Poseidon can remember your selection of model elements if you enable the checkbox titled, 'Remember current node selection'.



Note that the output and template folders are saved by project and language. This means that output of Java generation can be placed in one folder, and Perl generation from the same project can be placed in another folder. These folder preferences will be saved to the current project, but another project will not recognize these preferences and must have its own preferences set. This is to avoid accidentally overwriting the output from a previous project.

Figure 14-2. Code Generation dialog and settings - Java





You'll find the generated Java files in the specified output folder, sorted by packages.

14.1.2. Reverse Engineering

Software engineers often run into the problem of having to re-engineer an existing project for which only code and no models are available. This is where reverse-engineering comes into play; a tool analyzes the existing code and auto-generates a model and a set of Class diagrams for it.

Poseidon for UML can do this for Java programs, where source code is available and is in a state where it can be compiled without errors. With the corresponding JAR Import function (available only in the Professional and Enterprise editions), it even works with JAR files of compiled Java classes.

To launch this process, go to the Import Files menu and direct the file chooser to the sources' root package. It will then analyze this as well as all sub-packages. The outcome is a model containing the corresponding packages, all the classes, their complete interface, their associations, as well as one Class Diagram for each package. Note that the path that you select here will be automatically adopted by the generation dialog. The next time you open the code generation dialog, this path will be displayed as output folder by default.

If the imported file uses classes that are part of the JDK, these classes will be created in the model as required, so you may see some apparently empty classes in the package `java` in your model. This is of no concern and is done solely to have a smaller model. But these classes are necessary to have a consistent project. All classes that the imported files use must be present in the model.

Additionally, from the settings dialog you can give an import classpath. This is necessary to make the

model complete if a file references classes that are not imported themselves. Here you can specify one or more jar files, each entry separated by a colon. If you want to import files that make use of `foo.jar` , `anotherfoo.jar` and `stillanotherfoo.jar` , then it should look similar to this:

```
folder/subfolder/foo.jar:anotherfolder/anotherfoo.jar:stillanotherfoo.jar
```

Classes that are needed to make the model complete but are not present in the package structure are created on demand. If you give an import classpath but the imported file does not use any classes from it, then no additional classes will show up in your model.

14.1.3. Roundtrip Engineering

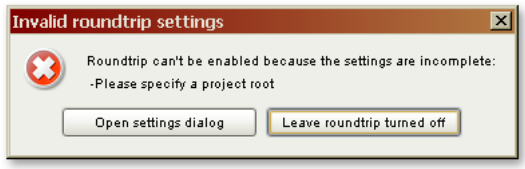
Generating code and reverse engineering that same code still does not make round-trip engineering. Reverse-engineering generates a new model from existing code, but it does not by itself reconnect the existing code to an existing model. This feature is only available in the Professional and Enterprise Editions, which contain the RoundTrip UML/Java Plug-in. It is one of the most recommended and highly sophisticated features provided by Poseidon for UML.

Generate a UML model from your existing code, change the model, re-generate the code, change the code and so on. All generated Java code files are watched, so that changes you have made with an external editor are imported into Poseidon's model of your project. Use your favorite source code editor to edit method bodies, add or remove methods and member variables. Poseidon keeps track of all changes, and all your project data is in one place - in Poseidon.

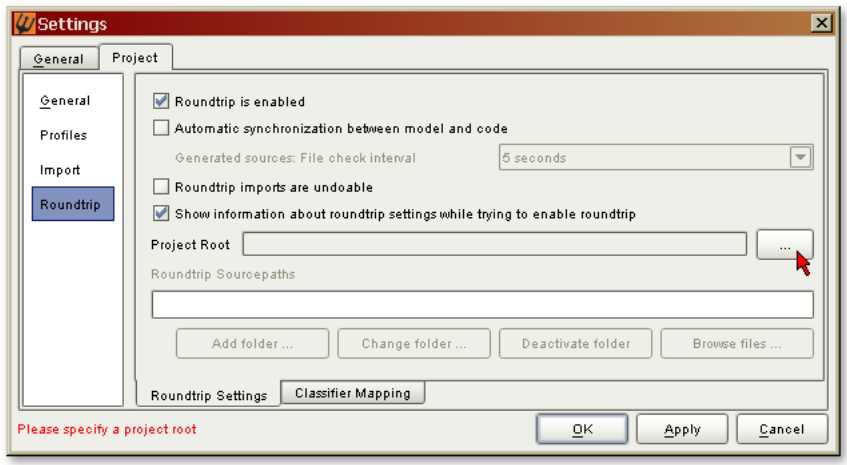
Please note that the round-trip plug-in is primarily an import tool; it imports changes in the source code for you and updates the model as necessary. Automatic code generation in the background is not yet implemented, but is planned for a future release.

14.1.3.1. Using Roundtrip

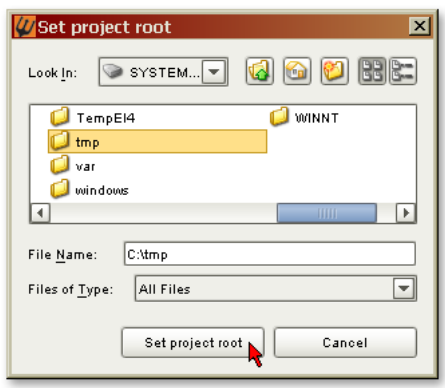
1. Create a model with Poseidon, or import sources into Poseidon. See Section 9.4 for more details on importing.
2. Enable roundtrip by clicking on the roundtrip button  . The button will turn from red to green. If roundtrip is already enabled, the green roundtrip button  will appear in the toolbar, and there is no need to click the button. The 'Invalid Roundtrip Settings' dialog will appear because no project root has been set yet. This is normal and expected. Click 'Open Settings Dialog' to set the project root.



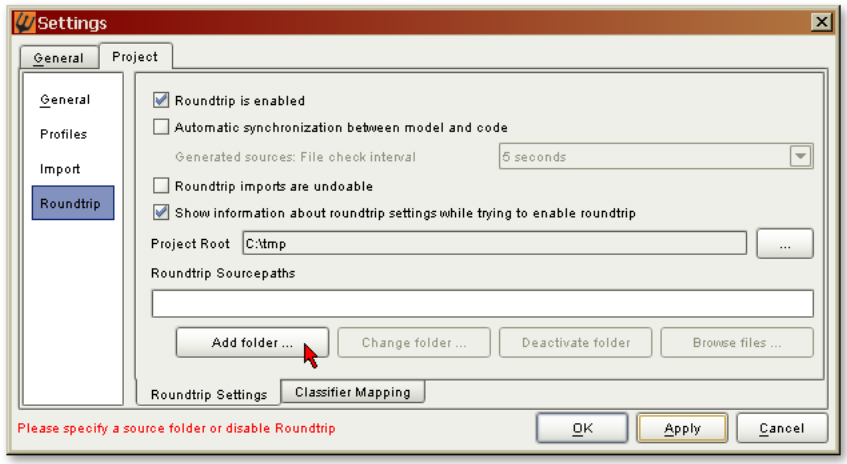
3. Set the project root by clicking on the ellipse button and navigating to the desired project root directory.



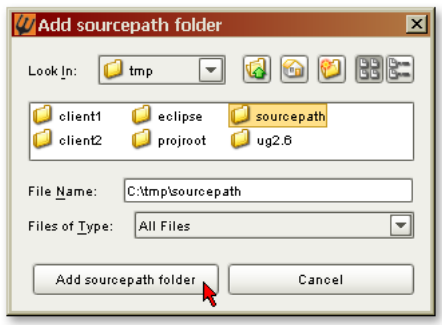
Once you have located the proper directory, click the 'Set Project Root' button. Note that soft links cannot be used here. Click 'Apply' in the Settings dialog.



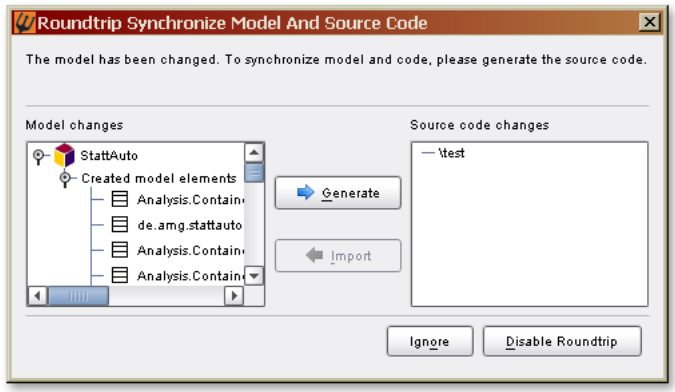
4. The message in the lower left corner of the Settings dialog will now prompt you to set one or more source paths in order to tell Poseidon where to store the generated code. This directory must be empty, or you will encounter an error message. Click the 'Add Folder...' button.



Once you have located the proper directory, click the 'Add Sourcepath Folder' button. Note that soft links cannot be used here. Click the 'Apply' button in the Settings dialog. Repeat this for all source paths you would like to add.



5. Poseidon will now check to see if the source and the model are currently in synch. If you have opened an existing project or added elements to a new project and then enabled roundtrip, there will be no corresponding source code so you will be prompted to generate this code.



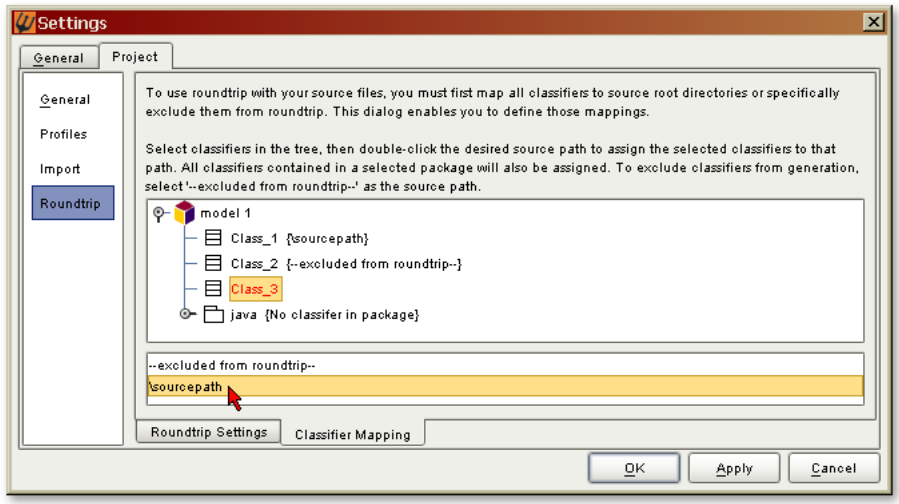
The synchronization dialog shows any new element that has not been generated yet, even if this element has been previously imported. This means that it is possible for this dialog to open more than once, for instance if you are reloading a modified project.

This dialog is always available from the toolbar.

6. Classifiers must be assigned to the source paths or explicitly set to be excluded from roundtripping before code generation can begin. Click on the 'Classifier Mapping' tab of the Settings dialog.

   Highlight a classifier by clicking on it in the upper field (or hold the control key while clicking to highlight more than one classifier at a time), then double-click the source path to which you would like to assign the classifier from the bottom field. Selecting a package inherently also selects the classifiers contained within that package. Once a classifier has been successfully assigned, the color of the name will change from red to black.

   You can also reassign a classifer by following the same procedure, regardless of whether the color of the name is red or black.



14.1.3.1.1. Edit Source Files

After you have enabled roundtrip, you can edit your source files and then import the changes into your Poseidon model.

1. Edit your source files as you normally would, for instance from a text editor or IDE.
2. Add the changes by clicking the Roundtrip Import button  or use the roundtrip import keyboard shortcut **Ctrl-Alt-I** in order to integrate the changes from the sources into your Poseidon project. It is important to note that this is completely separate from the regular import function. At this point, you will not be able to use the normal import for this project. An error message will appear if you attempt to import files from the roundtrip directory.

   If there are errors such as a missing classes during the import, you will encounter an error dialog alerting you to the problem.

3. After the import of source files, it is a good idea to then regenerate the code as described in Section 14.1.3.1.2 so that Poseidon identifiers are added where needed. For example, if you add an attribute to a class and import the code, the source code tab within Poseidon will display the Poseidon object

ID (which looks something like `@poseidon-object-id [Ilsmma581]`), but the ID will not be added to the code until explicitly told to do so.

You might temporarily have non-compiling source code that you do not want to import into Poseidon right away. For these instances, you can temporarily disable the automatic import with the button next to the `Import sources` button. It will turn to red, showing that automatic round-trip is disabled.

By clicking it again, it will turn back to green, designating that round-trip is enabled again.

14.1.3.1.2. Edit the Model

You may also choose to edit your model and have the changes reflected in your source code.

Poseidon will not generate new source code until it is explicitly told to do so. Edit source code from the Source Code tab in the Details pane, then click the 'Start Roundtrip Code Generation' button  from the toolbar or generate code with the roundtrip generation keyboard shortcut **Ctrl-Alt-G** . Only changed classifiers that have not been excluded from roundtripping will be generated. You can override this and generate all classifiers by holding the Ctrl button while clicking the code generation button.

Poseidon renames a class by generating a new class and deleting the old file.

14.1.3.1.3. Edit the Model and the Source Code Simultaneously

If both the model and the source code have changed, the Synchronization Dialog can be used to update the model and source. Access this dialog by clicking the 'Synchronization' button  on the toolbar. Once the Synchronization dialog has opened, you can select to update the model based on the code or to generate new source code based on the model. Any conflicts will be detected at this point, and you will be asked whether the model or the source code should take precedence. For more information about conflict resolution, see Section 14.1.3.2 .

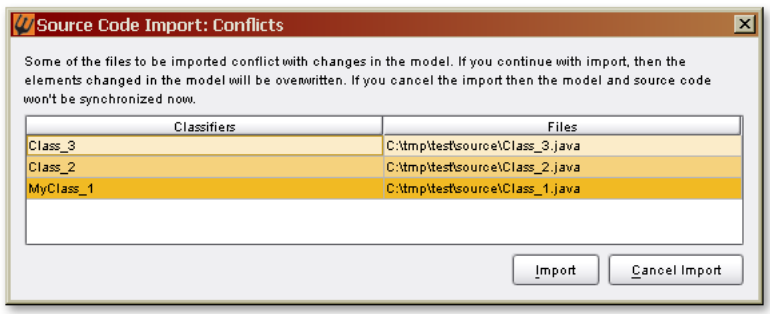
When generating source code from a model, Poseidon adds identifiers so that it can intelligently keep track of changes, therefore it is recommended that you first update the model and then generate the code. While it is possible to do this in reverse order, you may find that you have to generate the code twice in order to complete the synchronization.

Alternatively, you can import the source code as described in Section 14.1.3.1.1 and then generate the model for that code as noted in Section 14.1.3.1.2 without using the synchronization dialog..

14.1.3.2. Conflict Resolution

Import Conflicts

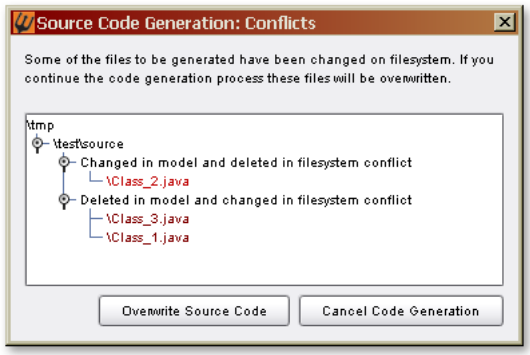
When using Roundtrip Engineering, you may encounter conflicts when trying to import source code that conflicts with the model. For instance, perhaps a class has been deleted from the model but still exists in the source code. Poseidon will alert you to these conflicts and allow you to decide whether or not to continue the import.



If you click the Import button, the model will be updated with the information from the source code. For example, if a class has been deleted from the source code, it will also be deleted in the model. A class that has been deleted from the model but still exists in the source code will be added back into the model.

Code Generation Conflicts

The case may also be that conflicts are encountered when generating code from a model that conflicts with the existing source code. In this case, however, changes are applied from the model to the source code. For instance, if a class has been deleted from the source code but still exists in the model, the class source code will be regenerated. A class that has been deleted from the model but still exists in the source code will be deleted from the source code.



Clicking 'Cancel Code Generation' will result in the model and the source code remaining out of synch.

Refactoring Warning

Poseidon keeps track of modifications you make to your code, but some changes like renaming attributes in the source code may render Poseidon unable to update references to these attributes. As a result, your code may not be compilable after this change takes effect. Poseidon helps you avoid this by presenting you with a warning.

This dialog may be turned off by checking 'Do not show this dialog again' in the dialog itself or uncheck 'Warn when doing changes to the model that might harm generated sourcecode' from Settings | General | Modeling.

14.1.3.3. Accessor Methods

You should unset the check box 'Generate accessor methods' after you have generated accessors once. Otherwise, they would be generated again, and would clutter up your classes. The preferred way to create set/get methods is by adding them in an attribute's Properties tab, and by checking 'Create accessor methods for new attributes' in the dialog Edit | Settings.

14.1.4. Fine Tuning Code Generation

There are several possibilities to fine-tune the appearance of the generated Java source code. Among them are the creation of accessor methods for attributes, the types for collection attributes, and the list of import statements in the files.

- **Accessor Methods**

  From Poseidon 1.4 on, you can create accessor methods for attributes automatically. This way, you can fine-tune the code so that some attributes have accessors, some not. In previous versions of Poseidon, you could only have setters/getters for all attributes, or for none.

  In Edit - Settings , there is a check box called 'Generate accessor methods for attributes'. Check this box to have accessor methods created for every attribute that is created. If the attribute has a multiplicity of 1..1 or 0..1, two simple `getAttribute()` and `setAttribute()` methods are created. For attributes with a finite multiplicity, an array is generated, and the accessor methods include `addAttribute()` and `setAttribute()`. For an unbounded multiplicity, a Collection is generated, and the appropriate access methods like `addAttribute()` and `removeAttribute()` are produced.

  You can fill the bodies of these access methods according to your business logic. Also, you can hide the display of accessors by setting the check box 'Hide accessor methods' in Edit-Settings-View .

  Additionally, you can generate the standard accessor methods for your attributes at code generation time. These will be visible only in the generated code, not in your Poseidon project.

- **Collection Types**

  Poseidon up to version 1.3.1 used the type Vector whenever an association had a multiplicity of ..*.

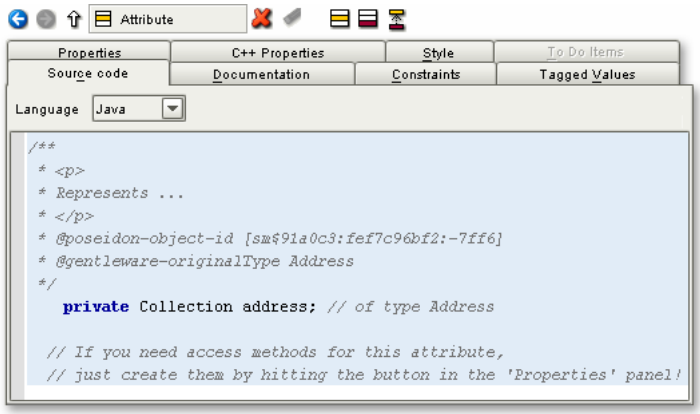
  From version 1.4 on, the rules are:

  1. Create an attribute of the element's type if the multiplicity is 0..1 or 1..1.
  2. Create a `Collection` type attribute if the multiplicity has an upper bound of *.
  3. Create an array of the element's type if the multiplicity has an upper bound that is not 1 and not * (that is, it is a number).

  In `Code Generation-Settings,` you can define what type of collection should be used for Collection types. The default is `ArrayList,` but you can enter any type (e.g., Vector) that implements `Collection.` Accessor methods are programmed against `Collection` .

  Beginning with version 2.6, attributes with an unlimited upper bound now use a JavaDoc tag called `@gentleware-originalType` to note the original type of the attribute that has been replaced with

the Collection type. This enables the roundtrip and import functions to determine the original type. Note that this tag can be added manually, just like any other JavaDoc tag.

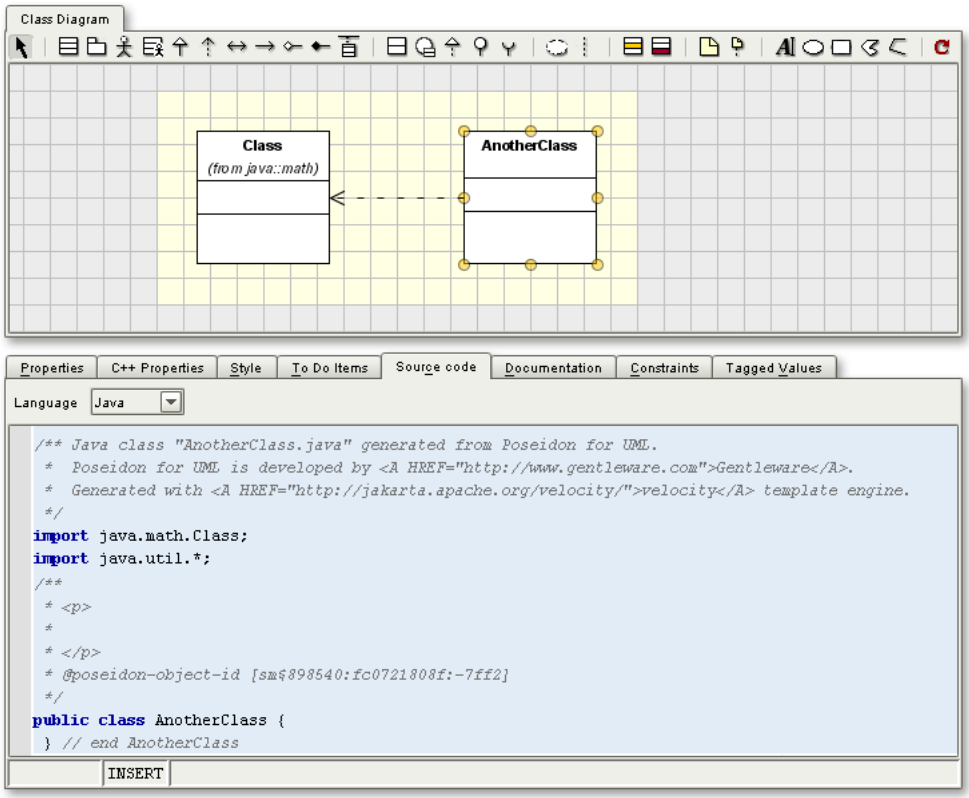


In this version of Poseidon, you are now able to distinguish between ordered, unordered and sorted attributes, and you will be able to give different kinds of implementation types such as `TreeSet` for unordered, `ArrayList` for ordered and `Vector` for sorted attributes.

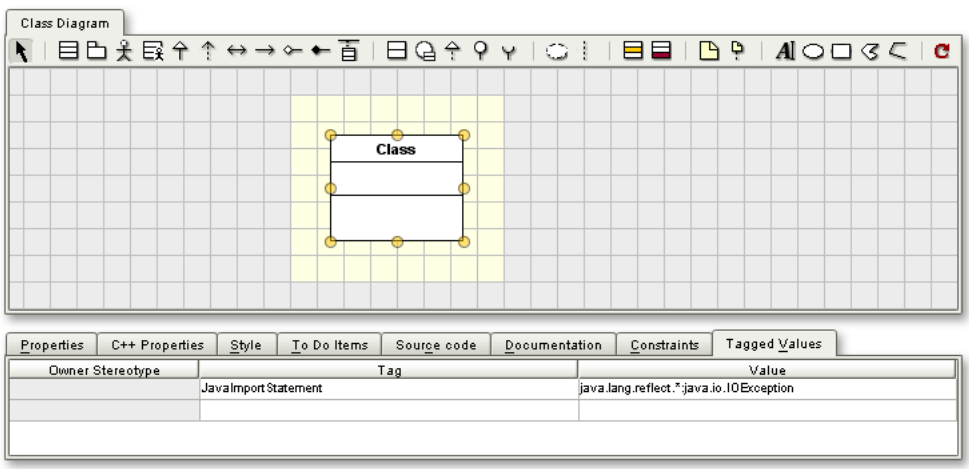
- **Import Statements**

  Import statements can be added to classes in two ways: By drawing dependencies or by entering tagged values.

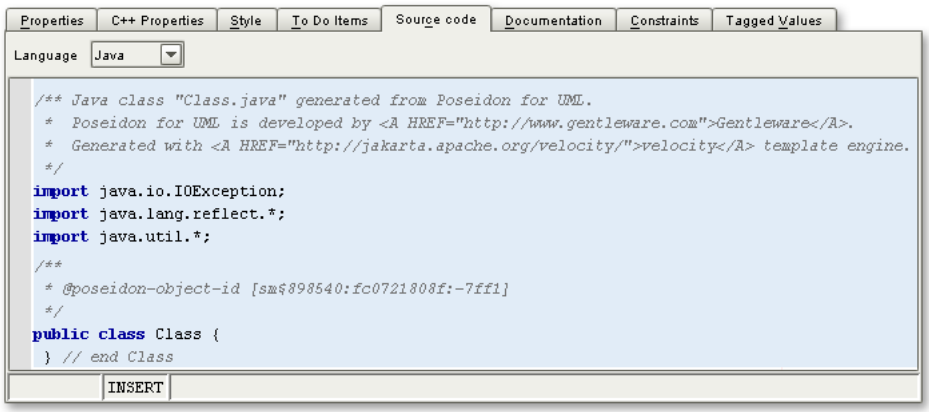
  The graphical way is to draw a dependency from the class to the class or package that you want to import. An appropriate import statement will be generated: Either `import package.*` or `import package.Class` .



The second way (that does not clutter up your diagrams) is to add a Tagged Value called `JavaImportStatement` to the class. Then enter a number of imports, separated with colons. Qualified names can be given in Java syntax. For example, import `java.lang.reflect.*` and `java.io.IOException` by setting the tagged value `JavaImportStatement` to `java.lang.reflect.*:java.io.IOException`.



Above, you can see the import statement in the Tagged Values tab, and below is the resulting Java code generated by Poseidon.



- **Modifying Templates** (Not available in the Community Edition)

  More advanced customizaton of the generated code is possible if you are using one of the Premium Editions. Modification of the templates that are used for code generation is possible with these editions. We cover this topic briefly in a separate chapter and more deeply in a separate document that is distributed with these editions and online under http://www.gentleware.com/index.php?id=174 (http://www.gentleware.com/index.php?id=174) .

- **Javadoc Tags**

  You may not want certain operations to be reverse engineered. Any operations with the Javadoc tag '@poseidon-generated' will be excluded from the reverse engineering process.

14.2. Advanced Code Generation

This chapter describes the code generation functions offered by Poseidon for UML and the options for customizing the code generation templates. Code generation based on standard templates is available in all editions of Poseidon for UML. The standard templates define code generation for Java and HTML. With the Developer and Professional Editions, you have the option of changing the code generation templates to suit your specific requirements. You can even create new templates to generate code for a different programming language such as C#.

14.2.1. Velocity Template Language

Code generation in Poseidon for UML is based on the Velocity Template Language. Velocity is an open source template engine developed as part of the Apache/Jakarta project. Originally designed for use in the development servlet based Web applications, it has also proved to be useful in other areas of application including code generation, text formatting and transformation.

The Velocity Template Language (VTL) supports two types of markup elements: references and directives. Both references and directives can be intermixed freely with the (non-VTL) content of a template, as shown in the examples below.

Since templates have been widely used in the field of Web page generation, we will begin with a simple HTML example. The second example demonstrates the use of VTL to generate Java code.

For further information on Velocity - including complete documentation of the Velocity Template Language - please go to the Velocity Web site at http://jakarta.apache.org/velocity/ (http://jakarta.apache.org/velocity/) .

14.2.1.1. References

References are variable elements referring to some entity provided by the context. A reference such as `$userName` or `$userList` can be used to access and store a particular data structure for use within a template; thus, references establish the connection between a template and the context of the Velocity engine.

Within a template it is possible to create a new reference at any time and to assign a value to the new reference. This is done using the `#set` directive (see directives). This means you can add references to the active context as required. If a reference name is used within a template for which no corresponding object or value exists in the active context, the reference name is treated as plain text, i.e. it is output "as is" just like the other (non-VTL) elements of the template.

Every reference must have a unique name. The name (also known as the VTL identifier) begins with a dollar sign `$` followed by a string of characters as described in the following table:

$	dollar sign - the dollar sign must be the first character in the reference name, and it may not occur in any other position.
a-z, A-Z	alphabetic characters - only standard characters are allowed, no accented or diacritical characters. The first character following the dollar sign must always be an alphabetic character.
0-9	numerical characters
-	minus sign (hyphen)
_	underscore

A regular expression describing the reference name syntax would be:

```
$[a-zA-Z][a-zA-Z0-9_/-]*
```

In addition to referencing variables, it is also possible to specify attributes and methods by means of the VTL reference syntax. Using references such as `$item.name` and `$item.price` , you can dynamically insert the attributes associated with the specified object. Likewise, you can access the methods of a referenced object (for example a Java object) using a reference such as `$item.getNameAsString()` . This will return the result of applying the given method to the specified object.

Taking this one step further, you will find that the standard Java templates supplied with Poseidon for UML make extensive use of the following syntax:

```
#set ($name = $currentOp.getNameAsString())
```

Here the reference `$name` is dynamically set to the string returned by the method `$currentOp.getNameAsString` . This use of references to elements of the context establishes a very powerful connection between the templates and the template API.

14.2.1.2. Directives

Directives in VTL are a defined set of commands that can be used for basic control functions within a template. For example you can us the directives to create typical procedural branches (if/else) and loops (foreach).

The current set of VTL directives comprises the following commands:

#set()	function for assigning a value to a reference
#if() #else#elseif()#end	common conditional functions used for branching
#foreach()#end	looping function
#include() #parse()	functions for including code from another template or static resource
#macro()#end	function for defining a reusable set of commands

For complete information on the use of these directives please refer to the Velocity documentation (see http://jakarta.apache.org/velocity/ (http://jakarta.apache.org/velocity/)).

14.2.1.3. Comments

Particularly in the case of templates used for code generation it may be advisable to use comments in the

templates to explain their use. Comments can be added to a template by means of the following syntax:

## Single line comment ...	The comment continues up to the end of the line. This is comparable to the syntax for single line comments in Java or C beginning with g.
#* Inline or multiline comment *#	The comment continues up to the closing character *#. This is comparable to the syntax for inline and multiline comments in Java or C beginning with /* and ending with */.

The use of comments in VTL is illustrated by the examples below.

14.2.1.4. Examples

Example 14-1. Simple HTML Template

This example uses VTL markup intermixed with HTML code to generate dynamic Web pages based on information retrieved from the context (e.g. from a database).

```
#* This is an example of a simple VTL template
                                for generating dynamic HTML pages. *#

                                <HTML>
                                <HEAD>
                                <TITLE>Holiday Weekend</TITLE>
                                </HEAD>

                                <BODY>
                                <B>$roomList.size() rooms available
                                at special holiday weekend rates! </B>
                                <BR>Check in for a luxurious holiday
                                weekend at these amazing prices.
                                <BR> Choose from: #set( $count = 1 )

                                <TABLE> #foreach( $room in $roomList )
                                <TR>
                                <TD>$count)</TD>
                                <TD>$room.type</TD>
                                <TD>$room.price</TD>
                                </TR> #set( $count = $count + 1 ) #end
                                </TABLE>

                                <BR> Call today for a reservation.
                                Toll free number: $freePhone
                                </BODY>
                                </HTML>
```

This example makes use of VTL references and directives to generate an HTML page based on data from an external data source, for example a database. The data source is referenced by means of the elements

`$roomList` and `$room` (with its attributes `$room.type` and `$room.price`). When this template is applied, the directives and references are interpreted and the results are inserted into the generated HTML code.

The resulting HTML page might look something like this:

```
<HTML>
                                    <HEAD>
                                    <TITLE>Holiday Weekend</TITLE>
                                    </HEAD>

                                    <BODY>
                                    <B>3 rooms available at special
                                    holiday weekend rates! </B>
                                    <BR>Check in for a luxurious holiday
                                    weekend at these amazing prices.
                                    <BR> Choose frome:

                                    <TABLE>
                                    <TR>
                                    <TD>1)</TD>
                                    <TD>Single Room</TD>
                                    <TD>$ 100.00</TD>
                                    </TR>

                                    <TR>
                                    <TD>2)</TD>
                                    <TD>Double Room</TD>
                                    <TD>$ 150.00</TD>
                                    </TR>

                                    <TR>
                                    <TD>3)</TD>
                                    <TD>Luxury Suite</TD>
                                    <TD>$ 250.00</TD>
                                    </TR>
                                    </TABLE>

                                    <BR> Call today for a reservation.
                                    Toll free number: 1-800-555-1212 </BODY>
                                    </HTML>
```

Example 14-2. Simple Java template

The following example demonstrates the generation of standard Java code and a number of options for changing the format of the generated code by making slight modifications to the template.

Note: The standard Java templates supplied with Poseidon for UML use defined indentation markers to format the code for better reading. The markers are of the format: `$(__)` . These indentation markers are defined as variables that resolve to an empty string. They should never show up in the generated Java code. If you find that the generated code contains such text elements, please ensure that the markers are defined and used correctly in the template.

Below is an excerpt from the template used for generating the class and method declarations.

(A)

```
## Template for standard Java output

                              ## .. snippet ..


                              #set ($vis
                              = $currentOp.getVisibilityAsString())
                              #set ($static
                              = $currentOp.getOwnerScopeAsString())

                              ..

                              #set ($thrownClause
                              = $currentOp.getThrownExceptionsSignature())
                              #set ($name
                              = $currentOp.getNameAsString())
                              #set ($methodBody
                              = $currentOp.getMethodBody())

                              ..

                              ${vis}${static}${final}${synch}${return}
                              ${name}($params) $thrownClause {
                              #renderMethodBody($currentOp.getMethodBody()
                              $currentOp.hasReturnType())
                              }

                              ## .. snippet ..
```

One step you could take to modify the Java code generated by this example would be to enter a line break before the "$thrownClause" references in the template so that the thrown exceptions appear in a separate line of the method declaration. In the following example the opening bracket has also been moved to a separate line:

(B)

```
## Template for reformatted Java output

                              ## .. snippet ..
```

```
${vis}${static}${final}${synch}${return} ${name}($params)

$thrownClause {
#renderMethodBody($currentOp.getMethodBody()
$currentOp.hasReturnType())
}

## .. snippet ..
```

The effects of such a change become clear if we compare a bit of Java code generated on the basis of these simple variations (A and B):

(Java code based on A)

```
public static void main(String[] params)
                            throws Exception {

                            doSomething()

                            }
```

(Java code based on B)

```
public static void main(String[] params)

                            throws Exception

                            {

                            doSomething()

                            }
```

14.2.2. Working with the Standard Templates

The standard templates supplied with Poseidon for UML can be used to generate Java and HTML code. The generated code is based on Class Diagrams only, but one may want to produce code from deployment diagrams or sequence diagrams. With the Professional Edition you can create your own templates to generate IDL files or C++ code.

The Java code generated on the basis of the standard Java templates is fully Java 2 compliant. The code can make use of all the features supported by Java 2, including exception handling, inner classes, and static initializers.

HTML code generated on the basis of the Standard HTML templates is simple HTML, similar to Javadoc. A separate page is generated for each class in a Class Diagram. As with the Java templates, the Professional Edition of Poseidon for UML allows you to modify the HTML templates to conform with your preferences and requirements.

14.2.3. Code Generation API

For a detailed description of the code generation API, please refer to the online API documentation (http://www.gentleware.com/index.php?id=95) (Javadoc) and the separate document describing the code generation framework (http://www.gentleware.com/index.php?id=174) . These files are part of the Developer and Professional distributions in the docs folder.

Also available are two demo plug-ins (http://www.gentleware.com/index.php?id=85) that show the capability of the code generation API and of the Poseidon plug-in API in general. The demo plug-ins are distributed as ready-to-run JAR files, along with the appropriate license keys. Also, the source code is distributed, including an ANT script for building the JARs. You may use these plug-ins as examples and as starting points for your own plug-ins. If you want to create your own plug-ins, please contact plugins@gentleware.com to receive a key for your plug-in.

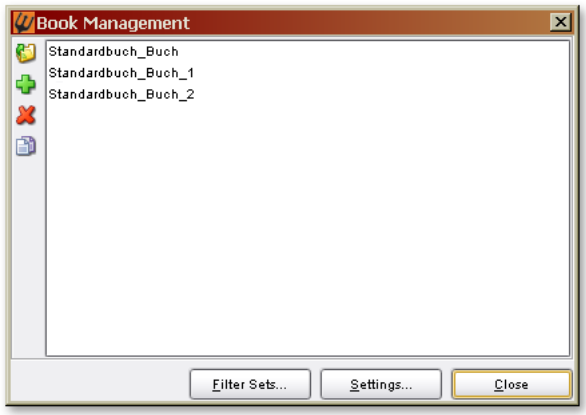
14.3. Documentation Generation

Generating corresponding HTML and Word 2003 documentation for your model is available in every edition *except* the Community Edition. The generated documentation is similar to Javadocin that it includes specific Javadoc information entered directly in your model (in the Documentation tab). This information, such as comments to your classes or methods, is included in the code. But Javadoc provides a view of the code only, not of the model - you do not see your diagrams. With the HTML doc feature you get the same information as with Javadoc, in addition to class, use case, activity, and state machine diagrams from your model. The remaining diagram types may be included in the documentation, but they are not yet supported or configurable.

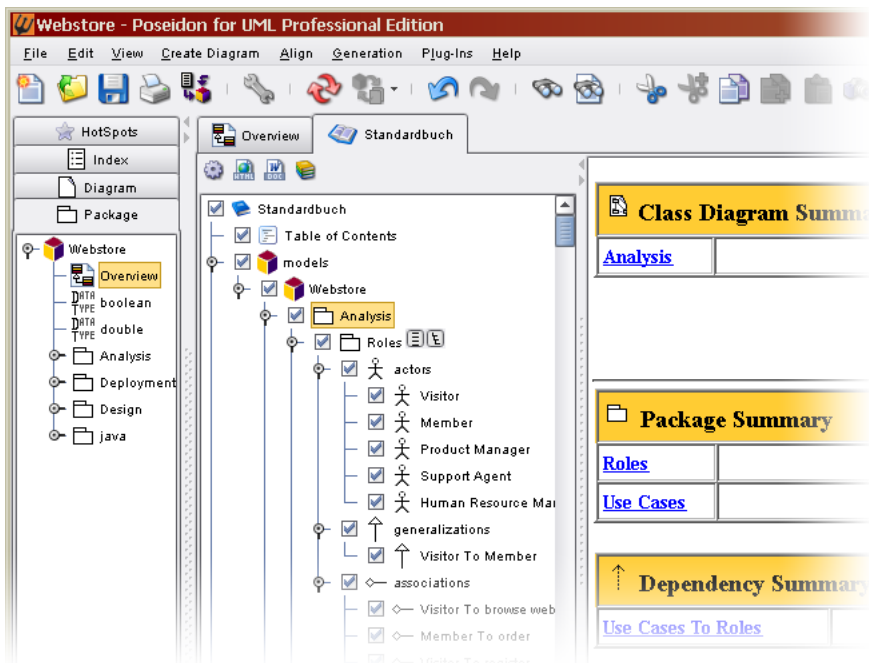
Note that from within Poseidon, the default browser is Netscape. This is not configurable in this version; however, workarounds are available. For instance, in Linux you can create a soft link from Netscape to Mozilla with the command: **ln -s /usr/local/share/mozilla netscape** . Now Poseidon should open external links on your system with Mozilla

14.3.1. HTML Preview

The HTML Preview  button opens the 'Book Management' dialog. To open an HTML preview, select the desired book(s) and click the 'Open Selected'  button.



A tab for each book is added to the Diagram pane, as when new diagrams are created. The left side displays the tree view of the documentation, the right side is the actual HTML preview.



In the tree view, items can be rearranged, added, and deleted as necessary. Please review Section 14.3.2.2 for more specific information about the behavior of the tree.

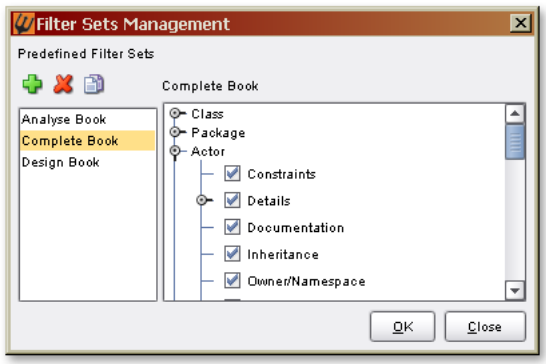
When the book has been completed to your satisfaction, you can generate HTML with the Generate HTML button, or generate Word with the Generate Word button.

Book Management Buttons

 Open Selected
 Add New
 Delete Selected
 Duplicate Book

Filter Set Management

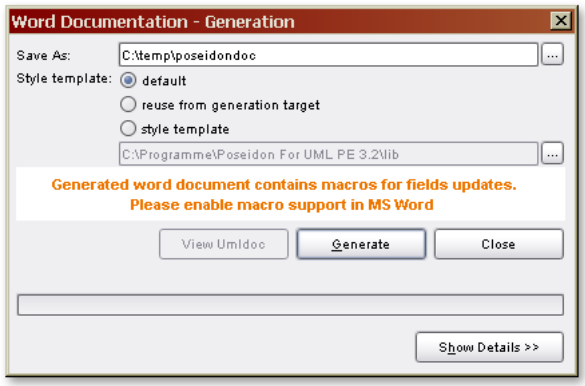
Also available from Book Management is the Filter Set Management dialog. This dialog allows you to create, edit, and delete the filter sets which determine the elements included in the documentation.



14.3.2. Generation Dialog (for Poseidon 5.x and lower)

The HTML Generation dialog is accessible from the toolbar  button or the Generation menu of every edition *except* the Community Edition, and can generate both HTML and Microsoft Word 2003 documentation from one dialog. To generate HTML, click the 'Generate...' button. For Word, click 'Word Generate...'

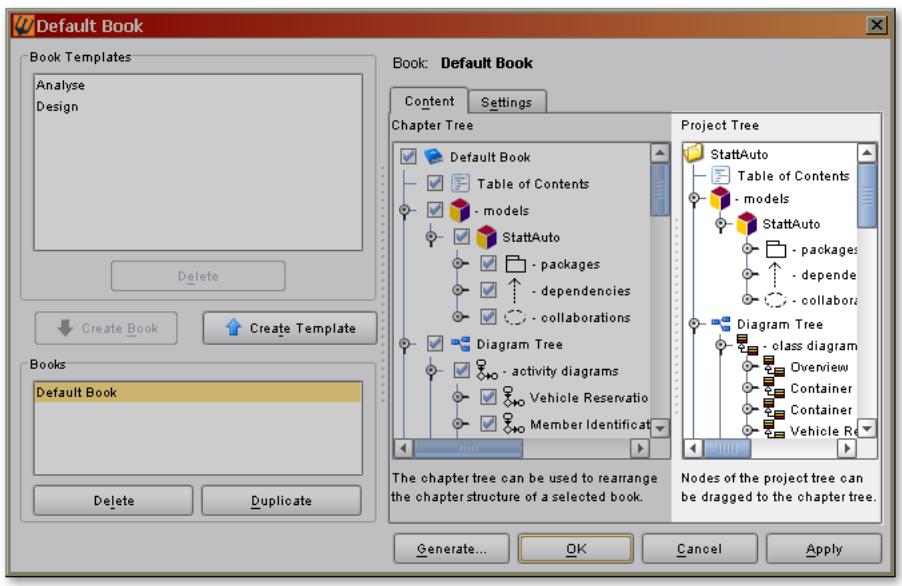
Select a Word Style Template



When generating word documents, a generation dialog prompts for the selection of a style template.

- Default - applies the styles included in umldoc.jar
- Re-use from generation target - applies the same styles as the current version of the document. This option is only available if the target document already exists.
- Style template - applies a specified XML file to the resulting documentation. Use the ellipsis button to browse to the location of the XML file or type the path directly.

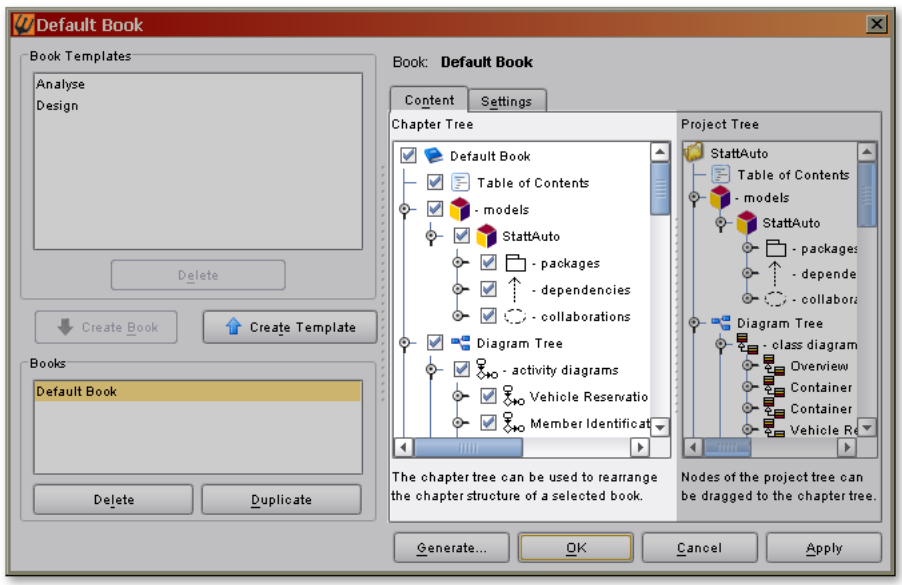
14.3.2.1. Project Tree



The project tree is a list of all available elements that can be dragged to the chapter tree. This tree cannot be reordered, nor can elements be deleted.

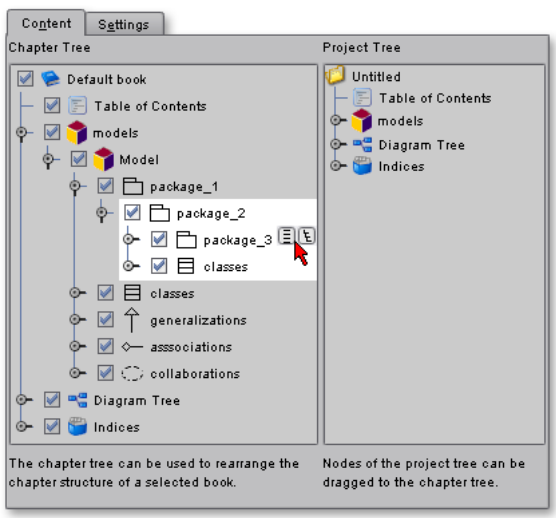
The top node of the tree is the project itself. The first element below that is the Table of Contents, followed by a package-centric view of the supported elements, then a diagram-centric view of the supported diagrams. Finally, the list of indices is presented.

14.3.2.2. Chapter Tree



The chapter tree displays all elements and diagrams that will be included in the documentation, in the order that they will appear.

Changing the Chapter Tree Nesting



Packages may be displayed with either deep or flat nesting. The project tree uses the standard deep nesting, where the packages are displayed indented and below their parent packages. The chapter tree allows for flat nesting that reduces the the number of hierarchy levels. Flat nesting can be assigned per package via the nesting toggle buttons that appear when hovering over the name of a package.

Reordering Elements

Elements can be reordered by dragging them to the appropriate location. A bar indicates where the element will be dropped, and a slashed circle indicates that the selected element cannot be dropped into that location.

Adding Elements

New elements are added by dragging them from the project tree; the elements are not removed from the project tree, meaning they may be reused and inserted in multiple locations in the document.

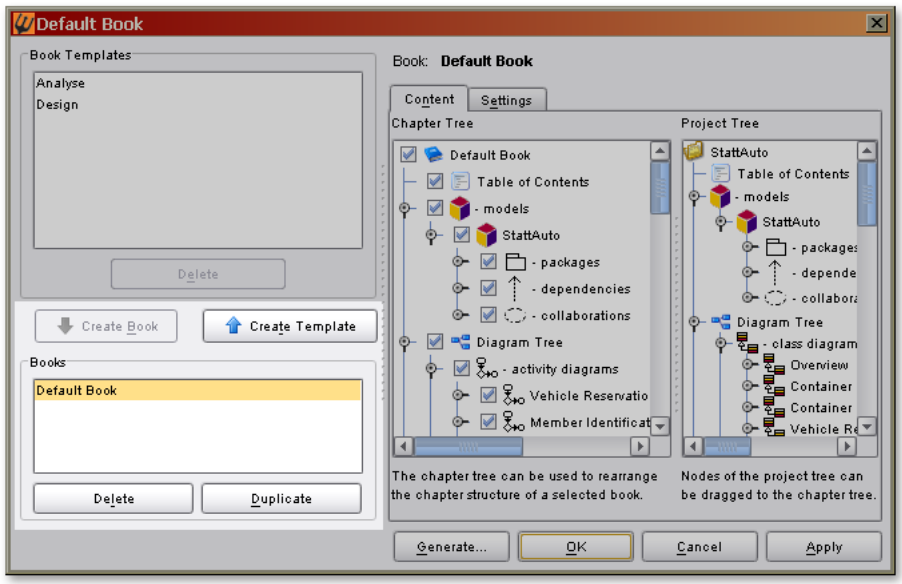
Removing Elements

Elements that appear automatically from the template are excluded from the documentation by disabling their checkboxes. Elements that have been dragged from the content tree and the top level nodes (’table of contents’, ’diagram tree’, ’models’, and ’indices’) can either be deleted from the chapter tree by pressing ’delete’ on the keyboard, or disabled with their checkbox.

Renaming Elements

Any element, book, or book template can be renamed by double-clicking on the name to open the inline editor. The name of a book or book template will simply change. The name of an element will display the new name, followed in parenthesis by the original name of the element in the model. For example, in the model, a class has the name ’Class_1’. After it is renamed, it will show ’NewClassName (Class_1)’.

14.3.2.3. Books



A single cohesive model can be used to communicate different aspects of the project to a variety of audiences. The diagrams include or exclude information as necessary to perform their intended function.

As of Poseidon 3.1, the same principle is applied to HTML documentation. One model can define separate books, each containing only the information necessary for the intended audience. Perhaps an analyst must communicate with non-technical colleagues in order to compile a collection of business practices. The non-technical people may not be interested in class diagrams, but would find certain use case diagrams or state machine diagrams ideal. These particular diagrams can be added to a book for just this audience. The same model can also contain a separate book aimed at programmers, which might contain all of the class diagrams.

A book is an organizational unit that not only defines the structure of the chapter hierarchy, but also the style settings for each document. The book list displays the names of all of the defined books for the current project. Click on a book name to display the book in the right-hand pane. Each book is stored within the Poseidon project file.

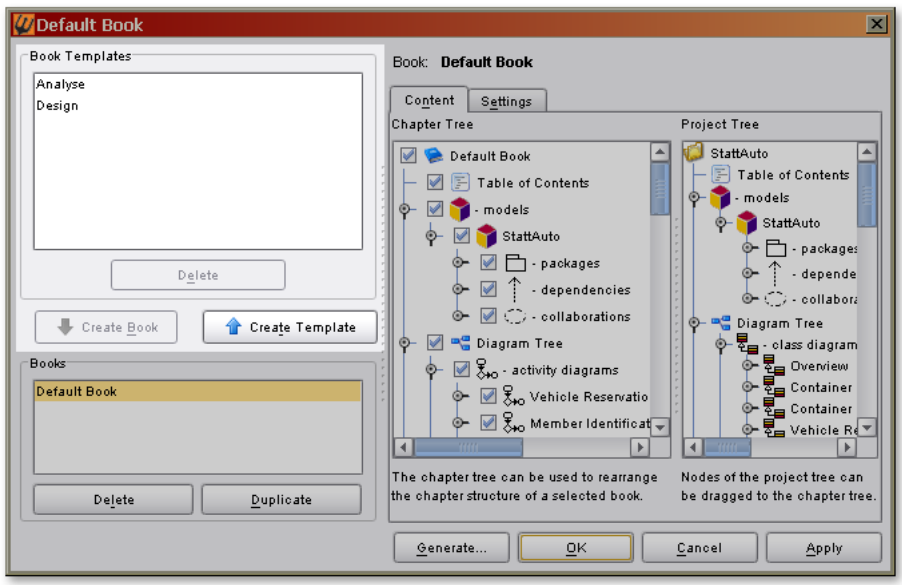
Renaming Books

A book can be renamed simply by double-clicking the book name to open the inline editor in either the chapter tree or the book list, then typing in the new name.

Creating a New Book

To create a new book, select a template from the book template list, then click the 'Create Book' button.

14.3.2.4. Book Templates



A book template is independent of any project and contains the chapter hierarchy on a type level. It refers to no specific model element, but instead to types of model elements, such as classes or states. Although specific elements of the book upon which the template is based will be displayed, these elements will have no effect on the template or any book subsequently using this template.

To illustrate, a book has a class 'Foo' and the documentation of this class is disabled. This book is used to create a book template. A new book in a different project is created based on the template, and this book also has the class 'Foo'. The template will not affect the second book.

Book templates are not stored in individual project files, so they are available to all projects.

Renaming Book Templates

A book template can be renamed simply by double-clicking the book template name to open the inline editor in the book template list, then typing in the new name.

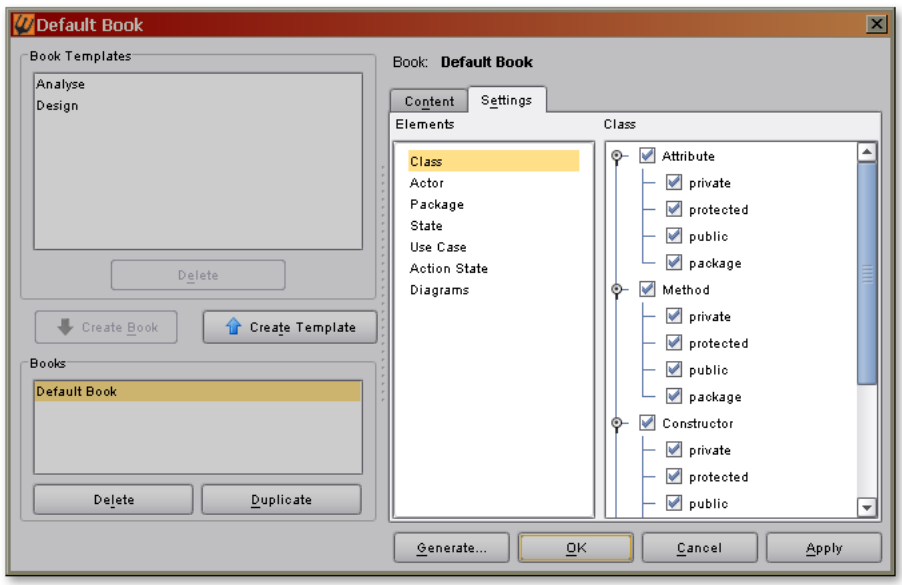
Creating a New Book Template

To create a new book template, select a book from the book template list, then click the 'Create Template' button.

Editing a Book Template

Book templates take the structure and settings of the current book. To edit a book template, select the book template, create a new book from that template, edit the book, then use the edited book to create a template.

14.3.2.5. Settings Tab



The Settings tab indicates which elements are to be included on a book level. Individual settings are not possible at this time, e.g. if the checkbox for class attributes is disabled, no class attributes will be included throughout the entire book.

14.3.3. Included Diagrams

Class, Use Case, State Machine, Sequence, and Activity diagrams are fully supported by the documentation plug-in. They are included in the diagram trees of the generation dialog, listed in the indices, and all of their element types may be documented.

Unsupported diagram graphics may still appear in the documentation, but the configuration options are limited. They are not listed in the indices or diagram trees, and the element types from these diagrams which may have documentation are element types which may appear in the supported diagrams.

For example, a deployment diagram graphic may appear in the documentation, but the name of the

diagram will not be listed and the summary section for the diagram will be empty. Associations in this diagram may appear in the indices and have their own documentation because they are allowed in the supported diagrams. Nodes, on the other hand, are not allowed in any of the supported diagrams, therefore they are not included in the documentation.

14.3.4. Supported Javadoc Tags

Currently UMLdoc generates output for the following Javadoc tags, all unknown tags are skipped and do not produce output.

@author *[author name]*

Adds the specified author name to the model element documentation, output is only produced if you have selected the *Generate authors doc* option in the UMLdoc code generation settings.

@deprecated *[text]*

Adds a comment indicating that this API should no longer be used (even though it may continue to work).

@exception, @throws *[exception type] [description]*

Adds an exception description to the method documentation.

@gentleware-originalType

Notes the original type of an attribute with an unlimited upper bound which has been typed as a Collection.

{@link *package.class#member label* }

Inserts an in-line link with visible text that points to the documentation for the specified package, class, or member name of a referenced class.

Also generates external links. Any types from Java will be automatically linked to Sun's Java site, and other links can be created utilizing the @link tag. Additionally, any URL included in the documentation will be automatically detected and the link will be activated without requiring any other notation.

@param *[param name] [description]*

Adds a parameter description to the method documentation.

@return *[description]*

Adds a return parameter description to the method documentation.

@see *[reference]*

Adds a "See Also" heading with a link or text entry that points to a reference.

@serial *[description]*

Adds a comment indicating a default serializable field. The optional description should explain the meaning of the field and list the acceptable values.

@serialData *[description]*

Documents the sequences and types of data written by the writeObject method and all data written by the Externalizable.writeExternal method.

@serialField *[name] [type] [description]*

Documents an ObjectStreamField component of a Serializable class' serialPersistentFields member.

@since *[release name]*

Adds a description indicating that this change or feature has existed since the software release specified.

@version *[version]*

Adds a version to the method documentation. A doc comment may contain at most one @version tag.

Chapter 15. Plug-Ins

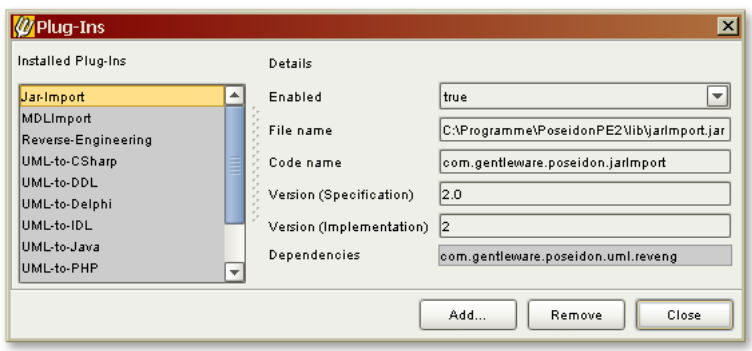
With Poseidon's plug-in interface it is possible to add extended functionality that is well beyond what is implemented in the core product. The Standard Edition of Poseidon for UML comes with this plug-in feature. Development teams from Gentleware AG as well as technology partners are working on plug-ins that meet specific designer and developer needs. The following sections give a brief overview of the most recent plug-ins that are available for shipping (or will be soon). For information on how to install a plug-in, please see the separate documentation (http://www.gentleware.com/index.php?id=instguides) available on the Gentleware Web site.

The Professional Edition of Poseidon comes with several options for code generation and one for documentation generation (UMLDoc). Java code generation is the default setting, but you can also choose to generate other types of code. To do this, you have to activate the plug-in that supports the desired language. (Via Plug-Ins | Plug-Ins Panel). When you do this, a set of stereotypes becomes available that can be used to control the result of the code generation. The next sections describe what stereotypes and tagged values you can use to control the output of code generation.

15.1. The Plug-In Manager

The Plug-In Manager is provides an easy interface to install, manage, and uninstall plug-ins. The left side displays all installed plug-ins, while the right side displays details about the selected plug-in.

The plug-in displayed in the figure below is named JarImport. It contains all of the information used by Poseidon to import archived Java files. This plug-in is standard in the Professional Edition.



Details available in the Plug-in Manager:

- **Enabled** - This dropdown allows you to determine whether or not a plug-in is used by Poseidon.

- **File Name** - Displays the location where the plug-in is installed. This field is not editable.
- **Code Name** - Displays the code name of the plug-in.
- **Version (Specification)** - Displays the version number of the plug-in.
- **Version (Implementation)** - Displays the internal build number of the plug-in.
- **Dependencies** - Lists the plug-ins from which it uses functions.

15.1.1. Installing a New Plug-In

Using Plug-Ins requires three steps. You must first add the license, then install the plug-in, and finally enable the plug-in.

15.1.1.1. Add the Plug-In License

1. Download the plug-in from the Gentleware web site. You must additionally purchase a license key from the Gentleware store (except in the case of beta versions).
2. From Poseidon, open the License Manager from the Help menu.
3. Paste the Serial Number into the 'New Key/Serial #' box at the bottom of the License Manager. This number should have arrived via email when you purchased the plug-in.

   Click the 'add' button.

4. The Serial Number now displays a valid status.

   The Serial Number must be registered in order to receive the Final Key. The Final Key allows you to use an unrestricted version of the plug-in. Failure to register the plug-in will cause the plug-in to cease operation after the grace period expires.

5. Click the 'register' button. You can choose to register online (directly from the dialog) or via the website. Once the registration is complete, close the License Manager.

15.1.1.2. Install the Plug-In

1. Check the documentation accompanying your plug-in to determine which directory you should use for installation. For most plug-ins, extract these files into the 'lib' directory under the Poseidon installation directory. There are exceptions, however. For example, if you have downloaded the GoVisual autolayout plug-in, extract these files into the 'lib/ext/'directory.
2. Now that the files are in place, it is time to add the Plug-In to Poseidon.

15.1.1.3. Enable the Plug-In

1. Open the Plug-In Panel (located in the Plug-Ins menu). Click the 'add' button.
2. Select the .jar file for the Plug-In from the 'lib' directory (or wherever you installed the plug-in, as this is the same file that was unzipped earlier in the process).

   Click the 'install' button.

3. Verify that the Plug-In has been installed and is enabled by highlighting the name of the Plug-In from the list of Installed Plug-Ins.

15.2. Removing Plug-Ins

If you decide to no longer use a plug-in, you have the option to disable it from within the Plug-In panel. You can also remove the plug-in from the panel by selecting the plug-in and clicking the 'Remove' button. This does not remove the files for the plug-in from the Poseidon directory, nor does it remove the license key for the plug-in.

To completely uninstall a plug-in, you must manually delete the files that were added during the installation process, then open the License Manager and remove the License Key.

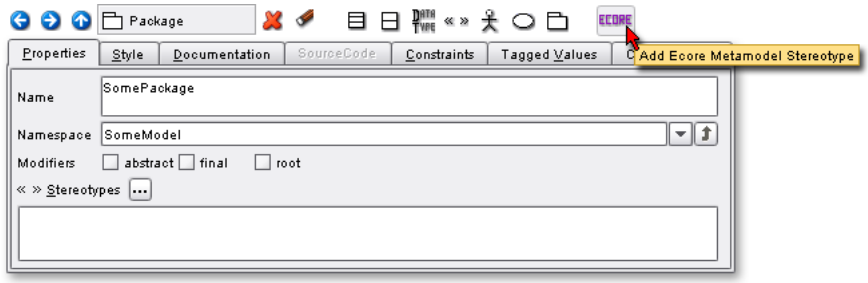
15.3. Available Plug-Ins

15.3.1. UML-to-Ecore Plug-In

Professional Edition and Enterprise Edition users with version 4.1 or higher can use the UML-to-Ecore Plug-In to easily create Ecore metamodels for use in the Eclipse Modeling Framework (http://www.eclipse.org/emf/) (EMF). The contained UML elements will be mapped to Ecore elements as described in Appendix A .

To export a package to Ecore:

1. Select the desired package in the diagram or via the Navigation Panel.
2. Click the 'Ecore' button in the Properties tab of the Package.



This will add the stereotype <<*ecoremodel*>> and the tagged values **nsURI** and **nsPrefix** to the package. Alternatively, you can opt to add these manually.

3. Right-click on the package to access the context menu, then select "Export to Ecore".
4. Browse to the appropriate location, enter a filename, and click "Save".

15.3.1.1. EMF and Ecore

(The following text is reproduced from the Eclipse Project. Please consult their website (http://www.eclipse.org/emf/) for more detailed information.)

EMF is a modeling framework and code generation facility for building tools and other applications based on a structured data model. From a model specification described in XMI, EMF provides tools and runtime support to produce a set of Java classes for the model, a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor. Models can be specified using annotated Java, XML documents, or modeling tools like Poseidon, then imported into EMF. Most important of all, EMF provides the foundation for interoperability with other EMF-based tools and applications.

EMF consists of three fundamental pieces:

1. EMF - The core EMF framework includes a meta model (Ecore) (http://download.eclipse.org/tools/emf/2.2.0/javadoc/org/eclipse/emf/ecore/package-summary.html#details) for describing models and runtime support for the models including change notification, persistence support with default XMI serialization, and a very efficient reflective API for manipulating EMF objects generically.
2. EMF.Edit - The EMF.Edit framework includes generic reusable classes for building editors for EMF models. It provides:
   - Content and label provider classes, property source support, and other convenience classes that allow EMF models to be displayed using standard desktop (JFace) viewers and property sheets.
   - A command framework, including a set of generic command implementation classes for building editors that support fully automatic undo and redo.

3. EMF.Codegen - The EMF code generation facility is capable of generating everything needed to build a complete editor for an EMF model. It includes a GUI from which generation options can be specified, and generators can be invoked. The generation facility leverages the JDT (Java Development Tooling) component of Eclipse.

15.3.2. JAR Import

The JAR Import Plug-in supports reverse-engineering and importing JAR archives into an existing model in Poseidon for UML. You can use and extend existing packages or frameworks in your own models, or browse and learn existing APIs. This feature is often requested by professional developers, for instance, to get a more vivid visualization of APIs than a standard Javadoc might provide.

15.3.3. RoundTrip UML/Java

With the RoundTrip UML/Java Plug-in you can generate Java code from your UML model, edit your code, reverse-engineer your code and synchronize with the model. Modeling and coding are not separated anymore.

15.3.4. MDL Import

The MDL Import Plug-in is now incorporated into the Embedded, Embedded Enterprise, Professional, and Enterprise editions and enables Poseidon to import UML models created by Rational Rose.

15.3.4.1. Using MDL Import

The MDL Import functionality is available from the Import Files dialog (accessible from the File menu or by clicking the icon in the toolbar), which allows you to select the file type `*.mdl` . Unlike jar and java import, the current model is discarded and you cannot add a Rose model to your current model. You can set the scaling factor by entering a different value into the text field below the general information about the plug-in (see Display Issues). By default, the import plug-in hides the package information in Class Diagrams - long package names tend to ruin the diagram layout. If you want package names to be displayed in classes and interfaces, you may activate the check box.

15.3.4.2. Supported Diagrams

This version of the import plug-in reads class, state, activity, usecase, and sequence diagrams. The other diagram types will be incorporated in the next release.

15.3.4.3. Unsupported Features

Some elements are changed during the import, others are ignored completely. Here is a list of known shortcomings:

- Poseidon currently supports comments for classes, interfaces, packages, use cases, actors and states, but not for transitions, associations or objects. If a comment is not supported, it is added to the diagram as ordinary text.
- Metaclass: Poseidon does not support meta classes, these classes are imported as ordinary classes.
- Synchronization States: Rose does not discriminate between fork and join states. There is no way of telling how to map synchronization states - this plug-in currently always assumes fork states if the number of outgoing transitions is bigger than one. You are informed about the decision.
- Subsystems: Subsystems are treated as packages - Poseidon does not support subsystems at the moment.

The following features are (at the moment) not being imported at all. You will get a warning after the import is complete that these elements will be missing.

- Destruction Markers
- Swim lanes
- References: MDL files support references to other files (`*.jar` or `*.cab` files, for example). This import tool ignores references, no warning is issued.

Other problems: Some older versions of Rose have a bug in sequence diagrams: Links between objects have a wrong target ID. These links will not be resolved correctly by this plug-in - you will get an error message. Rose does the resolving by name instead of by ID, which seems rather error-prone, so we do not try to do this. Loading and saving the model with a new Rose version like Rose 2000 solves the problem, and the sequence diagram can be correctly be imported.

15.3.4.4. Display Issues

MDL files contain information about the diagram layout. The import plug-in reads the diagram elements coordinates and positions the diagram elements accordingly. A few things should be considered, though. Poseidon uses "smaller" coordinates than Rose. In general, scaling down the coordinates by 40 percent does the job - the diagrams almost look like they did in Rose. You can change the value in the Configuration tab to the right. If you choose 80%, for example, the diagram elements are further apart (but not bigger!) - making it easy to add comments or further elements.

While the coordinates are read from the MDL file, the sizes of diagram elements are dependent on the information being displayed. For example, a classes size depends on the length of the contained methods names and parameters. Long names or lots of parameters may lead to overlapping classes. To solve this, you can either select a higher scaling factor, or (at least for Class Diagrams) you can can edit the display options (select menu item Edit/Settings , and click the tab Diagram display).

Sequence Diagrams

Poseidon performs an automatic layout of sequence diagrams - layout information contained in MDL files is ignored. Objects are currently placed arbitrarily, you might have to re-arrange them and any associated textual information. Apart from that, Rose allows activations to have arbitrary length, while Poseidon calculates the length of activations depending on the stimuli sent. Using the right mouse button, you can force an object to remain activated after the last message was sent.

15.3.4.5. Status

We did extensive testing, and any problems during import should be signaled. But before you use and extend an imported file for production work, you should check your models and diagrams in case some model element was forgotten. If you experience problems or want to request additional features, do not hesitate to contact us - details available from Section 1.1.4 .

Chapter 16. Advanced Features

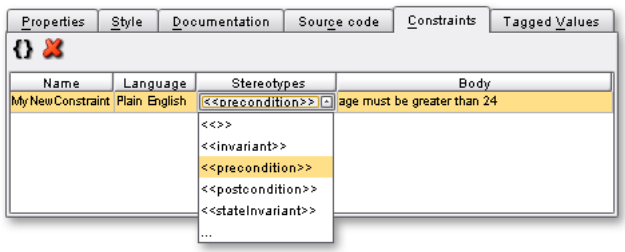
16.1. Constraints with OCL

UML is basically a graphical language. As a graphical language it is very suitable for expressing high-level abstractions for architectures, workflows, processes etc. But for expressing very detailed and fine-grained things like algorithms, equations or constraints, textual languages just tend to be more convenient.

The current UML recognizes this and comes with a supplementary textual language to express constraints. This language is called the Object Constraint Language, or abbreviated OCL. Although Poseidon does not require OCL, nor does it check OCL syntax, there are certainly no restrictions prohibiting you from using OCL.

Since OCL is noted as text, it is simple to support, and many UML tools do it just that way. You can simply enter lines of text in certain fields reserved for constraints. In Poseidon for UML you can do that in the Constraints tab on the Details pane, as shown in the figure below.

Figure 16-1. A Constraints tab.

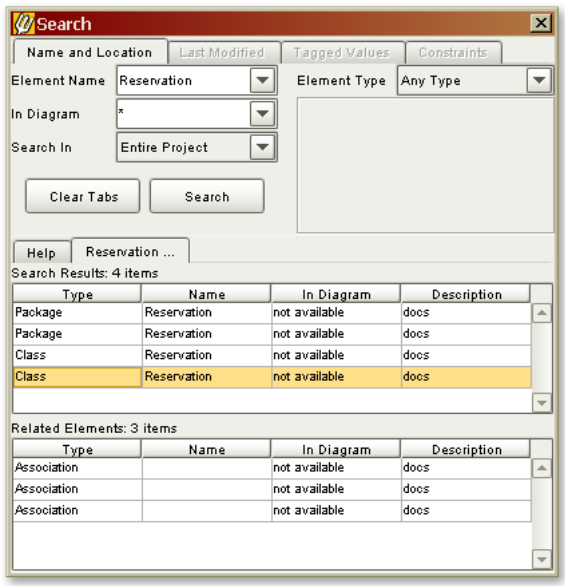


16.2. Searching for Model Elements

When your models start to grow, you will want a nice mechanism to search for elements. Poseidon offers a powerful search tool that is not just based on text but on model information. It allows you to look for specific types of elements. The search tool is invoked from the Edit menu, by selecting Find... or by directly pressing the function key `F3` .

Type in the name of the element you are looking for (you can also use the asterisk as a wildcard), and specify the type of element you are looking for. If you are looking for a class, this type would be Class.

Figure 16-2. Searching for a class



For each search, a new tab is created so that you can access older search results. You can also restrict the search space to be the result of an earlier search. Selecting one entry from the results list provokes that `Related Elements` are also shown. Double-click on one entry in the results list whereas effects that the element is selected in the Navigation pane.

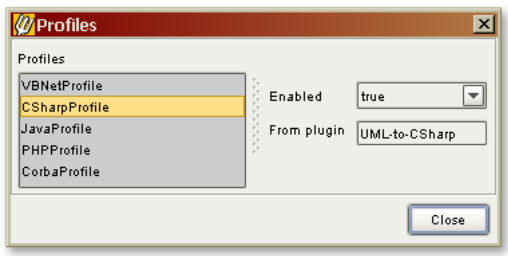
16.3. Profiles

Profiles generically extend the UML through the use of stereotypes that are most often language-specific, provide a common graphical notation and vocabulary, and define a subset ot the UML metamodel (that could possibly be the entire UML metamodel). For example, variable and operation types change based on the profile (and therefore the stereotypes) used. There is a profile associated with each of the language plug-ins, and the profiles that automatically appear in the Profile Manager directly correspond to the set of enabled language-specific plug-ins and are enabled by default. Likewise, if a plug-in is disabled from the Plug-in Manager, the associated profile is automatically disabled and will not appear in the Profile Manager.

It may be advantageous at times to disable these profiles. The Profile Manager displays those profiles that are currently available and allows you to enable and disable them with a simple dropdown menu.

The profile is saved to the project as long as the profile was enabled in the Profile Manager when the project was saved. If the originating plug-in or the profile was disabled at the time of the save, no data related to that profile is saved. Say you have disabled the profile, and then decide to disable the plug-in. If you enable the plugin again, the profile will be automatically enabled. The status of the profile is not saved when the plug-in is disabled.

Figure 16-3. The Profile Manager



Chapter 17. Poseidon in Eclipse

There are lots of ways to create UML projects and even more ways to write code, including standing at a whiteboard. But it isn't just a question of how to capture this information, it is what you can do with it once it has been created. Poseidon's integration into the Eclipse IDE means that the most intuitive UML tool can exchange data directly with one of the most flexible development environments. Code and model become integrated into one project.

Throughout the development process, both the UML model and the source code evolve and grow. Optimally, the two should always be synchronized. Previously this meant importing and exporting changes between tools based on a common denominator. Now with the push of a single button, Eclipse and Poseidon can seamlessly integrate the textual information that is Eclipse's specialty with the visual data Poseidon is expert at creating into one comprehensive project.

This is not a comprehensive guide to Eclipse, but highlights some of the more important features of Eclipse that are relevant to Poseidon. More information about Eclipse, including download mirrors, is available at the Eclipse Website (http://www.eclipse.org) . Full Eclipse Documentation (http://www.eclipse.org/documentation/main.html) is also available.

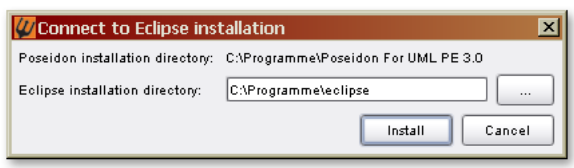
Much of the functionality of Poseidon within Eclipse is based on the Roundtrip Engineering that is available from the standalone version of Poseidon for UML. Many of the dialogs presented here are discussed in more detail in Section 14.1.3 . We suggest that you review this section before beginning with Eclipse.

Please note that Poseidon in Eclipse is available in the Professional Edition only.

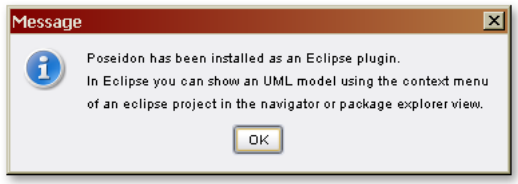
17.1. Installing Poseidon within Eclipse

Installing the Poseidon Professional Edition for use within Eclipse is very simple. Before you begin, be sure that you have Eclipse correctly installed on your machine and that Eclipse is not currently running.

1. Install Poseidon PE .x normally, as described in Section 3.3 . Start Poseidon.
2. From the Poseidon Plug-Ins menu, select 'Install Eclipse-Bridge'.
3. Select your Eclipse installation directory, then click 'Install'.



4. A confirmation dialog will appear. Click 'OK'.



5. Shut down Poseidon and start Eclipse.

If you do not have write permissions for the Eclipse directory, a new file will be placed in a temporary directory and you will be reminded to manually move this file into the 'links' directory under the Eclipse installation directory. Alternatively, you may run Poseidon with Administrator privileges.

Increasing Memory

To increase the memory available to Eclipse, start Eclipse by running " **eclipse -vmargs -Xmx512m** ", where the number 512 can be replaced with the amount of memory to be allocated to Eclipse. Providing more memory is recommended.

Running Linux

On Linux, Eclipse needs Java 5.0 to work properly. Set the environment variable `JAVA_HOME` to your Java 5.0 installation directory and make sure that the `PATH` environment variable contains `$JAVA_HOME/bin` . You can verify your Java installation by checking that the command " **java -version** " returns a text containing "build 1.5".

Importing files has changed in Java 5.0. The changes are outlined in Section 17.3.1 .

17.2. Start Poseidon in Eclipse

Before you can work with the Poseidon Eclipse integration, you must start the Poseidon application inside Eclipse. The following sections outline the three possible scenarios.

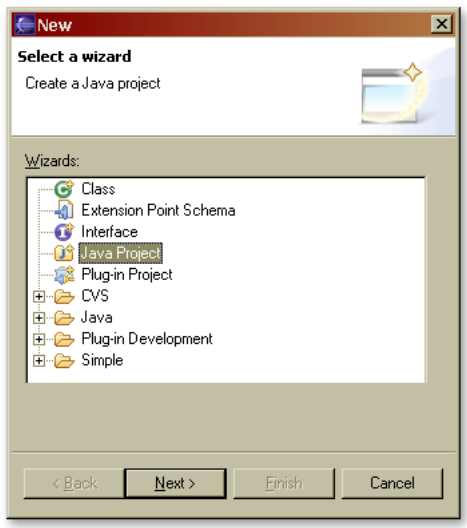
17.2.1. New Eclipse Project, New Poseidon Model

Poseidon projects exist somewhat independently from Eclipse projects, yet are closely related. The Poseidon project is stored as a part of the Eclipse project, but the .zuml file can be opened and edited as usual by a standalone version of Poseidon. Each Eclipse project may have only one Poseidon project.

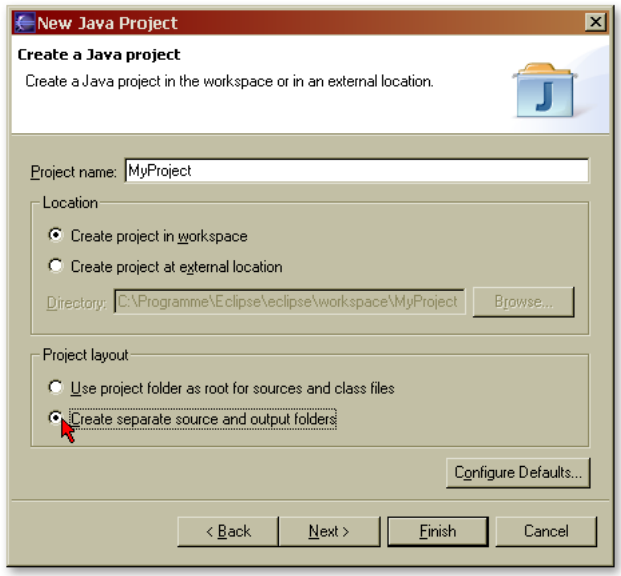
Because the Poseidon projects are part of the Eclipse project, the source directories defined in Eclipse will be the source directories available to Poseidon. You can add, edit, and delete source directories from Project | Properties | Java Build Path | Source.

17.2.1.1. Create a Project from Scratch

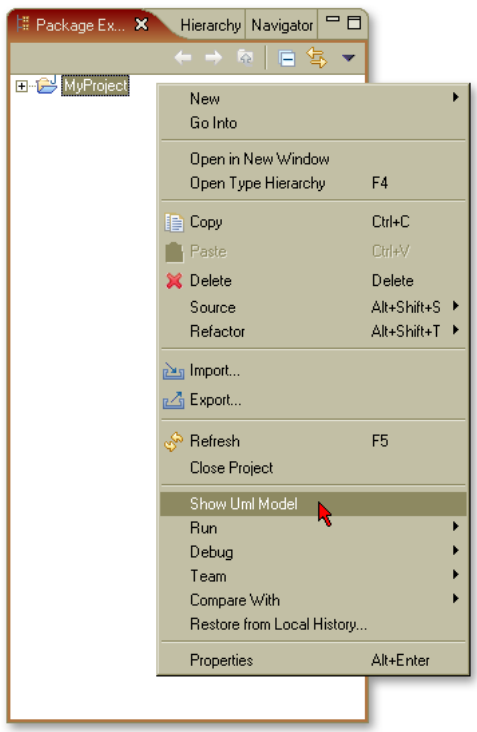
1. Click the  'New Project' button.
2. Select 'Java Project' from the New Project Wizard.



3. Name the new project. Although it is possible to use the project root as the source folder, separate source and output folders are highly recommended; if the project root is used as the source folder, roundtrip will open with an error message and the project root will have to be explicitly set in the resulting dialog.

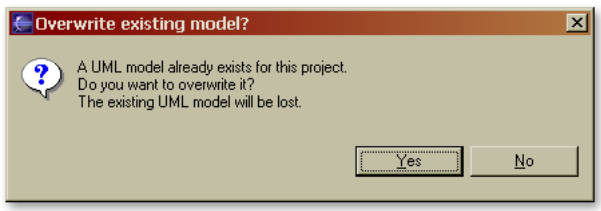


4. Select 'Show UML Model' from the new project's context menu to open the associated model.



17.2.2. Existing Poseidon Model

The simplest way to incorporate an existing Poseidon project into Eclipse is to directly import the UML model. As each Eclipse project can have at most one UML model, if a model currently exists for the Eclipse project, you will be asked for permission to overwrite it. If you decline to overwrite, the import will be cancelled.



To import a UML model:

1. Open or create an Eclipse project.
2. Right-click on the project name and show the UML model.
3. Poseidon will start if it isn't running already, and the UML menu option will appear. Select Import UML Project... from this menu. Note that the project will be imported into the current UML project displayed, regardless of the project selected in the Eclipse navigator.
4. A dialog will open where you can save changes to the project - you must select 'No'. If you save the changes to the model, the UML project will not be imported.
5. Roundtrip is enabled and the synchronization dialog appears because there are elements in the model that do not exist in the generated code. Click 'Synchronize Source Code' to generate the code for these model elements.
6. If the Eclipse project had elements like packages and classes prior to the import, you will be asked to import these into the Poseidon model. Click 'Synchronize Model' to import them into Poseidon.
7. Now that they have been added to the model, Poseidon has added identification information to these elements and therefore the source does not match the Eclipse source. Click 'Synchronize Source Code' once again to update the source in Eclipse.

It is possible to merge an existing .zuml file into the current UML model. Simply select UML | File | Merge Project into Current and then select the desired .zuml file from the file chooser.

You can also add a UML model to the Eclipse project by creating a directory called 'uml' in your Eclipse workspace directory under the project directory. Copy the desired .zuml file to this directory, keeping in mind that only one .zuml file may exist here. Refresh the Eclipse navigator with F5 to see the new directory.

Note that double-clicking the UML model name will open the model in standalone Poseidon. To open the model in Eclipse, use the 'Show UML Model' option from the context menu.

17.2.3. Existing Eclipse Project

Select the Eclipse project in the Eclipse navigator panel, then select 'Show UML Model' from the context menu of the project. An empty UML model is created. To see the code from the Eclipse project in Poseidon, you must now import the code. See Section 17.3.1 .

17.3. Working with Projects

While Eclipse and Poseidon work closely to maintain a single project, each has its own component within that project. It is important to remember that each component must be made aware of changes to the other, so that any time you modify the source code, you must update the model to see the changes and vice versa. This synchronization process is very simple, and most times will require only one click of a button.

17.3.1. Importing Source Code

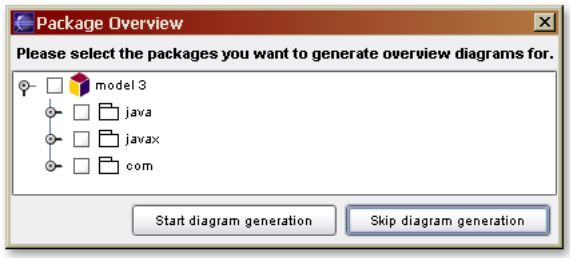
A note about importing using Java 5: At a command prompt, type 'java -version'. If you see 'build 1.5', then you are running Java 5. The file chooser has changed between Java 1.4 and 5; in 1.4 it was possible to select single files or subfolders of the current folder. In Java 5, when you select a subfolder and press "open", the chooser enters that folder, rendering it impossible to import the folder using only the mouse. Instead, you must type the name of the folder into the text field and then press 'open'.

17.3.1.1. Roundtrip Import - Roundtrip is enabled

Roundtrip import can be accomplished with one mouse click of the  'Roundtrip Import' button. If all classifiers have been mapped to source directories so that Poseidon knows in which folder to generate the code, import will proceed automatically. If some classifiers have not been mapped, a dialog will appear, allowing you to either map the classifier or exclude it from roundtrip altogether. More information on how to map classifiers can be found in Section 7.2.4.2 . After the import is complete, the imported items will be available from the model navigator pane and can be added to any diagrams as usual.

17.3.1.2. Normal Import - Roundtrip is disabled

When roundtrip engineering is turned off, source code can be imported into the UML model from 'UML | File | Import Files...', then select the appropriate files from the file chooser. Click 'Open'. From here you will be presented with a dialog where you can choose to generate diagrams for packages if you like.



17.3.2. Generating Source Code

17.3.2.1. Add Additional Source Directories

The source directories available to Poseidon within are not set in the roundtrip settings, as is standard in the standalone Professional Edition. Instead, the source directories are determined by Eclipse. You can specify these directories from 'Project | Properties | Java Build Path'.

17.3.2.2. Generate Source Code with Roundtrip

When roundtrip is enabled, simply click the  'Generate Code' button on the Eclipse toolbar. Source code will be generated to the directories specified in 'UML | Open Roundtrip Settings Dialog | Project | Roundtrip | Classifier Mappings'. If classifiers have been added to the model, they will not have a mapping yet, and a dialog will open allowing you to set the source folders for those classifiers. For more information about this dialog, see Section 7.2.4.2 .

17.3.2.3. Generate Source Code without Roundtrip.

This method of source generation works just as it does in standalone Poseidon, which is outlined in Section 14.1 . Start code generation from 'UML | Generation | <language>'.

To generate the code into Eclipse, you must select the output folder to be the source directory of your Eclipse project. If you created a simple Java project, this will be under your Eclipse workspace as the name of the project, which could look something like C:\Programme\Eclipse\workspace\myProject. The recommended way to create a project is with the 'Create Separate Source and Output Folders' option. In this case, your source path might be similar to C:\Programme\Eclipse\workspace\myProject\src. Refresh the Package Explorer or Navigator in Eclipse to see the changes.

This method of generation does not allow you to specify different source folders for each classifier, and as a result if more than one source directory is used for this project in Eclipse, is is likely that you will

end up with duplicates - one version from Poseidon and one from Eclipse.

You can also choose to generate the source code to a completely different directory, which will not affect the Eclipse version in any way.

17.3.3. General

17.3.3.1. Drag and Drop

Drag and Drop functions within each component of the Eclipse integration, but does not function between components. For instance, you can drag a class from the Eclipse navigator to the Eclipse Java editor, or from the Poseidon navigator to a diagram, but you cannot drag a class from Eclipse to a diagram, nor can you drag a class from Poseidon into the Eclipse Java editor.

Remember that while these two tools function closely, you must still import and export changes made from one tool to see the alterations in the other tool. The inability to drag and drop between the two is a reminder to generate the code or model in order to transfer data between the two components.

17.3.3.2. Working with the UML Model outside of Eclipse

Incorporated into the Eclipse project is a standard .zuml file, which can be edited in standalone Poseidon if you like. These revisions can then be incorporated into the Eclipse project by generating the source as usual.

17.3.3.3. Saving the UML Model

Two options exist to save your UML model. For an interim save while working, you can save your model from 'UML | Save Project'. This will save only your Poseidon project, not the entire Eclipse project. For a final save when shutting down Eclipse, Eclipse will automatically save your UML project information while saving the workspace.

17.3.4. Summary

17.3.4.1. Edit Projects with Roundtrip Engineering

The roundtrip engineering function is a convenient way to keep your source code and model synchronized, therefore this is the recommended way to work with UML models in Eclipse.

Edit code in Eclipse, see changes in the UML model

1. Edit code as usual in Eclipse
2. Save your code in Eclipse
3. Press the 'Import Code' button from the Eclipse toolbar
4. Changes to existing elements will be automatically updated, new elements will be available from the Poseidon navigator.

Edit the UML model, see changes in Eclipse

1. Edit the model from the Poseidon integration
2. Press the 'Generate Code' button from the Eclipse toolbar
3. Make the Eclipse navigator the active frame and press F5 to refresh the navigator and see the new code

17.3.4.2. Edit Projects with Roundtrip Disabled

You may have occasion to use Poseidon in Eclipse with Roundtrip disabled. Although this is not the recommended way to edit your code and model, it is certainly possible to do so.

Edit code in Eclipse, see changes in the UML model

1. Edit code as usual in Eclipse
2. Save your code in Eclipse
3. Import changes into Poseidon from 'UML | File | Import Files'

Edit the UML model, see changes in Eclipse

1. Edit the model from the Poseidon integration
2. Generate the source code from 'UML | Generation | Java'. As the output folder, use the source directory for the Eclipse project.
3. Refresh the Eclipse navigator to see the changes.

17.4. Interface

Please note that these interface elements are available only after Poseidon has been started within Eclipse. See Section 17.2 for instructions.

17.4.1. Poseidon Entries in the Eclipse Toolbar


Open UML Perspective


Roundtrip Disabled


Roundtrip Enabled

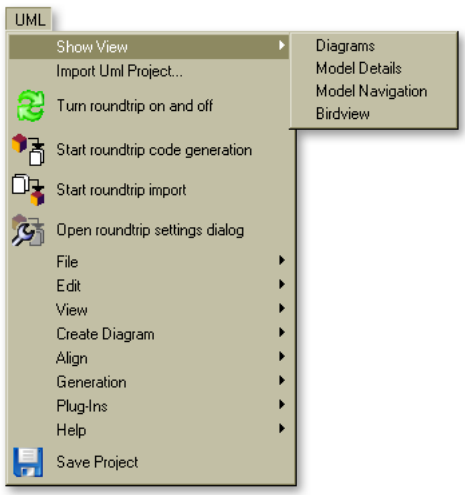

Synchronize Model and Code


Generate Source from Model


Import Changes from Source


Open Roundtrip Settings

17.4.2. UML Menu



- **Show View** - Opens the chosen Poseidon pane in Eclipse. See Chapter 6 for compete information about panes in Poseidon.
  - Diagrams
  - Model Details
  - Model Navigation
  - Birdview

- **Import UML Project...** - Opens the dialog to open a Poseidon project in the current Eclipse project.
- **Turn Roundtrip On and Off** - Toggles roundtrip engineering.

- **Start Roundtrip Code Generation** - Generate source code from the UML model.

  Quick Key - Ctrl-Alt-G

- **Start Roundtrip Import** - Update the UML model from the source code.

  Quick Key - Ctrl-Alt-I

- **Open Roundtrip Settings Dialog** - Open the Poseidon settings dialog.
- **Poseidon Menus** - Provides access to all Poseidon menu options. For a complete list of these options, see Section 5.2.1 .
  - File
  - Edit
  - View
  - Create Diagram
  - Align
  - Generation
  - Plug-Ins
  - Help

- **Save Project** - Saves the current UML project. It does not save the entire Eclipse project.

17.4.3. The Java Perspective

Perspectives in Eclipse are a collection of tabs and panes, each with a particular purpose in mind so that everything you need is conveniently available to you. These are highly configurable; you can drag the tabs around and open and close additional tabs as you need them.

The Java perspective is a standard perspective that has been designed for Java code development. Among other things, included are a navigator, java editor, and overview. Any of the Poseidon panes can be added to this perspective through UML | Show View | View Name. To relocate them, simply click and drag on the tab title.

17.4.4. The UML Perspective

The UML Perspective is the recommended way to work with Poseidon in Eclipse because it has been designed to mimic the familiar Poseidon interface. Each of the Poseidon panes is present and in the same

location as in the standalone version of Poseidon. Of course, you are able to customize this perspective by moving or adding other views, perhaps adding an Eclipse navigator or removing the Poseidon birdview.

To open the UML perspective, click the 'Open UML Perspective'  button from the toolbar or the perspective selector in the upper right hand corner.

Chapter 18. Epilogue

At this point we would like to express our thanks to everyone who, over the years, has contributed to ArgoUML and Poseidon. Without this active community of developers and users, Poseidon would not be what it is today.

Also, we would like to acknowledge the work of all the other open source projects we have made use of. We share with them the intention of developing high-quality software within the open source community. The Poseidon for UML Community Edition, as well as our feedback to the open source of ArgoUML (and to other OS projects) and our activities in the development of improved open standards, are a sustained expression of this support.

And let's not forget Jason Robbins, who started the quest that led us here.

Poseidon includes open source software by Antlr (Java source reverse engineering), Jakarta's log4j (logging), Jakarta's Velocity (Code Generation), Sun's Netbeans project (the UML repository MDR), TU Dresden (OCL support), Piccolo (diagram rendering), Apache's batik toolkit (SVG graphics export), and Freehep (Postscript and PDF rendering).

Appendix A. UML-to-Ecore Plug-In

A.1. Mappings

A.1.1. UML Package

- All directly contained classes will also be converted.
- A package with stereotype <<*ecoremodel*>> will be converted to an EPackage.
- Tagged values **nsURI** and **nsPrefix** are required. They will be transfered to the EPackage, e.g.:

```
nsURI http://www.mysite.org/myMetaModel
                              nsPrefix myMetaModel
```

A.1.2. UML Class

- Attributes must have primitive types. Model other attributes as associations.
- All attributes and associations are converted.
- Stereotype <<*enumeration*>> will be converted to EEnum.
  - All attributes of an <<*enumeration*>> class are converted to EEnumerationLiterals with the same name.
- Stereotype <<*metaclass*>> will be converted to the Ecore element with the same name.

A.1.3. UML Attributes

- Primitive types are converted to corresponding Ecore datatypes.
- The object versions of primitive types are converted to Ecore object datatypes, e.g. **Long** is converted to **ELongObject** .

A.1.4. UML AssociationEnds

- Will be converted to corresponding EReferences with the same name.
- Composite ends will result in a containment EReference.
- If an AssociationEnd is non-navigable, no EReference will be created.
- The multiplicity is converted accordingly.

Appendix B. Poseidon C# Code Generation Plug-In Guide

B.1. General Rules

B.1.1. Tagged Values

These tagged value keys are supported when the value is set to 'true' within the appropriate context:

- internal
- protected internal
- volatile
- override
- sealed
- extern
- internal
- virtual

B.1.2. Additional Stereotypes

- << *event* >>
- << *readonly* >>
- << *delegate* >>

B.2. Modeling Element Rules

B.2.1. Classes

- Uses the standard UML 'Class'
- Supports single inheritance only

B.2.1.1. Class Signature

- Additional visibilities for class signatures are set when the tagged values below are 'true':

  1. internal
  2. sealed

B.2.1.2. Class Attributes

- Additional visibilities for class attributes are set when the tagged values below are 'true':

  1. internal
  2. protected internal
  3. volatile

B.2.1.3. Class Operations

- Additional visibilities for class operations are set when the tagged values below are 'true':

  1. internal
  2. protected internal
  3. override
  4. sealed
  5. extern
  6. virtual

Everything else will use the checked visibility radio buttons

B.2.2. Interface

- Uses the standard UML 'Interface'
- Supports single inheritance only

B.2.2.1. Interface Signature

- Additional visibilities for interface signatures are set when the tagged value below is 'true':

  1. internal

B.2.2.2. Interface Members

- All interface members implicitly have public access. It is a compile-time error for interface member declarations to include any modifiers. In particular, interface members cannot be declared with the modifiers abstract, public, protected, internal, private, virtual, override, or static.

Everything else will use the checked visibility radio buttons.

B.2.3. Structure

- Uses the standard UML 'Class' with the << *struct* >> stereotype
- Supports single inheritance only

B.2.3.1. Structure Signature

Additional visibilities for structure signatures are set when the tagged value below is 'true':

- internal

Struct tapes are never abstract and are always implicitly sealed; therefore the 'abstract' and 'sealed' modifiers are not permitted in a struct declaration. Since inheritance isn't supported for structs, the declared accessibility of a struct member cannot be 'protected' or 'protected internal'.

B.2.3.2. Structure Members

Function members in a struct cannot be abstract or virtual, and the override modifier is allowed only to override methods inherited from the type System.ValueType. A struct may be passed by reference to a function member using a 'ref' or 'out' parameter.

Everything else will use the checked visibility radio buttons.

B.2.4. Enumeration

- Uses the standard UML 'Class' with an << *enum* >> stereotype
- By default, it generates an enum as type 'int'.
- Enum does not participate in generalizations or specifications
- Enum cannot have navigable opposite association ends, operations, or inner classifiers
- Anything else will default to 'int'.

B.2.4.1. Enumeration Signature

Additional visibilities for enumeration signatures are set when the tagged value below is 'true':

- internal

Everything else will use the checked visibility radio buttons.

B.2.5. Delegate

- Uses the standard UML 'Class' with a << *delegate* >> stereotype
- Delegate does not participate in generalizations or specifications

B.2.5.1. Delegate Signature

Additional visibilities for the delegate signatures are set when the tagged value below is 'true':

- internal

Everything else will use the checked visibility radio buttons.

B.2.6. C# Event

C# events are supported with an operation that has the stereotype << *event* >>.

B.2.7. Operations

There are some translations on the return type of C# operations:

- 'in/out' parameter direction will be translated to 'ref'

- ’in’ parameter direction will be translated to blank ("")
- ’out’ will be translated to ’out’
- ’root’ will be translated to ’new’

Appendix C. Poseidon CORBA IDL Code Generation Plug-In Guide

C.1. General Rules

- Everything is modeled using the standard UML 'Class' with an appropriate stereotype as defined by UML Profile for CORBA
- For details about modeling CORBA IDL, refer to the UML Profile for CORBA v1.0

C.2. CORBA Interface

- Uses the standard UML 'Class' with the << *CORBAInterface* >> stereotype
- Interface member has to be 'public'

C.3. CORBA Value

- Uses the standard UML 'Class' with the << *CORBAValue* >> stereotype
- Can only specialize one other concrete CORBA Value
- CORBA Value can only have 'public' or 'private' attributes and navigable opposite association ends
- CORBA Value's 'Factory' method is modeled using the << *CORBAValueFactory* >> stereotype with an Operation
- CORBA Value can have only 0 or 1 << *CORBAValueFactory* >>-stereotyped Operation
- CORBA Value can only have 'public' operations

C.4. CORBA Struct

- Uses the standard UML 'Class' with the << *CORBAStruct* >> stereotype
- CORBA Struct cannot participate in generalizations or specifications
- CORBA Struct can have only 'public' attribute and navigable opposite association end of single multiplicity
- CORBA Struct cannot have operations

C.5. CORBA Enum

- Uses the standard UML 'Class' with the << *CORBAEnum* >> stereotype
- CORBA Enum cannot participate in generalizations or specifications
- CORBA Enum can have only 'public' attributes
- CORBA Enum cannot have navigable opposite association ends
- CORBA Enum cannot have operations

C.6. CORBA Exception

- Uses the standard UML 'Class' with the << *CORBAException* >> stereotype
- Due to current Poseidon limitations, CORBA Exception names must end in the string 'Exception'
- CORBA Exception cannot participate in generalizations or specifications
- CORBA Exception can have only 'public' attributes with single multiplicity
- CORBA Exception cannot be an end of a navigable association end
- CORBA Exception cannot have operations

C.7. CORBA Union

- Uses the standard UML 'Class' with the << *CORBAUnion* >> stereotype
- CORBA Union cannot participate in generalizations or specifications
- CORBA Union can not have operations
- There are two ways to model CORBA Union as specified in UML Profile for CORBA:
  - Using a composition relationship that points to a 'switcher' and has the << *switchEnd* >> stereotype. Every attribute must have a tagged value with 'Case' as the key and the switch condition as the value.
  - Using an attribute with the << *switch* >> stereotype attribute acting as the 'switcher' in conjunction with a composition relationship. The navigable opposite association ends must have tagged values with 'Case' as the key and the switch condition as the value.

Please see UML Profile for CORBA v1.0 Â§3.5.15 for more details.

Appendix D. Poseidon Delphi Code Generation Plug-In Guide

D.1. Classfiers

- Class
- Interface
- Enumeration
- Record
- Set
- Sub Range
- Array
- Exception

D.2. Tagged Values

All strings input in the Tag column are case-sensitive.

D.2.1. Classifier

- Tag = 'uses' with Value = string that represents unit(s) name to be included in specified unit declaration, separated by comma.

  Description : Handles strings that represent the names of units to be included in specified unit declarations. A 'uses' tag with a blank value will be defaulted to 'SysUtils'.

  Example : UnitA, UnitB, UnitC

- Tag = 'setvalue' with Value = string that represents the value of 'Set', separated by comma.

  Description : Handles the way to input the value of the 'Set' type.

  Example : 1,9

- Tag = 'subrangevalue' with Value = string that represents the value of 'Sub Range' separated by comma.

  Description : Handles the way to input the value of the 'Sub Range' type.

  Example : 1,9

- Tag = 'arrayvalue' with Value = string that represents the value of 'Array', separated by comma.

  Description : Handles the way to input the value of the 'Array' type.

  Example : 1,9

- Tag = 'arraytype' with value = string that represents the type of 'Array'.

  Description: Handles the way to input the type of the 'Array' type.

D.2.2. Attribute

- Tag = 'published' with Value = 'true'.

  Description : Handles the published visibility of the classifier 'attribute'.

D.2.3. Operation

- Tag = 'published' with Value = 'true'.

  Description : Handles the published visibility of the classifier 'operation'.

- Tag = 'virtual' with Value = 'true'

  Description : Handles the way to set the specified operation into a 'virtual' type operation.

- Tag ='dynamic' with Value = 'true'

  Description : Handles the way to set the specified operation into a 'dynamic' type operation.

- Tag = ’override’ with Value = ’true’

  Description : Handles the way to set the specified operation into an ’override’ type operation.

- Tag = ’override’ with Value = ’true’

  Description : Handles the way to set the specified operation into an ’override’ type operation.

D.2.4. Exception

- Tag = ’published’ with Value = ’true’.

  Description : Handles the published visibility of ’Exception’.

D.3. Stereotypes

D.3.1. Attribute

- Stereotype = <<*Const*>>

  Description : Handles the way to specify a ’const’ type Attribute.

- Stereotype = <<*property*>>

  Description : This will handle the way to specify a ’property’ type Attribute.

D.3.2. Operation

- Stereotype = <<*function*>>

  Description : Handles the way to specify a ’function’ type Operation.

- Stereotype = <<*procedure*>>

  Description : Handles the way to specify a 'procedure' type Operation.

D.3.3. Classifier

- Stereotype = <<*Enum*>>

  Description : Handles the way to specify an 'Enumeration' type Classifier.

- Stereotype = <<*Record*>>

  Description : Handles the way to specify a 'Record' type Classifier.

- Stereotype = <<*Set*>>

  Description : Handles the way to specify a 'Set' type Classifier.

- Stereotype = <<*SubRange*>>

  Description : Handles the way to specify a 'Sub Range' type Classifier.

- Stereotype = <<*Array*>>

  Description : Handles the way to specify an 'Array' type Classifier.

- Stereotype = <<*Exception*>>

  Description : Handles the way to specify an 'Exception' type Classifier.

D.4. Modeling Element Rules

D.4.1. Class

- Uses the standard UML Class

- Participates in generalizations, associations and specifications
- Only supports single inheritance

D.4.2. Interface

- Uses the standard UML Interface
- Participates in generalizations
- Does not participate in associations or specifications
- Only supports single inheritance

D.4.3. Enumeration

- Uses the standard UML Class with << *Enum* >> stereotype
- Does not participate in generalizations or specifications
- Cannot have navigable opposite association ends or operations

D.4.4. Record

- Uses the standard UML Class with << *Record* >> stereotype
- Does not participate in generalizations or specifications
- Can have navigable opposite association ends
- Cannot have any operations

D.4.5. Set

- Uses the standard UML Class with << *Set* >> stereotype
- Does not participate in generalizations or specifications
- Cannot have navigable opposite association ends or operations

D.4.6. Sub Range

- Uses the standard UML Class with << *SubRange* >> stereotype
- Does not participate in generalizations or specifications
- Cannot have navigable opposite association ends or operations

D.4.7. Array

- Uses the standard UML Class with << *Array* >> stereotype
- Does not participate in generalizations or specifications
- Cannot have navigable opposite association ends or operations

D.4.8. Exception

- Uses the standard UML Class with << *Exception* >> stereotype
- The same as Class

D.5. Specific Rules

- An attribute with 'non-1' multiplicity will generate an Array that is defaulted to type 'int' with Lower Bound and Upper Bound values based on the specified multiplicity.

  Example : Attribute with multiplicity: 1..2 will generate : Array[1..2] of int;

- A blank value with the 'uses' Tag, will be defaulted to 'SysUtils'.
- A blank value with the 'setvalue' Tag will be defaulted to 'a'..'z'
- A blank value with the 'subrangevalue' Tag will be defaulted to 'a'..'z'
- A blank value with the 'arrayvalue' Tag will be defaulted to 1..10
- A blank value with the 'arraytype' Tag will be defaulted to int
- A blank value with the 'procedure' and 'function' Tag will be defaulted to procedure

Appendix E. Poseidon PHP4 Code Generation Plug-In Guide

This guide is based on the PHP4 Manual, available at http://www.php.net/docs.php (http://www.php.net/docs.php) .

E.1. General Rules

- The only classifier in PHP4 is 'Class'.
- PHP4 Class can not participate in an Association.
- There is no Exception in PHP4
- There are two files generated for each Class generation process:

1. '.inc' file that contains the class declaration
2. '.php' file that includes related the '.inc' on its first line

E.1.1. Tagged Values

The following tagged value keys are supported for PHP4 Class:

- '<<<' for Heredoc string
- 'initval' for an initial value of an operation parameter
- '&' for operation parameter passed by reference
- '&' for a function that returns a reference

E.2. PHP4 Class Modeling Rules

- Uses standard UML 'Class'
- Supports single inheritance only

E.2.1. Class Signature

- There are no visibilities for Class Signature

E.2.2. Class Attributes

- There are no visibilities for Class Attributes
- Tagged values supported:
  1. Heredoc

     Tagged value = '<<<', with value = 'true'

     Will return anything typed in the initial value with Heredoc string type.

     For example:

```
$str = <<<EOD
                                        Example of string
                                        spanning multiple lines
                                        using heredoc syntax.
                                        EOD;
```

E.2.3. Class Operations

- There are no visibilities for Class Operations
- Tagged values supported:
  1. Parameter initial value

     Tagged value= 'initval', with value = (specified parameter initial value).

     For example:

```
class ConstructorCart extends Cart {
                                        function ConstructorCart($item = "10", $num = 1) {
                                        $this->add_item ($item, $num);
                                        }
                                        }
```

  2. Parameter passed by reference

Tagged value='&' with value='true' in the parameter signature.

For example:

```
<?php function foo (&$var)
                                    {
                                    $var++;
                                    }

                                    $a=5; foo ($a);
                                    // $a is 6 here
                                    ?>
```

3. Function returns a reference

   Tagged value='&' with value='true' in the operation signature.

   For example:

```
<?php function &returnsReference()
                                    {
                                    return $someref;
                                    }

                                    $newref =& returnsReference();
                                    ?>
```

Appendix F. Poseidon Perl Code Generation Guide

F.1. General Rules

- The result of the code generation is saved as a Module file (ClassName.pm)
- Interfaces and their associations are not translated into Perl code
- Abstracts and their associations are not translated into Perl code
- Element's documentation are translated into Perl comment syntax (# comment)
- Attribute / Parameter types are ignored because there is no need to define data types for Perl variables

F.2. Classes

- Uses the standard UML 'Class'
- Classes are translated into Perl Class (package className)
- A constructor is generated for each class (sub new)

F.3. Class Attributes

- Attributes are translated into variables
- Attributes with single multiplicity are translated into scalar type variables (my $AttributeName)
- Attributes with multi-multiplicity are translated into array type variables (my @AttributeName)
- An attribute with a tagged value 'local' that is set to 'true' is translated into 'local $AttributeName' instead of 'my $attribute'
- An attribute that has non-1 multiplicity with a tagged value 'Map' set to 'true' is translated into '%AttributeName' instead of '@AttributeName'
- When the visibility of an attribute is public, 'use vars qw ($AttributeName)' is added to the code generation.

F.4. Class Operations

- Operations are translated into Sub-routines (Sub OperationName)
- Parameters are translated into Sub-routine variables
- Return value are not translated into Perl code

- When an operation is static and has the stereotype << *create* >>, a ’BEGIN { } ’ block is added to code generation.
- When an operation is static and has the stereotype << *destroy* >>, a ’END { } ’ block is added to code generation.

F.5. Associations

- 1 to 1 associations are translated into scalar type variables (my $className)
- 1 to N associations are translated into array type variables (my @className)

F.6. Aggregation

- 1 to 1 aggregations are translated into scalar type variables (my $className)

F.7. Inheritance

- Single inheritence is implemented using @ISA = qw(class)
- Multiple inheritence is implemented using @ISA = qw(class1 class2 ...)

Appendix G. Poseidon SQL DDL Code Generation Plug-In Guide

G.1. Modeling Element Rules

G.1.1. Classes

- Uses the standard UML 'Class'
- Each class is considered as a table.

G.1.2. Attributes

- Describes the columns in table. Each attribute can have stereotypes that will be treated as column constraints.

G.1.3. Association Ends

- Describes the relationships between tables. Foreign keys will be automatically generated in tables that have references to other tables.

G.2. Tagged Values

These tagged value keys are supported when the value is set with digit number within appropriate context: The values will specifically describe the column data type. Considered in the following order: length, precision, scale

G.3. Additional Stereotypes

Stereotypes apply in attributes. The stereotype for allowing NULL values is not included since it is the default behaviour of columns.

- *<<Primary Key>>*
- *<<Not Null>>*
- *<<Unique>>*

Appendix H. Poseidon VB.Net Code Generation Plug-In Guide

H.1. General Rules

- 'package' visibility will be translated into 'Friend'
- 'abstract' will be translated into 'MustInherit' or 'MustOverride'
- 'final' will be translated into 'NotInheritable' or 'NotOverridable'
- 'static' will be translated into 'Shared'

The following keys for tagged value pairs are supported when the value has been set to 'true' within the appropriate context:

- Shadows
- Overridable
- Protected Friend

H.2. Classes

- Uses the standard UML 'Class'
- Supports single inheritance only
- 'Protected Friend' visibility is determined by setting the tagged value 'Protected Friend' to 'true'. Everything else will use the checked visibility radio button.
- Classes with abstract operations must also be declared 'abstract'

H.3. Interfaces

- Uses the standard UML 'Interface'
- Interface identifiers must start with the 'I' character
- Interface operations must be 'Public' and cannot be 'Shared'
- Interfaces cannot have attributes or navigable opposite association ends

H.4. Modules

- Uses the standard UML 'Class' with the << *Module* >> stereotype

- Modules cannot be 'abstract' or 'final'
- Modules cannot participate in generalization or specification
- Modules cannot be an inner classifier or have an inner classifier
- Modules cannot have a 'Protected' or 'Protected Friend' member

H.5. Structures

- Uses the standard UML 'Class' with the << *Structure* >> stereotype
- Structures must have at least one member that is non-static (shared) and is either an attribute, a navigable opposite association end, or an operation with the stereotype << *Event* >>.
- Structures cannot have a 'Protected' or 'Protected Friend' member
- Structures cannot have an attribute or navigable opposite association end with an initialized value

H.6. Enums

- Uses the standard UML 'Class' with the << *Enum* >> stereotype
- By default, it generates an Enum as type 'Integer'
- Enums do not participate in generalizations or specifications
- Enums cannot have navigable opposite association ends, operations, or inner classifiers
- Other Enum types are supported by using the tagged value key 'type' with one of the following values:
  - Short
  - Byte
  - Integer
  - Long
- Anything else will default to 'Integer'

H.7. Operations

- Operations support the following tagged values:
  - Protected Friend
  - Shadows
  - Overridable
- Operations with no return parameter (returning 'void') are generated as 'Sub'

- Operations with a return parameter are generated as 'Function'

H.8. Operation's Parameters

- Parameter type 'in' is translated as 'ByVal', everything else is 'ByRef'
- The type 'ParamArray' is supported by using the stereotype << *ParamArray* >> with a parameter
- A 'ParamArray' parameter must be the last parameter
- A 'ParamArray' parameter must be of type 'in' or 'ByVal'

H.9. Visual Basic Properties

- Properties are supported with << *Property* >> stereotyped operations
- There are 3 type of stereotypes available:
  - 'Property' will generate 'Get' and 'Set' inside the Property block
  - 'ReadOnly Property' will generate only 'Get' inside the Property block
  - 'WriteOnly Property' will generate only 'Set' inside the Property block
- If no attribute is set in 'accessed attribute', it will by default generate an attribute with same type as the Property return type with the name set to 'm_operation_name'.

H.10. Visual Basic Events

- VB Events are supported by using << *Event* >> stereotypes with operations

H.11. Attribute & Association Ends

- Supports the tagged value 'Protected Friend'

Appendix I. Keyboard Shortcuts

Ctrl-A

Select All

Ctrl-B

Create Collaboration Diagram

Ctrl-C

Complete Copy

Ctrl-D

Create Deployment/Object/Component Diagram

Ctrl-E

Go to Element

Ctrl-G

Go to Diagram...

Ctrl-I

Invert Selection

Ctrl-L

Create Class Diagram

Ctrl-N

New Project

Ctrl-O

Open Project

Ctrl-P

Print Current Diagram

Ctrl-Q

Create Sequence Diagram

Ctrl-S

Save Project

Ctrl-T

Create State Machine Diagram

Ctrl-U

: Create Use Case Diagram

Ctrl-V

: Paste

Ctrl-X

: Complete Cut

Ctrl-Y

: Create Activity Diagram

Ctrl-Z

: Undo

Ctrl-Shift-C

: Copy Representation

Ctrl-Shift-G

: Go to Contained Diagram

Ctrl-Shift-X

: Cut Representation

Ctrl-Shift-Z

: Redo

Ctrl-Alt-C

: Copy to Windows Clipboard

Ctrl-Alt-G

: Generate Roundtrip Source Code

Ctrl-Alt-I

: Import Roundtrip Source Code

F3

: Find...

Alt-F4

: Exit Poseidon

Del

: Delete from Model

Pause

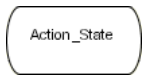
Toggle Rapid Buttons

Glossary

Action

An action is an atomic computation that cannot be terminated externally, and changes the state of the model or returns a value.

Action State

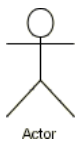


An action state is a simple state in an activity graph representing the execution of an noninterruptible and atomic action that is followed by a transition to another state.

Activation

An activation, also known as focus of control, shows the execution of an operation and the duration of time for that operation.

Actor

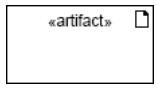


An actor is a representation of an entity that interacts with and derives value from the system.

Aggregation

An aggregation relationship is a 'whole-part' relationship, e.g. a page is a part of a book.

Artifact



An artifact is the specification of a physical piece of information that is used or produced by a software development process, or by deployment and operation of a system. Examples of artifacts include model files, source files, scripts, and binary executable files, or a table in a database system.

Association

An association is a represention of a semantic relationship between instances of objects.

Association End

An association end contains a reference to a target classifier and defines the participation of the classifier in the association.

Attribute

An attribute is a logical data value of a specified type in a class which is inherent to an object. Each object of the class separately holds a value of the type.

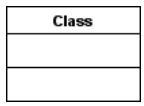
Boundary-Control-Entity-Schema

The boundary-control-entity-schema describes a three layer architecture. The boundary layer is the user interface, control decides what to do with the information gathered from the user interface, and entity holds the data.

Branch

A branch is an element in a state machine where a single trigger leads to more than one possible outcome, each with its own guard condition.

Class



A class is a descriptor for objects that share the same methods, operations, attributes, relationships, and behavior, representing a concept within the system being modeled.

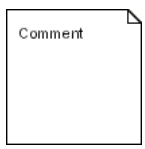
Classifier

A classifier is a model element that describes structural features and behavior. Some classifiers include: class, actor, component, data type, interface, node, and use case.

Collaboration

A collaboration describes a dynamic relationship that exists between objects. Additionally, a Classifier Role should be associated to the collaboration to illustrate the role an element plays in that collaboration.

Comment



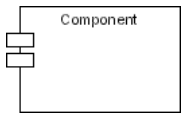
A comment is a textual annotation attached to an element or a collection of elements that has no direct semantics, but may display semantic information.

Previously referred to as a note.

Compartment

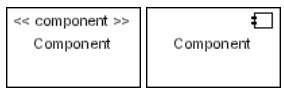
A compartment is a division of a symbol, such as a class rectangle divided vertically into smaller rectangles. Each compartment shows the properties of the represented element.

Component



A component is a replaceable, tangible part of a system that realizes of a set of interfaces, including software code, scripts, or command files, run-time objects, documents, databases, etc.

The new UML 2.0 notation for a component is a simple box with either the component text stereotype or the graphical stereotype depicting the old notation (not supported in this release of Poseidon).



Composition

A composition is a stronger form of aggregation. A part can only be a part of one composite, and the destruction of the whole automatically implies destruction of the parts. Parts with multiplicity that is not fixed can be created after the composite has been created, but once established they live and die with it. Parts can be explicitly removed before the death of the composite.

Constraint

Constraints are expressions that represent semantic conditions or restrictions that are used to limit the use of model elements.

Constructor

A constructor is an operation within the scope of a class that creates and initializes an instance of a class. It may be used as an operation stereotype.

Container

A container is an object that exists to encompass other objects and provide operations to access or iterate over its contents. Examples of containers include arrays, lists, and sets.

Contains

A 'contains' relationship is used to describe a composition relationship; for example, an airplane contains wings.

Control Flow

Control flow represents the relationship between actions in a sequence as well as between input and output objects, shown with messages attached to associations or as solid arrows between activity symbols.

Dependency

A dependency exists between elements and expresses that elements within one package use elements from the package on which it depends, implying that a change in one element may affect or supply information needed by the other element.

Details Pane

The Details pane is a quadrant of the Poseidon work area, located in the lower right corner, which provides advanced editing and viewing capabilities for all elements.

Descriptor

A descriptor is a model element that describes the commonalities of a set of instances, including their structure, relationships, behavior, constraints, and purpose. Most elements in a model are descriptors.

Diagram

A diagram is a graphical presentation of a compilation of model elements, rendered as a graph of shapes connected by paths. Comprehension resides mainly in the topology, not in the size or placement of the symbols.

Diagram Pane

The Diagram pane is the main working area of Poseidon, where all of the diagrams are displayed.

Drill-down Navigation

Drill-down navigation is a means of moving through a model by moving from element to element via the relationships of those elements.

Element

An element is a broad term with little in the way of specific semantics and refers to an atomic constituent of a model.

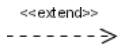
Evaluation Key

An evaluation key is a key granted to a user upon request to allow that user to operate Poseidon for a limited amount of time.

Event

An event is a non-trivial occurrence with a location in time and space.

Extend



An extend relationship exists between an extension use case and a basic use case, and indicates how the behavior of the extension use case can be directly applied to the behavior defined for the base use case. The extension use case incrementally modifies the base use case in a modular way.

Extension Point

An extension point is a named marker that references a location or set of locations within the behavioral sequence for a use case, at which additional behavior can be added.

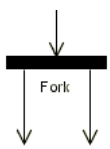
Final Key

A final key is a string provided to Poseidon in order to remove time limits and functionality limits from an evaluation copy of the software.

Final State

A final state is a state within a composite state that, when active, indicates that the activity of the enclosing composite state is complete.

Fork

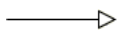


A fork is a complex transition where one source state is replaced by two or more target states, thus increasing the number of active states.

Friend

A friend dependency grants an operation or class permission to use the contents of a class where there would otherwise be insufficient permission.

Generalization



A generalization is a directed relationship between two like elements, where one element is the parent element and the other is the child element. This type of relationship is also referred to as 'kind of', meaning that the child is a kind of the parent.

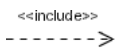
Guard Condition

A guard condition is a boolean expression that must be satisfied in order to enable an associated transition to fire.

Implementation Relations

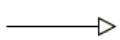
An implementation relation is a relation that exists only between interfaces and classes.

Include



An include relationship defines a dependency relationship between a source use case and a target use case in which the source use case explicitly incorporates the target use case. The source use case can see and use the target, but neither the source nor the target may access each other's attributes. Multiple include relationships may be applied to the same base use case. The same target use case may be included in multiple source use cases.

Inheritance Relations



An inheritance relation allows more specific elements to incorporate structures and behaviors that have been defined by more general elements.

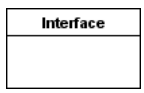
Initial State

An initial state is a syntactic notation indicating the default starting place for an incoming transition to the boundary of a composite state.

Instance

An instance is an individual, concrete entity with its own identity and value. An object is an instance of a class, a link is an instance of an association

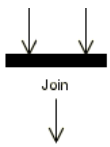
Interface



An interface is a named set of operations that characterize the behavior of an element. Interfaces do not have implementations, they lack attributes, states, and associations, they have only operations.

Interfaces may also be noted in lollipop and socket notations.

Join



A join is a location in a state machine, activity diagram, or sequence diagram where two or more concurrent threads or states combine into one thread or state.

Label

A label is a notational term for the use of a string on a diagram.

Link

A link is an instance of an association.

Lollipop

A lollipop is a type of notation used to denote an offered interface. It consists of a named circle (the interface) and an relationship drawn as a solid line. This is also known as ball notation.

Merge

A merge is a location in a state machine, activity diagram, or sequence diagram where two or more control paths come together.

Message

A message refers to the transfer of information, such as a signal or operation call, from one object to another with the expectation that activity will result. The receipt of a message instance is normally considered an instance of an event.

Metaclass

A metaclass is class whose instances are classes. Metaclasses are typically used to construct metamodels. A metaclass can be modeled as a stereotype of a class using the keyword metaclass.

Method

A method is an implementation of an operation that specifies the algorithm or procedure.

Model

A model is semantically complete abstraction of a system from a particular viewpoint.

Multiplicity

A multiplicity is a specification of the range of allowable cardinality values. It can be an explicit value, a range of values, or an expression that resolves to one or more values.

Name

A name is a string that is defined within a namespace and is used to identify a model element.

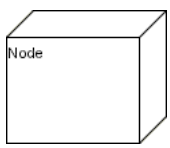
Namespace

A namespace is a part of the model in which names are defined and used, where each name has a unique meaning.

Navigation Pane

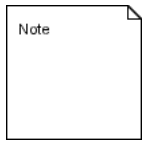
The Navigation pane is located in the top left corner of the Poseidon work area and displays model elements according to pre-determined schemas which can be selected from a dropdown menu.

Node



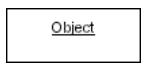
A node is a physical object that exists at runtime and represents a computational resource that executes components. It usually has at least a memory and often processing capability. Nodes can include, but are not limited to, computing devices, human resources, or mechanical processing resources.

Note



The Note element has undergone a name change. See ’Comment’.

Object

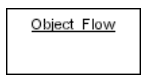


An object is a discrete entity with a well-defined boundary and identity that encapsulates state and behavior, an instance of a class.

Object Constraint Language (OCL)

Object Constraint Language (OCL) is a text language for specifying constraints, writing navigation expressions, boolean expressions, and other queries. It is not intended for writing actions or executable code

Object Flow State



An object flow state represents the existence of an object at a point within a computation. It can also represent the flow of control among operations in target objects.

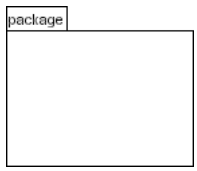
Operation

An operation is the specification of a transformation on the state of an object or a query that returns a value to the caller of the operation.

Overview Pane

The Overview pane is located in the bottom left-hand corner of the Poseidon application and helps the user keep the big picture in mind.

Package



A package, like a file directory, is a general way to put like things together to provide organization. Packages may be nested within other packages.

Parameter

A parameter is the placeholder for an argument that can be changed, passed or returned.

Path

A path is a graphical connection between symbols, usually used to show a relationship.

Plug-in

A Plug-in is a piece of code that extends the capabilities of Poseidon. It may or may not be authored by Gentleware.

Plug-in Key

A Plug-in Key is the string given to Poseidon to activate the Plug-in.

Port

A Port is a connectable element that specifies a set of required and provided interfaces.

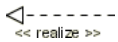
Profile

A profile takes a part of the UML and extends it with stereotypes for a particular purpose.

Project

A project is saved as a zipped .zuml file and contains all information regarding the model, both textual and graphical.

Realization



A realization is the relationship between an element that specifies behavior and one that provides an implementation. A specification describes a behavior or structure without specifying how the behavior will be implemented. An implementation provides the details about how to implement behavior in an effective way.

Relationship

A relationship is reified semantic connection among model elements. Types of relationships include association, generalization, and dependency.

Role

A role is a named slot within an object structure that represents the behavior of an element as it participates in a given context (in contrast to the inherent qualitites of the element).

Socket

A socket is a notation for a required interface. It is denoted as a semi-circle.

Specialization

A specialization produces a more specific description of a model element by adding children. A child element is the specialization of a parent element.

State

A state is a condition or situation during the life of an object during which it satisfies a condition, performs an activity, or waits for an event.

Stereotype

A stereotype characterizes a type of element without specifying its implementation and assists in teh creation of a new model element that is derived from an existing model element.

System

A system is collection of connected units organized to accomplish a purpose. A system can be described by one or more models, possibly from different viewpoints.

Tagged Value

A tagged value is consists of a tag-value pair and is attached to an element to hold some piece of information.

Transition

A transition is a relationship between two states within a state machine where an object in the first state will perform one or more actions and then enter the second state when a certain event occurs and guard conditions are satisfied.

Trigger

A trigger is an event whose occurrence makes a transition eligible to fire.

Type

A type is a declared classifier that the value of an attribute, parameter, or variable must hold. The actual value must be an instance of that type or one of its descendants.

Use Case

A use case defines a piece of behavior of a classifier without revealing its internal structure by describing the behavior of a system from a user's standpoint, providing a functional description of a system and its major processes, and providing a graphical description of users and interactions.

View

A view is a collection of diagrams that describe a particular aspect of the project.

Visibility

Visibility refers to an enumeration whose value determines whether a model element may be seen outside its enclosing namespace.