Poseidon for UML Users Guide

Dr. Marko Boger

Thorsten Sturm

Erich Schildhauer

Elizabeth Graham

Poseidon for UML Users Guide

by Dr. Marko Boger, Thorsten Sturm, Erich Schildhauer, and Elizabeth Graham

Table of Contents

List of Tables

List of Figures

List of Examples

Chapter 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 lead 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 highly 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. 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.

1.2. 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 its ties to the University of Hamburg. With our roots in academia and the community process of open projects, we continuously seek dialog with researchers along with the open source community and users.

1.3. 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.

1.4. How to get in touch with us

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 email. We offer different email addresses for different purposes

General information, feature requests, or suggestions:

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

Customer support (for all versions except the Community Edition):

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

Questions on purchase process, quotes, or volume sales:

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

Web Site

For general discussion we have installed an open forum (http://www.gentleware.com/forum/forum.php3) 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/products/order.php3), 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 (http://www.gentleware.com/products/order.php3) provided on our web site. Our fax number is +49 40 2442 5331.

You can also contact us by phone at +49 40 2442 5330. However, we ask you to use this responsibly. Please try to first find an answer to your question on our web pages (http://www.gentleware.com/support/), the FAQ list, or the Poseidon Users Guide.

Regular Mail

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

Gentleware AG
Schanzenstrasse 70
20357 Hamburg
Germany

1.5. New Features in Version 2.0

Many of the changes made in version 2.0 were implemented to improve the overall performance of Poseidon, but are not readily apparent to the user. Modifications of this sort that are not directly relevant to the User Interface have not been covered in this manual. A short list of UI modifications that have been covered:

- The look and feel of the diagrams has been completely revamped. Among these changes:
  - Moving an association end to a free area of the diagram creates a new class.
  - Waypoints of edges snap to their neighbors' X and Y coordinates.
  - Edge adornments move about the edges more intelligently.
  - Rapid Buttons now include directed associations, attribute creation, and operation creation.

- Diagram storage has been changed to the Diagram Interchange standard, a part of the UML 2.0 standard. This way, diagrams are written in the XMI 1.2 format, just like the model itself.
- Diagrams can be exported to pdf format.
- Project files now are saved with a ".zuml" extension. They are zip files containing a .proj file with project information, and an .xmi file with the model and layout information. All of this is in accordance with the Diagram Interchange standard.
- Undo and Redo is supported throughout Poseidon and for all actions.
- A new graphics engine has been implemented in order to render superior graphics, including anti-aliasing.

1.6. 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.

In the first few chapters we explain everything you need to know in order to install Poseidon for UML and get started. We then provide a guided tour to introduce you to the graphical user interface and the modeling capabilities of Poseidon for UML. In the later sections you will find information about more advanced features.

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 2.0.

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 purpose, commercial or not, for any duration and in any number. 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. A few features are available in the commercial editions but not 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 Community Edition also does not support printing, copy and paste to the Windows clipboard (to copy diagrams to Word or Powerpoint for example), and the zoom is restricted. 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 within the tool.
- Drag and drop within the tool.
- Internationalization and localization for English, German, French, Spanish, and Chinese.
- Code generation 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.

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. That is why we call it UMLdoc.

Poseidon for UML is constructed in a highly modular way and additional features can be purchased from our technology partners and added 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:

- 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 high-end 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. Enterprise Edition

Team support is provided in the Enterprise Edition. In this edition you will find version control, multi-user support, client-server architecture, and much more that

you might need for model-driven software engineering in larger teams. It supports multi-model editing and scales to high volume models.

The Enterprise Edition includes all features of the Professional Edition.

2.5. Embedded Edition

The Embedded Edition is specifically designed for embedded systems development. In order to meet the needs of embedded systems engineers, this version bundles the features of the Professional Edition with optimized code generation for ANSI C and C++. 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 diagrams as well as class diagrams.

2.6. 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	EE Enterprise Edition	Emb Embedded Edition
Simple install with Web Start	✓				
Platform independent	✓	✓	✓	✓	✓
All 9 diagram types	✓	✓	✓	✓	✓

Compliant to UML 2.0	✓	✓	✓	✓	✓
XMI supported	✓	✓	✓	✓	✓
Export as gif, ps, eps, svg	✓	✓	✓	✓	✓
Jpeg and png for JDK 1.4	✓	✓	✓	✓	✓
Internal Copy/cut/paste	✓	✓	✓	✓	✓
Internal Drag and drop	✓	✓	✓	✓	✓
Internationalization	✓	✓	✓	✓	✓
Sophisticated OCL support	✓	✓	✓	✓	✓
Forward Engineering Java	✓	✓	✓	✓	✓
Undo/Redo	Demo	✓	✓	✓	✓
Reverse Engineering Java		✓	✓	✓	✓
Printing		✓	✓	✓	✓
No watermark in graphic export		✓	✓	✓	✓
WMF Graphic Export		✓	✓	✓	✓
UMLdoc		✓	✓	✓	✓
Windows Clipboard for figures		✓	✓	✓	✓
Plugin Support		✓	✓	✓	✓
Support		✓	✓	✓	✓
Smooth zoom		✓	✓	✓	✓
AndroMDA		(✓)	(✓)	(✓)	(✓)
Autolayout		(✓)	(✓)	(✓)	(✓)
RoundTrip			✓	✓	C++
Changeable Code Templates			✓	✓	✓

JarImport			✓	✓	✓
MDL-Import			✓	✓	✓
CORBA IDL-Plugin			✓	✓	✓
Forward Engineering C#			✓	✓	✓
Forward Engineering VB.net			✓	✓	✓
Forward Engineering PHP			✓	✓	✓
XML-Schema-Plugin			(✓)	(✓)	(✓)
Multiuser Functionality				✓	

Notations:

(✓) — Indicates this item can be added to the Edition via plug-ins

Demo — Indicates this item is limited in its usage (Undo/Redo is limited to 3 steps saved in history)

Chapter 3. 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 Runtime Environment or Java Development Kit. JDK 1.4 or higher is required for Linux, Mac OS X, and Windows Platforms. Poseidon for UML will not run with JDK 1.2 or older.
- A computer with reasonable memory and CPU power. For memory, 128 MB is recommended, more is helpful. For CPU, a Pentium III or equivalent is recommended.
- 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 mostly developed and tested on Linux. However, on Windows platforms performance is known to be superior due to a faster Java environment.

Chapter 4. Installation and First Start

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 — this works for the Community Edition only.
- Download the compressed file (.zip file) over the internet and locally install Poseidon for UML.

4.1. 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, you can download the installer that includes Java.

The installer will ask you for an installation folder and where you would like your shortcuts. Free disk space of about 20 MB is required.

Following installation, you can start Poseidon for UML by selecting the icon placed in the “Start” menu by the installer.

4.2. Install through Java Web Start

Java Web Start is a mechanism provided by Sun Microsystems to automatically install and start applications from the internet. After Java Web Start is installed, all you need to do is double-click the provided link. The required files are then automatically loaded to a cache on your local disk and the program is started. The first time this may take a little while, but the second time around most information is taken from the local cache.

The big advantage of this mechanism is that the new version will start automatically as soon as the program is updated on the server. The program also works when you are not online — in this case, the local copy from the cache is used.

First, Java Web Start needs to be installed. If you are using JDK 1.4, Web Start may already have been installed along with the JDK. If not, follow these steps:

1. Download the Java Web Start installation file. You can get it from
   - Gentleware AG at http://www.gentleware.com
   - Or directly from Sun at http://java.sun.com/products/javawebstart/

   You will automatically be provided with a self-installing file for your platform.

2. Close all your browser windows
3. Execute the downloaded file.
4. Open your browser again, then go to http://www.gentleware.com and click on the icon provided for Java Web Start. After a few moments, the latest release version of Poseidon for UML (Community Edition) will start up automatically.

4.3. 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.
2. Navigate to the download area and follow the instructions. You will then have a single file stored on your local hard drive in the location you 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 PoseidonSE-##.##.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 for 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`.

4.4. 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`.

Chapter 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.

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.

5.2. Community Edition

The Community Edition comes complete with a Serial Number immediately upon download. This Serial Number must be registered online from within Poseidon or on the website in order to obtain the License Key required to start Poseidon. 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. The Serial Number will already be provided.

3. Click the ’Register’ button. A dialog will appear. Complete the form and click ’Next’.

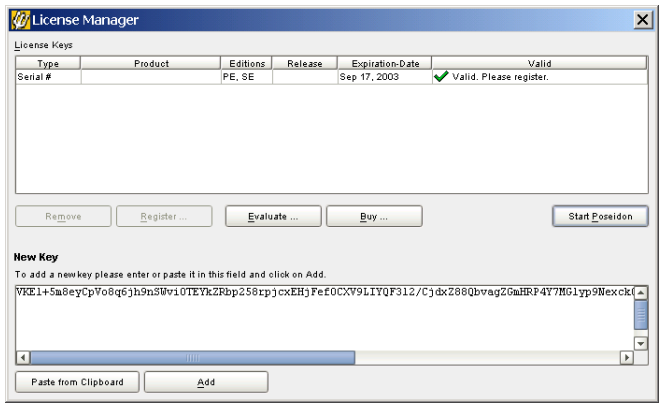
4. To finish the registration process, select either the online option or the web option (ideal for users who are behind a firewall).

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 15 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.



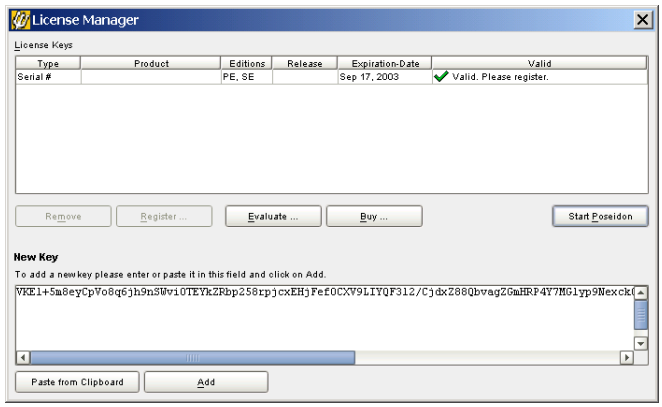
3. Click ’Add’.

5.4. Premium Version Purchase

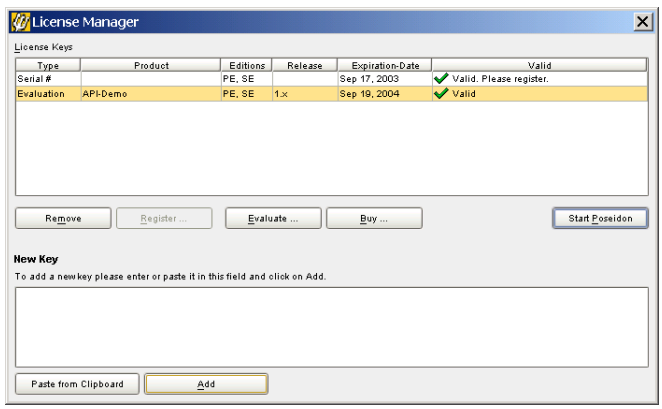
Registering a purchased copy of a Premium Edition follows a similar process as an evaluation copy.

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 which follow.



5.5. 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 Edition comes with four plug-ins. They do not need to be registered separately.

Chapter 6. A Short Tour of Poseidon for UML

This chapter introduces all basic concepts of Poseidon for UML by guiding you through an example model. 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 and thus will not explain all of the details. It will gradually teach you what you can do with Poseidon for UML and how you can use it for your own purposes.

6.1. 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 where a company called *Stattauto* 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.

6.2. 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 6-1. Poseidon for UML application work area.

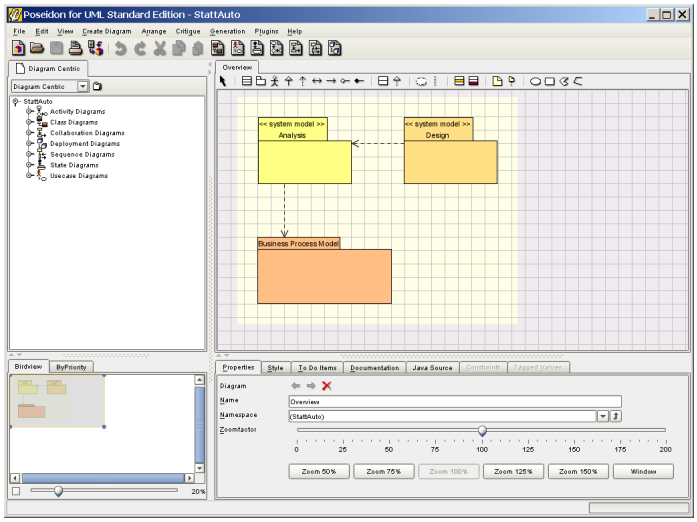


Diagram Pane

- 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

- Critiques assist in the creation of complete and accurate models and compileable code
- 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.

6.2.1. 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 a set of ways to structure this tree — these tree structures are 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 Centric` view.

The root node of the tree is the model itself, in our example it is called `Stattauto`. 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.

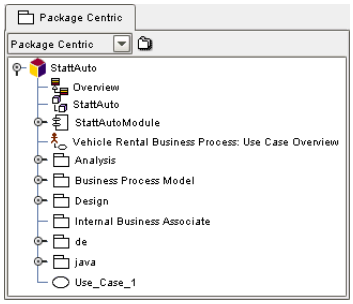
Some sample icons:

- **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:

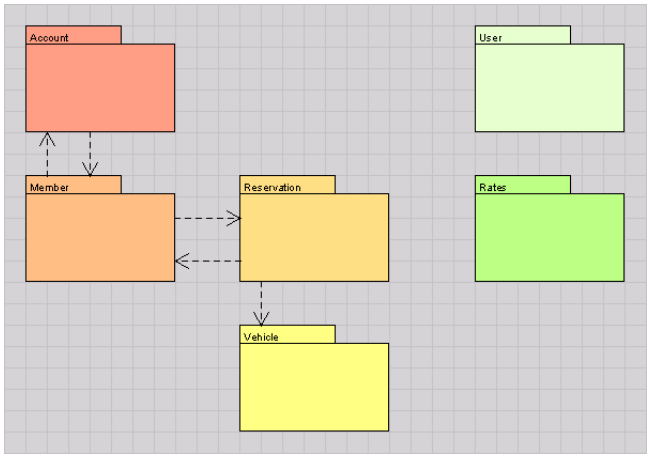
Figure 6-2. Navigation Pane in the Stattauto model.



The model `Stattauto` contains many packages (e.g. `Analysis`, `Business Process Model`, and `Design`) as well as a large number of diagrams (`Overview`, `Implementation: Overview`, ...).

Select the class diagram `Container Class Analysis-Packages` 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: `Account`, `Member`, `Reservation`, `Vehicle`, `User`, and `Rates`.

Figure 6-3. Class Diagram ’Container Class Analysis-Packages’

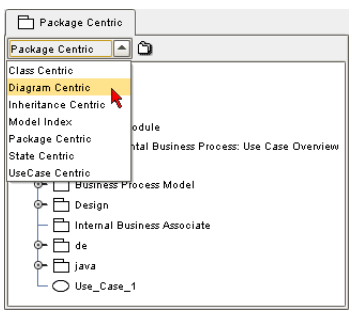


Inside the packages you can find further diagrams, but to quickly browse through the existing diagrams you need not navigate through the packages themselves. You can find diagrams directly (and much more quickly) using the **diagram tree**.

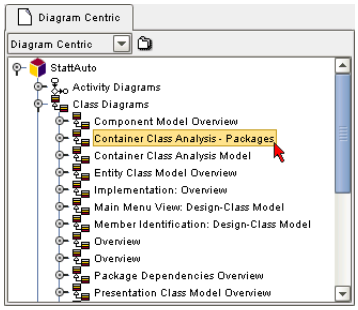
6.2.1.1. Changing the Navigation View

Let’s now take a quick look at the diagram tree, which can be seen in the Diagram Centric view. At the top of the Navigation pane, there is a drop-down selection box. Select the Diagram Centric view.

Figure 6-4. Change a View in the Navigation Pane



Now your Navigation pane should look like this:



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.

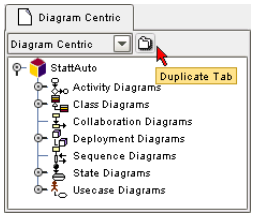
6.2.1.2. Opening Multiple Navigation Panes

As we have seen, views offer different advantages for different tasks. But we often find ourselves switching between these tasks regularly, and constantly changing the dropdown view selector would be a distraction. To give you several choices of views at one time, you can create multiple instances of the Navigation pane by creating additional tabs. The different Navigation panes are accessible through these tabs, and it is then possible to select a different view for each tab.

To open additional Navigation panes:

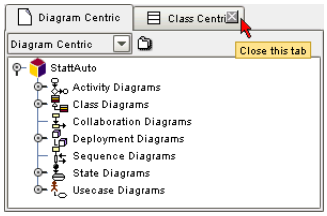
1. Click on the folder icon (called the ’duplicate tab’) that is located to the right of the drop-down selection box.
2. A new navigation view will be created behind the current view.
3. Now you can select the Package Centric view from the dropdown menu of one tab and the Diagram Centric view in the other tab. We will frequently need both views in the rest of the guided tour.

Figure 6-5. Add a Navigation View Tab



You can delete the navigation view tabs using the delete button which appears on the tab, next to the name of the view, whenever the mouse is placed there and two or more tabs are present.

Figure 6-6. Delete a Navigation View Tab



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

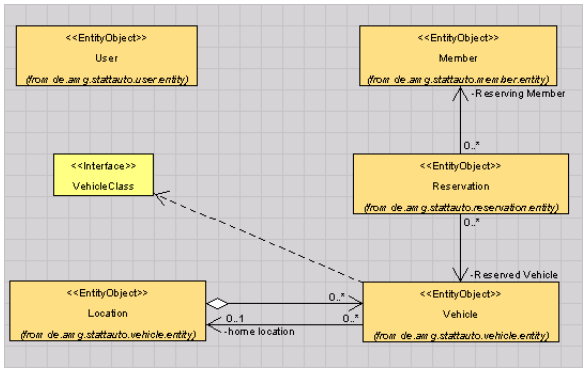
6.2.2. The Diagram Pane

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 `Entity Class Model Overview` in the Diagram pane by clicking on its name 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 6-7. The Diagram pane displaying the diagram 'Entity Class Model Overview'.



This is an overview diagram which provides a high level view of the main entities 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 de.amg.stattauto.user.entity)*). 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 the most important classes of our example model. It already tells you quite a bit about this example:

- It models `Reservations` that have (are associated with) a `Member` and a `Vehicle`.
- `Vehicles` are associated with a `Location`, and `Locations` have `Vehicles`.
- The `VehicleClass` is dependent upon the `Vehicle`.

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.

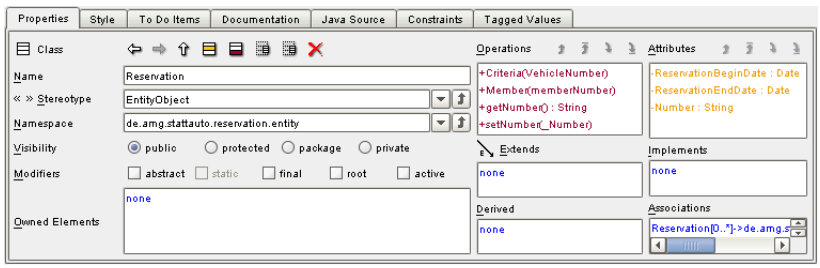
Try it for yourself by selecting one of the classes in this diagram from the Navigation pane. Notice how the class name is highlighted in the Navigation pane, while the Diagram pane displays the same class with its rapid buttons visible around it.

6.2.3. 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 `Reservation` by either clicking on the class itself in the diagram or clicking on the class name in the Navigation pane.

Figure 6-8. The Details Pane with class 'Reservation' selected.



The Details pane is composed of six 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:

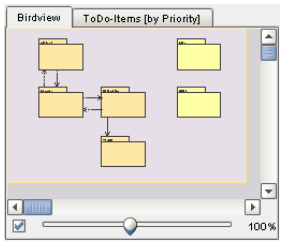
- Properties
- Style

- To Do Items
- Documentation
- Java Source
- Constraints
- Tagged Values

6.2.4. The Overview Pane

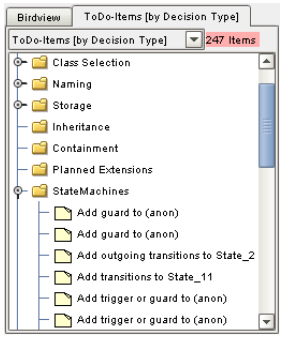
The larger a diagram becomes, the harder it gets to keep track of all of the elements, especially once they are out of the immediate viewing area. The Overview pane allows you to keep track of the elements already in the diagram. The pane, located at the bottom left, provides access to two tabs. First of the two is 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. The second tab, called the 'ByPriority' tab, contains a collection of critiques that have been compiled by Poseidon.

Figure 6-9. 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.

Figure 6-10. Critiques of the Stattauto example in the ByPriority Tab



Click on the tab called 'ByPriority'. This tab contains a collection of critiques that have been compiled by Poseidon. This is a feature that originates from ArgoUML and was one of the motivations for Jason Robbins to start the project. It is a powerful auditing mechanism that discretely generates critiques about the model you are building. Critiques can be hints to improve your model, reminders that your model is incomplete in some areas, or errors that would cause generated code to not compile.

6.3. 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.

6.3.1. 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

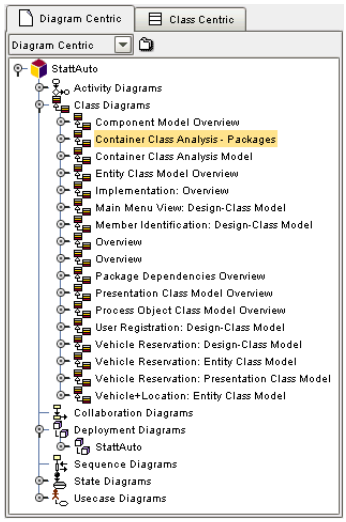
subtrees. At the top of the Navigation pane you will find a drop-down menu where you can choose between a number of views.

Views available from the Navigation pane

- Class Centric
- Diagram Centric
- Inheritance Centric
- Model Index
- Package Centric
- State Centric

Each view organizes the tree structure with its own different focus. By default, the Package Centric view is displayed. You have already seen how to change the view in a previous section.

Figure 6-11. The Navigation Pane in a Diagram Centric View.



Verify that the current view is the Diagram Centric 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 (Class Centric and Diagram Centric) 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.

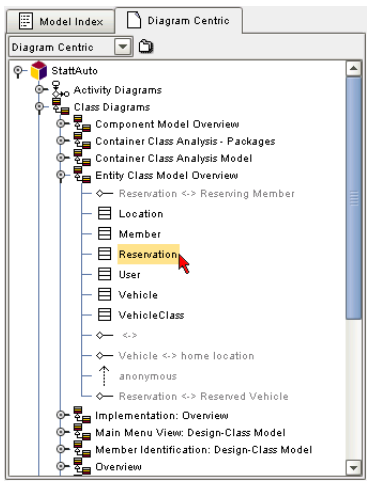
Take a look at the Model Index view by selecting Model Index from the drop-down menu. The Navigation pane will change to display an alphabetical list of all elements in the model. This illustrates yet another useful way to locate elements.

6.3.2. 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.

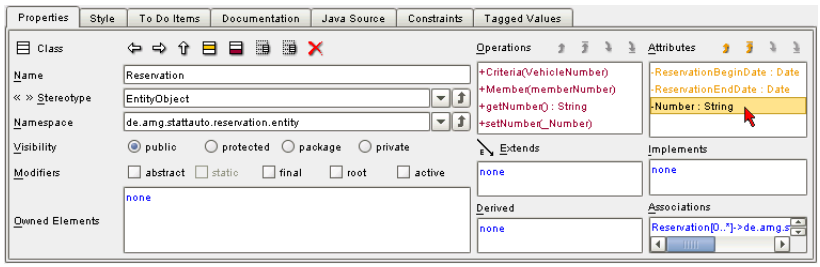
From the Diagram Centric view in the Navigation Pane, open the diagram ’Entity Class Model Overview’ subtree and select the class ’Reservation’.

Figure 6-12. Select class ’Reservation’ from Diagram Centric View



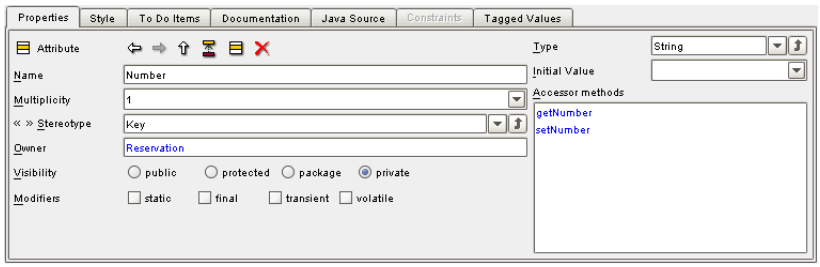
Take a look at the left side of the Properties tab. Listed here are properties of the class itself which can be modified, such as name and visibility. To the right are components of the class, elements in and of themselves. These components have their own properties in their own properties tab.

Figure 6-13. The Details Pane with the class ’Reservation’ selected.



Double-click on the attribute called ’number’. Notice that the Properties tab has changed and now displays the properties of the attribute. Notice, also, that the fields present on the left side have changed to details which are useful for attributes instead of classes. The right side shows us that this attribute has two accessor methods, and to modify those methods we need only double-click on their names to bring up the properties of the selected accessor method.

Figure 6-14. The Properties tab with the attribute ’number’ selected.



6.4. 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.

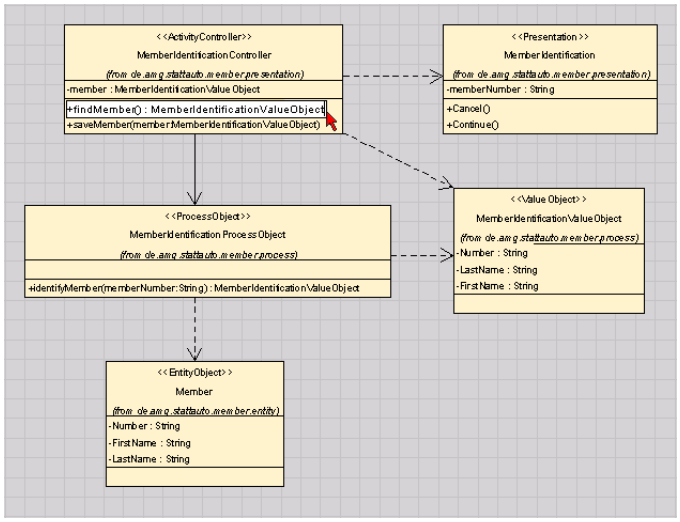
6.4.1. Change Element

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, ’Entity Class Model Overview’ and the Details pane should be displaying information about the attribute ’number’. This diagram has been set to hide attributes and operations, so we will change to a new diagram that displays this information. Select the diagram titled, ’Member Identification: Design-Class Model’ from the Navigation pane. We will now change the name of an operation from the class ’MemberIdentificationController’.

In the Diagram pane, double-click on the operation ’findMember()’ in the class ’MemberIdentificationController’.The Details pane will now display 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 ’searchMember()’ and then press return or click elsewhere in the diagram.

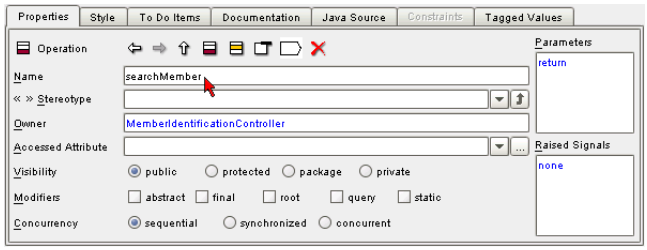
Figure 6-15. Change Operation Name in a Diagram



The name change will be propagated throughout the model, with 'MemberIdentificationController.findMember()' replaced by 'MemberIdentificationController.searchMember()' in every instance.

Another method of changing information is via the Details pane. Select the operation 'searchMember()' again. Notice that the Details pane provides lots of information about this operation. In the 'name' field, change the name from 'searchMember()' back to 'findMember()' 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 6-16. Change Operation Name from the Details Pane



6.4.2. Create Element

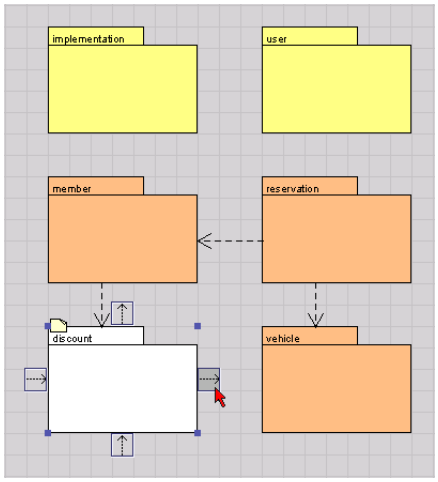
Creating new elements is just as simple as modifying existing ones. And just like changing elements, there are several ways to create new ones.

Perhaps we would like to associate a discount program with members. Let's create a new package, Discount. In the 'Package Dependencies Overview' 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 'member'. 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 'member' 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 6-17. Add a Package to a Diagram with the Rapid Buttons

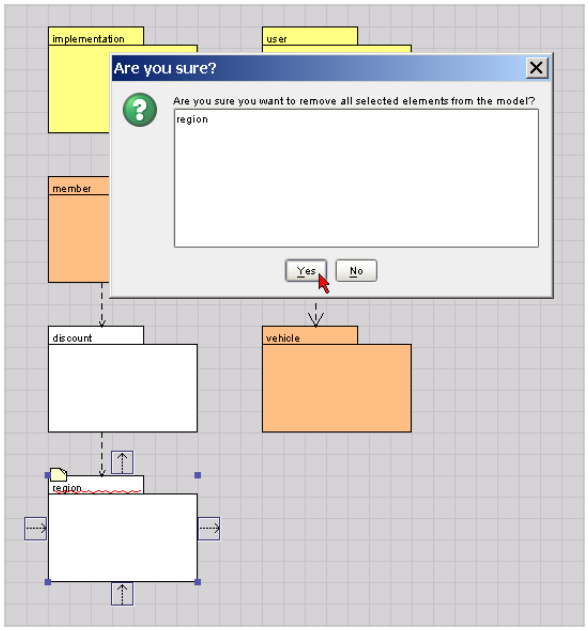


6.4.3. Delete Elements

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.

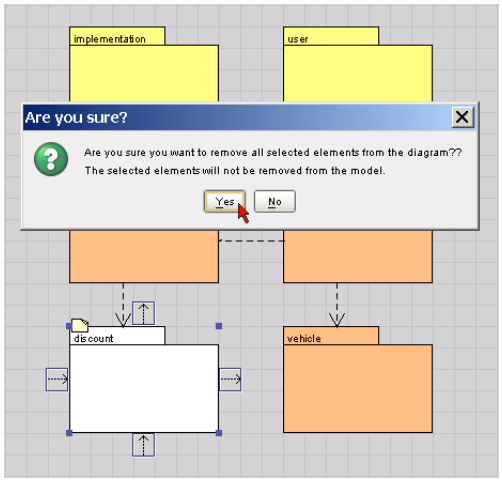
Select the package ’region’ in the Diagram pane. Now press the ’Del’ key. A dialog box will prompt you before removing the class. Notice that the association has been deleted as well, as there is no point to an association with only one end.

Figure 6-18. Delete an Element from a Model



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 class in the Diagram pane. Now cut it from the diagram using either the Cut option from the main toolbar, Cut from the Edit menu, or the shortcut Ctrl-X. You will encounter a warning here as well, but it explains that the element is not deleted completely.

Figure 6-19. Remove an Element from a Diagram



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.

Take some time now to experiment with the example. Our introductory tour through the default example `Stattauto` ends here. You received an impression of how the main tools are used, saw that most operations can be accomplished in more than one way, and eventually saw that there is a lot more to discover in and about Poseidon for UML. The next chapters will show you how edit a model and how to create your own model.

Chapter 7. Working with Diagrams

UML is a graphical language. Therefore, from a users 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.

7.1. The Diagram Pane

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.

7.1.1. 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:

7.1.1.1. 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.

7.1.1.2. Notes

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 note element. Notes are extra comments that are included and displayed in a diagram. These notes can be added to any element including other notes, or they can stand alone in the diagram.

Notes are ignored by the code generator and are therefore never seen in the code output. They are likewise never seen in the Navigation pane.

To add a note to a diagram:

1. Click the Note button in the diagram toolbar.
2. Position the crosshairs in the diagram and click to place the note in the diagram. At this point it is a freestanding note that is not connected to any element.

To connect a note to an element using the toolbar buttons:

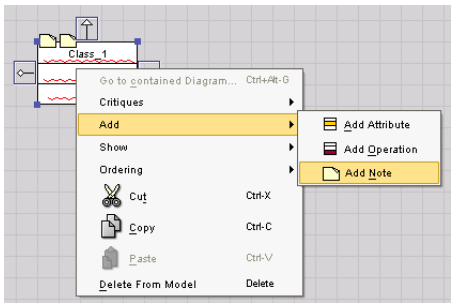
1. Click the Connect Note button from the toolbar.
2. Place the crosshairs over the note to be connected. Click and hold the mouse button.
3. Drag the crosshairs to the element to be connected. Release the mouse button.

To connect a note to an element using the rapid buttons:

1. Click one of the rapid buttons around the note to be attached. Hold the mouse button down.
2. Drag the crosshairs to the element to be connected.
3. Release the mouse button.

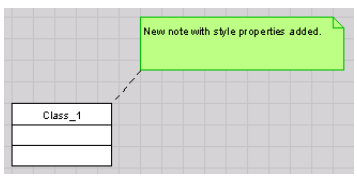
You can either use the ’select’ tool to make the note the current active element and then begin typing, or double-click to open the editable text field.

Figure 7-1. Adding a note through a context menu



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 7-2. A new note



7.1.1.3. 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 and therefore they don't show up in the model tree in the Navigation pane.

The Drawing Tools:

- **Rectangle** — Click in the diagram area and drag the mouse to create a rectangle.
- **Circle** — Click in the diagram area and drag the mouse to create an ellipse.
- **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 clicking to add a third waypoint. Double-click the last waypoint to cease the addition of lines.
- **Polygon** — Click once everywhere the polygon is to have a corner. Double-click the last corner to close and render the polygon.

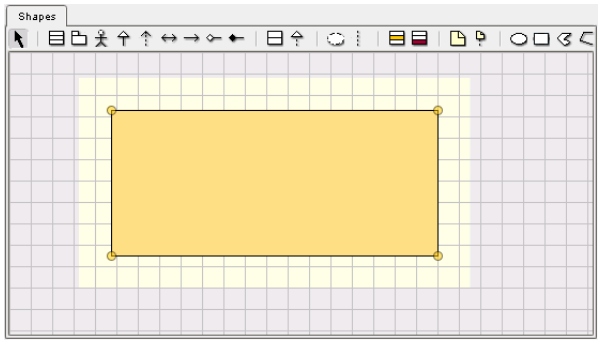
7.1.1.4. Toggle Between Editing Modes

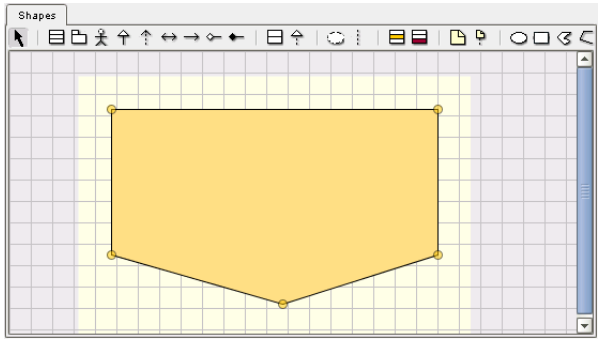
Two modes of editing are available for modifying shapes. You can switch between modes by double-clicking on a shape.

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 7-3. Add a Waypoint to a Rectangle

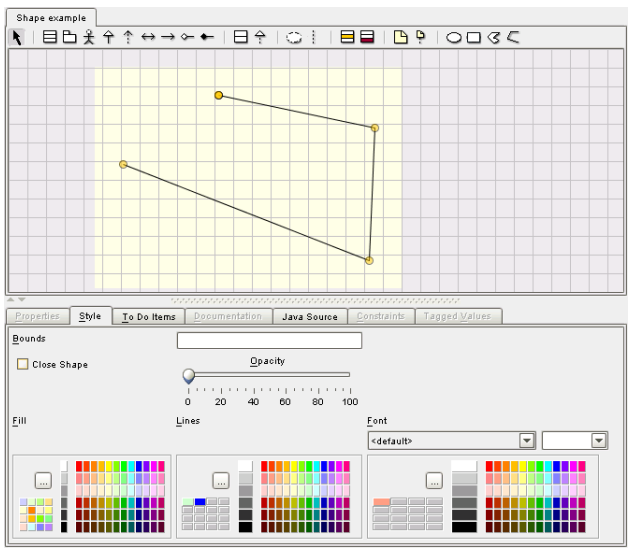


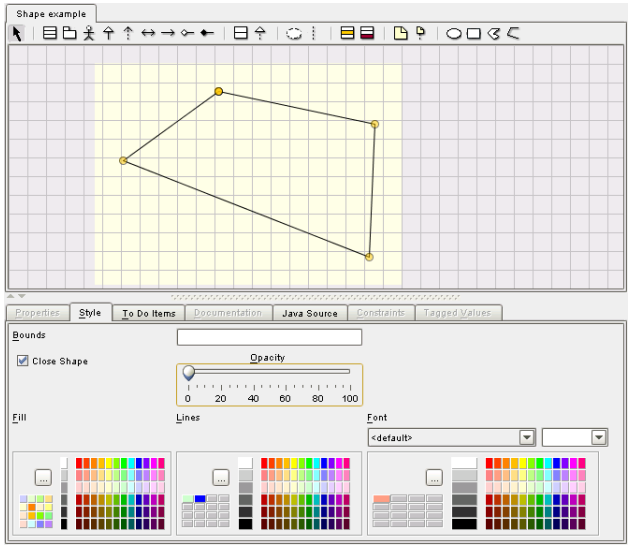


7.1.1.5. Close Shape

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 7-4. Open and Closed Lines

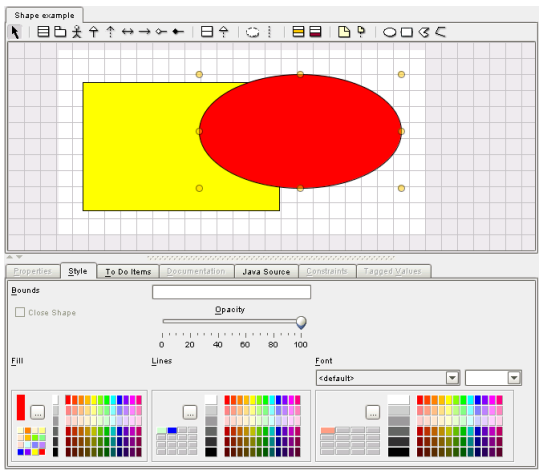


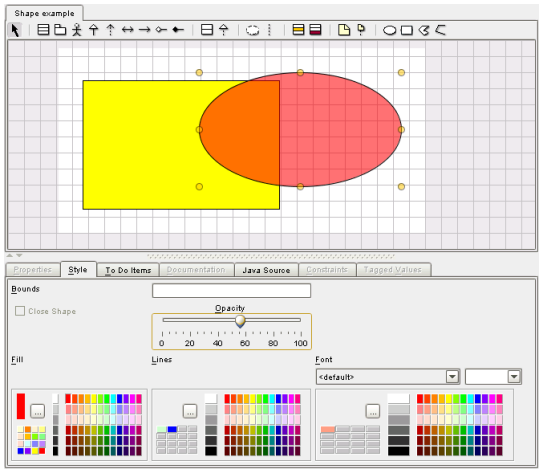


7.1.1.6. Opacity

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 7-5. Changing Opacity





7.1.1.7. 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.

7.1.1.8. 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 similar 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’.

7.2. Viewing Diagrams

Viewing a single diagram is easy. You simply select the diagram 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. 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. 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, but use caution as the entire field is highlighted and it is possible at this point to overwrite the entire field (this is similar to selecting and changing a file name in Windows). Changing the class name here perpetuates the name in the model everywhere this model element is used.

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. For 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.

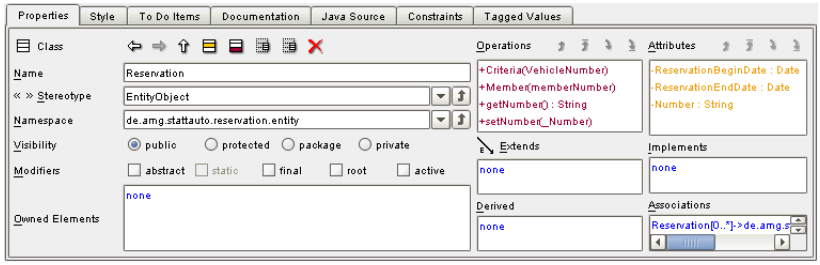
7.2.1. The Details Pane

To explore the facets of the diagram, you can select elements with the select tool and probe 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 also an ⇧ up button to access the element at the next higher level. For a class this is the package or namespace to which it belongs, for example.

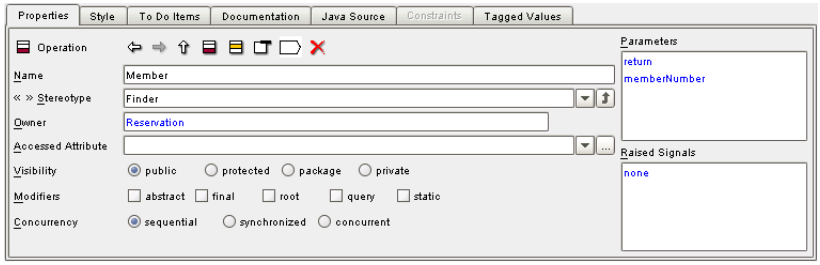
Figure 7-6. Properties tab displaying class ’Reservation’



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 a 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 us navigate to one of the operations of this class. Click the `Member` operation and have a look at its properties.

Figure 7-7. Properties tab with Operation 'Member' Selected.



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

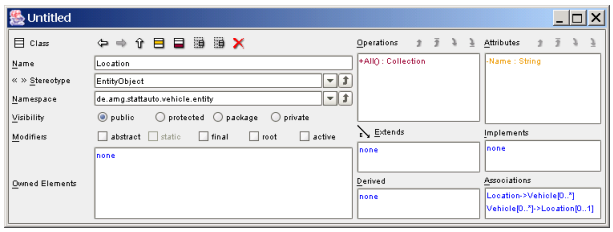
Click on the parameter `return`. The UML specification treats return types as special parameters. Thus every operation has a return parameter that by default is set to `void`. This type can be changed to any other type.

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.

Opening a Tab from the Details Pane in a Separate Window

Screen space is valuable real estate, and this is especially true when working with large diagrams. Space for diagrams can be increased by closing or shrinking the Details pane, but this is not really practical when you need to access that information often. Poseidon allows you to open tabs from the Details pane in separate windows, freeing up screen space.

To open a tab in a new window, double-click on the name of the tab. The new window will open on top of Poseidon.



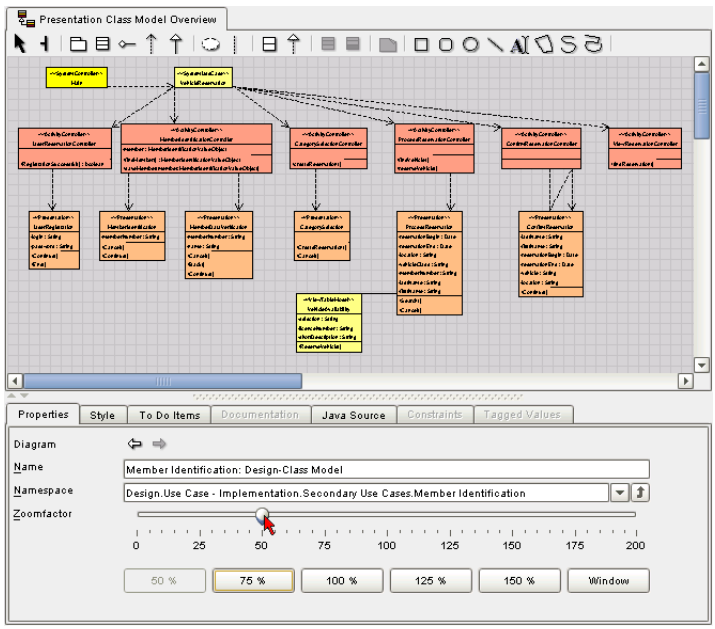
7.2.2. Zooming

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 7-8. 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.
- Hold down the CTRL key and use the mouse wheel to zoom in and out.
- Choose a zoom factor from the menu (View—Zoom).

7.3. 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 for that package. If anything else is selected in the Navigation pane, the new diagram will be created in the topmost package.

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 Diagram — *Ctrl-T*
  - Activity Diagram — *Ctrl-Y*
  - Use Case Diagram — *Ctrl-U*

7.4. 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 se in the next couple of sections.

7.4.1. 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 to this are associations. Any sort of relationships need to exist between two model elements, therefore both of these elements 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.

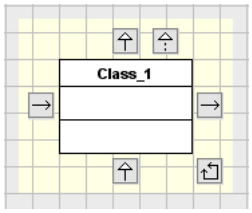
7.4.2. The Rapid Buttons

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 two 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 7-9. Rapid Buttons for a class element.

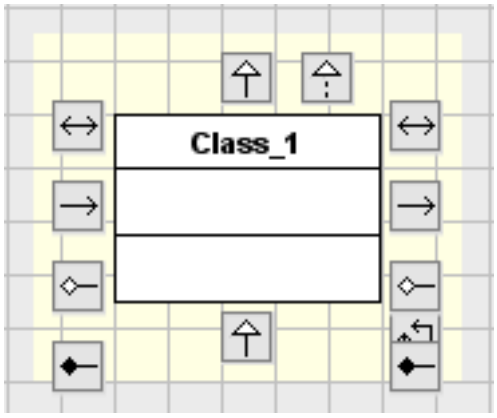


For a class element, the rapid buttons to the left and right represent directed associations, the button on top represents specialization of superclasses, below is the generalization of subclasses, and self-associations are in the bottom right corner.

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.

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 7-10. Additional rapid buttons for a class element.



7.5. Editing Elements

Now that new elements have been created, they must be modified in order to be meaningful to the model.

7.5.1. 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 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.

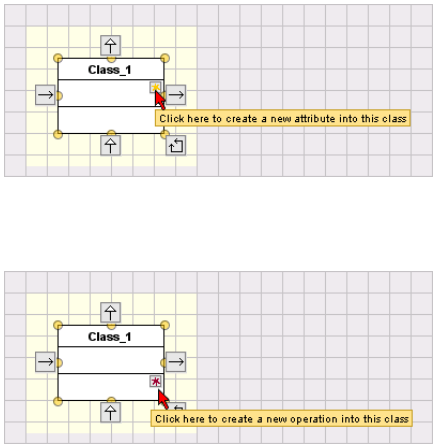
Classes and interfaces offer far more options for editing values than just editing their names. Both of them are constructed of different parts called **compartments**. The first compartment holds the values for the name, the stereotype and the package of the class or interface. You can edit the name 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.

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 7-11. Add a new attribute or operation to a class inline



7.5.2. Editing via the Details pane

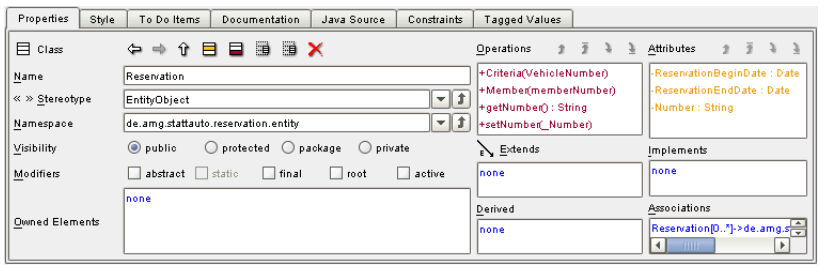
The first tab you will see in the Details pane is the Properties tab.

7.5.2.1. 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 7-12. 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 Stereotype and Namespace must be selected from the dropdown list of available options. The Owned Elements section is automatically populated.

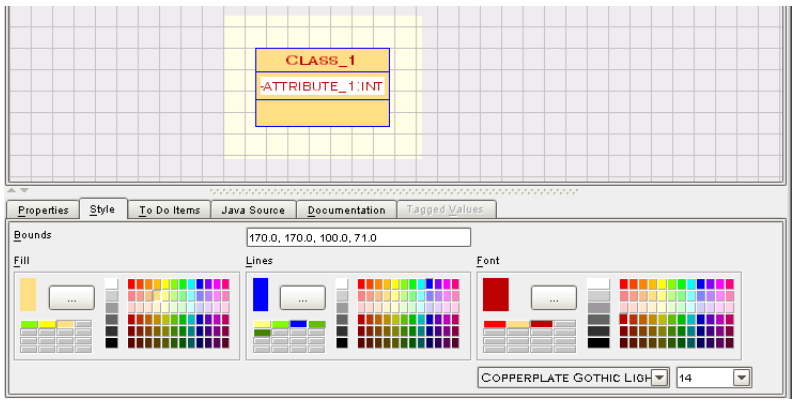
All changes made to the class are propagated throughout the model. For instance, when a namespace is changed, the navigation tree is updated and the class is moved from the original package to the new one that was just selected. This change is also reflected in the Diagram pane: the top compartment of 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.

7.5.2.2. The Style Tab

Next we can look at the Style tab, which determines how the element is rendered in the diagram.

Figure 7-13. Style tab for a class



The Style tab determines 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. 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.

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.

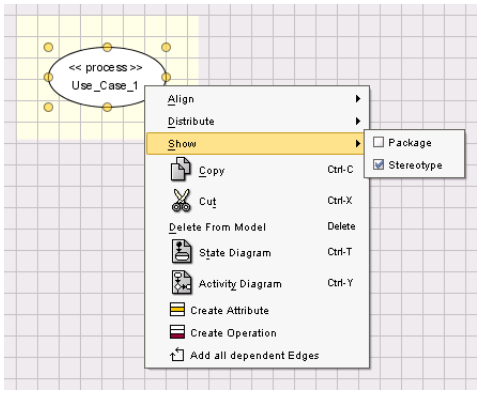
7.5.3. Editing via the Context menu

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 7-14. Context menu options for a Use Case



7.6. Editing Diagrams

Now that you have learned to add and edit elements to a diagram, it is time to learn how to refine the appearance of the diagram itself.

7.6.1. 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.

Drag and Drop can also be used to move a class from one package to another. This can be accomplished by selecting a class in the Navigation pane and dragging it to the destination package. Once the class has been moved, the description that shows the origin of the class is immediately updated.

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

7.6.2. 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.

7.6.3. Layout functions

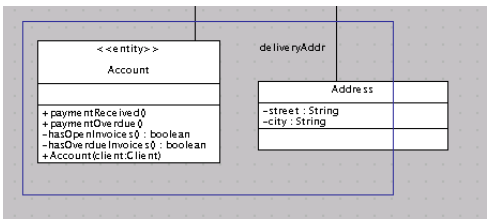
You already know that you can layout your diagram by using the select tool. But there are a number of other ways to rearrange your diagrams.

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 blue rectangle appears and all elements that are completely enclosed in it will be selected.

Figure 7-15. 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 hot key: CTRL-A.

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. They are divided into two groups: align and distribute. The align tools move elements to a specified axis.

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 adjust the spacing of the elements without regard to their alignment.

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

Both groups of tools may be used alone or in conjunction with another tool of a different type. For example,

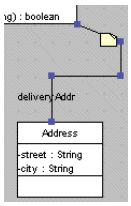
The layout process is supported by a grid. It you want a finer or a coarser grid than the default or if you want the grid to be displayed in a different manner, you can change this in the View menu.

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 7-16. Adding Waypoints.

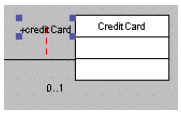


In Poseidon for UML version 2.0, waypoints have changed from blue boxes to yellow circles. Despite the change in appearance, they function in the same way.

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 7-17. 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.

7.6.4. 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. 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. 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

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

  Or the entry ’Delete from Model’ in the context menu; but be careful, this means that it will be removed from the current diagram as well as from all other diagrams. Once an element is deleted, there is no way to get it back again. Additionally, all connections to other elements, such as associations or inheritances, are completely removed.

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 and 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.

7.7. Undo/Redo

Sometimes when working with your models, you might have done something you did not really intend to do. If this happens, the possibility to undo what you did 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-W for redo

Chapter 8. Working with Models

By now you have learned how to navigate an existing model, how to work with existing diagrams and how to add new diagrams. Now let’s take it to the next level and look at what you can do with whole models in Poseidon for UML.

8.1. 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.

8.2. 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.

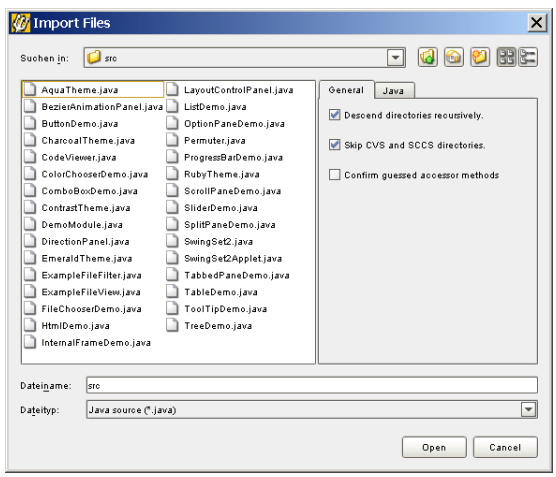
Poseidon 1.5 and 1.6 use a different XMI format than previous Poseidon versions (up to version 1.4.1). Support for XMI 1.1 and 1.2 as well as UML 1.4 was added, and all .zargo files that were created with older Poseidon versions are converted when necessary, and there is no need to care at all about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

8.3. Importing Files

Poseidon provides a slick dialog to assist with the importation of source code. Several options are available when importing this code:

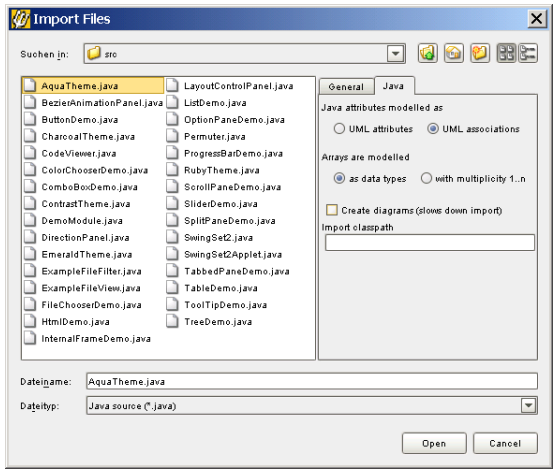
General Tab

- **Descend directories recursively** — Easily add all files below the selected directory
- **Skip CVS and SCCS directories** — Ignore version control files
- **Confirm guessed accessor methods** — A dialog appears to verify that methods are indeed accessor methods

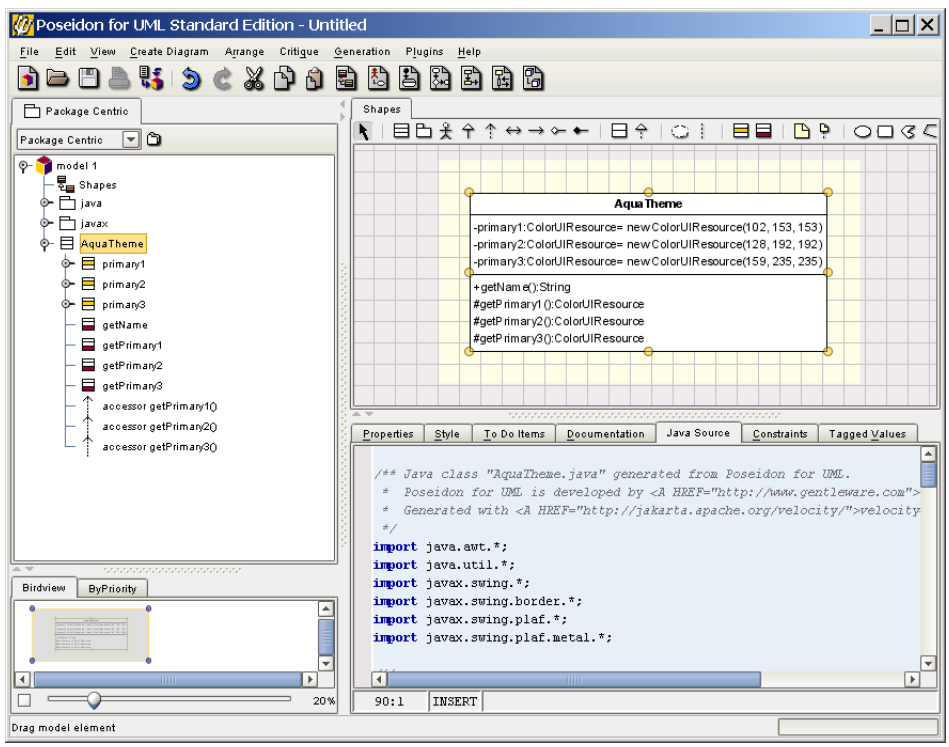


Java Tab

- **Java attributes modelled as** — Select either attributes or associations
- **Arrays are modelled** — Select either as data types or with a multiplicity of 1..n
- **Create diagrams (slows down import)** — Unchecking this box will speed up importation.



Here is an example of what an imported Java class looks like in the Navigation Pane and in the Diagram pane, as well as showing a bit of the source code:



8.4. Importing Models

Models can be imported directly into other 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 for UML 1.6 can import files stored in MDL format — the file format used by Rational Rose. This feature is available in the Professional Edition.

To Import Sub-Models:

- **Main Menu** — Select 'Import Files' from the main menu
- **Quick-Key** — Use the quick-key CTRL-I to import files.

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

8.5. 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:

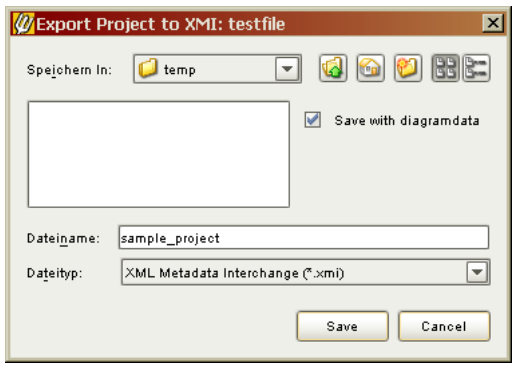
Some tools are especially well prepared for capturing models designed in cooperative sessions on the white-board. The model is sketched on the white-board, the gestures are tracked and transformed to UML model elements, and a laptop in conjunction with a projector places the newly-created diagram back 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 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 8-1. Export Project to XMI



8.6. 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 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.

Beginning with Poseidon for UML version 2.0.4, graphics generated from the Community Edition contain a watermark that appears in the background of the exported graphic file but does not affect any of the diagram information. Watermarks are not generated from any of the Premium Editions. Figures 8–1 and 8–2 depict the same diagram, but 8–1 was saved from the Community Edition and 8–2 was created in the Professional Edition.

Figure 8-2. Watermarked Community Edition Diagram Graphic

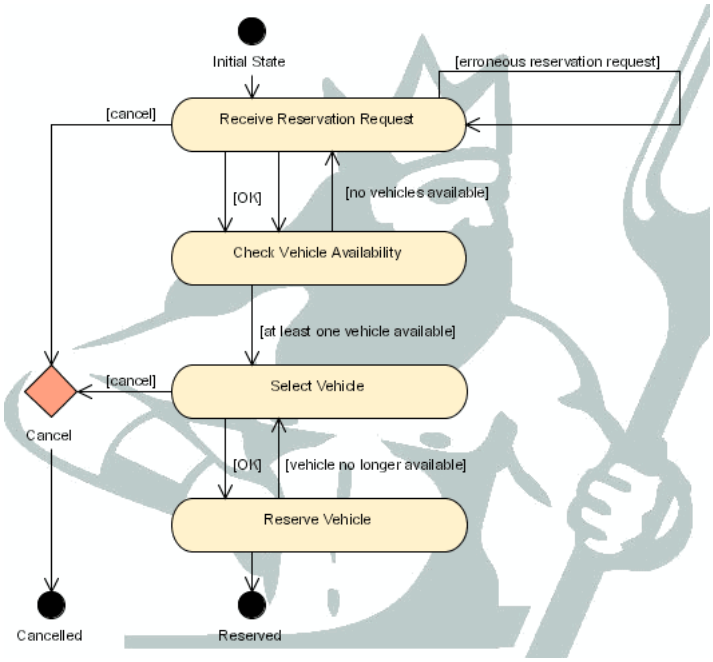
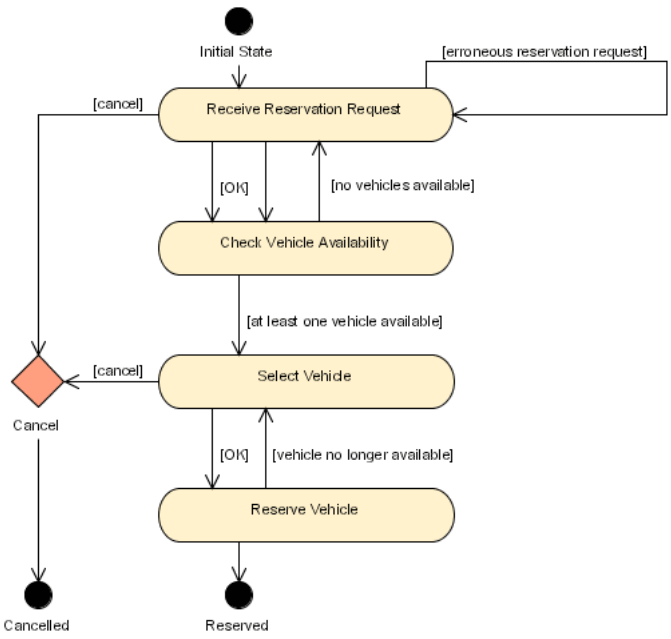


Figure 8-3. Premium Edition Diagram Graphic Without Watermark



Printing

You can also directly print diagrams to a printer. In the Page Setup dialog, you can specify how many diagrams to print per page — this allows you to place several diagrams on each print, e.g. 2x2. The Print function (CTRL-P) prints the current diagram. 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. These printing functions are not available in the Community Edition.

Chapter 9. A Walk through the Diagrams

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 6.

9.1. 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, these are *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 9-1. A Use Case Diagram.

9.1.1. 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.

9.1.2. Toolbar


Select


Package


Actor


Use Case


Generalization


Dependency


Association


Directed Association


Aggregation

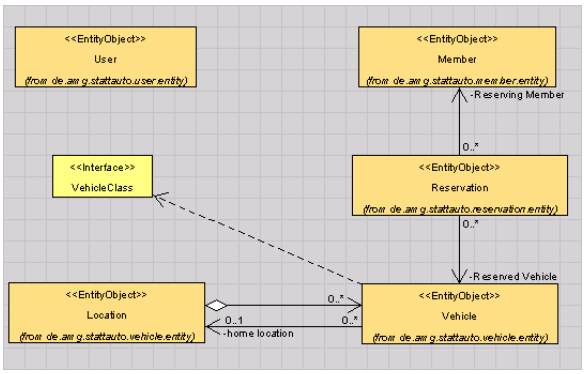
- Composition
- Include
- Extend
- Extension Point
- Collaboration
- Classifier Role
- Note
- Connect Comment to Element
- Ellipse
- Rectangle
- Polygon
- Polyline

9.2. Class Diagrams

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 9-2. 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.

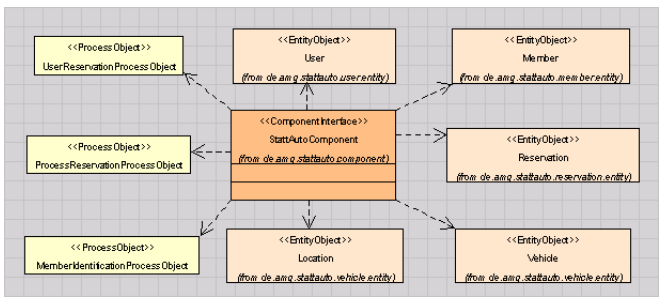
If a class is shown in a diagram of a different package, the text *(from package.subpackage)* is displayed just under the class name in the diagram. You can turn it off with the Context menu of the class. Move the mouse over the class, right-click, and select Display — Hide Package display.

9.2.1. 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 9-3. A Class Diagram making use of 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.

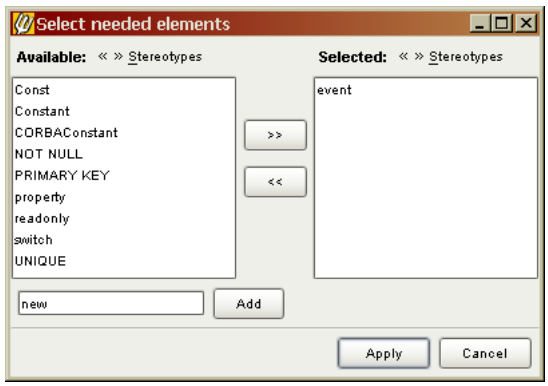
Poseidon supports multiple stereotypes for single elements. Adding, editing, and removing these stereotypes is accomplished via a dialog that is accessible from the Details pane.

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. Right-click in the Stereotype box.
4. Select ’Edit’ from the menu.

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.

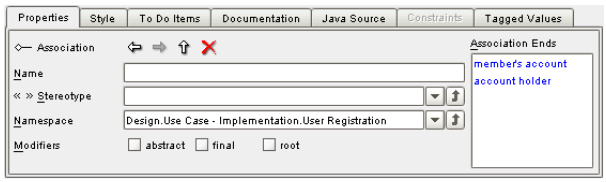
Figure 9-4. Stereotype Dialog



9.2.2. Associations

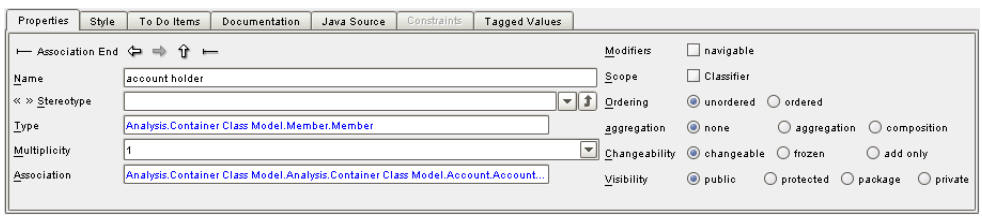
Associations are very important in UML. They have properties in and of themselves, as well as consisting of other model elements. Every association has two ⊢ association ends that are model elements in their own right, as defined in the UML specification. Figure 9–4 shows the Properties tab for an association, 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 9-5. 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.

Figure 9-6. 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.

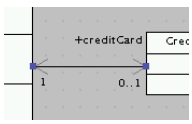
9.2.2.1. Navigability

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 9-7. Highlight hints for associations.



9.2.2.2. 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:

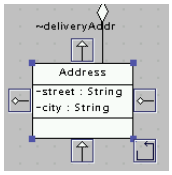
1. Select the diagram where you want to change the setting.

2. Go to the Style tab.
3. Activate or deactivate the ’Show association multiplicity of 1’ check box.

9.2.2.3. 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.

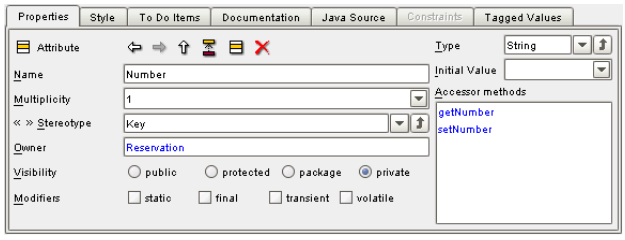
Figure 9-8. The rapid button for self-associations (lower right).



9.2.3. 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 9-9. Properties of an Attribute.



Attribute Properties

- **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.

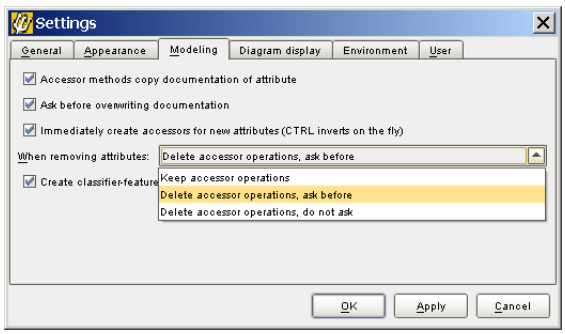
- **Modifiers** — You can also set whether the attribute is write-only by checking the **final** check box. An attribute can also be **static** (class scope instead of instance scope) or **transient** or **volatile**. An initial value can be given as a Java expression.
- **Multiplicity** — The multiplicity field determines how many references the class has to this attribute.
- **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 and thus, will not be marked as accessor any more. You

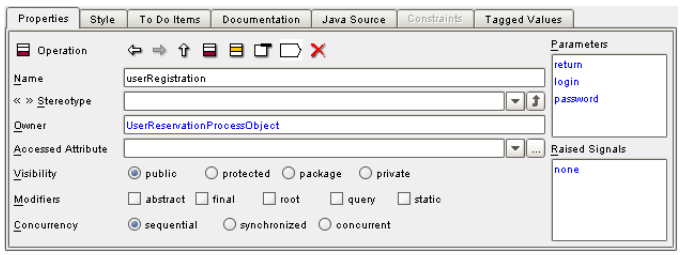
will be asked if you want to remove this relation only or if you want to delete the operation completely. Whether this dialog appears or not is determined by your settings, which can be accessed in Edit—Settings. The 'Modeling' tab is displayed in Figure 9–9. Notice the 'When removing attributes' drop-down 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 9-10. 'Remove Attributes' Setting



9.2.4. Operations

Figure 9-11. 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).

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.

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.

The last list, 'Raised signals', is used to define whether this operation throws exceptions. Select 'Add...' from the context menu, 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 can define a constructor by setting the stereotype `<<create>>`. When you do this, the name of the operation is automatically set to the class name, and the return type is also of the class. The Java code generation respects this fact by correctly generating a constructor signature.

9.2.5. 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 which implements the interface is also responsible for implementing the operations.
- **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.

9.2.6. Toolbar


Select


Class


Package


Actor


Generalization


Dependency


Association


Directed Association


Aggregation


Composition


Interface


Realization


Collaboration


Classifier Role


Attribute


Operation


Note


Connect Note to Element


Circle


Rectangle

- Polygon
- Polyline

9.3. Object Diagrams

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 deployment 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 deployment diagrams.

9.3.1. Diagram Elements

- **Nodes and Instances of Nodes** — Nodes represent the hardware elements of the deployment system.
- **Components and Instances of 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.
- **Objects, Classes, Interfaces** — Components and nodes can include objects, classes or interfaces.

9.3.2. Toolbar

- Select
- Node
- Instance of a Node
- Component
- Instance of a Component
- Dependency
- Class
- Interface
- Association
- Directed Association
- Aggregation
- Composition
- Object
- Link
- Note
- Connect comment to element
- Circle
- Rectangle
- Polygon
- Polyline

9.4. 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 diagrams.

The following example shows an activity diagram that depicts the rules and the process of paying an order. In the following example, `Softsale` will not accept an order if you have overdue payments open, will only allow payment by invoice if your e-mail and home address have been verified, and a few other rules. Take a closer look for yourself in order to become more familiar with the notation.

Figure 9-12. An Activity Diagram.

9.4.1. Diagram Elements

- ● **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.
- ⊛ **Synchronization States** — Used in concurrent sub-sequences to synchronize producer-consumer relations.
- → **Transitions** — The ingredient that keep states active and the model elements together. Each transition can be given *guards* [A], *triggers* ✳, and *actions* A 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.

9.4.2. Toolbar


Select


Action State


Object Flow State


Transition


Initial State


Final State


Synchronization State


Branch

Fork

- Join
- Note
- Connect comment to element
- Circle
- Rectangle
- Polygon
- Polyline

9.5. State 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.

Let’s take a short look at the States themselves. In the editors toolbar you find three different symbols:

- **State**

  In a state 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 actions or activities that are to be taken with it.

- **Composite State**

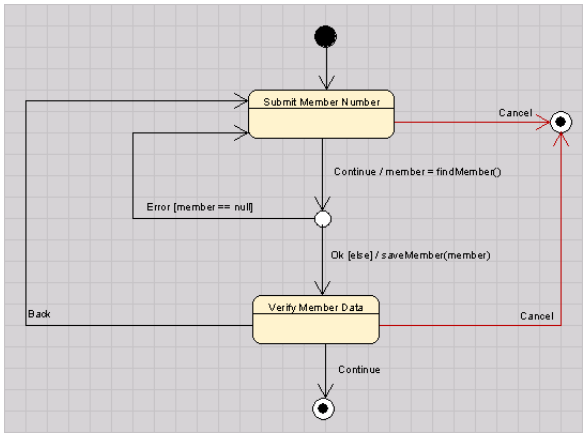
  Composite States make visual use of the second compartment that encloses refinements of the given state. Enclosed states don’t have to have an initial state. Ingoing as well as outgoing transitions might be connected directly to one of them. When the corresponding object is in the composite state, it is exactly in one of the sub-states (OR relation).

  If you find yourself needing to change a simple state to a composite state, you have to delete the former and again add the new state via the toolbar.

- **Concurrent State**

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

Figure 9-13. A State Diagram



9.5.1. Diagram Elements

- ● **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.
- ⊛ **Synchronization States** — Used in concurrent sub-sequences to synchronize producer-consumer relations.

- → **Transitions** — The ingredient that keep states active and the model elements together. Each transition can be given *guards* [A], *triggers* ✳, and *actions* A 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.

- ○ **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.

9.5.2. Toolbar


Select


Simple State


Composite State


Concurrent State


Transition


Initial State


Final State


Synchronization State


Deep History


Shallow History


Choice


Junction


Fork


Join


Note


Connect comment to element


Circle


Rectangle


Polygon

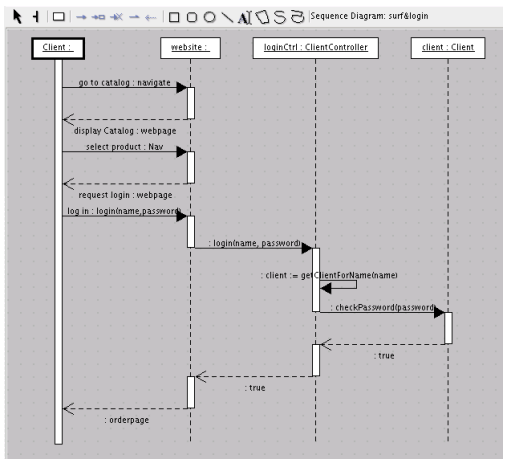

Polyline

9.6. 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. 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).

The following diagram shows a typical example:

Figure 9-14. A Sequence Diagram.



Objects

After creating or changing objects, they are automatically arranged in the Diagram pane. You can specify which of the objects is to have control by enabling the corresponding check box 'Focus of control' in the Properties tab. Afterwards, you'll see how the graphical representation of this object changes: it gets a thick border and its lifeline is no longer a dashed line but a solid rectangular area.

Self messages

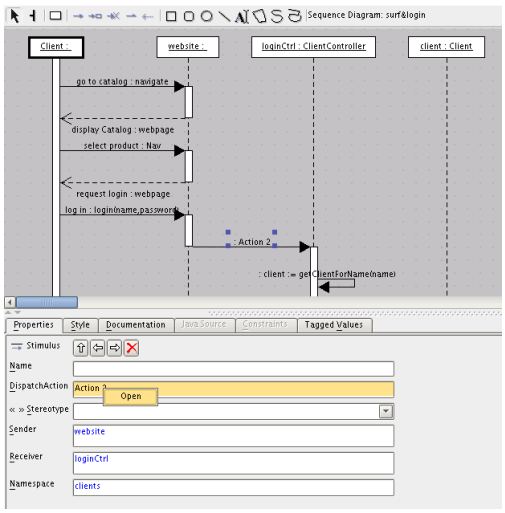
In Poseidon, all stimulus types, except create stimuli, can be created as self stimuli. In the case of a stimulus to itself, the arrow starts and finishes on the object's lifeline. A self stimulus is created by selecting the desired stimulus on the toolbar

and then double-clicking on the object's lifeline at the position you want the stimulus to be placed.

Selecting an operation

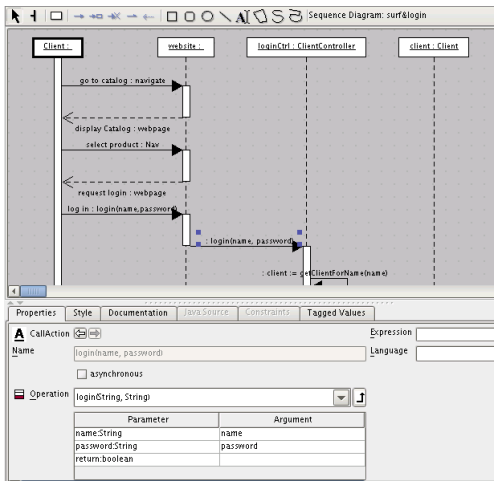
A call stimulus is regarded as a procedure call and can be connected with any operation provided by the receiving object, depending on its type. This is achieved by connecting the stimulus with an action that will cause the class operation to be called. The following two figures show an example for selecting an operation and attaching actual arguments to the call.

Figure 9-15. Selecting the action of a stimulus in a sequence diagram.



After selecting the stimulus in the diagram, the Details pane shows the properties of the stimulus. It is there that you have to open the dispatched action field displayed in the Details pane, which is directly below the name field of the stimulus. This causes the Details pane to change the view to the properties of the action.

Figure 9-16. Selecting an operation and attaching arguments to it.



The properties of the action allow you to select an operation and edit the arguments attached to the procedure call. The set of possible operations includes all operations of the receiving object's class, as well as any operations inherited from direct and indirect superclasses or interfaces. If an operation is selected, the name of the action is updated according to the name of the operation and the given values of the arguments. An empty argument value is displayed as an 'x'. Keep in mind that you cannot edit the name field while an operation is selected.

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.

9.6.1. Diagram Elements

- **Objects** — Elements responsible for sending and receiving messages.
- **Call stimuli** — Represents a synchronous message, which means that it is regarded as a procedure call.
- **Send stimuli** — 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.
- **Return stimuli** — Represents the return statement of a call stimulus.
- **Create stimuli** — Used to create a new object at a certain point in the sequence. The created object will then be placed at this specific point and not at the top of the Diagram pane.
- **Destroy stimuli** — Used to destroy an object at a specific point in the sequence. The lifeline of the destroyed object will then end with a cross at this point and not at the bottom of the Diagram pane.

9.6.2. Toolbar


Select


Object


Call Stimulus


Create Stimulus


Destroy Stimulus


Send Stimulus


Return Stimulus


Note


Connect comment to element


Circle


Rectangle


Polygon


Polyline

9.7. Collaboration Diagrams

Collaboration diagrams are also a means for representing the interaction 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.

9.7.1. 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 .

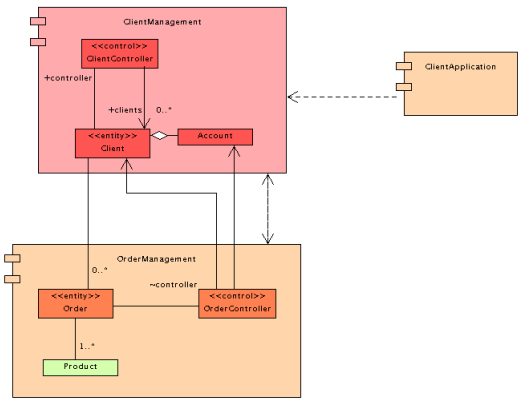
9.7.2. Toolbar

- Select
- Object
- Link
- Call Stimulus
- Create Stimulus
- Destroy Stimulus
- Send Stimulus
- Return Stimulus
- Note
- Connect comment to element
- Circle
- Rectangle
- Polygon
- Polyline

9.8. 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. As mentioned above, component diagrams are, like object diagrams, edited with the deployment diagram editor and therefore the corresponding model elements are explained in that section.

Figure 9-17. A Component Diagram.



9.8.1. Diagram Elements

- **Nodes and Instances of Node**s — Nodes represent the hardware elements of the deployment system.
- **Components and Instances of 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.
- **Objects, Classes, Interfaces** — Components and nodes can include objects, classes or interfaces.

9.8.2. Toolbar


Select


Node


Instance of a Node


Component


Instance of a Component


Dependency


Class


Interface


Association


Directed Association


Aggregation


Composition


Object


Link


Note


Connect comment to element


Circle


Rectangle

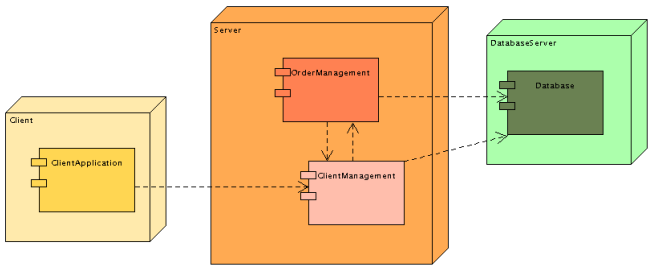

Polygon


Polyline

9.9. Deployment Diagrams

Finally the way the individual components are deployed to a hardware system can be described using the deployment diagram. Because we decided to merge the different diagram types, the editor contains a wide set of elements to be used (see also: Object diagrams Component diagrams). Deployment diagrams are defined on two levels: object or instance level and class level. For this reason, Poseidon for UML provides both nodes and instances of nodes.

Figure 9-18. A Deployment Diagram.



9.9.1. Diagram Elements

- **Nodes and Instances of Nodes** — Represent the hardware elements of the deployment system.
- **Components and Instances of Components** — Represent software elements that are deployed to the hardware system.
- **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.
- **Objects, Classes, Interfaces** — Components and nodes can include objects, classes or interfaces.

9.9.2. Toolbar

- Select
- Node
- Instance of a Node
- Component
- Instance of a Component
- Dependency
- Class

 Interface
 Association
 Directed Association
 Aggregation
 Composition
 Object
 Link
 Note
 Connect comment to element
 Circle
 Rectangle
 Polygon
 Polyline

Chapter 10. Panes

The panes in Poseidon for UML divide the application workspace into 4 sections, each with a specific purpose. They make the process of modelling easier by providing quick access to all parts of the project. These panes can be resized and hidden as you require.

10.1. 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 ’Softsale’.

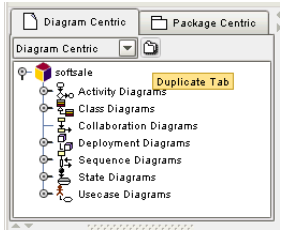
The views offered by Poseidon are as follows:

- **Class Centric**
- **Diagram Centric**
- **Inheritance Centric**
- **Model Index**
- **Package Centric**
- **State Centric**

10.1.1. Add a tab

Adding a tab allows you to view several navigation views at one time. You can add

as many tabs as you like to the Navigation pane, up to the number of views, that is.

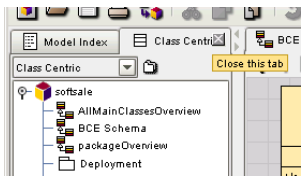


To add a tab to the Navigation pane:

1. Click the Add Tab button. A new tab will appear in front.
2. Select a different view for this tab from the dropdown list.

10.1.2. Delete a tab

Deleting a tab is equally as easy.



To delete a tab:

1. Move the mouse to the right side of the name of the tab. A ’close’ button with an ’X’ on it will appear.
2. Click this button to close the tab.

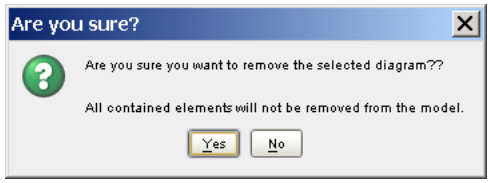
10.1.3. Delete a diagram

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.

10.2. Diagram Pane

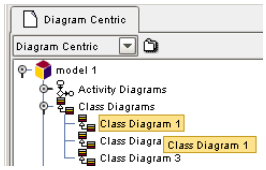
The Diagram Pane is the area used to do most of the diagram creation and modification. It is generally the largest pane.

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.

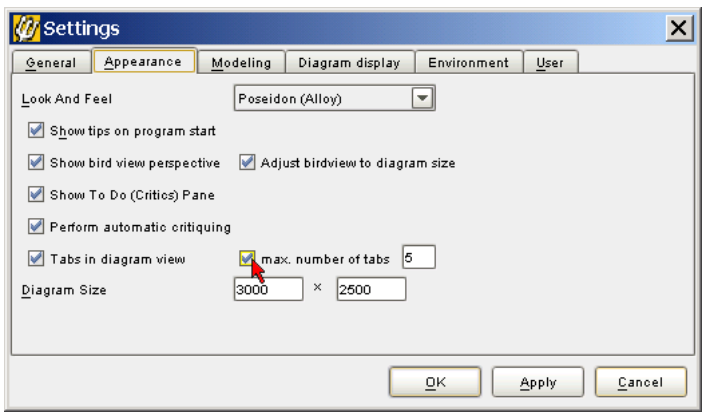
10.2.1. Open Diagrams

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.

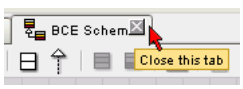


The number of diagrams which can be open at one time is set to 5 by default. This number can be changed in the Appearance tab of the Settings dialog. Unchecking the ’max number of tabs’ box removes any limits to the number of tabs.



10.2.2. 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.



10.2.3. Create Diagrams

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.

An alternative to the create buttons on the main toolbar is the create diagram menu from the main menu. All of the items available in this menu also have quick-keys assigned to them.

10.2.4. Edit Diagrams

Of course, central to any model is the collection of diagrams. They provide a means to communicate ideas to the viewer in a format which is easily comprehensible. And as they are responsible for clearly relating important aspects of the system, they must also be completely accurate. Poseidon makes it easy to modify the diagrams as the model progresses in development.

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.

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.

10.2.5. Change properties of the Diagram Pane

10.2.5.1. Grid Settings

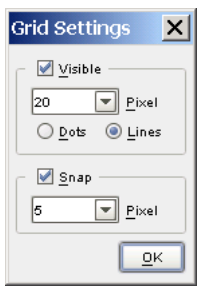
The first thing you may notice about the Diagram Pane is the grid that is drawn over the drawing area. By default, the drawing area displays this grid. The visible grid is only a collection of lines, they have no functions of their own.

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 View—Adjust Grid...

Figure 10-1. 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.

10.2.5.2. Other Settings

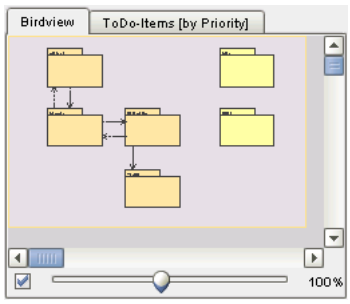
The grid is not the only setting that can be changed for the Diagram pane.

- **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.

10.3. Overview Pane

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

10.3.1. 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.

10.3.1.1. 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.

10.3.1.2. Zoom in 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.

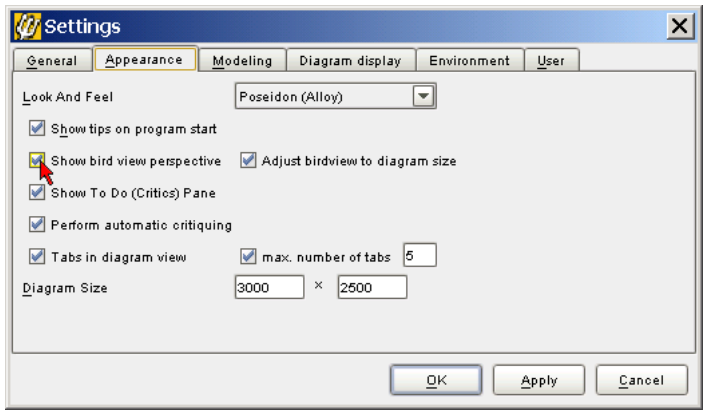
To change the zoom factor of the diagram in the Diagram pane:

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.

10.3.1.3. 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.



10.3.2. Critique tab

The second tab in the Overview pane, called To-Do-Items, is a collection of critiques. This is a feature that originates from ArgoUML and was one of the motivations for Jason Robbins to start the project. It is a powerful auditing mechanism that discretely generates critiques about the model you are building. Critiques can be hints to improve your model, reminders that your model is incomplete in some areas, or errors that would cause generated code not to compile.

10.3.2.1. Open a Critique

Critiques are arranged within this tab in a variety of ways. The following options are available for viewing critiques:

- by Decision Type
- by Knowledge Type
- by Offenders
- by Posters
- by Priority

To view the details of a critique, simply click on the critique. The details will be displayed in the ’To Do Items’ tab of the Details pane, located to the right of the Overview pane.

10.3.2.2. Navigate to critiqued area

You can move directly from a critique to the diagram where the issue occurs by double-clicking on the critique name in the Overview pane. The appropriate diagram will then be opened in the Diagram pane.

10.3.2.3. Snooze Critique

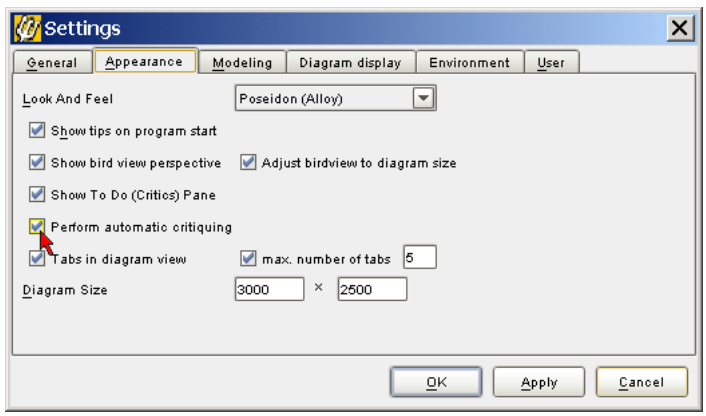
The Snooze critique button temporarily turns off a single critique. The critique will return without a user specifically re-enabling it.

10.3.2.4. Toggle Critique

The Toggle critique button allows you to turn off and on single critiques. This feature is available in the Standard and Professional editions of Poseidon.

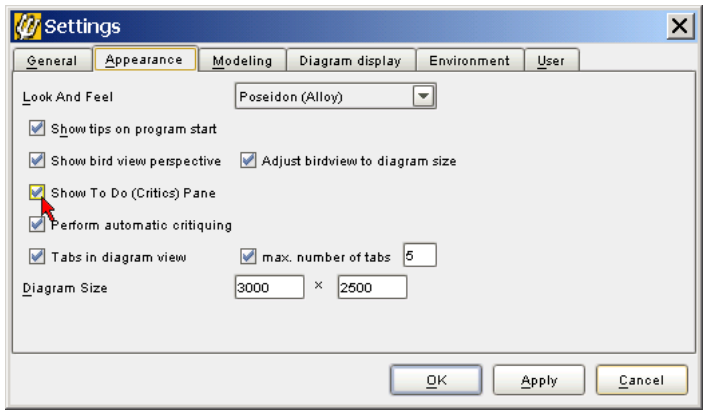
10.3.2.5. Turn off Autocritique

Critiquing can be turned off completely from the Appearance tab of the Settings dialog. Uncheck the box titled, 'Perform Automatic Critiquing' to disable critiques. The tab will still be visible in the Overview pane, but no critiques will be listed.



10.3.2.6. Hide/display Critique window

The Critique tab can be hidden regardless of whether critiquing has been enabled or disabled. To hide the tab, uncheck the box titled, 'Show To Do (Critics) Pane'.



10.4. 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 six tabs:

- Properties
- Style
- To Do Items
- Java Source
- Documentation
- Tagged Values

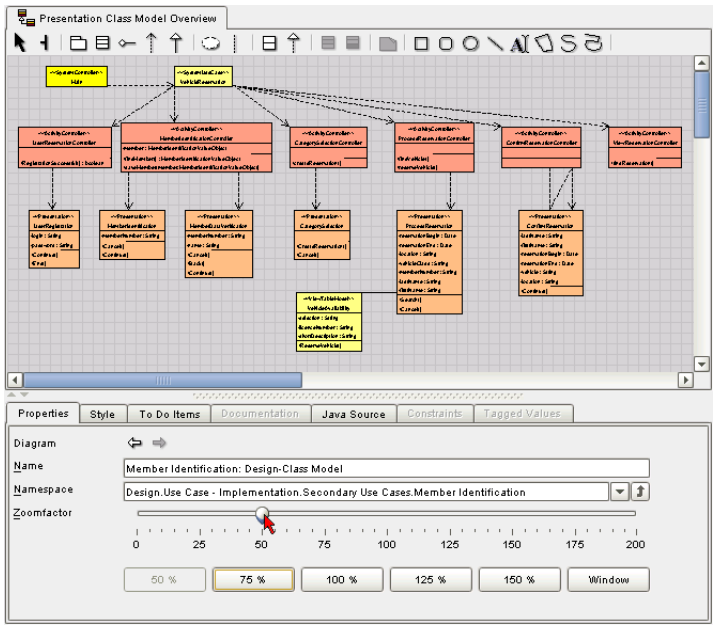
The following sections investigate these tabs in greater detail.

10.4.1. 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.

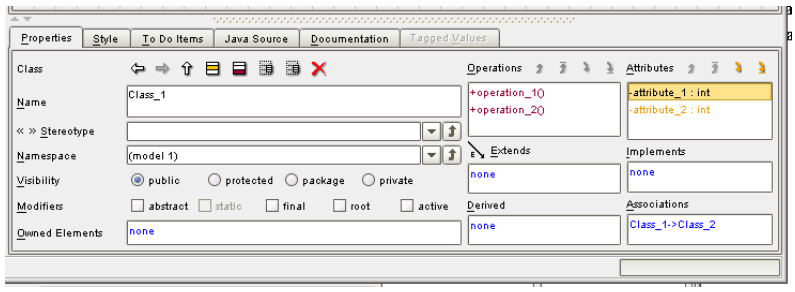
An important property, the zoom factor, becomes visible in the Details pane when a diagram name is selected in the Navigation pane. You can use the slider to change the zoom factor interactively or use the buttons to set it to pre-selected zoom factors (The range of zoom factors is limited in the Community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

Figure 10-2. 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 left. On the right, related model elements are displayed. By 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.

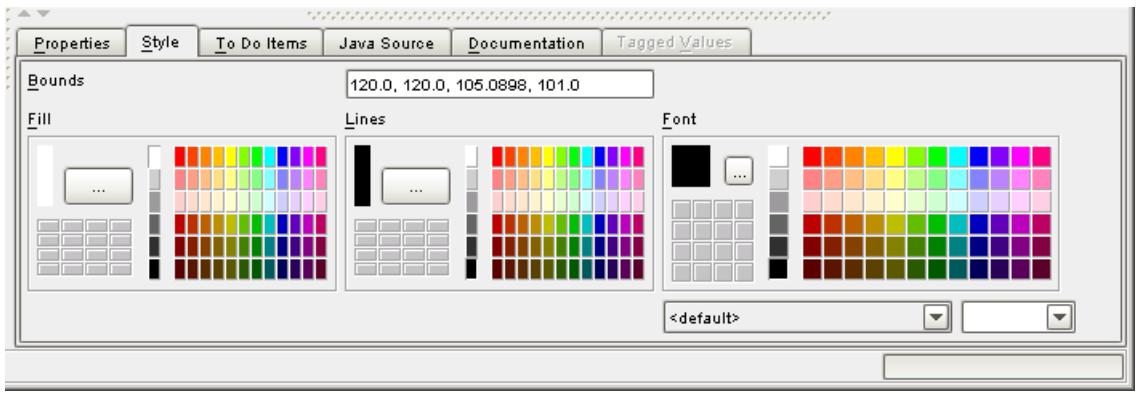
Figure 10-3. Drill-down Navigation



10.4.2. 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 10-4. Style tab for a class element



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 to change their color to green. You can also change the line color.

In addition, you can specify whether you want attribute or operation compartments to be shown or hidden.

10.4.3. To Do Items Tab

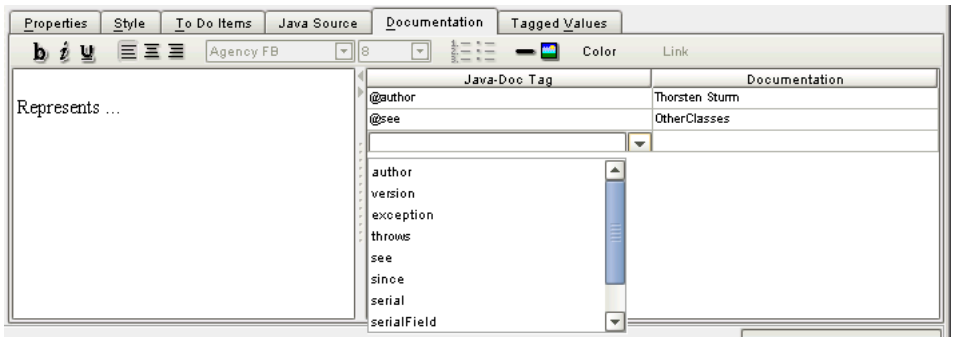
- Displays the critique selected in the Overview pane
- Sets the activity of the critique via the Toggle and Snooze buttons

The To Do Items tab is not functional in this Beta version.

10.4.4. 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 10-5. Documentation Tab for a class

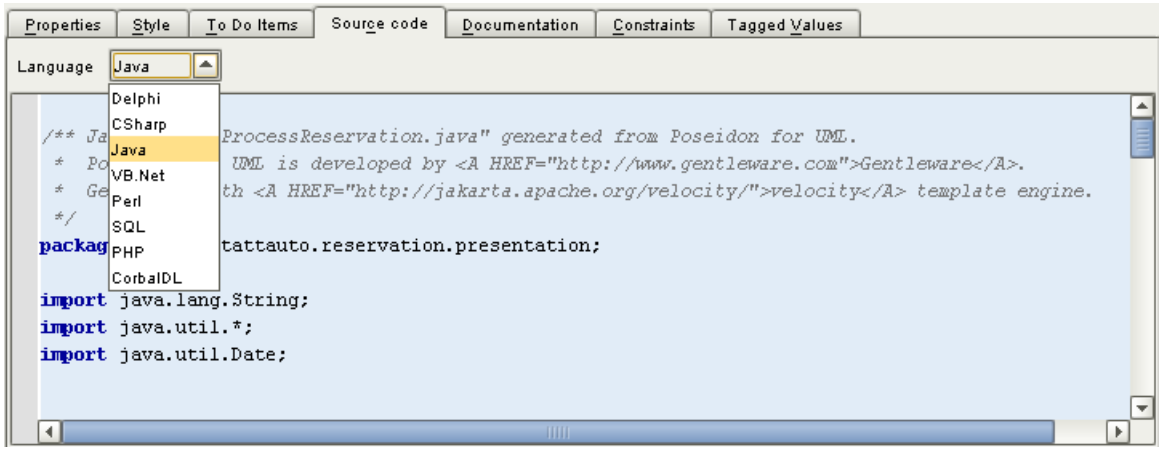


The Documentation tab provides a mechanism for adding your own freeform text as well as supported JavaDoc tags to the generated code. This information is stored as tagged values and can be previewed in the Java Source tab. Any entries made via the editor on the left side of the tab are placed in paragraph tags by default and are displayed before the JavaDoc entries.

10.4.5. Source Code Tab

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

Figure 10-6. 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 plugins and profiles. Each language must have both the plugin and profile specific to that language enabled.

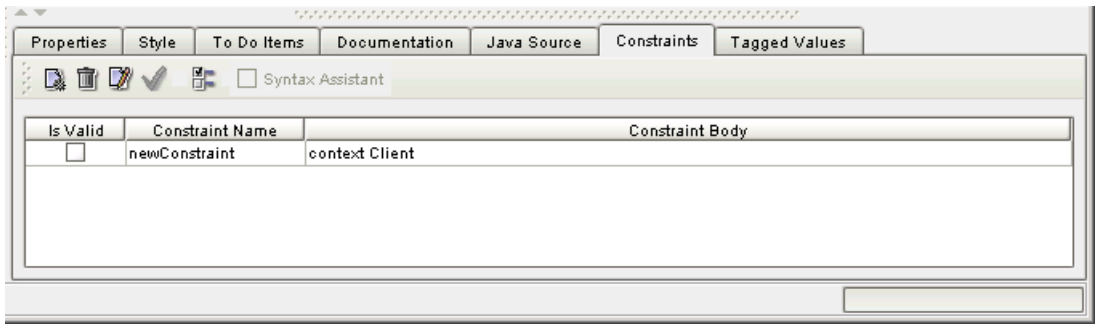
The same diagrams may be used to generate code in different languages. Any code written in the ’your code here’ sections is available only in the language selection in which it was written. For example, any code manually entered into the editable section of this tab while Java is the selected language 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.

10.4.6. Constraints Tab

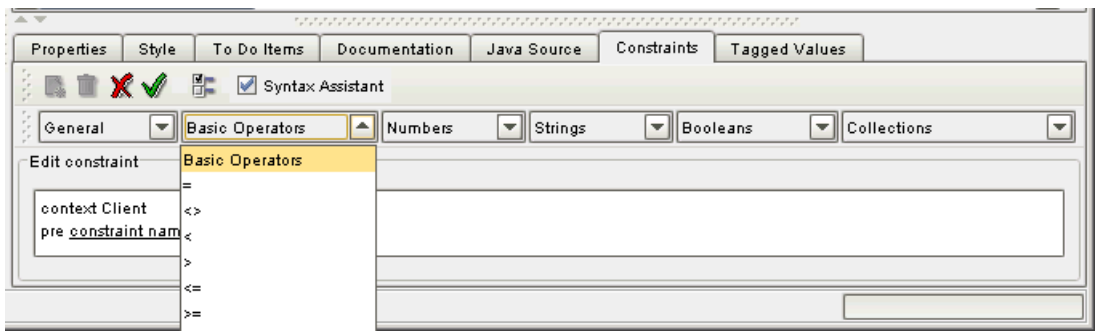
- Holds the OCL constraints for the given element

Figure 10-7. New constraint in the Constraints Tab



Since many people find OCL difficult to learn we have provided an assistant that helps you with the syntax of the OCL language. The syntax of the constraint and its consistency with the model can be checked for correctness.

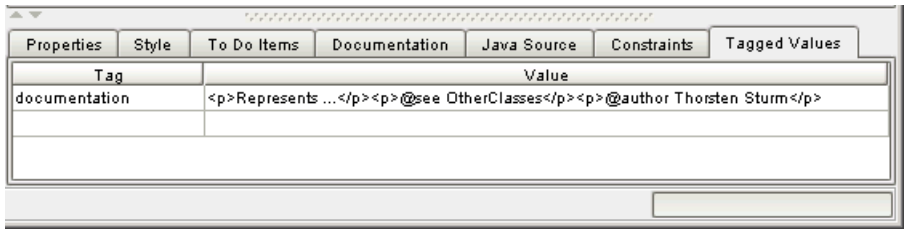
Figure 10-8. Syntax Assistant in the Constraints Tab



10.4.7. 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 10-9. Documentation stored in the Tagged Values Tab



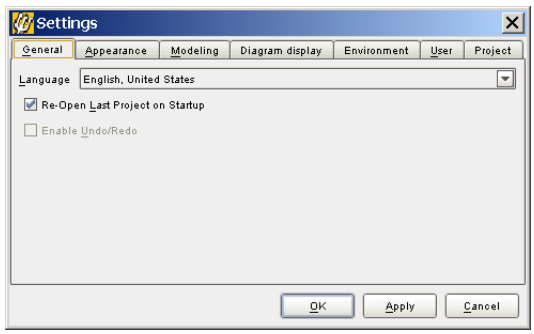
For example, if you need special information for external processing of the model you can add this information here. This is also where Poseidon stores any documentation entered in the Documentation tab.

Chapter 11. 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.

11.1. General

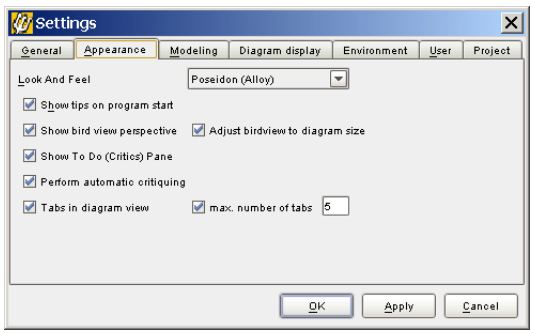
Figure 11-1. The General settings tab.



- **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.

11.2. Appearance

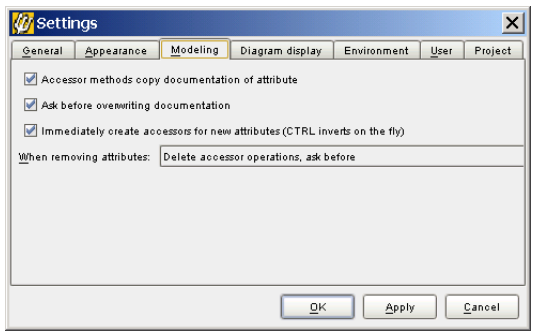
Figure 11-2. The Appearance settings tab.



- **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.
- **Show tips on program start** — If checked, a `Tip of the Day` dialog appears when you start up the program.
- **Show bird view 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.
- **Show To Do (Critics) pane** — If checked, the Overview pane displays the ToDo/Critics tab.
- **Perform automatic critiquing** — If checked, suggestions and possible conflicts or flaws will be automatically logged in the ToDo/Critics tab.
- **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.

11.3. Modeling

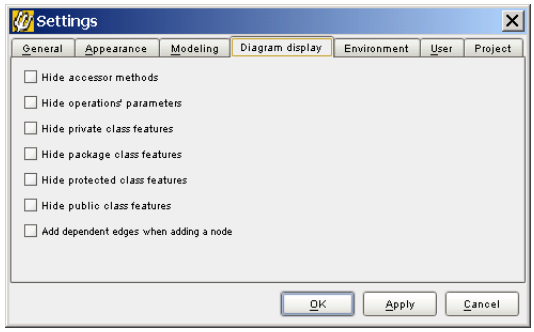
Figure 11-3. The Modeling settings tab.



- **Accessor methods copy documentation of attribute** — If checked, the documentation of the attribute is passed to its accessor methods.
- **Ask before overwriting documentation** — If unchecked, Poseidon will prompt before overwriting existing documentation of an attribute with the documentation of its accessor methods.
- **Immediately create accessors for new 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.

11.4. Diagram Display

Figure 11-4. The Diagram display settings tab.

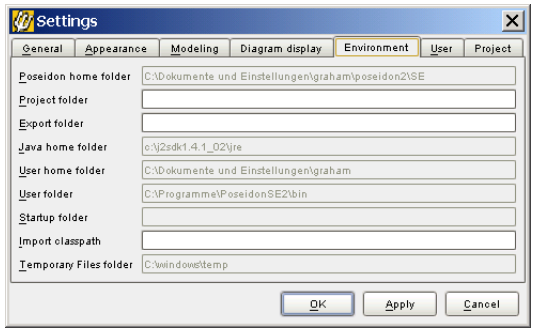


The Diagram Display tab contains properties regarding the display of information within the diagrams. Currently, most of the properties refer to Class Diagrams only.

- **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.

11.5. Environment

Figure 11-5. The Environment settings tab.

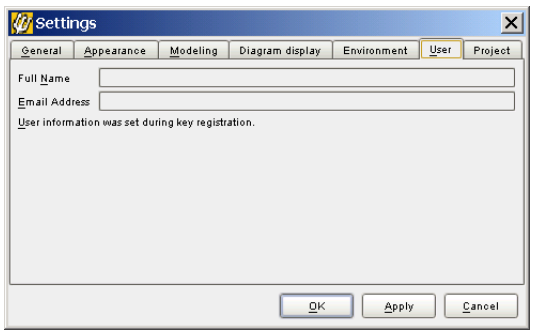


The Environment tab contains properties regarding the local environment and the directories used for loading and saving files.

- **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.
- **Import Classpath** — Additional classpath that is used when importing source code or jar files.
- **Temporary Files folder** — The folder used when any temporary files are created.

11.6. User

Figure 11-6. The User settings tab.

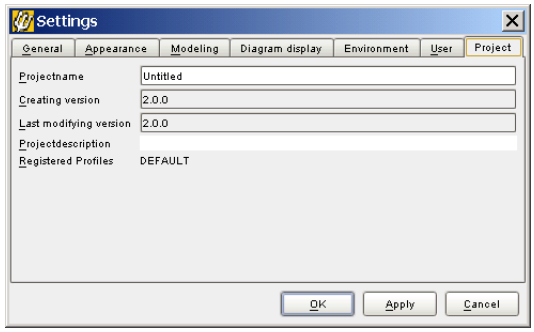


The User tab contains properties regarding information about the user. These properties cannot be changed from the settings dialog, because they are part of the product registration. They can be changed using the license manager, but any change would require a new registration of the product.

- **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.

11.7. Project

Figure 11-7. The Project settings tab.

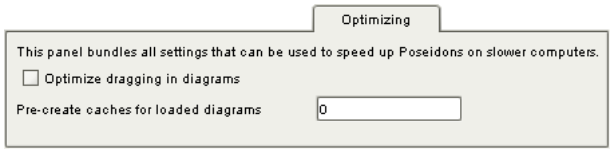


The Project tab contains properties regarding information about the project

- **Projectname** — Name under which the current project has been saved. If the project has not yet been saved, this will be ’Untitled’.
- **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.
- **Projectdescription** — A short, user-defined description of the current project.
- **Registered Profiles** — The Profiles registered for this project.

11.8. Optimizing

Figure 11-8. The Optimizing tab.



The Optimizing tab contains options to increase the speed of Poseidon.

- **Optimize dragging in diagrams —**
- **Pre-create caches for loaded diagrams —**

Chapter 12. Code Generation and Round-trip Engineering

12.1. Generating Code

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 Java and HTML code, but it is flexible enough to generate any kind of programming language, or other output, such as C++ or XML.

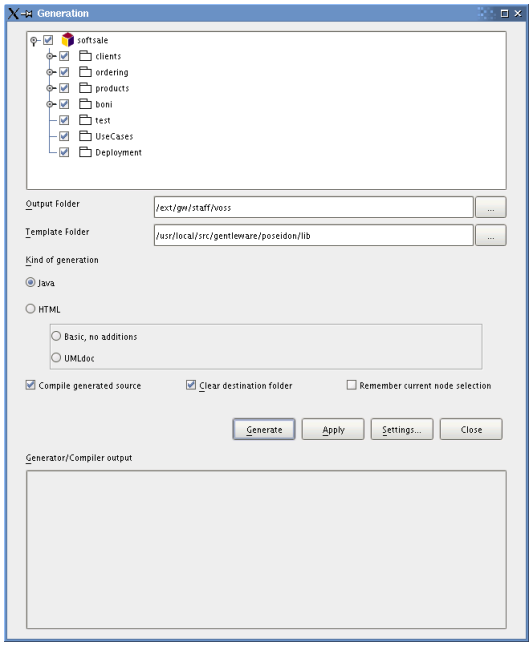
Java code generation is usually based on the classes of a model and other information displayed in the respective Class Diagrams. Additionally, Poseidon can generate setter and getter methods for the fields of each class.

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.

For generating the corresponding HTML documentation for your model you need the UMLdoc feature, which is available in every edition *except* the Community Edition. The look and feel of the generated documentation is very similar to Javadoc. Poseidon for UML allows you to specify Javadoc information directly in your model (in the Documentation tab). This information, like comments to your classes or methods, is included in the code. But Javadoc alone gives a view on the code only, not on the model. For example, you do not see your diagrams. With this feature you get the same information as with Javadoc, in addition to all diagrams from your model. This includes Class Diagrams, your use case diagrams, sequence diagrams etc. This is valuable information that you would want in your documentation.

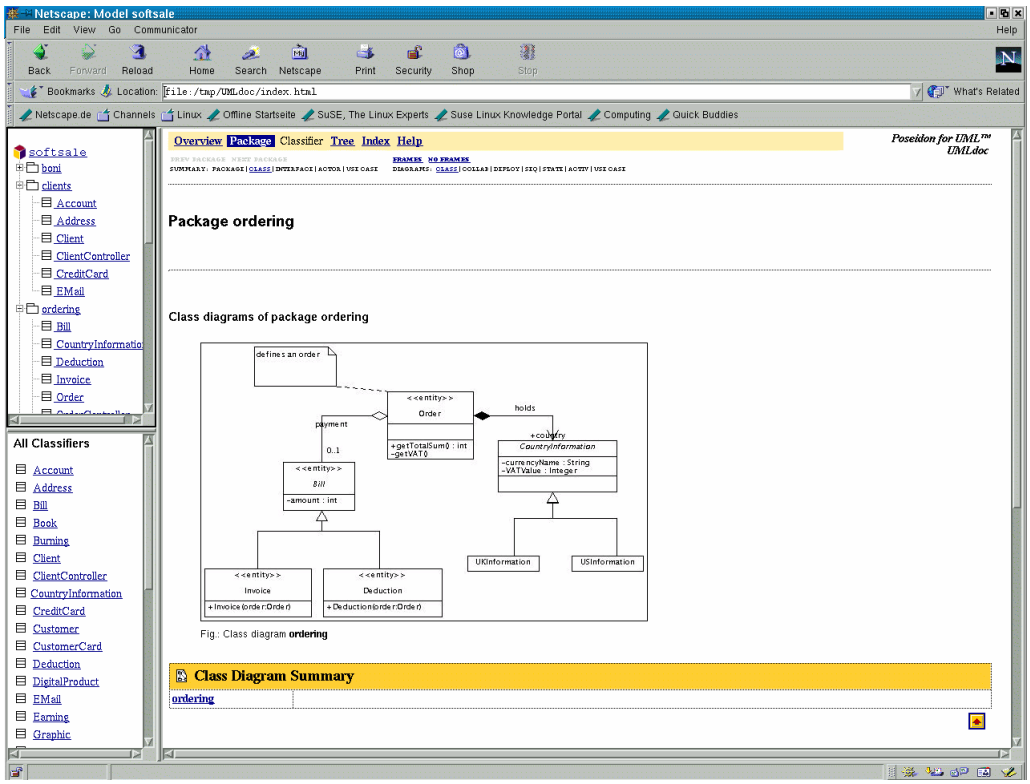
Both kinds of generation can be invoked from the generation menu. Select `Generate Classes of Model` and a dialog will appear. Here you can select or deselect model elements from the tree, specify an output and a template folder, indicate if the destination folder should be cleared and select either `Java` for code generation or `HTML` and `UMLdoc` for the documentation generation.

Figure 12-1. Code Generation Dialog — Java.



After the generation is finished, you will get a corresponding message in the `Generator/Compiler output`. Now you can open the HTML documentation in your favorite browser. UMLdoc generates an HTML page for the model overview, each package and each classifier (actors, use cases, classes, interfaces). They are connected by hyperlinks, so that you can easily navigate through the whole document.

Figure 12-2. Generated UMLdoc opened in Netscape.



You'll find the generated java files in the specified output folder, sorted by packages.

12.2. 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`.

  Future versions of Poseidon, you will be 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`.

- **Modifying templates** (Developer and Professional Editions)

  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/support/developer (http://www.gentleware.com/support/developer/index.php3).

- **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.

12.3. Reverse-Engineering Code

Software engineers often run into the problem of having to re-engineer an existing project for which only code but 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

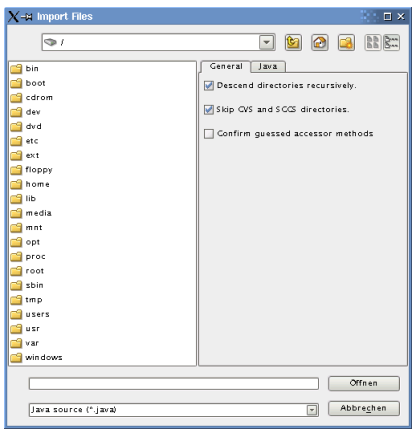
generation dialog. The next time you open the code generation dialog, this path will be displayed as output folder by default.

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, 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
```

Figure 12-3. Import Files Dialog.



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.

12.4. Round-Trip 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 Edition, which contains the RoundTrip UML/Java Plug-in. It is one of the most recommended and highly sophisticated features provided by Poseidon for UML.

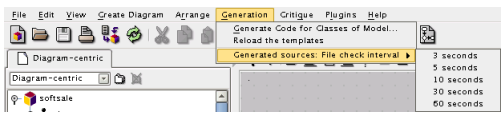
Generate a UML model from your existing code, change the model, re-generate the code, change the code and so on. All generated Java code files are watched, so that changes you have made with an external editor are imported into Poseidon's model of your project. Use your favorite source code editor to 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 will be in one of the next minor releases.

How to use this feature:

- Create or load a model in Poseidon
- Set the interval after which Poseidon checks for file modifications

Figure 12-4. Select File Check Interval.



- Generate the code (the generation window will automatically pop up if you load a model)
- Use your preferred editor to modify the code (especially the method bodies), modify identifier names, add or remove methods and/or variables
- Save the file in your editor

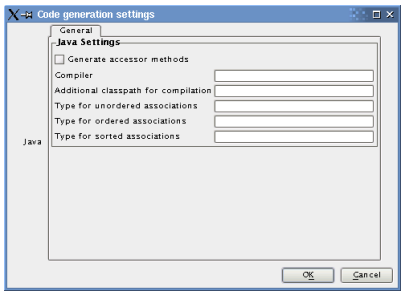
And the changes will appear in Poseidon!

Some words on how to handle 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.

Figure 12-5. Java Code Generation — Settings.



In the code generation settings dialog, you have the ability to specify an additional classpath for compilation.

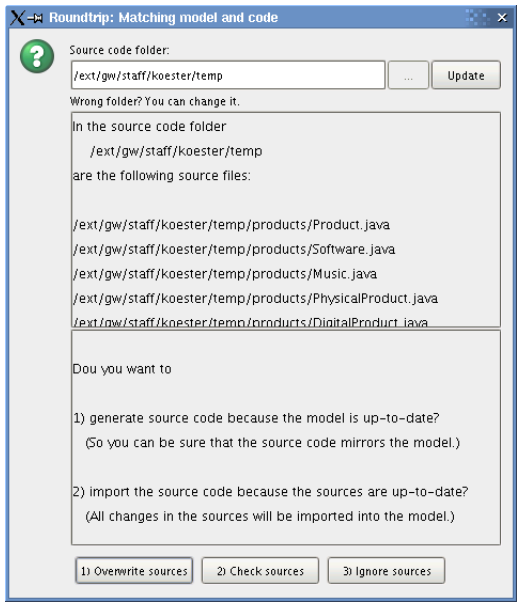
You might temporarily have non-compiling source code that you do not want to import into Poseidon right away. For these instances, you can temporarily disable the automatic import with the button next to the `Import sources` button. It will turn to red, showing that automatic round-trip is disabled.

By clicking it again, it will turn back to green, designating that round-trip is enabled again.

When the round-trip plug-in is running and you have imported files, Poseidon asks if you would like to keep these source files and the model in sync.

When you load a new model, Poseidon asks you if you want to generate the source code now or if you want to import existing source code. Choose the first alternative if the you want to ensure that the source code you have reflects the current model. Using the latter choice (import), you can synchronize the code and model even if

you have changed the source code while Poseidon was not running and thus could not keep track online of the changes you did to the source.

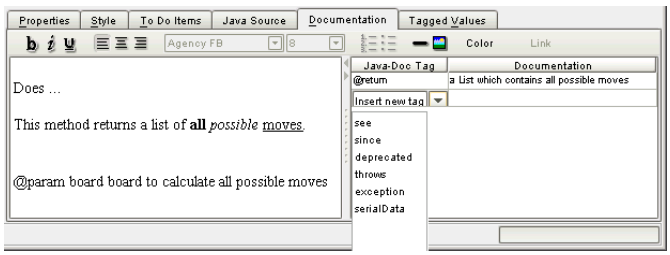


Note that a folder for the source code is always set in Poseidon. This may not be the one that is suitable for your new project. To change it, select ’Update’ and view the contents of another folder. This way, you can make sure that after opening a project, you can either update the project with the correct source code (and select option 2), or generate fresh code if the project is the latest version (select option 1).

Chapter 13. Documentation Generation (UMLdoc)

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 13-1. Editing a method documentation.



13.1. UMLdoc

The UMLdoc Plug-in generates HTML documentation files, that look similar to Javadoc. But it includes your UML diagrams as jpeg images, and offers an improved navigation. Currently UMLdoc generates documentation for models, packages, classes, interfaces, operations, methods, associations, actors, use cases, extend and include relationships.

UMLdoc is also capable of generating 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.

13.2. Code generation settings

The code generation settings dialog of UMLdoc provides the following settings:

Generate author docs

If you disable this setting, @author tags are skipped in the output.

Generate class doc for

Here you can select for which classes documentation should be generated. You can enable/disable the documentation output of public, protected and private classes.

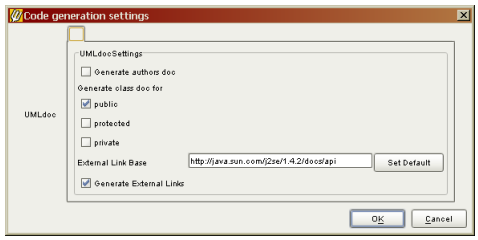
External Link Base

The site noted here will be used as the base link for all external links within the document. The default points to Sun’s Java site, and this site can be restored after modifications by clicking ’Set Default’.

Generate External Links

With this option enabled, @link destinations that are external will be activated within the document.

Figure 13-2. UMLdoc Code Generation — Settings.



13.3. 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.

{@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.

@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 14. Advanced Features

14.1. 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.

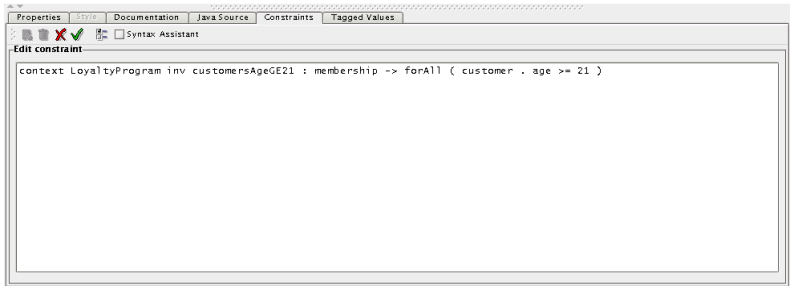
The current UML recognizes this and comes with a supplementary textual language to express constraints. This language is called the Object Constraint Language, or abbreviated OCL.

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 14-1. A Constraints tab.

But also because it is text, it is quite difficult to tell — just by looking at it — whether syntax and semantics are used correctly. Poseidon does help you there extensively, and to our knowledge is best at doing this. The Constraints tab on the Details pane holds an OCL editor that comes with its own syntax assistance mode, which you have to enable first.

Figure 14-2. Edit Constraints.

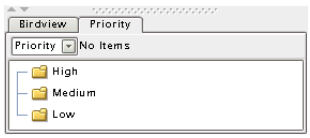


14.2. Critiques

Before we start generating code, lets first check if our model is as well-formed as it should be. There are certain design rules for software that are generally acknowledged by developers. The implementation of these kinds of rules into Poseidon for UML is in fact one of its finest features. This feature of cognitive support, which acts like a built in auditor, is called ’critique’.

When activated in the critiques menu, the critiques are constantly analyzing and criticizing your design. The Critiques pane in the bottom left corner of the working area shows three priority nodes.

Figure 14-3. Critiques Pane.

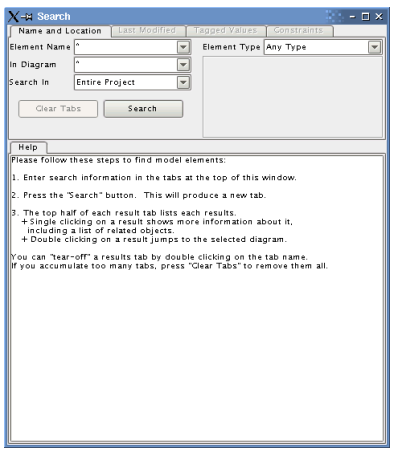


Broadening and improving this feature is part of each development cycle of Poseidon.

14.3. Searching for Model Elements

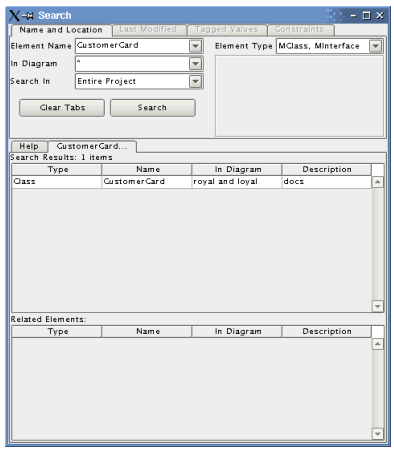
When your models start to grow, you will want a nice mechanism to search for elements. Poseidon offers a powerful search tool that is not just based on text but on model information. It allows you to look for specific types of elements. The search tool is invoked from the Edit menu, by selecting Find... or by directly pressing key F3.

Figure 14-4. Find Dialog



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 14-5. Searching 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.

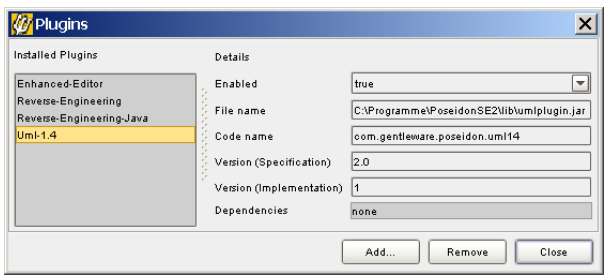
Chapter 15. Plug-ins and Profiles

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 (http://www.gentleware.com/Partner/index.php3) 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/products/documentation/) available on the Gentleware Web site.

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 Uml-1.4. It contains all of the information used by Poseidon to render the diagrams. This includes items such as the appearance of the elements and relationship definitions.



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.

15.2. Plug In Guides

The Professional Edition of Poseidon comes with four options for code generation and one for documentation generation (UMLDoc). Java code generation is the default setting, but you can also choose to generate CORBA IDL, Visual Basic .NET or C# code. To do this, you have to activate the plugin that supports the desired language. (Via Plugins -> Plugins 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.

15.2.1. Poseidon C# Code Generation Plugin Guide

15.2.1.1. General Rules

15.2.1.1.1. 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

15.2.1.1.2. Additional Stereotypes

- *<<event>>*
- *<<readonly>>*
- *<<delegate>>*

15.2.1.2. Modelling Element Rules

15.2.1.2.1. Classes

- Uses the standard UML ’Class’
- Supports single inheritance only

Class Signature

- Additional visibilities for class signatures are set when the tagged values below are ’true’:

  1. internal
  2. sealed

Class Attributes

- Additional visibilities for class attributes are set when the tagged values below are ’true’:

  1. internal
  2. protected internal
  3. volatile

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

15.2.1.2.2. Interface

- Uses the standard UML ’Interface’
- Supports single inheritance only

Interface Signature

- Additional visibilities for interface signatures are set when the tagged value below is ’true’:

1. internal

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.

15.2.1.2.3. Structure

- Uses the standard UML ’Class’ with the <<*struct*>> stereotype
- Supports single inheritance only

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. The ’abstract’ and ’sealed’ modifiers are therefore 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’.

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.

15.2.1.2.4. 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’.

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.

15.2.1.2.5. Delegate

- Uses the standard UML ’Class’ with a <<*delegate*>> stereotype
- Delegate does not participate in generalizations or specifications

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.

15.2.1.2.6. C# Event

C# events are supported with an operation that has the stereotype <<*event*>>.

15.2.1.2.7. 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’

15.2.2. Poseidon CORBA IDL Code Generation Plugin Guide

15.2.2.1. 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

15.2.2.2. CORBA Interface

- Uses the standard UML ’Class’ with the <<*CORBAInterface*>> stereotype
- Interface member has to be ’public’

15.2.2.3. 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

15.2.2.4. 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

15.2.2.5. 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

15.2.2.6. 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

15.2.2.7. 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 §3.5.15 for more details.

15.2.3. Poseidon VB.Net Code Generation Plugin Guide

15.2.3.1. 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

15.2.3.2. 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’

15.2.3.3. 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

15.2.3.4. 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

15.2.3.5. 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

15.2.3.6. 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’

15.2.3.7. 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’

15.2.3.8. 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’

15.2.3.9. 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’.

15.2.3.10. Visual Basic Events

- VB Events are supported by using <<*Event*>> stereotypes with operations

15.2.3.11. Attribute & Association Ends

- Supports the tagged value ’Protected Friend’

15.2.4. Poseidon PHP4 Code Generation Plugin Guide

This guide is based on the PHP4 Manual, available at http://www.php.net/docs.php.

15.2.4.1. 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

15.2.4.1.1. 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

15.2.4.2. PHP4 Class Modelling Rules

- Uses standard UML 'Class'
- Supports single inheritance only

15.2.4.2.1. Class Signature

- There are no visibilities for Class Signature

15.2.4.2.2. 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;
```

15.2.4.2.3. 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(); ?>
```

15.2.5. Poseidon Delphi Code Generation Plugin Guide

15.2.5.1. Classfiers

- Class
- Interface
- Enumeration
- Record
- Set
- Sub Range
- Array
- Exception

15.2.5.2. Tagged Values

All strings input in the Tag column are case-sensitive.

- **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.

- **Attribute**
  - Tag = ’published’ with Value = ’true’.

    Description: Handles the published visibility of the classifier ’attribute’.

- **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 = ’overload’ with Value = ’true’

  Description: Handles the way to set the specified operation into a ’overload’type operation.

- **Exception**
  - Tag = ’published’ with Value = ’true’.

    Description: Handles the published visibility of ’Exception’.

15.2.5.3. Stereotypes

- **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.

- **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.

- **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.

15.2.5.4. Modelling Element Rules

- **Class**
  - Uses the standard UML Class
  - Participates in generalizations, associations and specifications
  - Only supports single inheritance

- **Interface**
  - Uses the standard UML Interface
  - Participates in generalizations
  - Does not participate in associations or specifications
  - Only supports single inheritance

- **Enumeration**
  - Uses the standard UML Class with << *Enum* >> stereotype
  - Does not participate in generalizations or specifications
  - Cannot have navigable opposite association ends or operations

- **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

- **Set**
  - Uses the standard UML Class with << *Set* >> stereotype
  - Does not participate in generalizations or specifications
  - Cannot have navigable opposite association ends or operations

- **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

- **Array**
  - Uses the standard UML Class with << *Array* >> stereotype
  - Does not participate in generalizations or specifications
  - Cannot have navigable opposite association ends or operations

- **Exception**
  - Uses the standard UML Class with << *Exception* >> stereotype
  - The same as Class

15.2.5.5. 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

15.2.6. Poseidon Perl Code Generation Guide

15.2.6.1. 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

15.2.6.2. Classes

- Uses the standard UML ‘Class’
- Classes are translated into Perl Class (package className)
- A constructor is generated for each class (sub new)

15.2.6.3. 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.

15.2.6.4. 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.

15.2.6.5. 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)

15.2.6.6. Aggregation

- 1 to 1 aggregations are translated into scalar type variables (my $className)

15.2.6.7. Inheritance

- Single inheritence is implemented using @ISA = qw(class)
- Multiple inheritence is implemented using @ISA = qw(class1 class2 ...)

15.2.7. Poseidon SQL DDL Code Generation Plugin Guide

15.2.7.1. Modelling Element Rules

15.2.7.1.1. Classes

- Uses the standard UML ‘Class’

- Each class is considered as a table.

15.2.7.1.2. Attributes

- Describes the columns in table. Each attribute can have stereotypes that will be treated as column constraints.

15.2.7.1.3. Association Ends

- Describes the relationships between tables. Foreign keys will be automatically generated in tables that have references to other tables.

15.2.7.2. 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

15.2.7.3. 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

15.3. Available Plug Ins

15.3.1. 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.

15.3.2. 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. This is especially interesting with tight integration in an IDE like Forte. Modeling and coding are not separated anymore.

15.3.3. Statechart-to-Java

With the Statechart-to-Java Plug-in you can directly generate java code from a UML state diagram that describes the behavior of a class. So in addition to the static structure of the class — i.e. operations, attributes and associations — the dynamic behavior is generated as java code.This unique module enables you to visualize, test and even manipulate the behavior of objects using their corresponding state diagrams. The visualization is based on pure UML, just adding some color and action.

The triggers in the state chart are mapped to ordinary operations of a class. The behavior of these operations is determined by the actions that are specified in entry and exit actions of states and as effects of transitions in state charts.

In order to validate the semantics of the specified behavior, the Statechart-to-Java Plug-in allows you to run a simulation in which state diagrams can be instantiated. The generated java code of a class assures that by creating an object that owns a state chart, its state chart diagram is simulated graphically. You can send triggers to these objects and step through the various states that are defined in the state chart diagram. By dealing with several objects, you can validate not only the intra-object behavior of one object, but also the inter-object behavior.

15.3.4. OCL Code Generation

OCL code generation can immensely enhance your productivity using Poseidon. This module is particularly beneficial to development teams who make excessive use of UML and OCL. It also works very well for small and medium projects, if the developers already “speak” OCL fluently.

15.3.5. Refactoring Browser

The refactoring browser module is the latest extension of the cognitive support for Poseidon. It provides a very handy set of functions to change the structure of your design for the better, without changing the functional outcome. The refactoring is actively assisted according to acknowledged rules, so that with bigger projects you still don’t run the risk of side effects that ruin hitherto working models.

To put it in a nutshell, refactoring your program means cleaning up your program’s internal structure without implementing new features or introducing side effects. The term "refactoring" was coined by the famous thesis “Refactoring Object-Oriented Frameworks" (ftp://st.cs.uiuc.edu/pub/papers/refactoring/) by William Opdyke in 1992. Nowadays, refactoring is an important practice within eXtreme Programming (XP) (http://www.extremeprogramming.org/). In contrast to the popular saying "Never change a running system", XP advises developers to routinely refactor their programs in order to prevent them from deteriorating. Further information about refactoring in general can be found on Ward Cunningham’s extraordinary Wiki-Web (http://c2.com/cgi/wiki?WikiPagesAboutRefactoring).

To refactor a program, you don’t need a tool as everything may be done manually. But a dedicated tool can save you a lot of time (and trouble) by automating much of the work and relieving you of tedious routine checks. The aim of the "Refactoring Browser for Poseidon" is to aid developers in refactoring not "just" code but also UML models. Currently 13 refactorings for class, state, and activity diagrams are supported — with far more to come.

Using the Refactoring Browser is easy. Every time you select a model’s element, the browser checks its list of refactorings. If a refactoring is applicable for the current selection, you may select and customize it. Before performing the refactoring, the browser issues warnings if the refactoring is likely to alter your model’s behavior. Error messages are generated if a modification will result in a defective model. You perform the refactoring with a final click of the mouse.

15.3.6. MDL Import

The MDL Import Plug-in enables Poseidon to import UML models created by Rational Rose.

15.3.6.1. Installing and Using

After the plug-in has been installed, the Import Files dialog (accessible from the File menu or by clicking the icon in the toolbar) 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.

15.3.6.2. 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.

15.3.6.3. Unsupported Features

Some elements are changed during the import, others are ignored completely. Here is a list of known shortcomings:

- Poseidon currently supports notes for classes, interfaces, packages, use cases, actors and states, but not for transitions, associations or objects. If a note 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.
- Association Class: Association classes are imported, but not displayed correctly: They appear as simple associations. If you click on an association class, the Details pane displays the properties correctly, though.
- 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.

15.3.6.4. 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.

15.3.6.5. 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 at <support@gentleware.com>

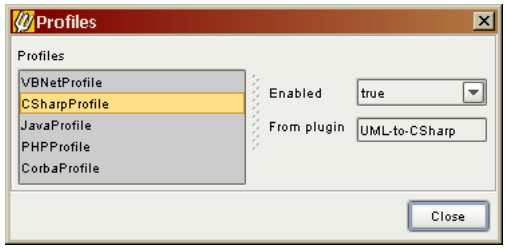
15.4. Profile Manager

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 plugins, and the profiles that automatically appear in the Profile Manager directly correspond to the set of enabled language-specific plugins and are enabled by default. Likewise, if a plugin is disabled from the Plugin 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 plugin 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 plugin. If you enable the plugin again, the profile will be automatically enabled. The status of the profile is not saved when the plugin is disabled.

Figure 15-1. The Profile Manager



Chapter 16. More on 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#.

16.1. The 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/.

16.1.1. 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.

16.1.2. 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/).

16.1.3. 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.

16.1.4. Examples

Example 16-1. 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 16-2. 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()
}
```

16.2. 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.

16.3. The Code Generation API

For a detailed description of the code generation API, please refer to the online API documentation (http://www.gentleware.com/support/developer/codegen-api/) (Javadoc) and the separate document describing the code generation framework (http://www.gentleware.com/products/documentation/PoseidonCodeGenFramework.html). These files are part of the Developer and Professional distributions in the docs folder.

The Professional Edition also includes two demo plug-ins 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 info@gentleware.com to receive a key for your plug-in.

Chapter 17. 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).