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

1. About Gentleware and Poseidon for UML 1
1.1. About Gentleware and Poseidon for UML 1
1.1.1. Our Vision 1
1.1.2. Innovation 1
1.1.3. Cooperation 2
1.1.4. Contact 2
1.2. New Features in Version 2.x 3
1.3. About This Document 4
2. Editions 7
2.1. Community Edition 7
2.2. Standard Edition 8
2.3. Professional Edition 9
2.4. Enterprise Edition 10
2.5. Embedded Edition 11
2.6. Edition Comparison 12
3. Installation and First Start 15
3.1. Prerequisites 15
3.1.1. Additional Requirements for MacOS X Users 15
3.1.1.1. Jaguar (MacOS X 10.2.x) 15
3.1.1.2. Panther (10.3.x) 15
3.2. Community, Standard, Professional, and Embedded Editions 16
3.2.1. Install Using InstallAnywhere 16
3.2.1.1. Windows 16
3.2.1.2. *NIX 17
3.2.2. Install Through Java Web Start (Community Edition Only) 17
3.2.3. Install from a ZIP File 18
3.2.4. Silent Installation 19
3.2.4.1. Installer Properties File 19
3.2.4.2. Command Line Parameter 19
3.2.4.3. Uninstallation 19
3.2.5. Uninstallation 20
3.3. Enterprise Edition 21
3.4. Environment Variables 21
3.5. Keys and Registration 22
3.5.1. Types and Terminology 22
3.5.2. Community Edition 23
3.5.3. Evaluation Copy 24
3.5.4. Premium Version Purchase 24
3.5.5. Keys for Plug-Ins 25

4. A Short Tour of Poseidon for UML 27
4.1. Opening the Default Example 27
4.2. Introducing the Work Area 27
4.2.1. The Navigation Pane 29
4.2.1.1. Changing the Navigation View 31
4.2.1.2. Opening Multiple Navigation Panes 32
4.2.2. The Diagram Pane 34
4.2.3. The Details Pane 35
4.2.4. The Overview Pane 36
4.3. Navigation 38
4.3.1. Navigating with the Navigation Pane 38
4.3.2. Navigating in the Properties Tab 39
4.4. Modify Elements 41
4.4.1. Change Element 41
4.4.2. Create Element 43
4.4.3. Delete Elements 44
5. Interface 47
5.1. Toolbar 47
5.2. Menus 47
5.2.1. File 48
5.2.2. Edit 50
5.2.3. View 52
5.2.4. Create Diagram 53
5.2.5. Align 54
5.2.6. Critique 56
5.2.7. Generation 56
5.2.8. Plug-Ins 57
5.2.9. Help 58
6. Panes 61
6.1. Navigation Pane 61
6.1.1. Add a Tab 62
6.1.2. Delete a Tab 62
6.1.3. Delete a Diagram 63
6.2. Diagram Pane 64
6.2.1. Diagram Pane Toolbar 64
6.2.2. Remove Tabs 64
6.2.3. Change Properties of the Diagram Pane 65
6.2.3.1. Grid Settings 65
6.2.3.2. Other Settings 66
6.3. Details Pane 66
6.3.1. Properties Tab 67
6.3.2. C++ Properties 69

6.3.3. Style Tab 69
6.3.4. To Do Items Tab 71
6.3.4.1. Snooze Critique 71
6.3.4.2. Toggle Critique 72
6.3.4.3. Turn Off Autocritique 72
6.3.5. Source Code Tab 72
6.3.6. Documentation Tab 73
6.3.6.1. Toolbar 74
6.3.6.2. Dropdowns 75
6.3.7. Constraints Tab 76
6.3.8. Tagged Values Tab 77
6.4. Overview Pane 77
6.4.1. Birdview Tab 77
6.4.1.1. Redisplay a Section of a Diagram 78
6.4.1.2. Zoom in Birdview Only 78
6.4.1.3. Zoom in a Diagram 78
6.4.1.4. Turn Off Birdview in Settings 79
6.4.2. Critique tab 79
6.4.2.1. Open a Critique 79
6.4.2.2. Navigate to a Critiqued Area 80
6.4.2.3. Turn Off Autocritique 80
6.4.2.4. Hide/Display Critique Window 81
7. Setting Properties 83
7.1. General 83
7.2. Appearance 84
7.3. Modeling 85
7.4. Environment 86
7.5. User 87
7.6. Project 88
7.7. Key Mappings 89
7.8. Diagram Display 89
8. Model Reference 91
8.1. Views 91
9. Using Models 93
9.1. Creating New Models 93
9.2. Saving and Loading Models 93
9.3. Importing Files 95
9.4. Importing Models 99
9.5. Exporting Models 100
9.6. Exporting Graphics and Printing 101

10. Diagram Reference 105
10.1. Use Case Diagrams 105
10.1.1. Diagram Elements 106
10.1.2. Toolbar 107
10.2. Class Diagram 107
10.2.1. Diagram Elements 108
10.2.2. Toolbar 109
10.3. Object Diagram 110
10.3.1. Diagram Elements 111
10.3.2. Toolbar 111
10.4. Activity Diagrams 112
10.4.1. Diagram Elements 113
10.4.2. Toolbar 114
10.5. State Diagrams 114
10.5.1. Diagram Elements 116
10.5.2. Toolbar 117
10.6. Sequence Diagrams 118
10.6.1. Diagram Elements 121
10.6.2. Toolbar 121
10.7. Collaboration Diagrams 122
10.7.1. Diagram Elements 122
10.7.2. Toolbar 122
10.8. Component Diagrams 123
10.8.1. Diagram Elements 124
10.8.2. Toolbar 124
10.9. Deployment Diagrams 125
10.9.1. Diagram Elements 125
10.9.2. Toolbar 126
11. Using Diagrams 129
11.1. Creating New Diagrams 129
11.2. Opening Diagrams 130
11.3. Viewing Diagrams 131
11.3.1. Details Pane 132
11.3.2. Zooming 134
11.3.3. Scrolling 136
11.3.4. Birdview Tab 136
11.4. Navigation 136
11.4.1. Navigation Pane 137
11.4.2. Details Pane 137
11.4.3. Diagram Pane 137
11.5. Editing Diagrams 138
11.5.1. Drag and Drop 139
11.5.2. Changing Namespaces 140

11.5.3. Layout Functions 141
11.5.4. Undo/Redo 145
11.5.5. Non-UML Additions 145
11.5.5.1. Select 145
11.5.5.2. Comments 146
11.5.5.3. Drawing Tools 147
11.5.5.4. Toggle Between Editing Modes 147
11.5.5.5. Close Shape 149
11.5.5.6. Opacity 150
11.5.5.7. Waypoints 152
11.5.5.8. Diagram-specific Tools 152
12. Element Reference 155
12.1. Relationships 155
12.1.1. Types of Relationships 156
12.1.2. Navigability 156
12.1.3. Hiding and Displaying Multiplicity of 1 157
12.1.4. Self-Associations 158
12.2. Classes 159
12.2.1. Attributes 159
12.2.2. Operations 162
12.2.3. Association Classes 164
12.3. Interfaces 164
12.3.1. Box Notation 164
12.3.2. Lollipop Notation 165
12.3.2.1. Sockets 165
12.3.3. Ports 166
13. Using Elements 169
13.1. Creating New Elements 169
13.1.1. Diagram Pane Toolbar 169
13.1.2. The Rapid Buttons 170
13.2. Editing Elements 172
13.2.1. Inline Editing Text Values 172
13.2.2. Editing Via the Details Pane 174
13.2.2.1. The Properties Tab 174
13.2.2.2. The Style Tab 175
13.2.2.3. The Documentation Pane 176
13.2.3. Editing Via the Context Menu 177
13.2.4. Undo/Redo 178
13.2.5. Stereotypes 178
13.2.6. Removing and Deleting Elements 180

14. Generation 183
14.1. Code Generation 183
14.1.1. Generation Settings 183
14.1.2. Reverse Engineering 185
14.1.3. Roundtrip Engineering 186
14.1.4. Fine Tuning Code Generation 189
14.2. Advanced Code Generation 192
14.2.1. Velocity Template Language 192
14.2.1.1. References 193
14.2.1.2. Directives 194
14.2.1.3. Comments 194
14.2.1.4. Examples 195
14.2.2. Working with the Standard Templates 197
14.2.3. Code Generation API 197
14.3. Documentation Generation 198
14.3.1. UMLdoc 198
14.3.2. Generation Settings 200
14.3.3. Supported Javadoc Tags 201
15. Plug-Ins 205
15.1. The Plug-In Panel 205
15.1.1. Installing a New Plug-In 206
15.1.1.1. Add the Plug-In License 206
15.1.1.2. Install the Plug-In 206
15.1.1.3. Enable the Plug-In 207
15.2. Removing Plug-Ins 207
15.3. Available Plug-Ins 207
15.3.1. JAR Import 208
15.3.2. RoundTrip UML/Java 208
15.3.3. Refactoring Browser 208
15.3.4. MDL Import 209
15.3.4.1. Installing and Using MDL Import 209
15.3.4.2. Supported Diagrams 209
15.3.4.3. Unsupported Features 209
15.3.4.4. Display Issues 210
15.3.4.5. Status 211
16. Advanced Features 213
16.1. Constraints with OCL 213
16.2. Critiques 213
16.3. Searching for Model Elements 215
16.4. Profiles 216

17. Using The Enterprise Edition 217
17.1. Interface 217
17.1.1. Connection Status 217
17.1.2. Toolbar 217
17.1.3. Menu 218
17.1.4. License Manager 218
17.1.4.1. Test Connection 219
17.2. Modeling with Others 219
17.2.1. Collaborations 219
17.2.1.1. New Collaboration 220
17.2.1.2. Join Collaboration 220
17.2.1.3. Leave Collaboration 220
17.2.2. Projects 221
17.2.2.1. Load and Start Project 221
17.2.2.2. Upload Project 221
17.2.3. Model Locking and Conflict Checking 221
17.2.3.1. Java-Import 223
17.3. Enterprise Server Administration Tool 223
17.3.1. Collaboration Administration 224
17.3.1.1. Removing locks 224
17.3.1.2. Renaming collaborations 224
17.3.1.3. Ending collaborations 224
17.3.2. Project Administration 224
17.3.3. CVS Support 224
17.3.3.1. Configuring Your System for CVS Support 225
17.3.3.2. Using the CVS Support 226
18. Epilogue 229
A. Poseidon C# Code Generation Plug-In Guide 231
A.1. General Rules 231
A.1.1. Tagged Values 231
A.1.2. Additional Stereotypes 231
A.2. Modeling Element Rules 231
A.2.1. Classes 232
A.2.1.1. Class Signature 232
A.2.1.2. Class Attributes 232
A.2.1.3. Class Operations 232
A.2.2. Interface 233
A.2.2.1. Interface Signature 233
A.2.2.2. Interface Members 233
A.2.3. Structure 233
A.2.3.1. Structure Signature 234
A.2.3.2. Structure Members 234

A.2.4. Enumeration 234
A.2.4.1. Enumeration Signature 234
A.2.5. Delegate 235
A.2.5.1. Delegate Signature 235
A.2.6. C# Event 235
A.2.7. Operations 235
B. Poseidon CORBA IDL Code Generation Plug-In Guide 237
B.1. General Rules 237
B.2. CORBA Interface 237
B.3. CORBA Value 237
B.4. CORBA Struct 237
B.5. CORBA Enum 238
B.6. CORBA Exception 238
B.7. CORBA Union 238
C. Poseidon Delphi Code Generation Plug-In Guide 241
C.1. Classfiers 241
C.2. Tagged Values 241
C.2.1. Classifier 241
C.2.2. Attribute 242
C.2.3. Operation 242
C.2.4. Exception 243
C.3. Stereotypes 243
C.3.1. Attribute 243
C.3.2. Operation 243
C.3.3. Classifier 244
C.4. Modeling Element Rules 244
C.4.1. Class 245
C.4.2. Interface 245
C.4.3. Enumeration 245
C.4.4. Record 245
C.4.5. Set 245
C.4.6. Sub Range 246
C.4.7. Array 246
C.4.8. Exception 246
C.5. Specific Rules 246
D. Poseidon PHP4 Code Generation Plug-In Guide 249
D.1. General Rules 249
D.1.1. Tagged Values 249
D.2. PHP4 Class Modeling Rules 249
D.2.1. Class Signature 249
D.2.2. Class Attributes 250
D.2.3. Class Operations 250

E. Poseidon Perl Code Generation Guide 253
E.1. General Rules 253
E.2. Classes 253
E.3. Class Attributes 253
E.4. Class Operations 254
E.5. Associations 254
E.6. Aggregation 254
E.7. Inheritance 254
F. Poseidon SQL DDL Code Generation Plug-In Guide 255
F.1. Modeling Element Rules 255
F.1.1. Classes 255
F.1.2. Attributes 255
F.1.3. Association Ends 255
F.2. Tagged Values 255
F.3. Additional Stereotypes 255
G. Poseidon VB.Net Code Generation Plug-In Guide 257
G.1. General Rules 257
G.2. Classes 257
G.3. Interfaces 257
G.4. Modules 258
G.5. Structures 258
G.6. Enums 258
G.7. Operations 259
G.8. Operation’s Parameters 259
G.9. Visual Basic Properties 259
G.10. Visual Basic Events 260
G.11. Attribute & Association Ends 260
Glossary 261

List of Tables

2-1. Edition Comparison 12

List of Figures

4-1. Poseidon for UML application work area. 28
4-2. Navigation pane in the Stattauto model. 30
4-3. Class diagram ’Container Class Analysis-Packages’ 31
4-4. Change a view in the Navigation pane 32
4-5. Add a navigation view tab 33
4-6. Delete a navigation view tab 33
4-7. The Diagram pane displaying the diagram ’Entity Class Model Overview’ 34
4-8. The Details pane with class ’Reservation’ selected. 35
4-9. Class diagram as seen in the Birdview tab 37
4-10. Critiques of the Stattauto example 37
4-11. The Navigation pane in a Diagram Centric view. 38
4-12. Select class ’Reservation’ from Diagram Centric view 40
4-13. The Details pane with the class ’Reservation’ selected. 40
4-14. The Properties tab with the attribute ’number’ selected. 41
4-15. Change an operation name in a diagram 42
4-16. Change operation name from the Details pane 43
4-17. Add a package to a diagram with the rapid buttons 44
4-18. Delete an element from a model 45
4-19. Remove an element from a diagram 46
6-1. Panes in Poseidon 61
6-2. Grid Settings dialog 65
6-3. Properties tab with zoom 67
6-4. Drill-down navigation 68
6-5. C++ tab for an attribute 69
6-6. Style tab for an element without compartments 70
6-7. Style tab for an element with compartments 70
6-8. To Do tab in the Details pane 71
6-9. Source code tab for a class 72
6-10. Documentation tab for a class - WYSIWYG and source 74
6-11. New constraint in the Constraints tab 76
6-12. Documentation stored in the Tagged Values tab 77
7-1. The General settings tab. 83
7-2. The Appearance settings tab. 84
7-3. The Modeling settings tab. 85
7-4. The Environment settings tab. 86
7-5. The User settings tab. 87

7-6. The Project settings tab. 88
7-7. The Key Mappings settings tab. 89
7-8. The Diagram display settings tab. 90
9-1. Export a project to XMI 101
9-2. Watermarked Community Edition diagram graphic 102
9-3. Premium Edition diagram graphic without watermark ..... **??**
10-1. A Use Case diagram. 105
10-2. A Class diagram 108
10-3. An Activity diagram. 112
10-4. A State diagram 115
10-5. A Sequence diagram. 118
10-6. Selecting the action of a stimulus in a sequence diagram. ..... **??**
10-7. Selecting an operation and attaching arguments to it. 119
10-8. A Component diagram. 123
10-9. A Deployment diagram. 125
11-1. Tooltip displaying documentation 131
11-2. Properties tab displaying class 'Reservation' 133
11-3. Properties tab with operation 'Member' selected. 133
11-4. Zooming by changing the properties of a diagram. 134
11-5. Zooming from the Birdview tab 135
11-6. Selecting multiple elements with the mouse. 141
11-7. Adding waypoints. 144
11-8. Moving adornments. 144
11-9. A new comment 147
11-10. Add a waypoint to a rectangle 148
11-11. Open and closed lines 149
11-12. Changing opacity 150
12-1. Properties tab for an association. 155
12-2. Properties tab for an association end. 155
12-3. Highlight hints for associations. 157
12-4. Style tab with multiplicity unset and set 157
12-5. The rapid button for self-associations 159
12-6. Properties of an attribute. 160
12-7. 'Remove Attributes' setting 161
12-8. Properties of an operation. 162
12-9. Uni-Directional Ports 166
12-10. Bi-Directional Port 167
13-1. Rapid buttons for a class element. 170
13-2. Toggled representation of actors and interfaces 171
13-3. Additional rapid buttons for a class element. 172
13-4. Add a new attribute or operation to a class inline 173
13-5. Properties tab for a class 174
13-6. Style tab for a class 175

13-7. Editing a method documentation. 177
13-8. Context menu options for a Use Case 177
13-9. A Class diagram making use of stereotypes. 179
13-10. Stereotype dialog 180
14-1. Generation menu 183
14-2. Code Generation dialog and settings - Java 184
14-3. Import Files dialog ..... **??**
14-4. Select file check interval. 187
14-5. Java code generation - settings. 187
14-6. Generated UMLdoc opened in Netscape. 199
14-7. Code Generation dialog and settings - UMLdoc 200
14-8. UMLdoc code generation - settings 201
16-1. A Constraints tab. 213
16-2. Critiques pane. 214
16-3. Searching for a class 215
16-4. The Profile Manager 216

List of Examples

14-1. Simple HTML Template 195
14-2. Simple Java template 196

Chapter 1. About Gentleware and Poseidon for UML

1.1. About Gentleware and Poseidon for UML

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

About 4000 years later, Jason Robbins started an open source project for a UML modeling tool and named it ArgoUML. Many others joined him in this adventurous undertaking, including a group of software developers 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 very popular. They realized the great demand for a tool like ArgoUML, as well as the amount of work necessary to shape it into a professionally usable tool. They finally took the risk of starting a company with the goal of bringing the most usable tool to a broad audience. With respect to their open-source origin, the company is called Gentleware and their tool is called Poseidon for UML.

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

1.1.1. Our Vision

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

1.1.2. Innovation

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

1.1.3. Cooperation

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

1.1.4. Contact

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

Email

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

General information, feature requests, or suggestions:

http://www.gentleware.com?redirect=contact

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

http://www.gentleware.com?redirect=contact

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?redirect=forum) 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?redirect=order), 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?redirect=order) provided on our web site. Our fax number is +49 40 2442 5331.

Please try to find an answer to your question on our web pages (http://www.gentleware.com/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

1.2. New Features in Version 2.x

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:

- Interfaces can be rendered in box or lollipop notation
- Ports are now available in Object, Collaboration, and Deployment diagrams
- Performance has been greatly improved
- 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.

- New elements, such as association classes and actors-as-classifiers, and diagram helpers that do not appear in the source code, such as text objects, have been added.
- 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.
- A new documentation editor with full HTML markup capabilities has been included.
- Derived and ordered attributes can be defined.
- Multiple stereotypes can be applied to all elements.
- Source code target languages are chosen from separate code generation menu items, and the source tab can be set to display different languages
- More languages are included in the Professional Edition: Java, Perl, VB.Net, Delphi, CorbaIDL, PHP, C#, and SQL, in addition to UMLdoc generation.

A complete list of changes can be found at
http://www.gentleware.com?redirect=changelog.

1.3. About This Document

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

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

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

Chapter 2. Editions

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

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

Poseidon for UML is released in *Versions* as well as in *Editions*. All Editions are based on the same source base and therefore carry the same version number. New versions are released a couple of times per year. This document refers to version 2.x.

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. Some of the features that are available in the commercial editions are not included in the free Edition. These features are nice to have to increase your productivity, but are not necessarily required to build UML models. Perhaps most important, the Community Edition does not support reverse or round-trip engineering, and it cannot load plug-ins. The Community Edition also does not support printing, copy and paste to the Windows clipboard (to copy diagrams to Word or Powerpoint for example), the zoom is restricted, and graphic exports include watermarks. 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.
- Some diagrams allow drag and drop within the tool.
- Internationalization and localization for English, German, French, Spanish, and Simplified Chinese.
- Forward engineering for Java.
- Simple installation and updates with Java Web Start.
- Full Undo and Redo.

2.2. Standard Edition

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

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

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

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

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

2.3. Professional Edition

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

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

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

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

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

2.4. Enterprise Edition

Team support is provided in the Enterprise Edition, the premier version of Poseidon for UML. It features version control, multi-user support, client-server architecture, and many more features that you might need for model-driven software engineering in a highly collaborative development environment. It supports multi-model editing and scales to high volume models.

The Enterprise Edition comes in two parts, the server and the client component. Both of them feature an easy installation process. Transmission of files between clients and server is secured by using standard ssh encryption and authentication. In order to support the communcation aspect of collaborative modeling, the client also features an integrated instant messenger based on the Jabber protocol.

The Enterprise Edition includes all features of the Professional Edition, in addition to the following:

- Collaborative modeling environment based on a client-server architecture.
- Exclusive locking of model parts and elaborated conflict checking.
- Server-based project handling.
- Instant messaging features for client-to-client communication.
- Secure transmission of files between clients and server using SSH.
- Easy installation process for client and server.

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 most of 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	EmbEd Embedded Edition	PE Professional Edition	EE Enterprise Edition
Simple Install with WebStart	✔				
UML 2.0 Diagram Interchange	✔	✔	✔	✔	✔
All 9 Diagram Types	✔	✔	✔	✔	✔
Forward Engineering Java	✔	✔	✔	✔	✔
XMI Supported	✔	✔	✔	✔	✔

Platform Independent	✔	✔	✔	✔	✔
Export as GIF, JPG, PNG, PS, EPS, SVG	✔	✔	✔	✔	✔
Copy/Cut/Paste Within Poseidon	✔	✔	✔	✔	✔
Internal Drag and Drop	✔	✔	✔	✔	✔
Internationalization	✔	✔	✔	✔	✔
OCL Support	✔	✔	✔	✔	✔
Undo/Redo	*	✔	✔	✔	✔
UMLdoc (HTML Export)	**	✔	✔	✔	✔
Reverse Engineering Java		✔	✔	✔	✔
Printing		✔	✔	✔	✔
WMF Graphic Export		✔	✔	✔	✔
No Watermark in Graphic Export		✔	✔	✔	✔
Copy/Cut/Paste of Diagrams to Other Applications		✔	✔	✔	✔
Plug-In support		✔	✔	✔	✔
Support		✔	✔	✔	✔
JAR Import			✔	✔	✔
MDL Import			✔	✔	✔
C++ Generation			✔	✔	✔
ANSI C Generation			✔		
C# Code Generation				✔	✔
CORBA IDL Code Generation				✔	✔
Delphi Code Generation				✔	✔

Perl Code Generation				✔	✔
PHP Code Generation				✔	✔
SQL DDL Code Generation				✔	✔
VB.net Code Generation				✔	✔
Changeable Code Templates				✔	✔
Round Engineering Java				✔	✔
Synchronous Team Modeling					✔
Version Control					✔
Element Locking					✔
Secure Communication					✔
GoVisual Autolayout Plug-In by Oreas		(✔)	(✔)	(✔)	
yWorks Autolayout Plug-In by yWorks		(✔)	(✔)	(✔)	
AndroMDA Plug-In by M. Bohlen		(✔)	(✔)	(✔)	(✔)
b+m EJB Plug-In by b+m Informatik AG		(✔)		(✔)	(✔)

Notations:

✔ Included in this Edition

(✔)Not included in this Edition, can be purchased separately

***** Limited to 3 steps in history

****** Available externally at http://www.umldoc.org

Chapter 3. Installation and First Start

3.1. Prerequisites

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

- Java 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 1.3 or older.
- A computer with reasonable memory and CPU power. For memory, 512 MB is recommended, more is helpful. For CPU, a Pentium III or equivalent is the recommended minimum.
- A specific operating system is not required. Poseidon for UML is known to run on Windows 98, 2000, NT, and XP, on Linux SuSe 6.X, 7.X, Red Hat, and MacOS X. It has been predominantly developed and tested on Linux. However, on Windows platforms performance is known to be superior due to a faster Java environment.

3.1.1. Additional Requirements for MacOS X Users

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

3.1.1.1. Jaguar (MacOS X 10.2.x)

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

3.1.1.2. Panther (10.3.x)

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

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

3.2. Community, Standard, Professional, and Embedded Editions

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

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

A complete installation guide is available at
http://www.gentleware.com?redirect=installguide
(http://www.gentleware.com/products/documentation/installguide/InstallGuide.pdf)

3.2.1. Install Using InstallAnywhere

The easiest way to install Poseidon for UML is to use the InstallAnywhere (http://www.installanywhere.com) installer for your platform. If you already have a recent Java version installed, you can download the installer for your platform that does not include Java. If you do not have Java installed or if you are not sure of the version, download the installer that includes Java.

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

3.2.1.1. Windows

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

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

3.2.1.2. *NIX

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

Ensure that your installation file executable with:

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

Then execute the file. If ’.’ iIncluded in this Editions not in your path you should use:

./PoseidonCE_2_1_1Installer.bin to start the file.

3.2.2. Install Through Java Web Start (Community Edition Only)

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, 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.

Web Start provides a big advantage in 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?redirect=webstart
   - 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?redirect=webstart 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.

3.2.3. 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?redirect=download.
2. Follow the download instructions. You will then have a single file stored on your local hard drive in the location you have indicated.
   - The file is compressed using zip format. Move this file to the folder where you would like to install Poseidon for UML. Then, to decompress it, call the zip program used on your platform. Here are some examples:
   - On Linux or Unix, open a command shell, go to the folder where the downloaded file is stored (using the `cd` command), and call **unzip PoseidonPE-##.##.zip**. The file may be named differently, depending on the edition you downloaded.
   - On Windows, start your Zip program. The Zip program should automatically start if you double-click on the downloaded file. Then extract the file to a folder of your choice by selecting extract and following the instructions.
3. All Poseidon files are extracted into a folder `PoseidonForUML_XX_##.##`.
4. Switch to the `/bin` subfolder.
5. Run the start script provided authentificationfor your platform:
   - On Linux or Unix, open a command shell, enter the `poseidon.sh` command, and press return.
   - On Windows, open the file explorer and double-click the start script `poseidon.bat`.

3.2.4. Silent Installation

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

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

3.2.4.1. Installer Properties File

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

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

Here are the contents of a sample properties file:

USER_INSTALL_DIR=/programs/PoseidonPE2/

INSTALLER_UI=silent

3.2.4.2. Command Line Parameter

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

installername -i
silent

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

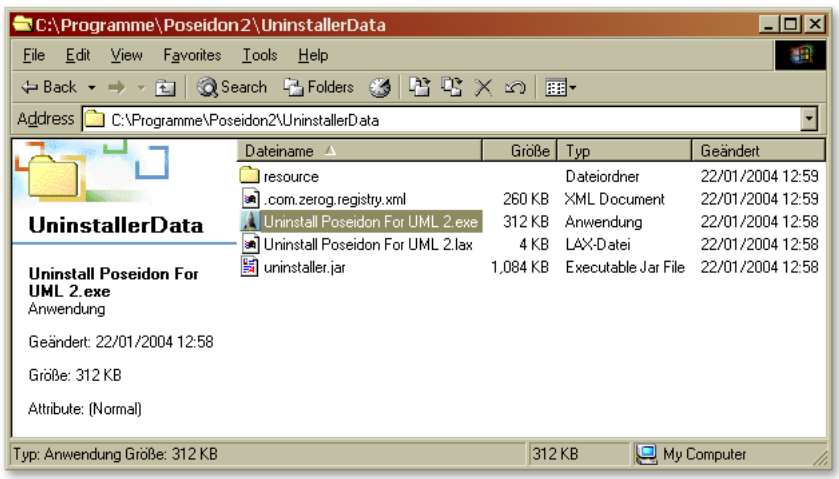
installername -f
<properties file>

3.2.4.3. Uninstallation

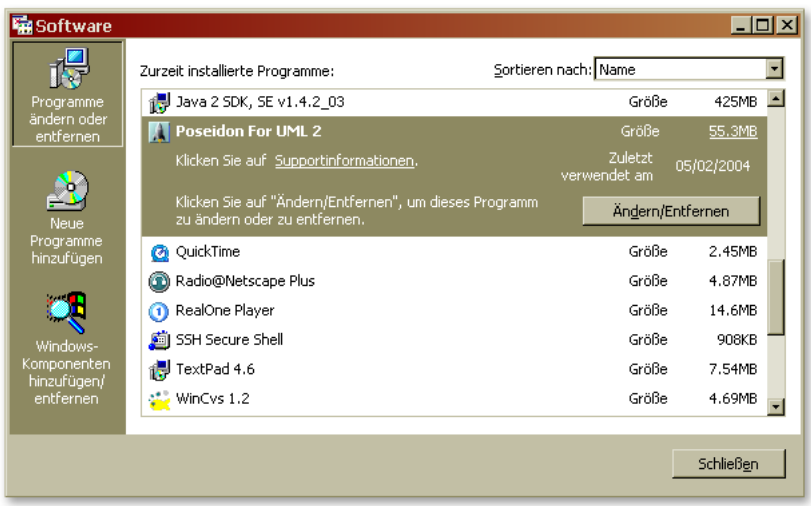
When silent mode using a properties file has been used to install Poseidon and this file still exists, uninstallation is also completed silently.

3.2.5. Uninstallation

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

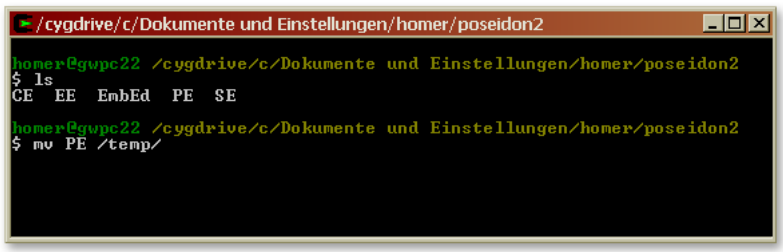


From Windows, you have the additional option of uninstalling Poseidon from the Software section of the Control Panel.



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

If you are planning to re-install Poseidon, you should copy your Poseidon settings directory to another location before starting the uninstall procedure in order to avoid losing your keys and registration data. In general, the Poseidon settings directory is located under your user's home directory in poseidon2/<*poseidon edition*>, where <*poseidon edition*> is the abbreviation for the Edition. Below, the user 'homer' has moved his Professional Edition directory to a temporary location before proceeding with the uninstallation.



3.3. Enterprise Edition

The Enterprise Edition is made up of two components, the Client application and the Server application. At least one server and one client must be installed in order to use the Enterprise Edition, but the extended capabilities of the Enterprise Edition will not be realized without the installation of at least two client applications. A single Installer program is used to set up both components.

The complete configuration is beyond the scope of this document. Please refer to the Enterprise Edition Installation Guide, available from http://www.gentleware.com?redirect=ee-installguide for complete installation information.

3.4. Environment Variables

The installation processes described above should enable Poseidon to run properly on your system. However, some adjustments can be made by using environment variables in order to make Poseidon fit even better in your personal environment.

Please refer to the instructions of your operating system to see how environment variables can be set and persisted.

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

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

3.5. Keys and Registration

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

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

3.5.1. Types and Terminology

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

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

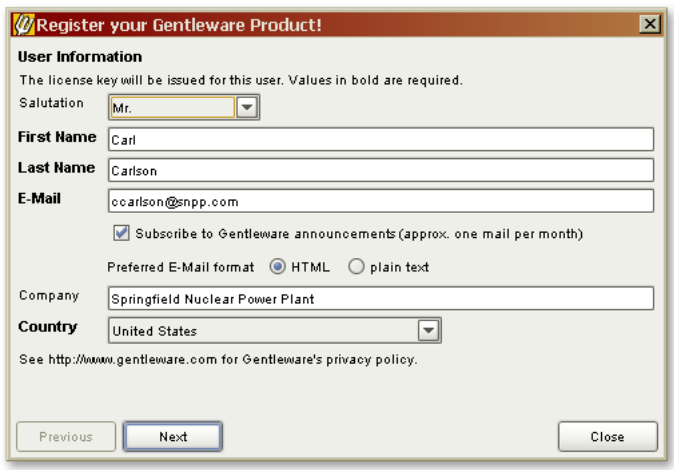
License Key - provided for the Community Edition and purchased copies of Premium Editions. These keys are made available after the registration data has been received by Gentleware. Once a License Key is in place, the registration process is complete. These keys need no further attention, unless the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

3.5.2. 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’.



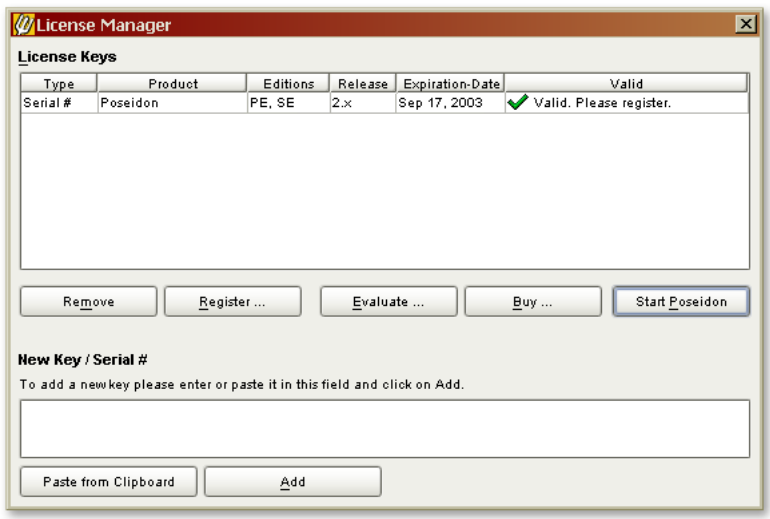
4. To finish the registration process, select either the online option or the web option (ideal for users who are behind a firewall).

3.5.3. Evaluation Copy

Premium Editions of Poseidon (Standard, Professional, Enterprise, or Embedded) can be evaluated free-of-charge, but be aware that some functionality is limited, e.g. saving is limited to eight diagrams. The evaluation key is valid for 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.



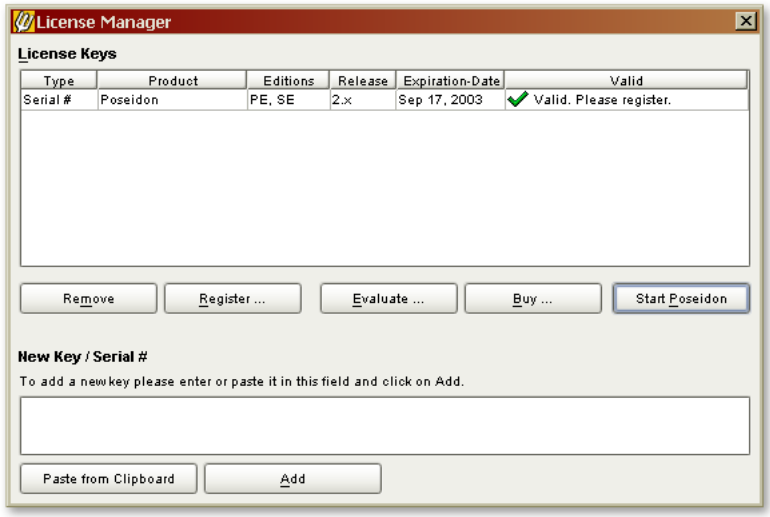
3. Click ’Add’.

3.5.4. Premium Version Purchase

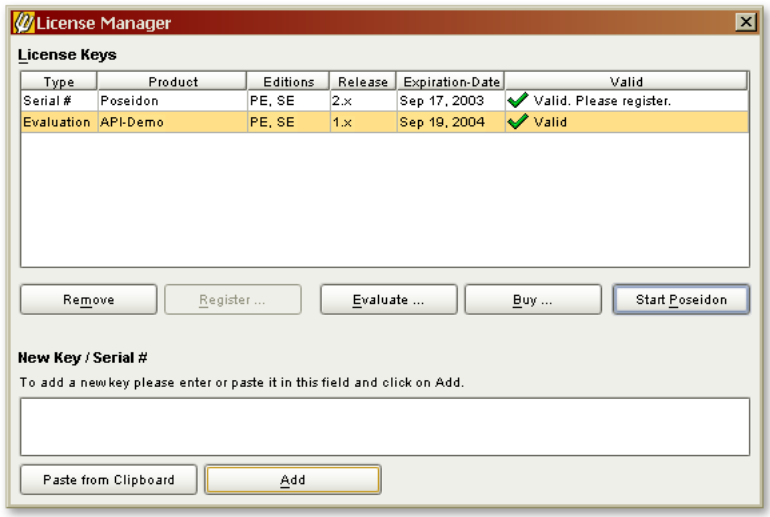
Registering a purchased copy of a Premium Edition follows a similar process as an evaluation copy.

To register a Premium Edition:

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



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



Note that if you try to register via the website, Poseidon searches for Netscape as the default browser. You can change the default browser from the Settings dialog in the Edit menu. Note that this setting does not currently work for Macs.

3.5.5. Keys for Plug-Ins

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

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

Chapter 4. A Short Tour of Poseidon for UML

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

4.1. Opening the Default Example

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

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

This example is based on a car rental scenario in which a company named *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.

4.2. Introducing the Work Area

The work area of Poseidon is separated in five parts. At the top of the window, there is a main menu and a toolbar that provide access to the main functions. Below this

are four **panes:**

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

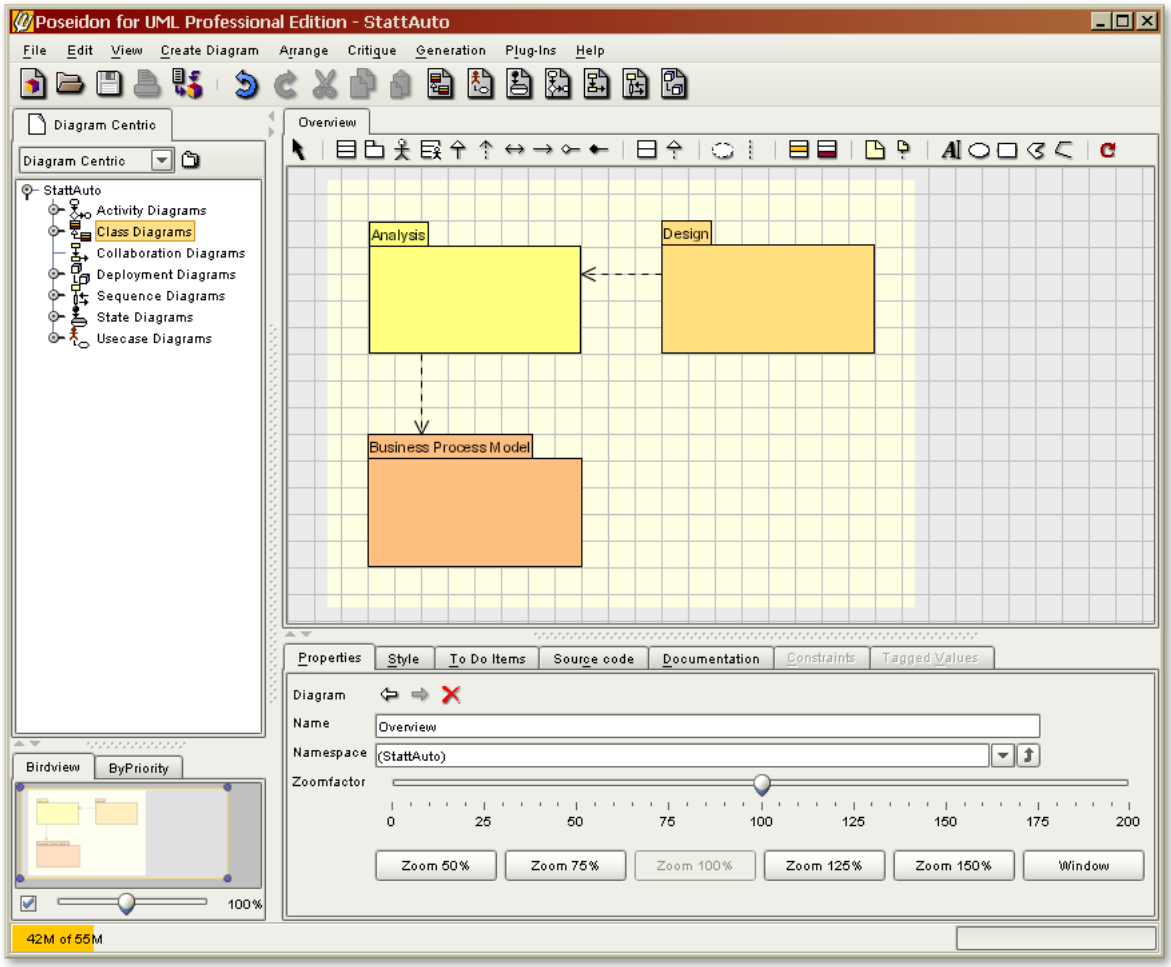


Diagram Pane

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

Navigation Pane

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

Overview Pane

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

4.2.1. The Navigation Pane

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

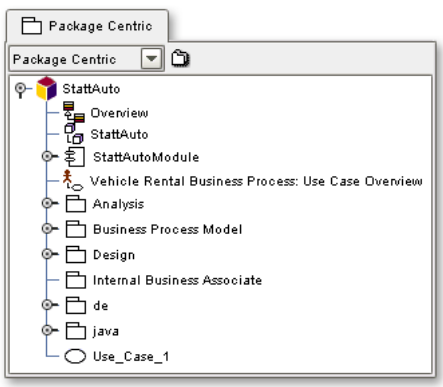
Some sample icons:

- **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 4-2. Navigation pane in the Stattauto model.

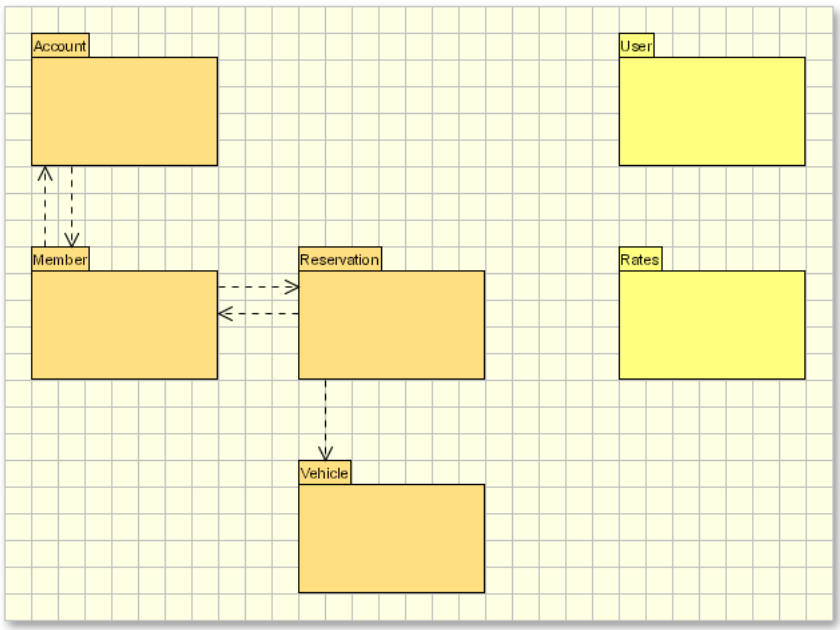


The model `Stattauto` contains many packages (e.g. `Analysis`, `Business Process Model`, and `Design`) as well as a large number of

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 4-3. Class diagram 'Container Class Analysis-Packages'



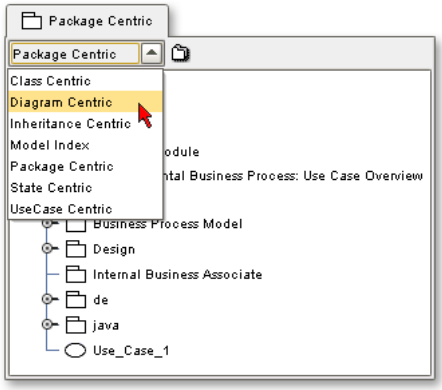
Inside the packages you can find further diagrams, but to quickly browse through the existing diagrams you need not navigate through the packages themselves. You can find diagrams directly (and much more quickly) using the **diagram tree**.

4.2.1.1. Changing the Navigation View

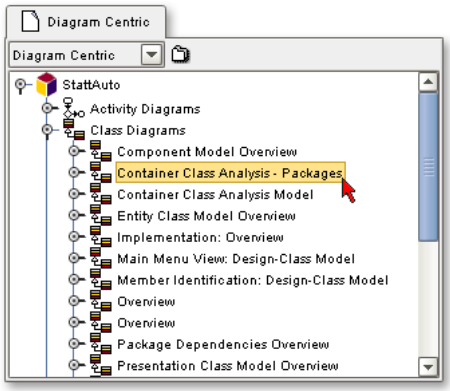
Let's now take a quick look at the diagram tree, which can be seen in the Diagram Centric view. At the top of the Navigation pane, there is a drop-down selection box.

Select the Diagram Centric view.

Figure 4-4. 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.

4.2.1.2. Opening Multiple Navigation Panes

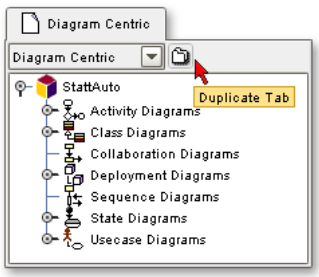
As we have seen, views offer different advantages for different tasks. But we often

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:

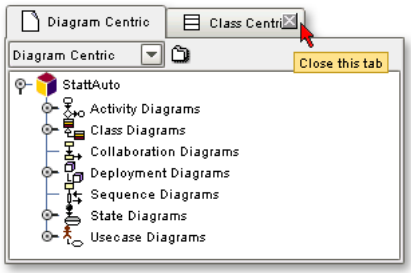
1. Click on the folder icon (called the 'duplicate tab button') that is located to the right of the drop-down selection box.
2. A new navigation view will be created behind the current view.
3. Now you can select the Package Centric view from the dropdown menu of one tab and the Diagram Centric view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

Figure 4-5. Add a navigation view tab



You can delete the navigation view tabs using the delete button that 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 4-6. 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.

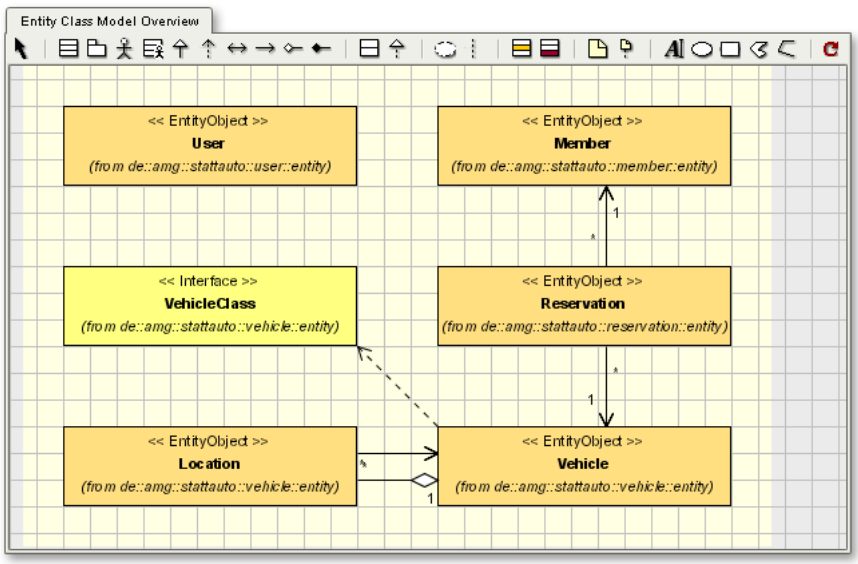
4.2.2. The Diagram Pane

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

Open the diagram `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 4-7. The Diagram pane displaying the diagram 'Entity Class Model Overview'.



This is an overview diagram that 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.

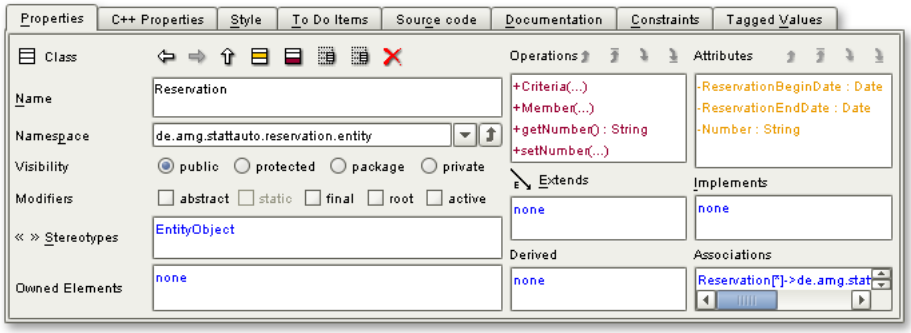
Try it for yourself by selecting one of the classes in this diagram from the Navigation pane. Notice how the class name is highlighted in the Navigation pane, while the Diagram pane displays the same class with its rapid buttons visible around it.

4.2.3. The Details Pane

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

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

Figure 4-8. The Details pane with class 'Reservation' selected.



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

Tabs available in the Details pane:

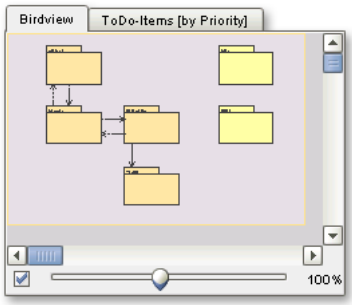
- Properties
- Style
- To Do Items
- Source Code
- Documentation
- Constraints
- Tagged Values

4.2.4. The Overview Pane

The larger a diagram becomes, the harder it gets to keep track of all of the elements, especially once they are out of the immediate viewing area. The Overview pane allows you to keep track of the elements already in the diagram. The pane, located at the bottom left, provides access to two tabs. First of the two is the 'Birdview' tab,

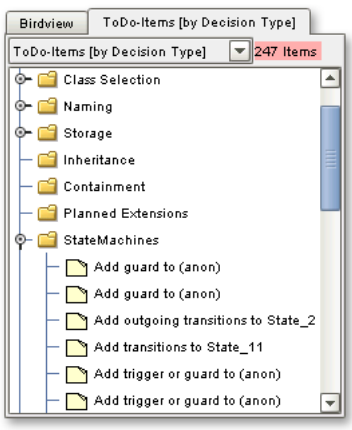
which displays a graphic summary of the diagram currently displayed in the Diagram pane. From this tab you can zoom and/or pan in either the Diagram pane or the Overview pane. The second tab, called the 'ByPriority' tab, contains a collection of critiques that have been compiled by Poseidon.

Figure 4-9. 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 4-10. Critiques of the Stattauto example



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.

4.3. Navigation

A UML model can become quite complex as it expands to include more and more information. Different aspects of the model are important to a variety of people at particular times. Additionally, there is no one correct approach to viewing a model and the information it contains. A UML tool should provide comprehesive yet simple-to-use mechanisms to access and change that information as each individual requires. Therefore, Poseidon for UML offers various ways of navigating between model elements to accomodate all of these needs. We will now take a closer look at some of the most important ones.

4.3.1. Navigating with the Navigation Pane

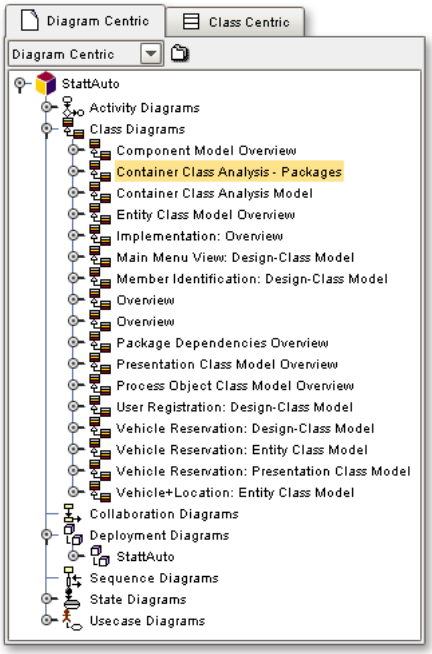
The central mechanism for moving through the models is the Navigation pane, mentioned above. It organizes the complete UML model into a tree view that provides access to almost all parts of that model via the opening and closing of subtrees. At the top of the Navigation pane you will find a drop-down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

Views available from the Navigation pane

- 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 4-11. 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.

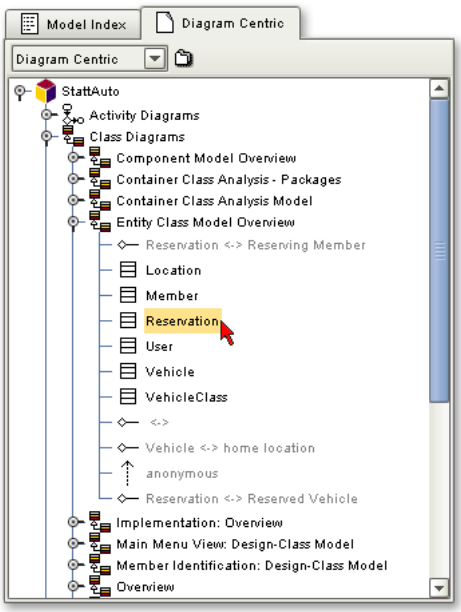
Take a look at the Model Index view by selecting Model Index from the drop-down menu. The Navigation pane will change to display an alphabetical list of all elements in the model. This illustrates yet another useful way to locate elements.

4.3.2. Navigating in the Properties Tab

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

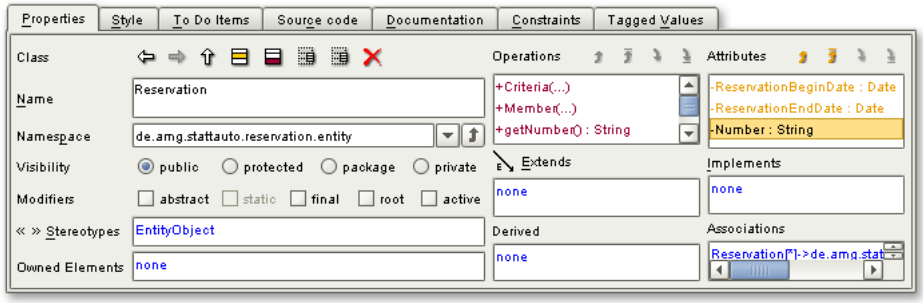
From the Diagram Centric view in the Navigation Pane, open the diagram ’Entity Class Model Overview’ subtree and select the class ’Reservation’.

Figure 4-12. Select class ’Reservation’ from Diagram Centric view



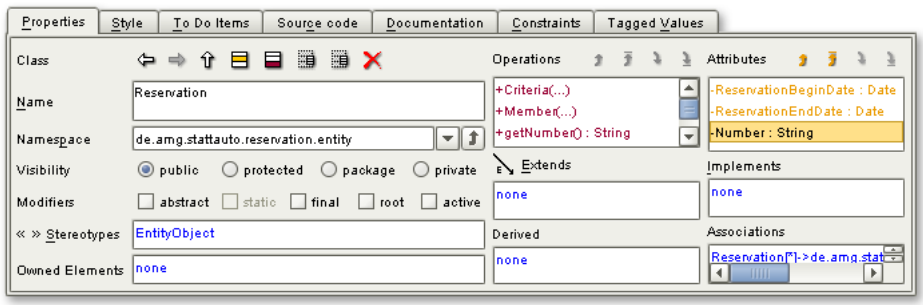
Take a look at the left side of the Properties tab. Listed here are properties of the class itself which can be modified, such as name and visibility. To the right are components of the class, elements in and of themselves. These components have their own properties in their own properties tab.

Figure 4-13. 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 4-14. The Properties tab with the attribute 'number' selected.



4.4. Modify Elements

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

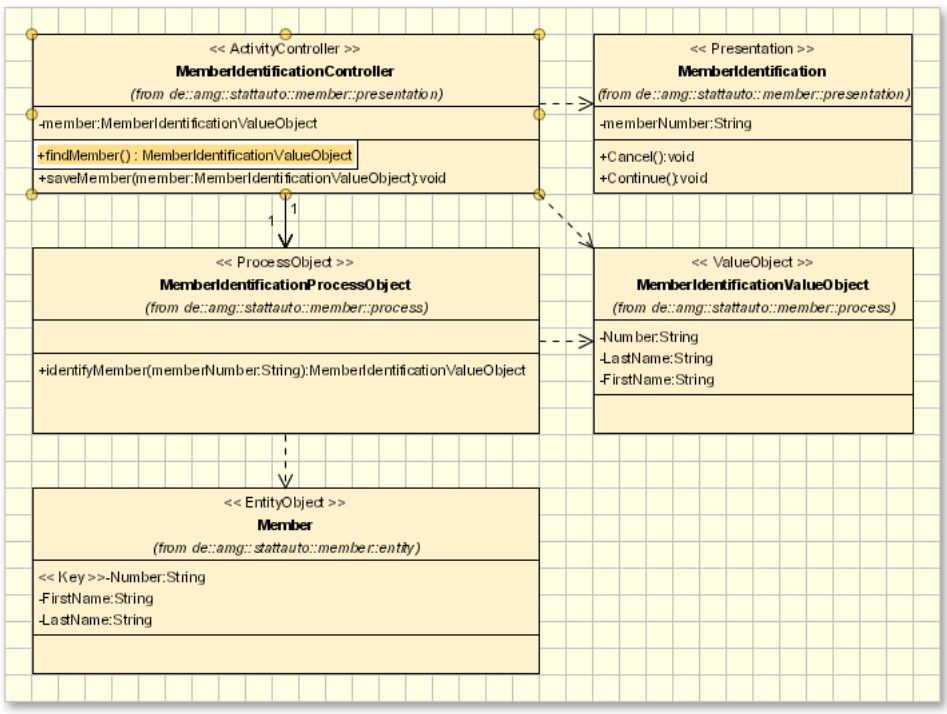
4.4.1. Change Element

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 displays the information about this operation, and the text in the Diagram pane itself is now editable from a text box. Change the name of the operation to 'searchMember()' and then press Ctrl-Return or click elsewhere in the diagram.

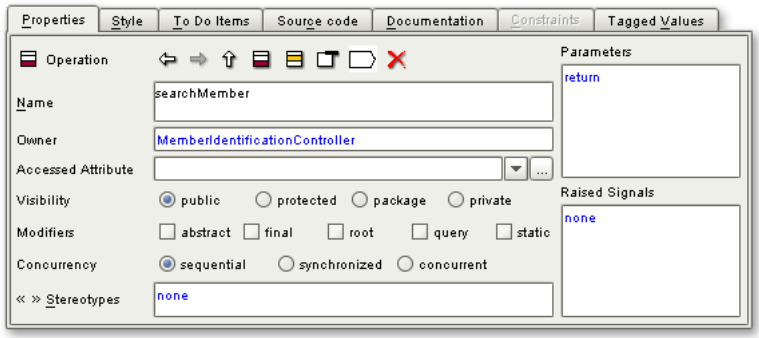
Figure 4-15. Change an operation name in a diagram



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 4-16. Change operation name from the Details pane



4.4.2. 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.

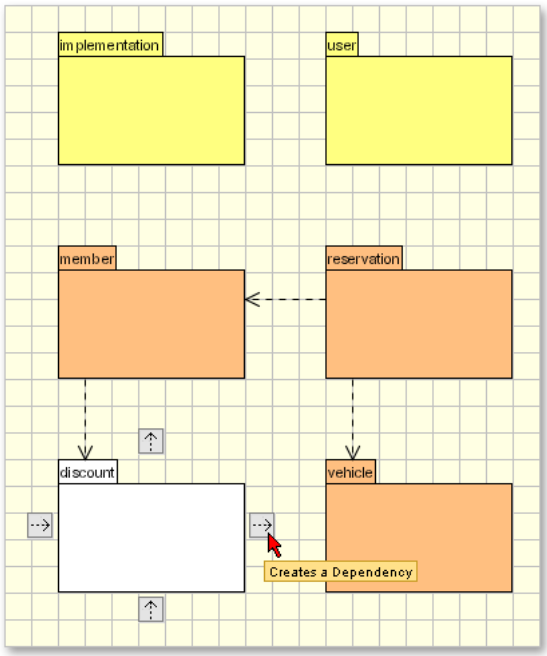
Perhaps we would like to associate a discount program with members. Let's create a new package, Discount. In the 'Package Dependencies Overview' class diagram, click on the 'Package' button from the Diagram pane toolbar. A cross-hair appears. Click in the Diagram pane to place the new package into the 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 4-17. Add a package to a diagram with the rapid buttons

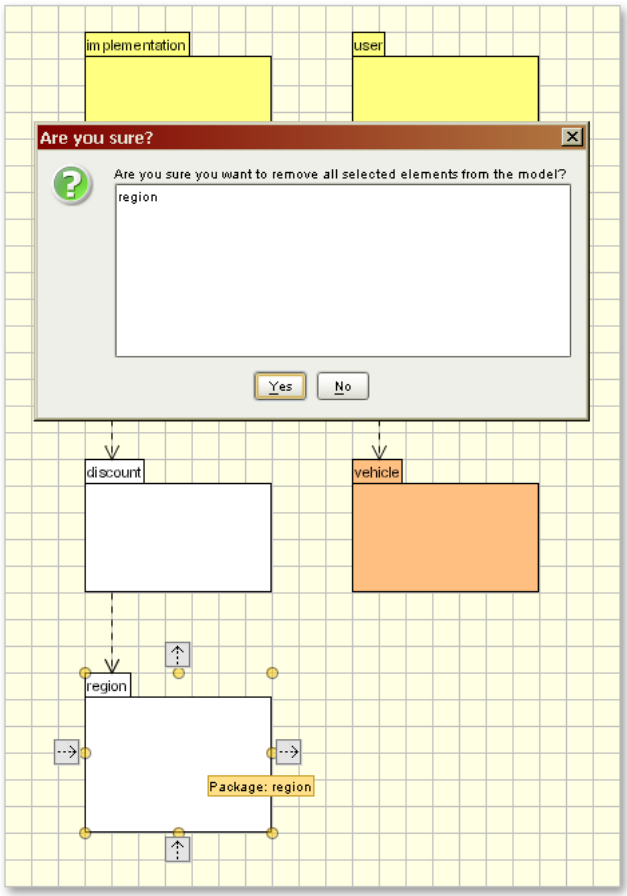


4.4.3. Delete Elements

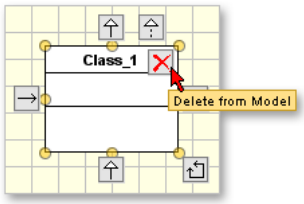
So after further review, we have decided that the package ’discount’ is a good idea, but not for this diagram. We have further decided that the ’region’ package is unnecessary and will not be used elsewhere in the model. Let’s first delete ’region’ completely.

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

Figure 4-18. Delete an element from a model



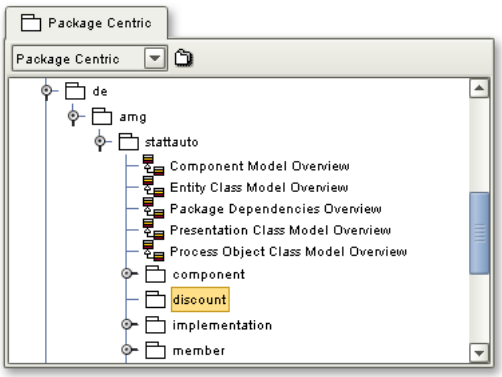
To delete an element without encountering the dialog box, you can press Ctrl and mouseover the element. A delete rapid button will appear in the upper right corner.



The package ’discount’ is a different story. We may need to use this again elsewhere, so we just want to remove it from this one diagram, not from the entire model. To remove it from this diagram, select the 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 not encounter a warning here, but the package still exists within the model and can be viewed from the Navigation pane.

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

Figure 4-19. Remove an element from a diagram



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

Chapter 5. Interface

5.1. Toolbar

 **New Project**

 **Open Project**

 **Save**

 **Print***

 **Import Files**

 **Roundtrip Enabled*****

 **Roundtrip Disabled*****

 **Undo****

 **Redo****

 **Cut**

 **Copy**

 **Paste**

 **New Class Diagram**

 **New Use Case Diagram**

 **New State Diagram**

 **New Activity Diagram**

 **New Collaboration Diagram**

 **New Sequence Diagram**

 **New Deployment/Object/Component Diagram**

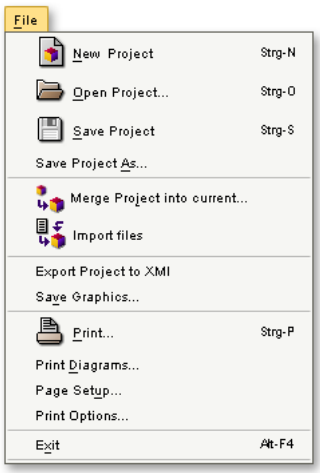
* Not available in the Community Edition

** Limited functionality in the Community Edition

*** Available in the Professional and Enterprise Editions only

5.2. Menus

5.2.1. File



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

  Quick Key - Ctrl-N

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

  Quick Key - Ctrl-O

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

  Quick Key - Ctrl-S

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

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

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

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

> **Note:** Not available in the Community Edition

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

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

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

  Quick Key - Ctrl-P

  > **Note:** Not available in the Community Edition

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

  > **Note:** Not available in the Community Edition

- **Page Setup...** - Opens a dialog to select page options such as paper size and orientation.

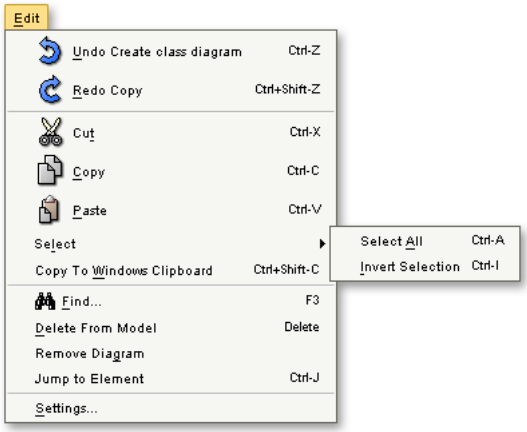
  > **Note:** Not available in the Community Edition

- **Print Options...** - Opens the Print Options dialog to select options such as scaling.

  > **Note:** Not available in the Community Edition

- **Exit** - Exits Poseidon. If there are any edited projects, a save prompt will appear.

5.2.2. Edit



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

  Quick Key - Ctrl-Z

  > **Note:** Limited functionality in the Community Edition

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

  Quick Key - Ctrl-Alt-Z

  > **Note:** Limited functionality in the Community Edition

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

  Quick Key - Ctrl-X

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

  Quick Key - Ctrl-C

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

  Quick Key - Ctrl-V

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

    Quick Key - Ctrl-A

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

    Quick Key - Ctrl-I

- **Copy to Windows Clipboard** - Places the element(s) into the Windows clipboard without altering the element selection.

  Quick Key - Ctrl-Alt-C

  Note: Not available in the Community Edition

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

  Quick Key - F3

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

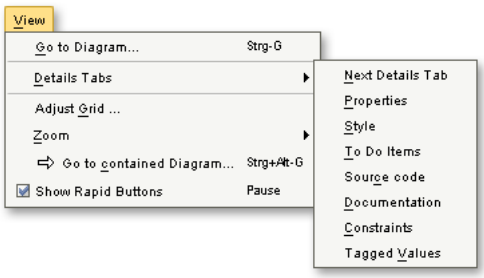
  Quick Key - Del

- **Remove Diagram** - Removes the selected diagram from the project without deleting the elements contained within that diagram.
- **Jump to Element** - Opens a dialog to quickly navigate to a specified element

Quick Key - Ctrl-J

- **Settings...** - Opens the Settings dialog, where you can change some of the default properties of Poseidon.

5.2.3. View



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

  Quick Key - Ctrl-G

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

- **Adjust Grid...** - Opens the grid dialog, where the grid display can be modified.
- **Zoom** -
  - **Zoom Factors** - List of zoom factors to change the diagram display.

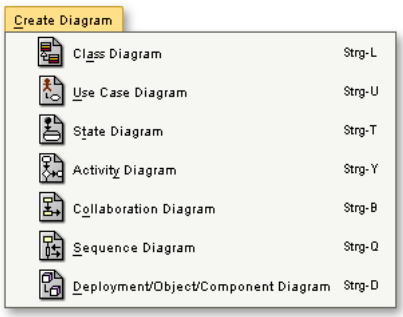
- **Go To Contained Diagram...** - Navigates to the sub-diagram of the current element. If the element has more than one sub-diagram, the diagram navigation dialog will open, filtered on the current element.

  Quick Key - Ctrl-Alt-G

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

  Quick Key - Pause

5.2.4. Create Diagram



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

  Quick Key - Ctrl-L

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

  Quick Key - Ctrl-U

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

  Quick Key - Ctrl-T

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

  Quick Key - Ctrl-Y

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

  Quick Key - Ctrl-B

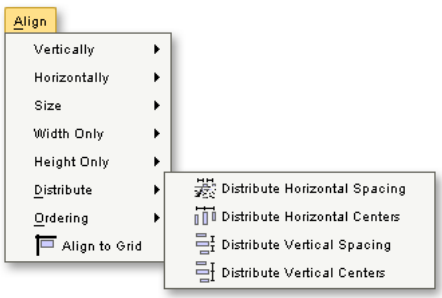
- **Sequence Diagram** - Creates a new blank diagram.Limited functionality in the Community Edition

  Quick Key - Ctrl-Q

- **Deployment/Object/Component Diagram** - Creates a new blank diagram.

  Quick Key - Ctrl-D

5.2.5. Align



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

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

- **Size** -
  - **Greatest Current Width and Height** - Uniformly resizes selected elements so that each is the size of the largest selected element.

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

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

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

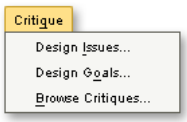
- **Distribute** -
  - **Distribute Horizontal Spacing** - Places a uniform vertical gutter between elements. Vertical spacing and element size are not changed.
  - **Distribute Horizontal Centers** - Places a uniform amount of space between the center vertical axes of the selected elements. Vertical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

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

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

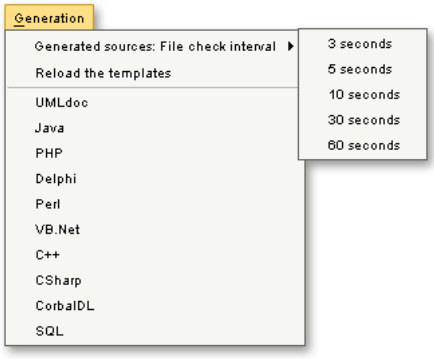
- **Align to Grid** - Snaps the top and left edges of an element to the grid. Element size is not changed.

5.2.6. Critique



- **Design Issues...** - Opens the issues dialog, where you can specify the importance of each design criterion. This affects the order of critiques.
- **Design Goals...** - Opens the goals dialog, where you can specify the importance of each design goal. *Note: this is not functional in the current version of Poseidon.*
- **Browse Critiques...** - Displays a list of all possible critiques. Here you can toggle the critiques and change their importance.

5.2.7. Generation



- **Generated Sources: File Check Interval** -
  - **Interval Listing** - Determines the length of time between comparison of the source code and the current project.

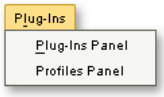
    Note: Available in the Professional and Enterprise Editions only

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

    Note: Available in the Professional and Enterprise Editions only

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

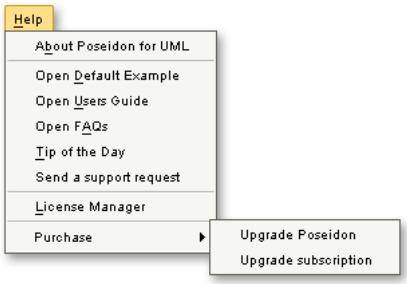
5.2.8. Plug-Ins



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

> **Note:** Not available in the Community Edition

5.2.9. Help



- **About Poseidon for UML** - Displays Poseidon version information and credits.
- **Open Default Example** - Opens the Stattauto project.
- **Open Users Guide** - Opens the local html version of the Poseidon User Guide.
- **Open FAQs** - Opens the local html version of the FAQ.
- **Tip of the Day** - Opens the Tip of the Day dialog.
- **Send a Support Request** - Opens the Support Request web page at the Gentleware web site. Poseidon will launch a web browser, if necessary.

> **Note:** Not available in the Community Edition

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

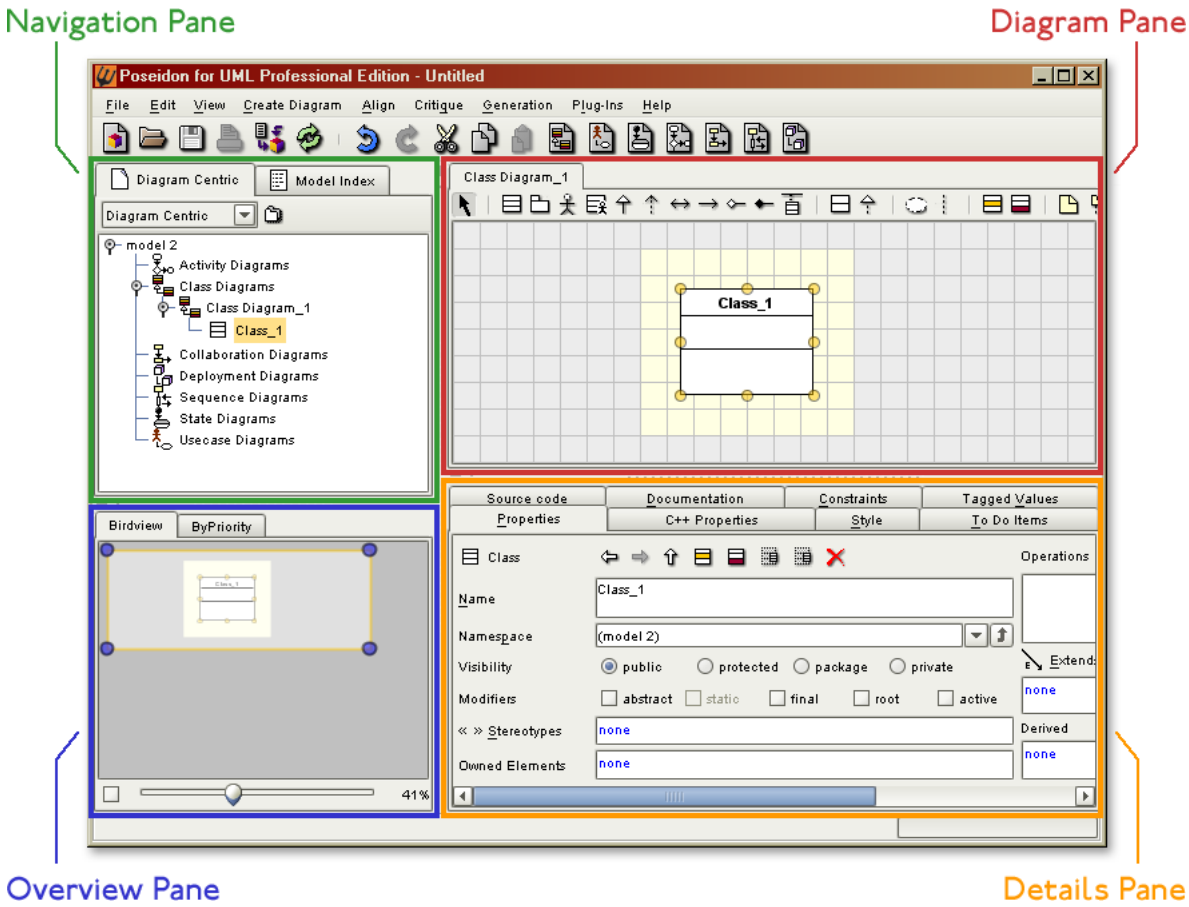
  Note: Not available in the Community Edition

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

Chapter 6. Panes

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

Figure 6-1. Panes in Poseidon



6.1. Navigation Pane

As a model grows, its complexity likewise increases. It becomes more and more necessary to have different organizations of the model to facilitate easy navigation. This is what the Navigation pane has been designed to do; present the elements of a

model in different arrangements based on pre-determined criteria. Poseidon calls these arrangements ’views’.

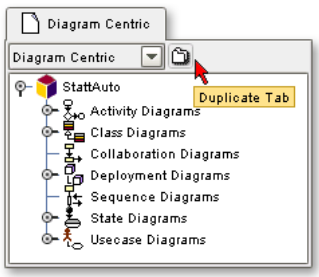
Each view positions the elements within the model hierarchy differently. Views are not required to display all of the elements of a model, only those which pertain to their organization schema. The one similarity between all of the views is the root node, which is always the model itself. In the case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

The views offered by Poseidon are as follows:

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

6.1.1. 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.

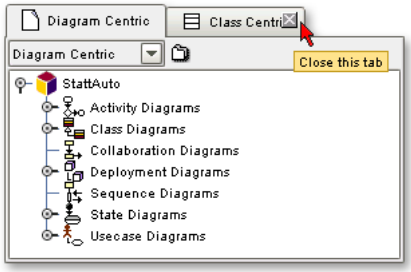


To add a tab to the Navigation pane:

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.

6.1.2. 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.

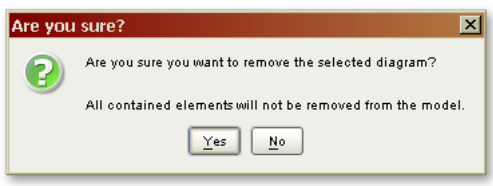
6.1.3. Delete a Diagram

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.

6.2. Diagram Pane

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

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

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

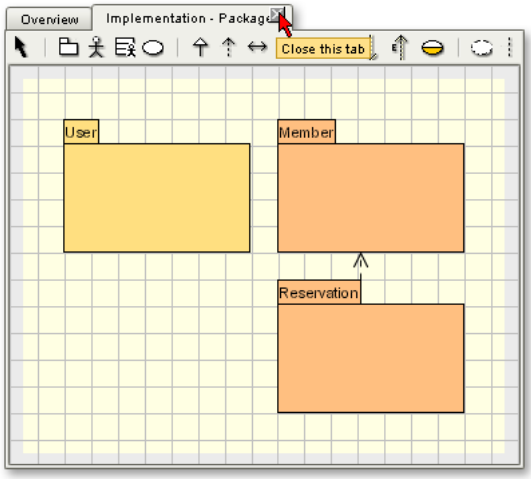
6.2.1. Diagram Pane Toolbar

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

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

6.2.2. 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.



6.2.3. Change Properties of the Diagram Pane

6.2.3.1. Grid Settings

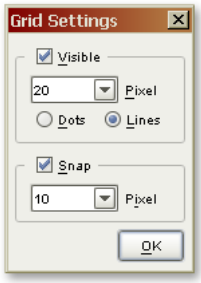
The first thing you may notice about the Diagram Pane is the grid that is drawn over the drawing area. By default, the drawing area displays this grid. The visible grid is only a collection of lines, they have no functions of their own.

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 6-2. 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.

6.2.3.2. Other Settings

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

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

6.3. Details Pane

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

The pane is composed of eight tabs:

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

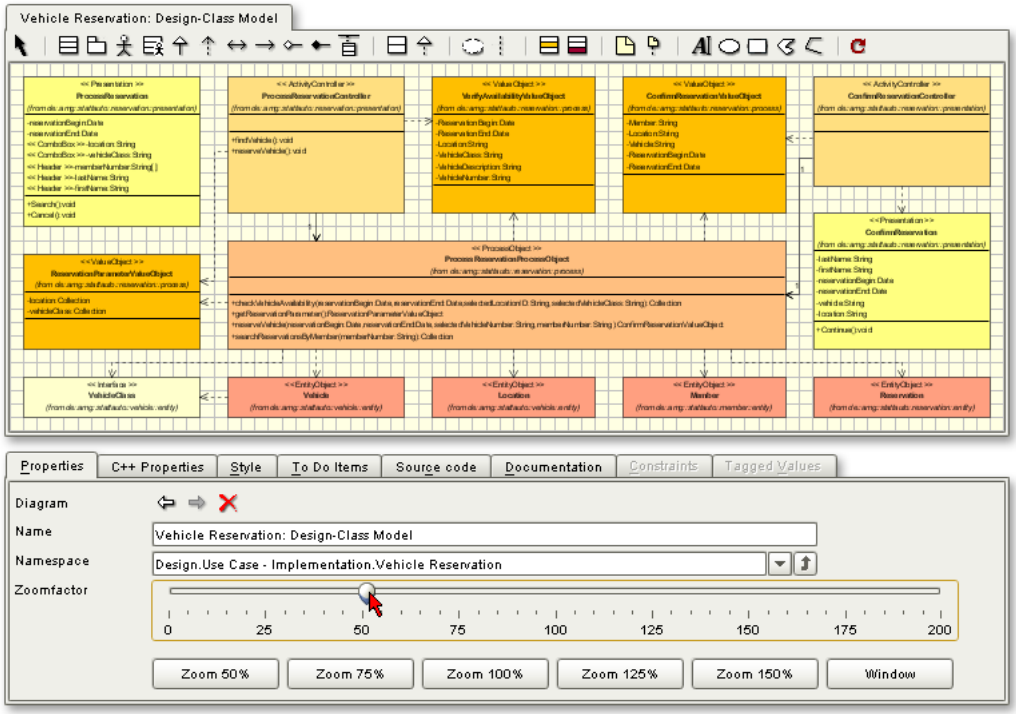
The following sections investigate these tabs in greater detail.

6.3.1. 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.

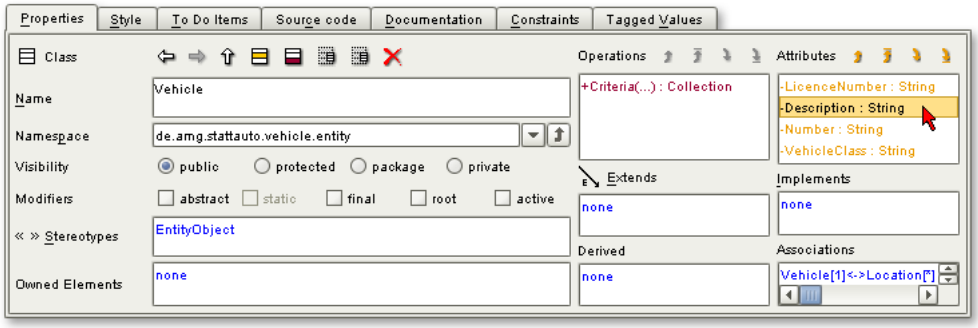
An important property, the zoom factor, becomes visible in the Details pane when a diagram name is selected in the Navigation pane. You can use the slider to change the zoom factor interactively or use the buttons to set it to pre-selected zoom factors (The range of zoom factors is limited in the Community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

Figure 6-3. 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.

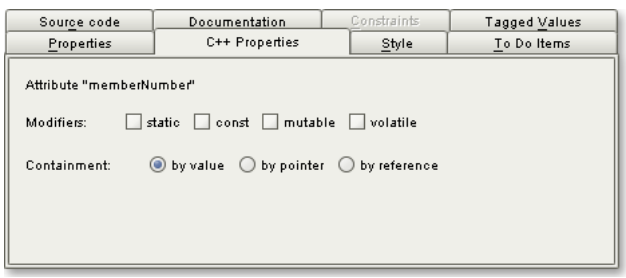
Figure 6-4. Drill-down navigation



6.3.2. C++ Properties

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

Figure 6-5. C++ tab for an attribute

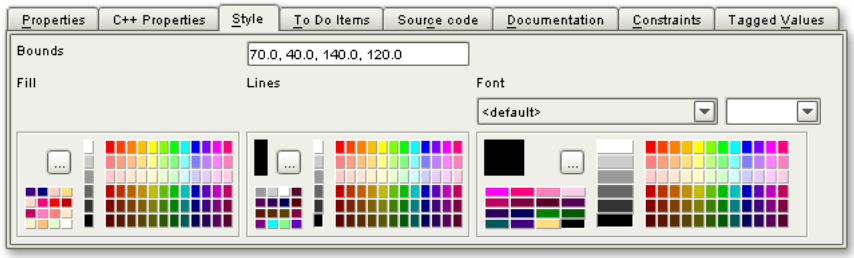


Note: Not available in the Community or Standard Editions

6.3.3. Style Tab

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

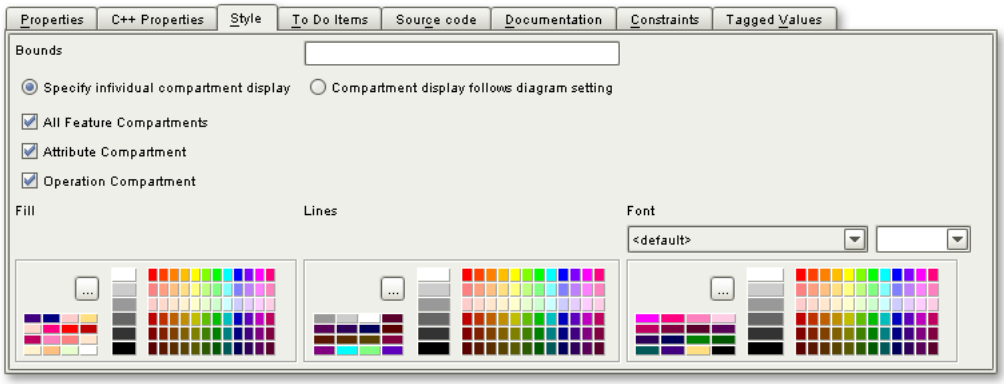
Figure 6-6. Style tab for an element without compartments



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

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

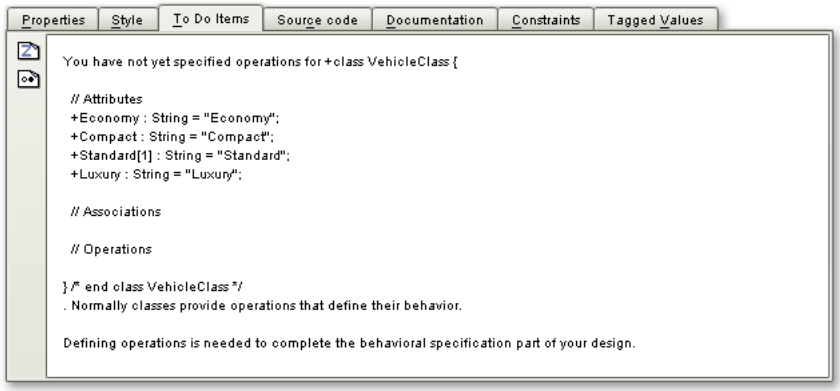
Figure 6-7. Style tab for an element with compartments



6.3.4. To Do Items Tab

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

Figure 6-8. To Do tab in the Details pane



6.3.4.1. Snooze Critique

The Snooze critique button temporarily turns off a single critique. The critique

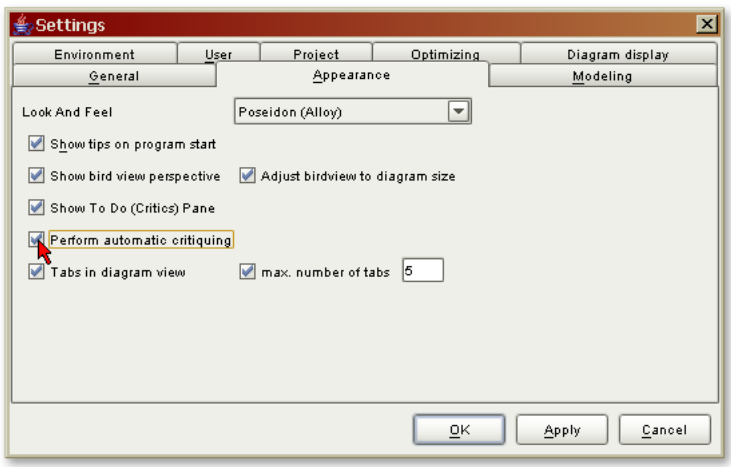
will return without a user specifically re-enabling it.

6.3.4.2. 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.

6.3.4.3. Turn Off Autocritique

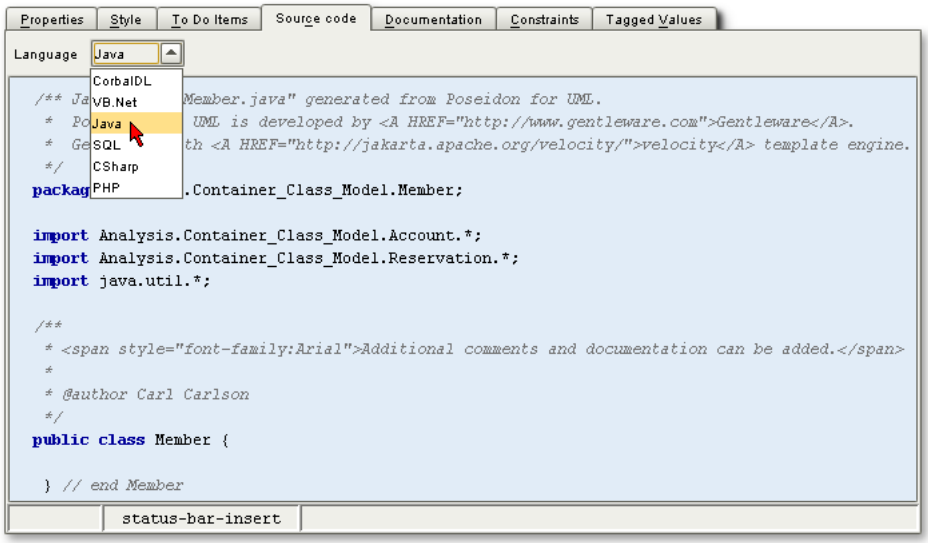
Critiquing can be turned off completely from the Appearance tab of the Settings dialog. Uncheck the box titled, 'Perform Automatic Critiquing' to disable critiques. The tab will still be visible in the Details pane, but no critiques will be listed.



6.3.5. Source Code Tab

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

Figure 6-9. Source code tab for a class



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

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

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

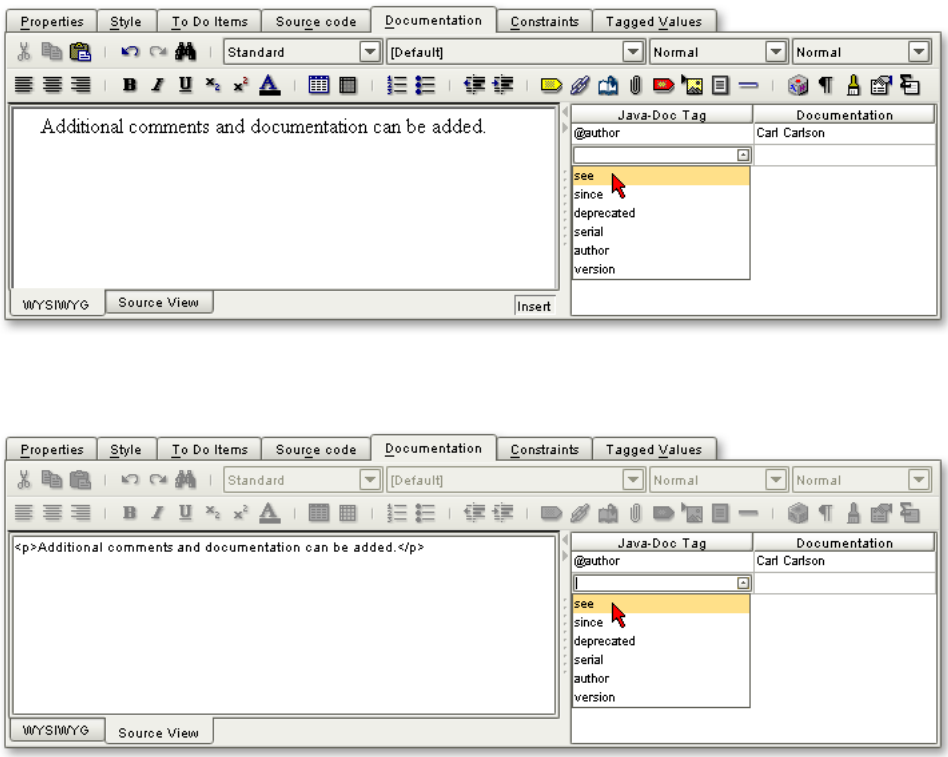
The same diagrams may be used to generate code in different languages. Any code written in the ’your code here’ sections is available only in the language selection in which it was written. For example, any code manually entered into the editable section of this tab while Java is the selected language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

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

6.3.6. Documentation Tab

- Contains a WYSIWYG editor, where you can easily add your own documentation for model elements
- You can also use javadoc tags, like `@author`, `@see` and others

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



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

There are a couple of choices for entering text - you can use the WYSIWYG editor and format the text using the documentation toolbar, or you can edit in plaintext mode and add html formatting tags by hand. Both methods result in html-formatted text; for instance, pressing 'Enter' while in the WYSIWYG tab generates a new paragraph with the   tags, and pressing 'Ctrl-Enter' generates the
 tag.

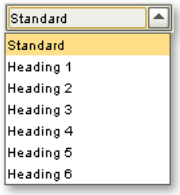
This information is stored as tagged values and can be previewed in the Java Source tab. Any entries made via the editor on the left side of the tab are placed in paragraph tags by default and are displayed before the JavaDoc entries.

6.3.6.1. Toolbar

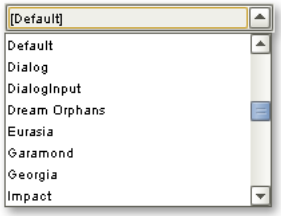
- Cut
- Copy
- Paste
- Undo
- Redo
- Search and Replace
- Align Left
- Align Center
- Align Right
- Bold
- Italic
- Underline
- Subscript
- Superscript
- Font Color
- Insert Default Table
- Insert Table
- Numbered List
- Bulleted List
- Decrease Indent
- Increase Indent
- Link to Model Element
- Insert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

6.3.6.2. Dropdowns

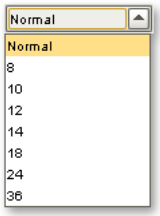
Paragraph Style



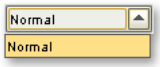
Font Type



Font Size



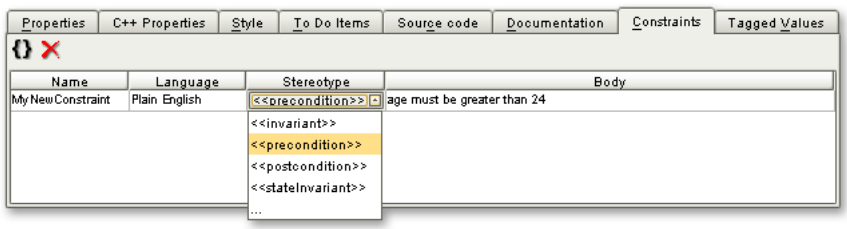
Style Sheet



6.3.7. Constraints Tab

- Holds the constraints for the given element

Figure 6-11. New constraint in the Constraints tab

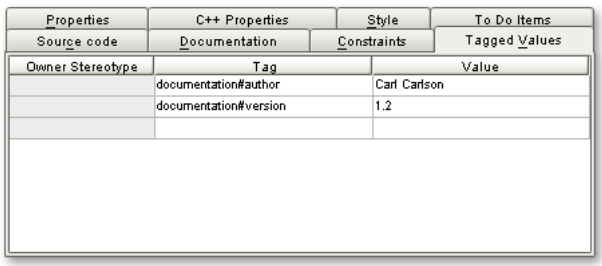


The UML does not include specifications regarding constraint language. Poseidon is now able to hold constraint information that is language independent, including plain English. Of course, you are still free to use OCL if you so choose.

6.3.8. Tagged Values Tab

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

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

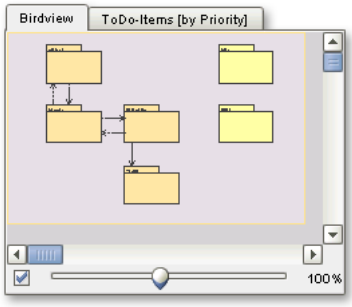


If, for example, you need special information for external processing of the model you can add this information here. This is also where Poseidon stores any documentation entered in the Documentation tab.

6.4. Overview Pane

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

6.4.1. Birdview Tab



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

6.4.1.1. Redisplay a Section of a Diagram

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

6.4.1.2. Zoom in Birdview Only

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

To change the zoom factor of the Birdview tab:

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

6.4.1.3. Zoom in a Diagram

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

enlarged or reduced.

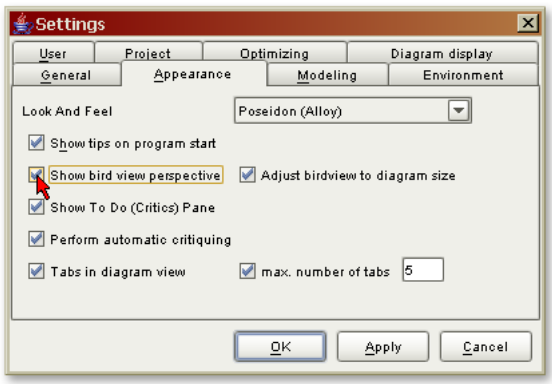
To change the zoom factor of the diagram in the Diagram pane:

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

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

6.4.1.4. Turn Off Birdview in Settings

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



6.4.2. 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.

6.4.2.1. 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 - Uncategorized, Behavior, Class Selection, Naming, Storage, Inheritance, Containment, Planned Extensions, State Machines, Design Patterns, Relationships, Instantiation, Modularity, Expected Usage, Methods, Code Generation, Stereotypes
- by Diagrams - Seems to be broken
- by Knowledge Type - Correctness, Syntax, Presentation, Completeness
- by Offenders - Model Elements
- by Posters - arranged according to the critic who reported them
- by Priority - High, Medium, Low

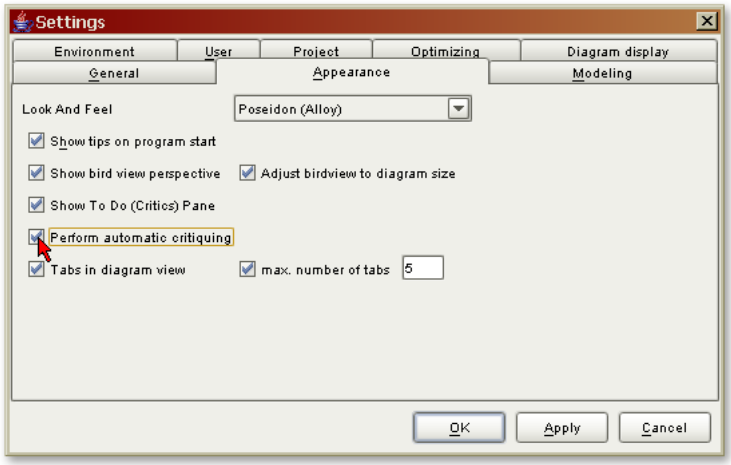
To view the details of a critique, simply click on the critique. The details will be displayed in the ’To Do Items’ tab of the Details pane, located to the right of the Overview pane.

6.4.2.2. Navigate to a 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.

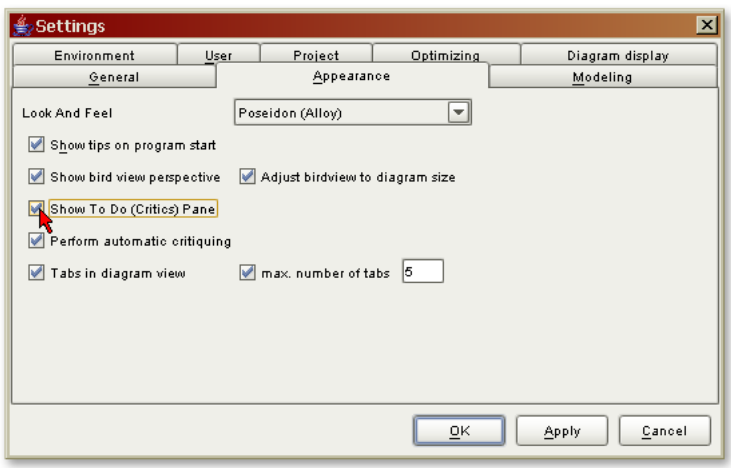
6.4.2.3. Turn Off Autocritique

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.



6.4.2.4. 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'.

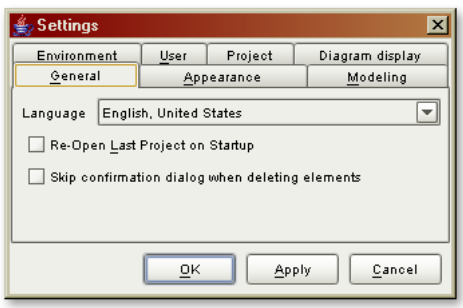


Chapter 7. Setting Properties

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

7.1. General

Figure 7-1. The General settings tab.

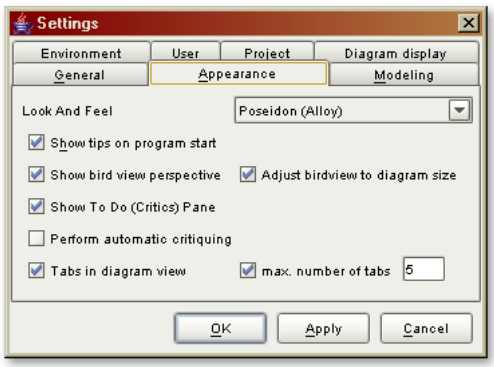


- **Language** - This is the language used for the Poseidon user interface. You can switch the interface to a different language by choosing your preferred language from this selection list. Poseidon currently supports English, German, French, Spanish, Italian and Chinese. By default, the system language is used - or English, if the system language is not available. In other words, if you start Poseidon on a Spanish system, for example, the program will start in Spanish - but on a Swedish system the program will start in English, as Swedish is not currently supported.
- **Re-Open Last Project on Startup** - If checked, Poseidon opens the most recently used project on startup.
- **Skip confirmation dialog when deleting elements** - If checked, the dialog asking if you really want to delete selected elements will not appear.

- **Browser** - This sets the default web browser used by Poseidon. To change the browser, click on the ellipse button and navigate to the location of your favorite web browser. This does not currently work on Macs.

7.2. Appearance

Figure 7-2. The Appearance settings tab.



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

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

  1. Start Poseidon, then shut it down again.
  2. Open the Poseidon.properties file in a text editor. In Windows, this is found in: `C:\Documents and Settings\<User>\poseidon2\<Poseidon Version>`
  3. Change the line

     `poseidon.init.laf=...`

to

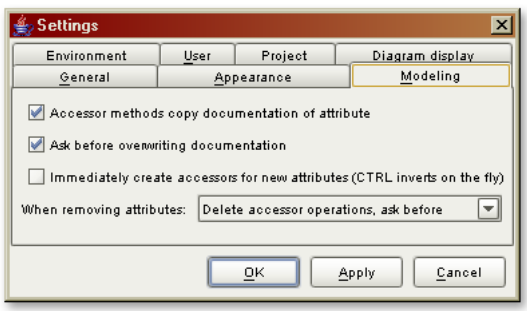
```
poseidon.init.laf=com.incors.plaf.alloy.AlloyLookAndFeel
```

4. Save this file and restart Poseidon.

- **Show tips on program start** - If checked, a `Tip of the Day` dialog appears when you start up the program.
- **Show 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.

7.3. Modeling

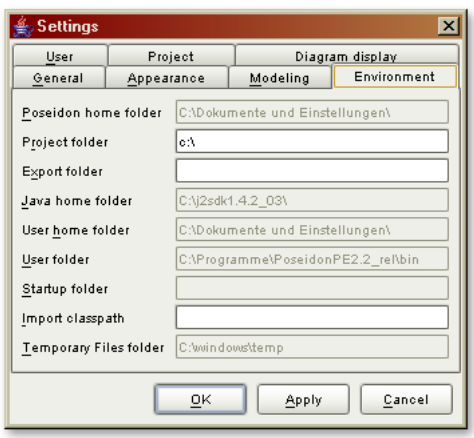
Figure 7-3. The Modeling settings tab.



- **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.

7.4. Environment

Figure 7-4. The Environment settings tab.



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

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

7.5. User

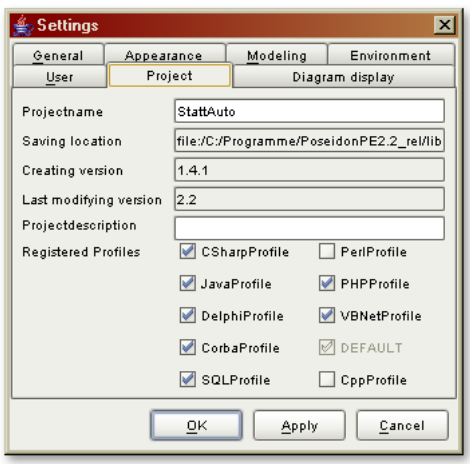
Figure 7-5. The User settings tab.

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

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

7.6. Project

Figure 7-6. The Project settings tab.



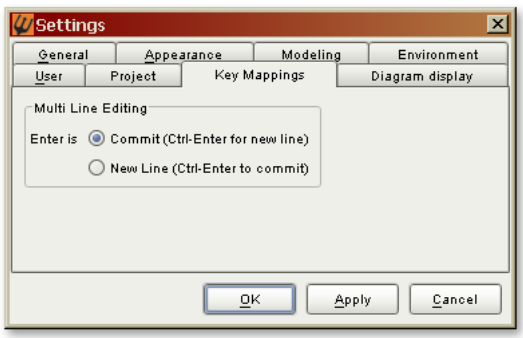
The Project tab contains properties regarding information about the project

- **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.

7.7. Key Mappings

Figure 7-7. The Key Mappings settings tab.

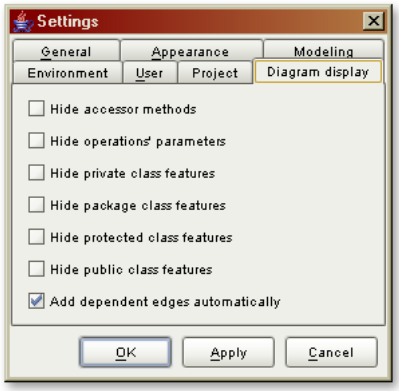


The Key Mappings tab allows you to determine the behavior of particular keys.

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

7.8. Diagram Display

Figure 7-8. 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.

Chapter 8. Model Reference

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

8.1. Views

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

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

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

Chapter 9. Using Models

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

9.1. Creating New Models

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

To create a new model:

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

9.2. Saving and Loading Models

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

Open Standards Support

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

Introducing the .zuml File

The previous version of UML had no standards for storing the graphical information of a diagram. This made exchanging a model between tools very difficult. UML 2.0 has solved this issue with the inclusion of the **Diagram**

Interchange standard, which specifies exactly how graphics are to be stored and rendered. Gentleware was at the forefront of the development of this standard and is therefore uniquely able to implement it in Poseidon.

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

To Save a Model:

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

To Load a Model

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

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

Components Of A .zargo File

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

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

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

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

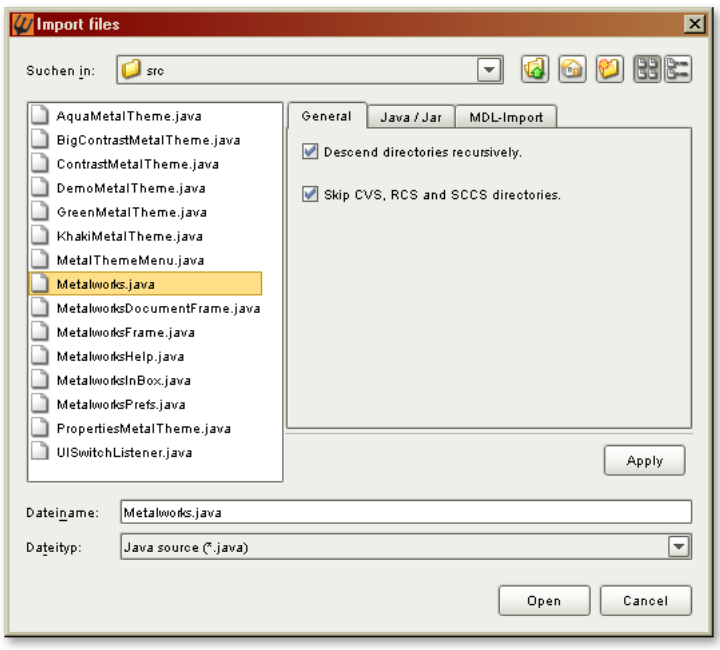
9.3. Importing Files

Poseidon provides a slick dialog to assist with the importation of source code. You can save any changes you make to the settings, such as modeling Java attributes as UML associations, by clicking the 'Apply' button in the lower right section of the dialog.

Several options are available when importing code:

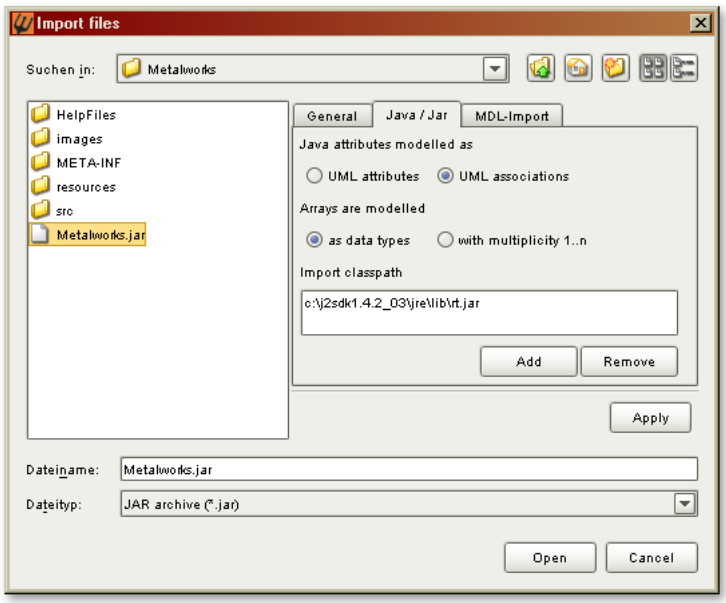
General Tab

- **Descend directories recursively** - Easily add all files below the selected directory
- **Skip CVS and SCCS directories** - Ignore version control files

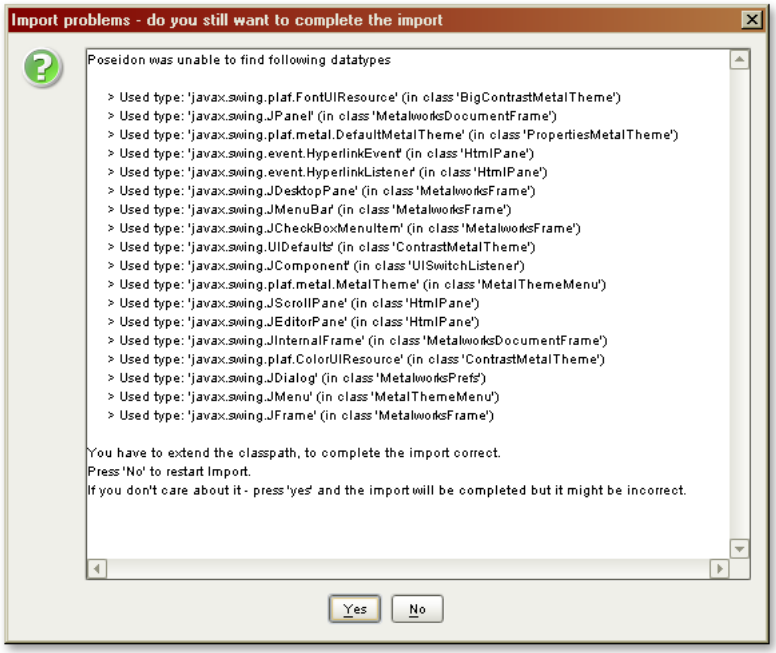


Java Tab

- **Java attributes modeled as** - Select either attributes or associations
- **Arrays are modelled** - Select either as data types or with a multiplicity of 1..n
- **Import Classpath** - Add classpaths by clicking the 'Add' button and navigating to the classpath in the browser that opens automatically.



Should the import fail because needed files have not been located, an error dialog will appear. If you would like to continue without adding to the classpath and create dummy classes instead, click 'Yes'. To return to the previous dialog and add the necessary files to the classpath, click 'No'.

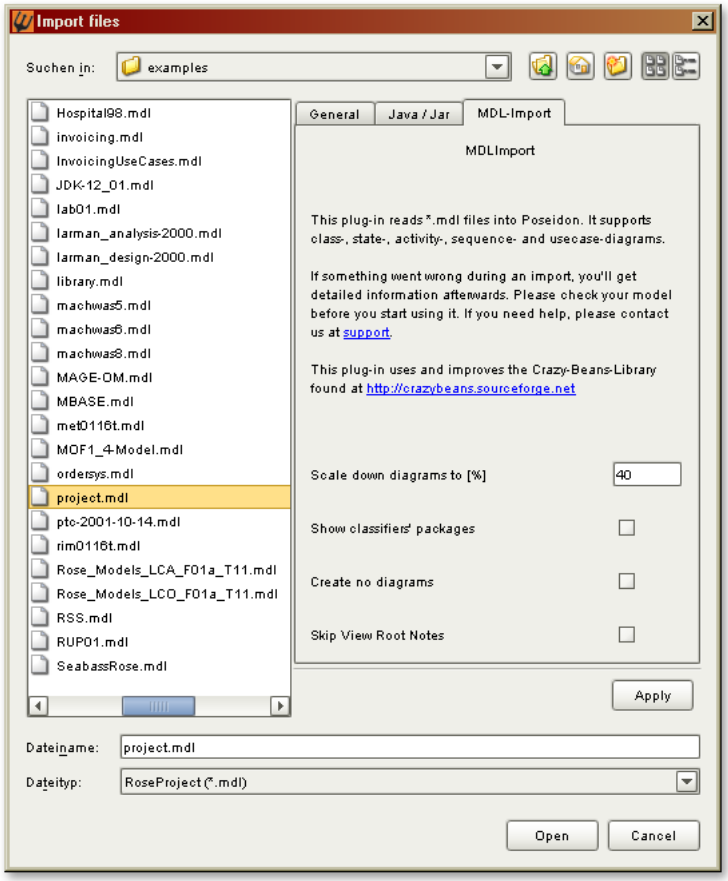


Once the import has begun, a dialog will appear asking if you would like to generate diagrams. A tree of the project is also presented, so that you can choose which packages and classes you would like to include in the diagrams. By default, any packages that contain new classes are automatically selected.

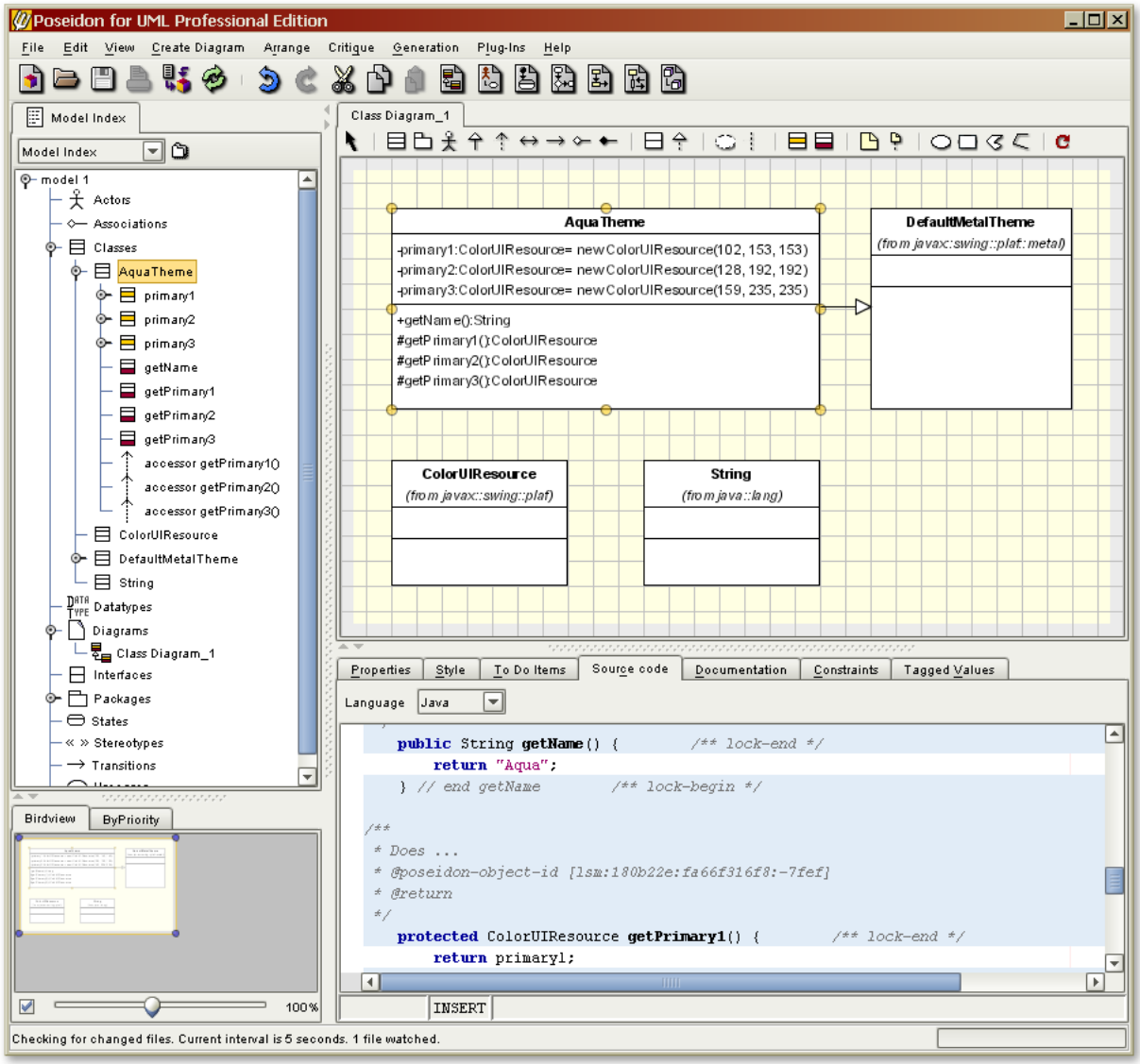
MDL Import Tab

The MDL Import tab is used to import diagrams saved to the Rational Rose format.

- **Scale down diagrams to [%]** - Automatically resizes diagrams for easier readability
- **Show classifiers' packages** - Hides or displays the package of a classifier
- **Create no diagrams** - Import just the elements, or import the elements and create all diagrams
- **Skip View Root Notes** - Does not import the view root notes



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:



9.4. 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

9.5. 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.

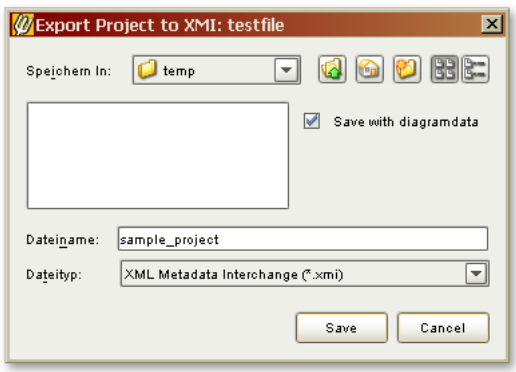
Different tools from other vendors are specialized on generating code. They might not even have a graphical user interface, they simply read in a model and produce code for a specific type of application or platform. Such tools can be connected to

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 recognized standard.

To Export a Project

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

Figure 9-1. Export a project to XMI



9.6. 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 Joint Photographic Experts Group (JPG), CompuServe Graphics Interchange (GIF), Portable Networks Graphics (PNG),

Portable Document Format (PDF), Postscript (PS), Encapsulated Postscript (EPS), Scalable Vector Graphics (SVG), and Windows Meta File (WMF).

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

Export a Diagram to a Graphic File:

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

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. The figures below depict the same diagram, but the first was saved from the Community Edition and the second from the Professional Edition.

Figure 9-2. Watermarked Community Edition diagram graphic

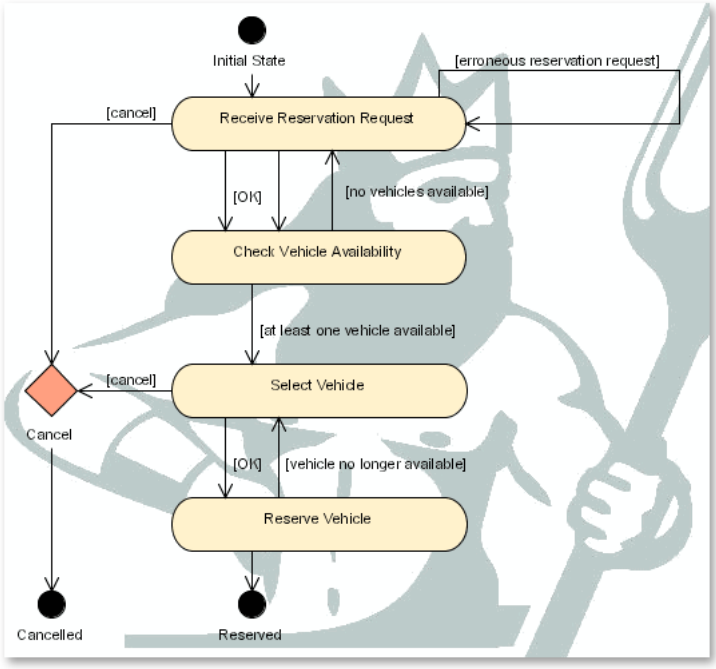
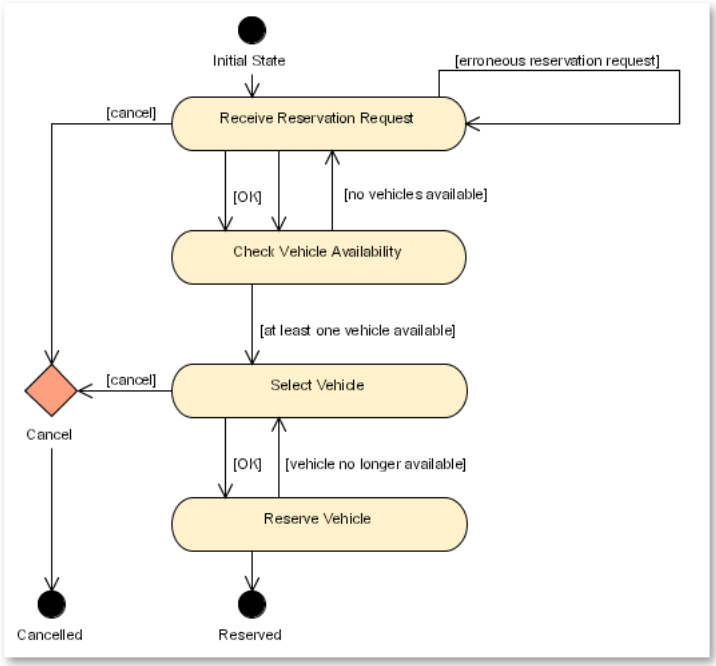


Figure 9-3. Premium Edition diagram graphic without watermark



Printing

You can also directly print diagrams to a printer. 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 10. Diagram Reference

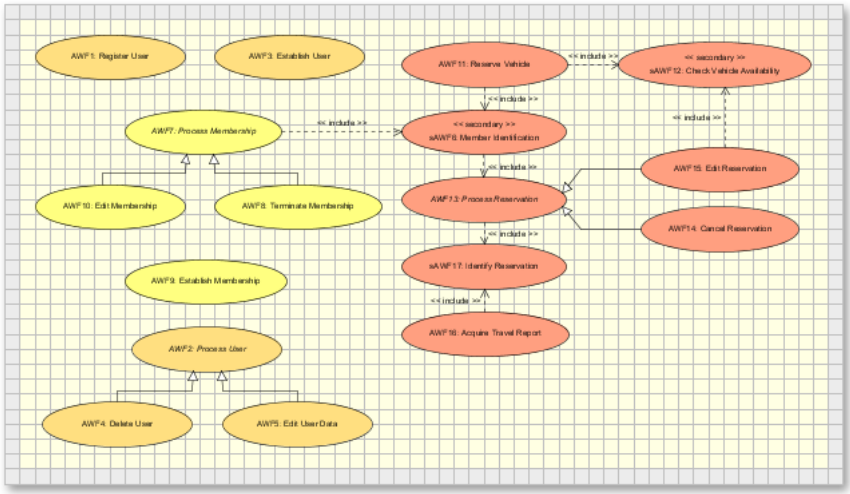
There is a lot to say about when to use which diagram type when developing a design, and what the role of it should be. The different answers to this are referred to as the design process or design method. This document is not intended to describe a concrete design process. Poseidon for UML can be used for any such process. Instead, in this chapter we will look at the various diagram types and how the corresponding model elements are created or edited in Poseidon. For many of these diagrams, a short example has already been given in the default model `Stattauto`, which we looked at in Chapter 6.

10.1. 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, or *use cases*, they are involved in. It is often used in early stages of the design process to collect the intention requirements of a project.

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

Figure 10-1. A Use Case diagram.



10.1.1. 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.

10.1.2. Toolbar

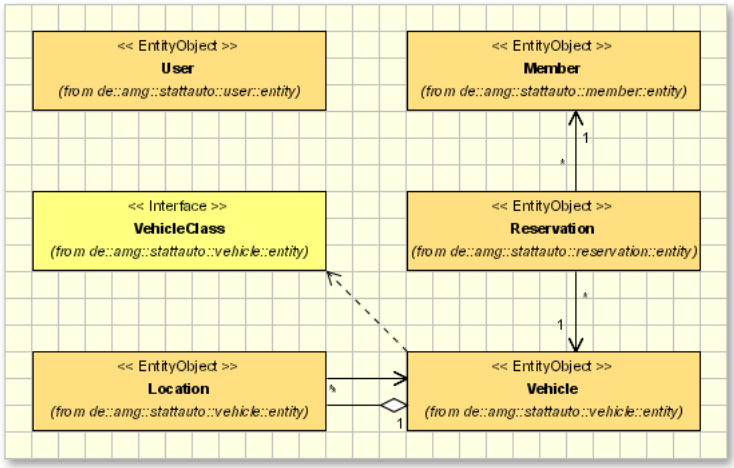
 Select
 Package
 Actor
 Actor as Classifier
 Use Case
 Generalization
 Dependency
 Association
 Directed Association
 Aggregation
 Composition
 Include
 Extend
 Extension Point
 Collaboration
 Classifier Role
 Comment
 Connect Comment to Element
 Text
 Ellipse
 Rectangle
 Polygon
 Polyline
 Repaint

10.2. Class Diagram

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

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

Figure 10-2. 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.

10.2.1. Diagram Elements

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

10.2.2. Toolbar

- Select
- Class
- Package
- Actor
- Actor as Classifier
- Generalization
- Dependency

- Association
- Directed Association
- Aggregation
- Composition
- Association Class
- Interface
- Interface as Circle
- Realization
- Lollipop
- Socket
- Collaboration
- Classifier Role
- Attribute
- Operation
- Comment
- Connect Comment to Element
- Text
- Circle
- Rectangle
- Polygon
- Polyline
- Repaint

10.3. Object Diagram

Object diagrams show classes at the instance level.

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

10.3.1. 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.

10.3.2. Toolbar


Select


Node


Instance of a Node


Component


Port


Instance of a Component


Dependency


Class


Interface As Circle


Lollipop


Socket


Association


Directed Association


Aggregation


Composition


Association Class


Object

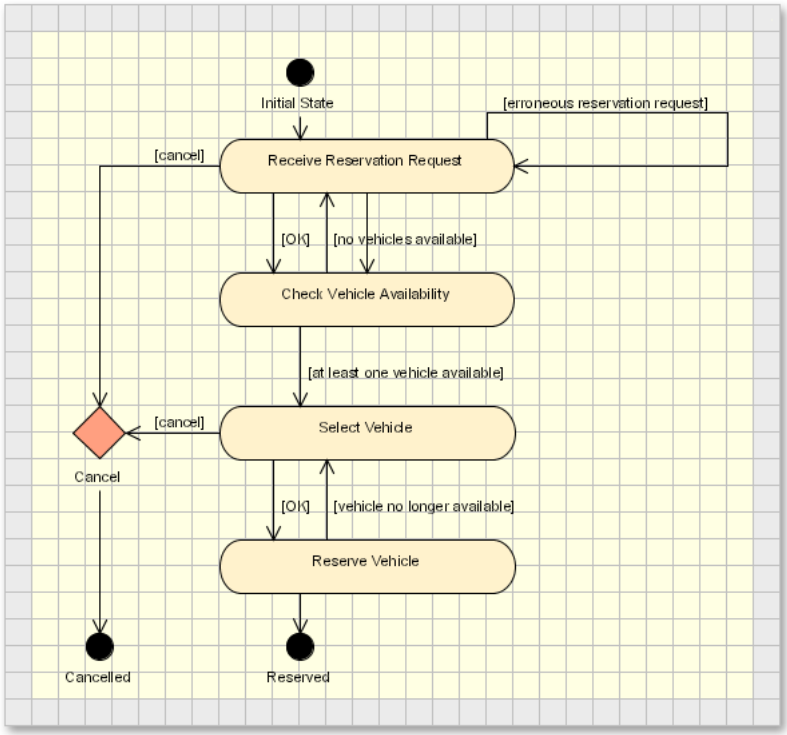
 Link
 Comment
 Connect Comment to Element
 Text
 Circle
 Rectangle
 Polygon
 Polyline
 Repaint

10.4. 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 10-3. An Activity diagram.



10.4.1. 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.

10.4.2. Toolbar


Select


Action State


Object Flow State


Transition


Initial State


Final State


Synchronization State


Branch


Fork


Join


Send Signal


Receive Signal


Note


Connect Comment to Element


Text


Circle


Rectangle


Polygon


Polyline


Repaint

10.5. 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**

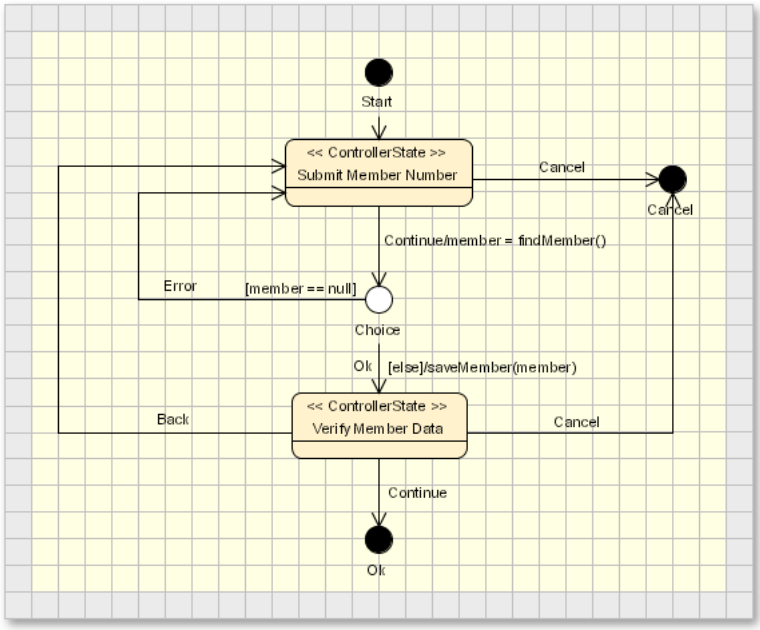
  Composite States make visual use of the second compartment that encloses refinements of the given state. Enclosed states don’t have to have an initial state. Ingoing as well as outgoing transitions might be connected directly to one of them. When the corresponding object is in the composite state, it 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; therefore, they are focused in the second compartment. When the corresponding object enters the concurrent state, all initial sub-states are enabled at once (AND relation).

Figure 10-4. A State diagram



10.5.1. 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.

10.5.2. Toolbar


Select


Simple State


Composite State


Concurrent State


Transition


Initial State


Final State


Synchronization State


Deep History


Shallow History


Choice


Junction


Fork


Join


Comment


Connect Comment to Element


Text


Circle


Rectangle


Polygon


Polyline

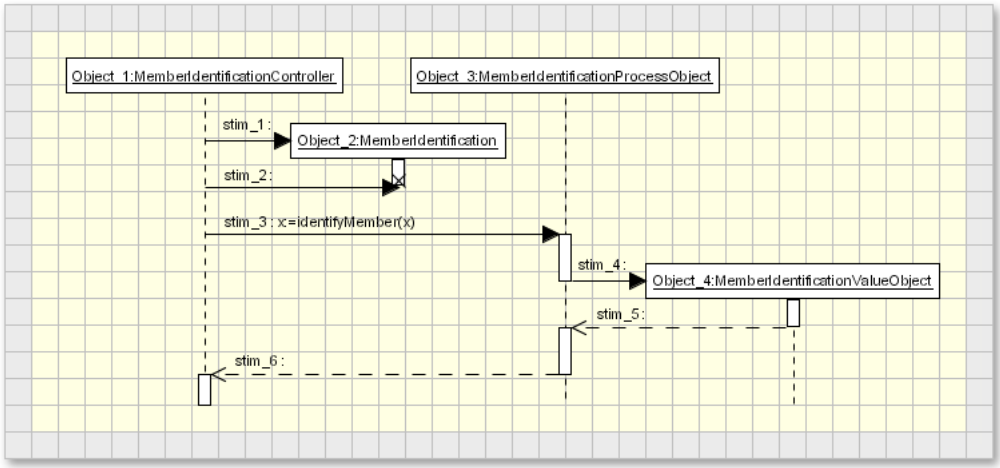

Repaint

10.6. 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 10-5. 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.

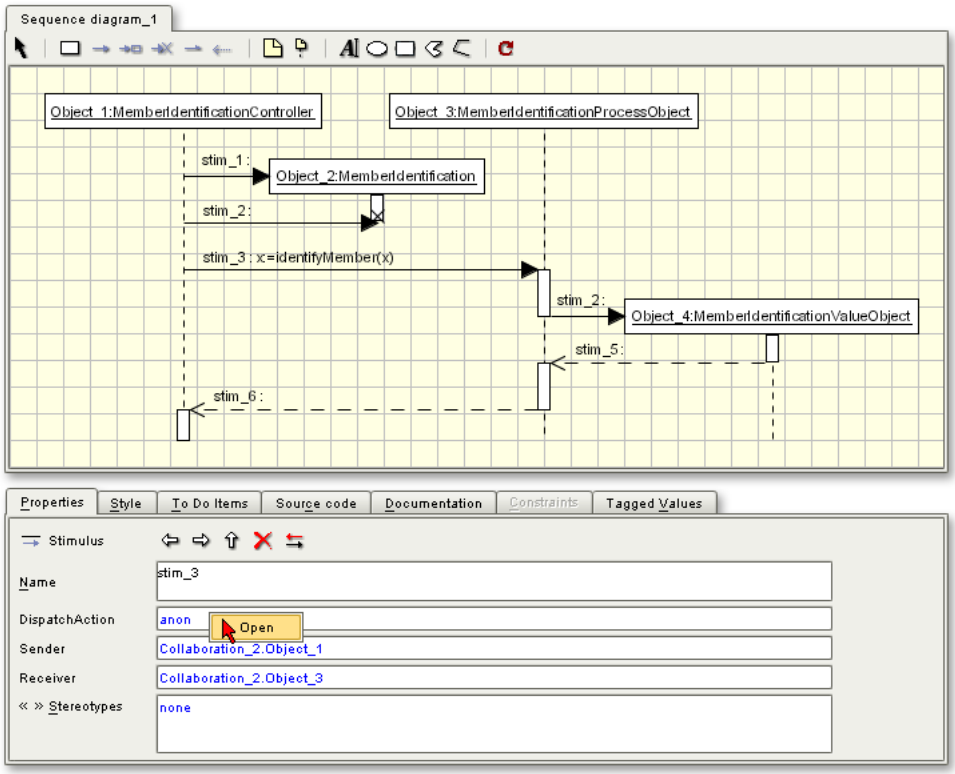
Self messages

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 dropping the stimulus target point on the source object.

Selecting an operation

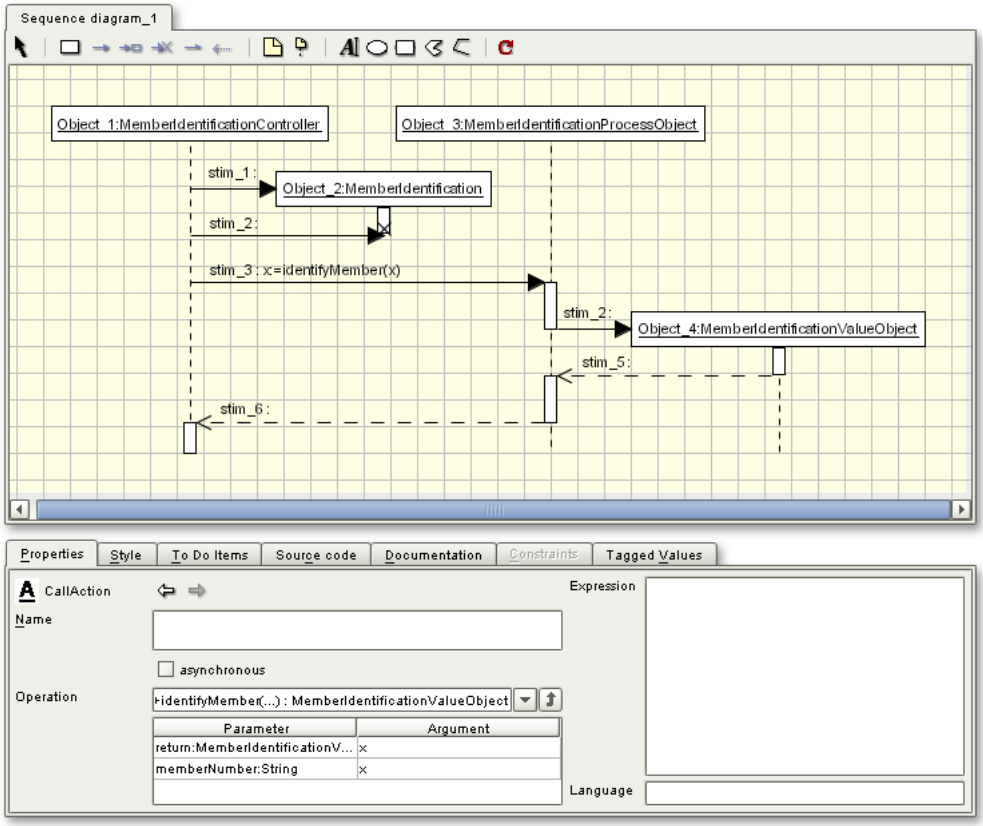
A call stimulus is regarded as a procedure call and can be connected with any operation provided by the receiving object, depending on its type. This is achieved by connecting the stimulus with an action that will cause the class operation to be called. The following two figures show an example for selecting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

Figure 10-6. 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 10-7. 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.

10.6.1. 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.

10.6.2. Toolbar

 Select
 Object
 Call Stimulus
 Create Stimulus
 Destroy Stimulus
 Send Stimulus
 Return Stimulus
 Comment
 Connect Comment to Element
 Text
 Circle
 Rectangle
 Polygon

- Polyline
- Repaint

10.7. Collaboration Diagrams

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

10.7.1. 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.

10.7.2. Toolbar

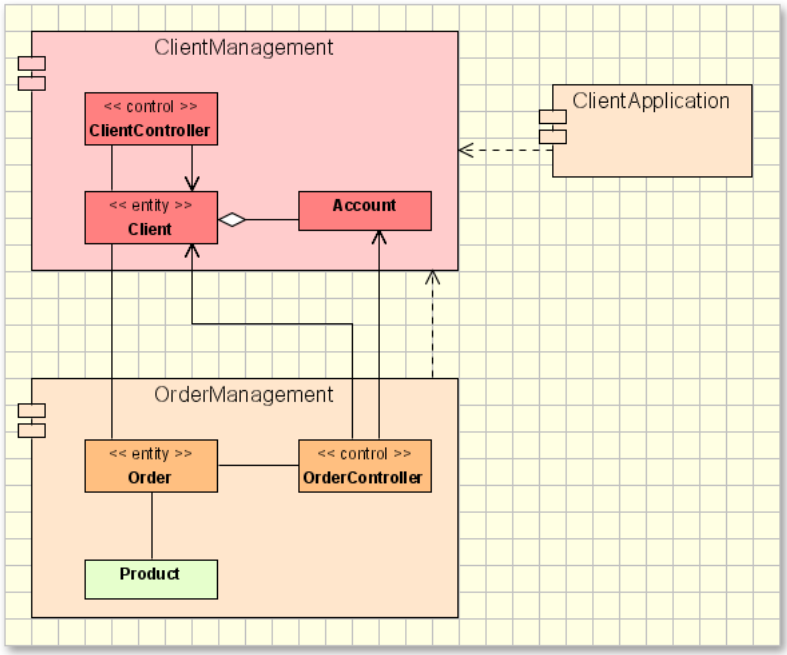
- Select
- Object
- Link
- Call Stimulus
- Create Stimulus
- Destroy Stimulus
- Send Stimulus
- Return Stimulus

- Comment
- Connect Comment to Element
- Text
- Circle
- Rectangle
- Polygon
- Polyline
- Repaint

10.8. 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; therefore, the corresponding model elements are explained in that section.

Figure 10-8. A Component diagram.



10.8.1. 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.

10.8.2. Toolbar


Select


Node


Instance of a Node


Component


Port


Instance of a Component


Dependency


Class


Interface As Circle


Lollipop


Socket


Association


Directed Association


Aggregation


Composition


Association Class

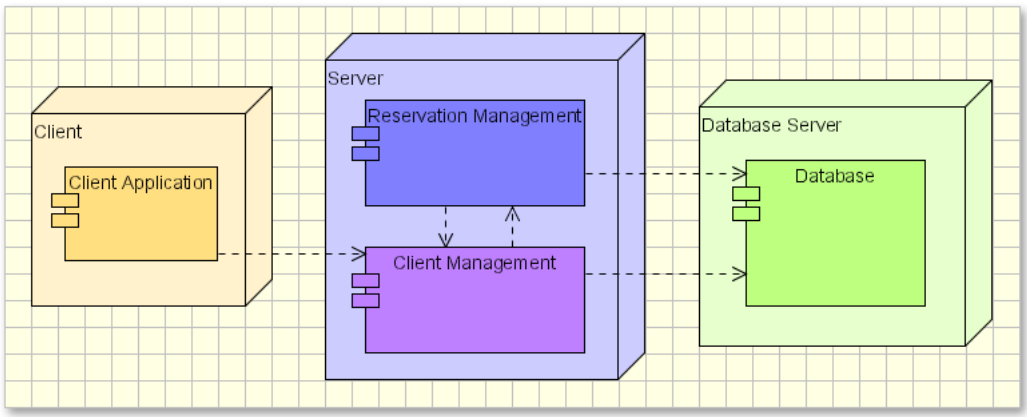

Object

- Link
- Comment
- Connect Comment to Element
- Text
- Circle
- Rectangle
- Polygon
- Polyline
- Repaint

10.9. 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 10-9. A Deployment diagram.



10.9.1. 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.

10.9.2. Toolbar


Select


Node


Instance of a Node


Component


Port


Instance of a Component


Dependency


Class


Interface As Circle


Lollipop


Socket


Association


Directed Association


Aggregation


Composition


Association Class


Object


Link


Comment


Connect Comment to Element


Text

 Circle

 Rectangle

 Polygon

 Polyline

 Repaint

Chapter 11. Using Diagrams

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

11.1. Creating New Diagrams

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

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

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

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

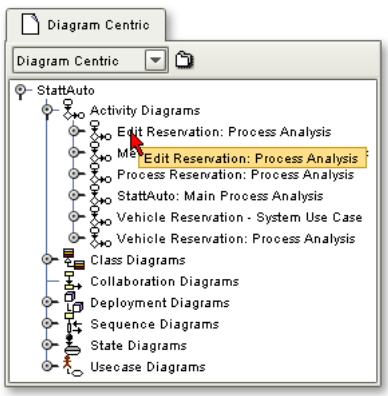
New diagrams can be created in several ways:

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

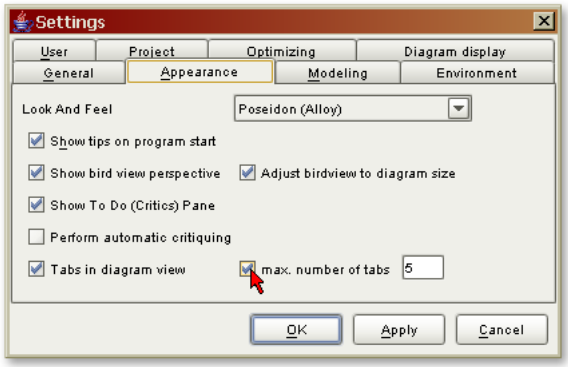
- Class Diagram - *Ctrl-L*
- Collaboration Diagram - *Ctrl-B*
- Deployment / Object / Component Diagram - *Ctrl-D*
- Sequence Diagram - *Ctrl-Q*
- State Diagram - *Ctrl-T*
- Activity Diagram - *Ctrl-Y*
- Use Case Diagram - *Ctrl-U*

11.2. Opening Diagrams

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



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



11.3. Viewing Diagrams

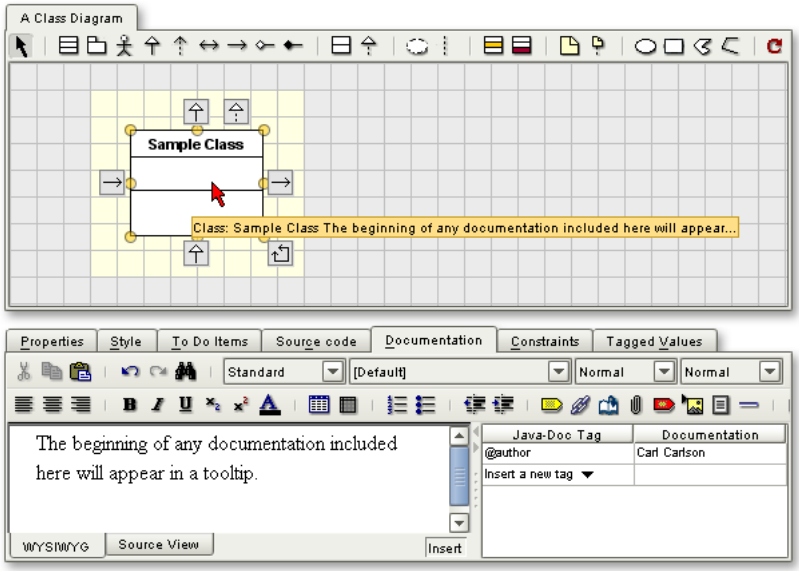
Viewing a single diagram is easy. You simply select the diagram you wish to view from the Navigation pane and the chosen diagram is then displayed in the Diagram pane. Much more interesting are the relationships between elements and how specific elements are represented in different diagrams. Each element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

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

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

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

Figure 11-1. Tooltip displaying documentation



The model can be changed directly from the diagram. For example, double-click on the class name of any class in a class diagram. The text field now slightly changes its look and becomes editable. Changing the class name here perpetuates the name in the model everywhere this model element is used. Most name fields accept multiline text; therefore, the Enter key will add a newline. To commit a change to such a field, use Ctrl-Enter in place of Enter, or simply click elsewhere in the application.

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

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

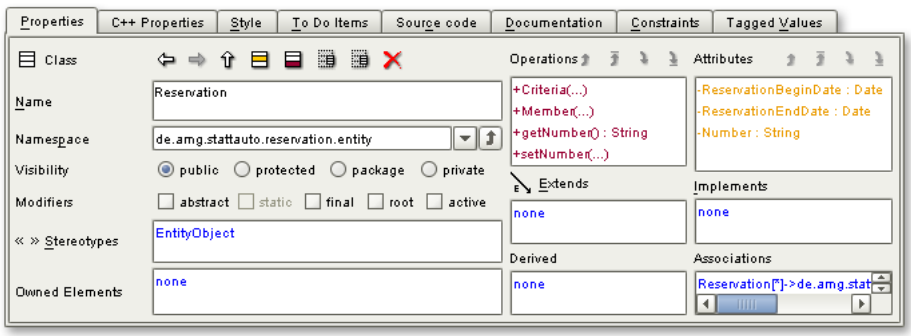
There are, however, some details that can conceptually not be changed in place or where it makes more sense to provide a special graphical user interface. It is for these purposes that the Details pane is used.

11.3.1. Details Pane

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

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

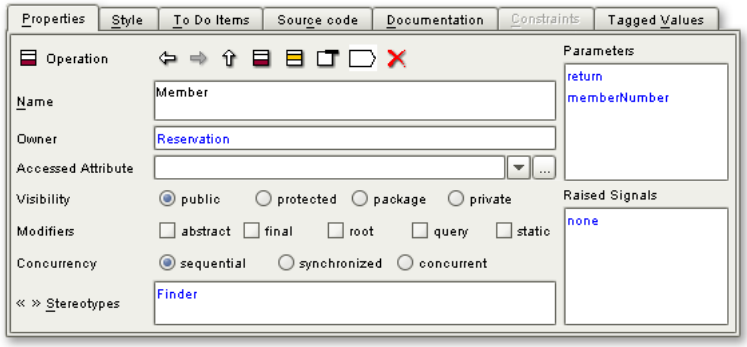
Figure 11-2. Properties tab displaying class ’Reservation’



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

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

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



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

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

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

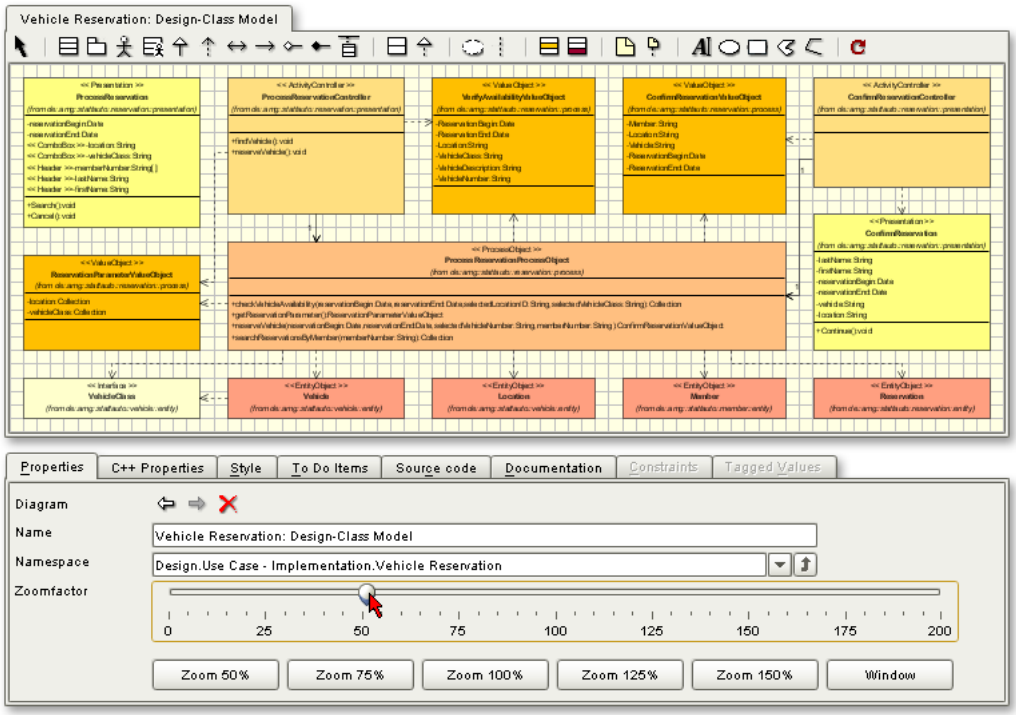
11.3.2. Zooming

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

There are several ways to zoom in and out:

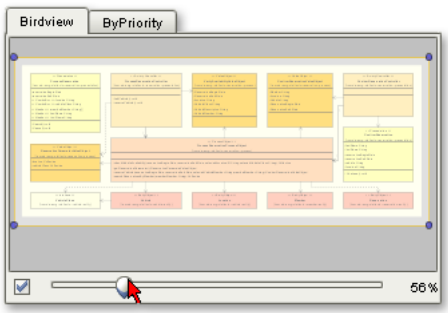
- Change the zoom factor of a diagram by clicking on an empty space in the diagram and using the slider (or the buttons with predefined zoom values) on the Properties tab in the Details pane.

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



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

  Figure 11-5. Zooming from the Birdview tab

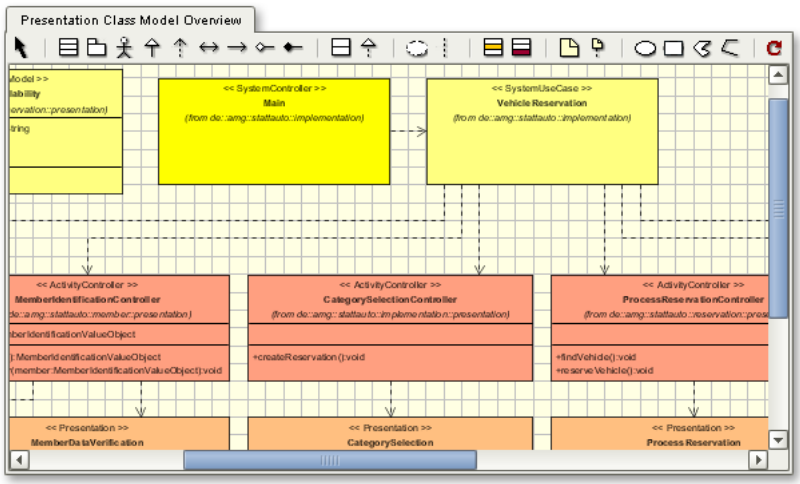


- Hold down the Ctrl key and use the mouse wheel to zoom in and out.

- Choose a zoom factor from the menu (View->Zoom).

11.3.3. Scrolling

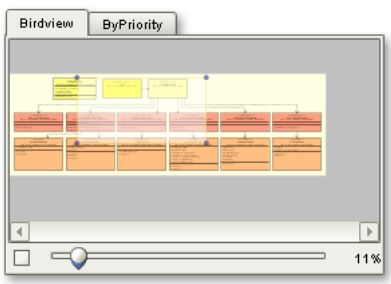
As diagrams get bigger and bigger, the scroll bars become more and more useful.



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

11.3.4. Birdview Tab

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



11.4. Navigation

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

11.4.1. Navigation Pane

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

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

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

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

11.4.2. Details Pane

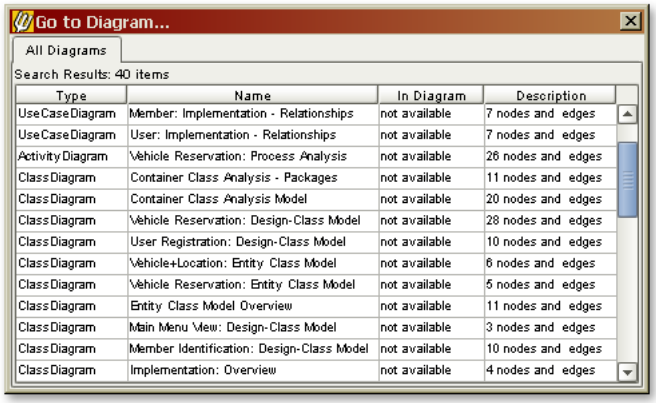
The Details pane contains numerous tabs, and it is the Properties tab that provides the means for drill-down navigation. Each element contains or is a part of other elements, as an operation is both a part of a class and contains parameters, for example. Moving between these elements is as simple as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

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

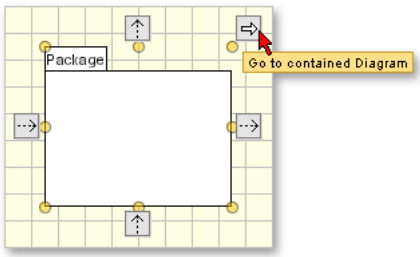
11.4.3. Diagram Pane

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

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



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



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

11.5. 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.

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.

11.5.1. 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.

Another 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.

Note that Drag and Drop is not currently available in Activity, Collaboration, Sequence, and State diagrams.

11.5.2. 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.

11.5.3. Layout Functions

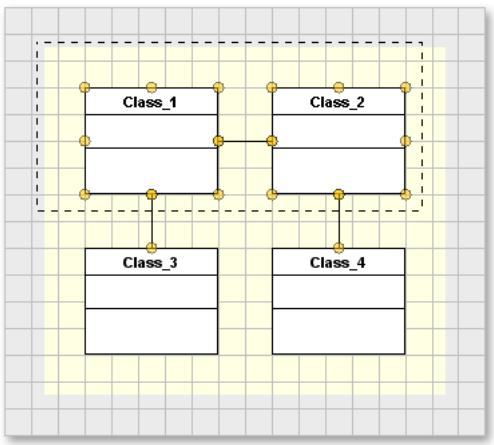
You already know that you can layout your diagram by using the select tool. But there are a number of other ways to rearrange your diagrams.

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 dashed line appears and all elements that are partially or wholly enclosed in it will be selected.

Figure 11-6. Selecting multiple elements with the mouse.



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

Elements can be moved along invisible 'rails' by holding the Ctrl key while dragging the elements. This limits the movement to the X and Y axes of the original element. If multiple elements are selected, the center of the selected elements is

used as the origin of the rails. This means that an element may or may not reside at the origin.

Arrange Elements

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

The **Align Tools** include:

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

The **Distribute Tools** include:

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

The **Size Tools** include:

- **Greatest Current Width and Height** - Uniformly resizes selected elements so that each is the size of the largest selected element.

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

The **Ordering Tools** include:

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

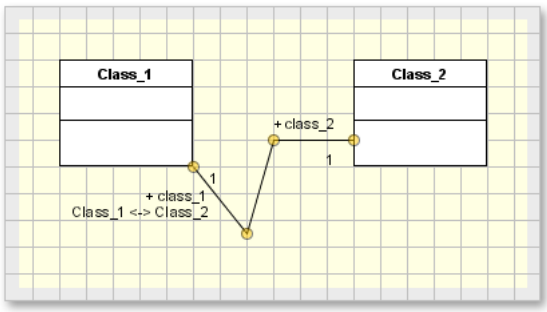
These groups of tools may be used alone or in conjunction with another tool of a different type.

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

Changing the Shape of Relationships

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

Figure 11-7. Adding waypoints.

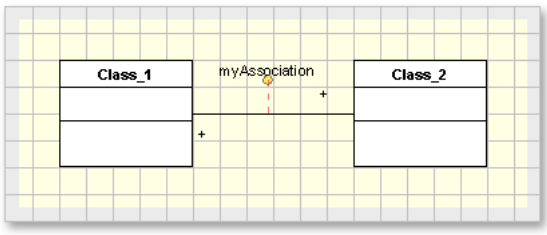


In Poseidon for UML version 2.0, waypoints have changed from blue boxes to yellow circles. Despite the change in appearance, they function in the same way.

Moving Adornments

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

Figure 11-8. Moving adornments.



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

11.5.4. Undo/Redo

Poseidon maintains a history of changes made to the model. The undo and redo buttons step forwards and backwards through this history.

Undo and redo is limited to three steps in the Community Edition.

11.5.5. Non-UML Additions

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

11.5.5.1. Select

The first tool in the toolbar is called the 'select' tool, and is the default active tool. It is used to select, move, and scale diagram elements, as well as modify the element directly from the diagram. When an element has been selected and is now the current active element, it will appear with yellow circles (called 'handles') surrounding it.

A brief list of functions:

- **Select an element** - Click on the desired element.
- **Move an element** - Click and hold the mouse button inside the element, then drag the element to its new location.
- **Resize an element** - Click and hold the mouse button on an element handle, then drag the handle.
- **Edit an element inline** - Double-click on a text element to activate the text edit box.

Try it Yourself - *Resize an Element*

1. Select the ▤ `Client` class from a diagram.
2. Small round yellow handles appear on the corners of the element.
3. Click and hold the mouse button on one of these handles and drag it around the diagram to resize the class.

11.5.5.2. Comments

Sometimes a diagram requires a bit of extra explanation. This information is not a part of the final code, yet it helps the viewer better understand the diagram. This information can be included in a comment element. Comments are extra notes that are included and displayed in a diagram. These comments can be added to almost any element including other comments, or they can stand alone in the diagram. Comments cannot be added to relationships, transitions, or shapes created with drawing tools in any diagram, and objects in sequence diagrams.

Comments are ignored by the code generator; therefore they are never seen in the code output. They are likewise never seen in the Navigation pane.

To add a comment to a diagram:

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

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

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

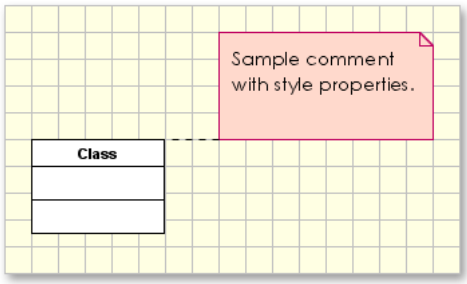
To connect a comment to an element using the rapid buttons:

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

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

Just as with any other element, notes can be resized with its handles and the color can be changed through the style panel of the Details pane. This makes it easy to introduce a color-coding scheme to diagram notations.

Figure 11-9. A new comment



11.5.5.3. Drawing Tools

The set of tools which appears at the end of the toolbar are for general drawing purposes. With these tools you can add other graphical elements such as shapes to your diagram. You should keep in mind that, although useful sometimes, these graphics are not part of UML; therefore, they don't show up in the model tree in the Navigation pane.

The Drawing Tools:

- **Text** - Click in the diagram area and begin typing to create a text object.
- **Circle** - Click in the diagram area and drag the mouse to create an ellipse.
- **Rectangle** - Click in the diagram area and drag the mouse to create a rectangle.
- **Polygon** - Click once everywhere the polygon is to have a corner. Double-click the last corner to close and render the polygon.
- **Polyline** - Click in the diagram area and to create a waypoint. Click again to create another waypoint and a line between them. A connected line can be added by lclicking to add a third waypoint. Double-click the last waypoint to cease the addition of lines.

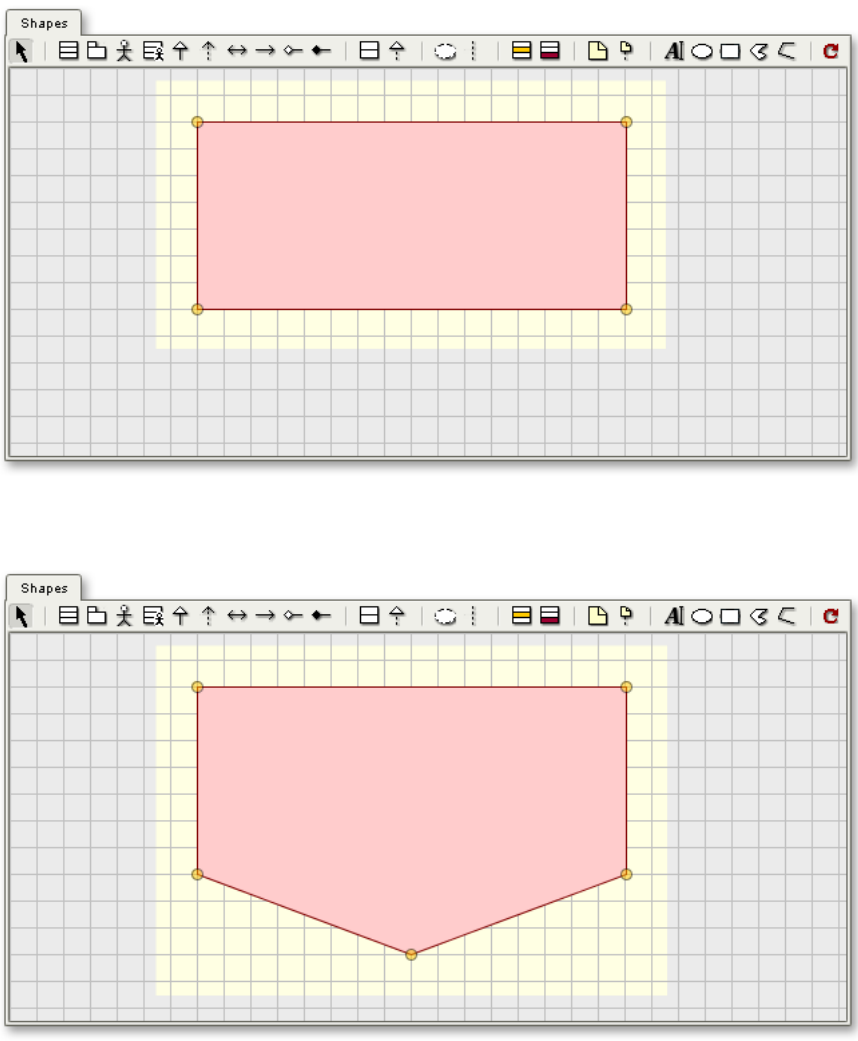
11.5.5.4. Toggle Between Editing Modes

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

The first is a resize mode, which allows you to change the size of the shape by dragging the handles (gold circles) that surround the shape. Dragging one of the corner handles enlarges and shrinks the shape without changing its proportions. Dragging the side handles expand and compress the shape.

The second editing mode is available for all shapes except circles. It allows you to add, remove, and move waypoints to change the shape of the element. For example, you can create a rectangle, double-click on it, and then add a waypoint to create a new polygon.

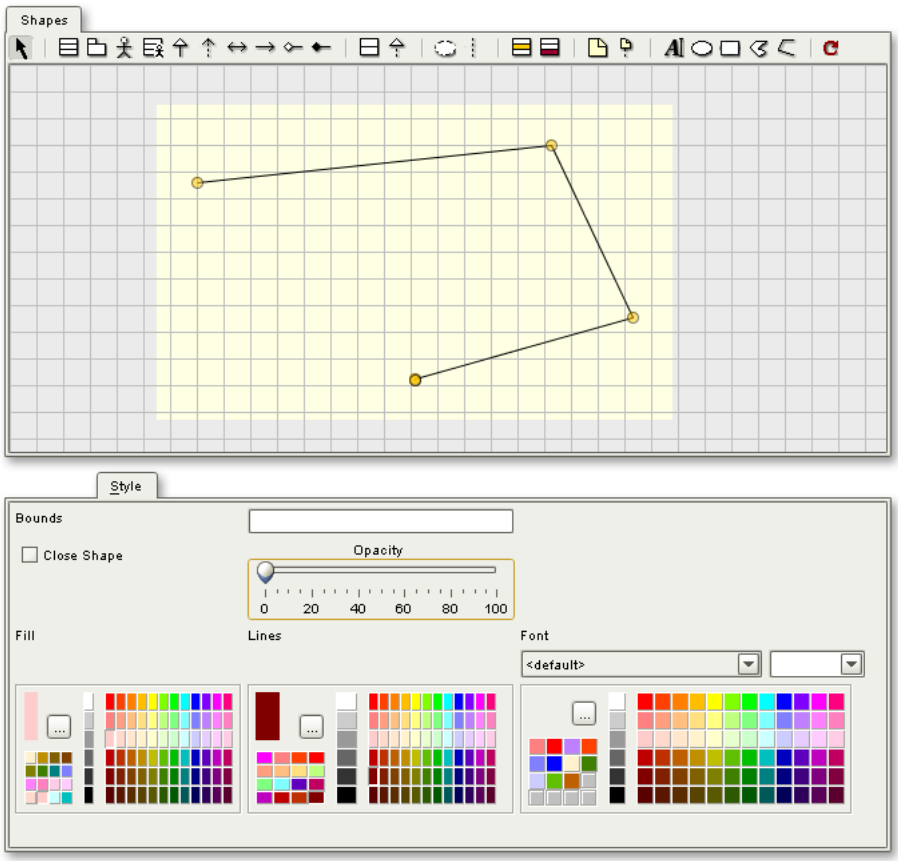
Figure 11-10. Add a waypoint to a rectangle

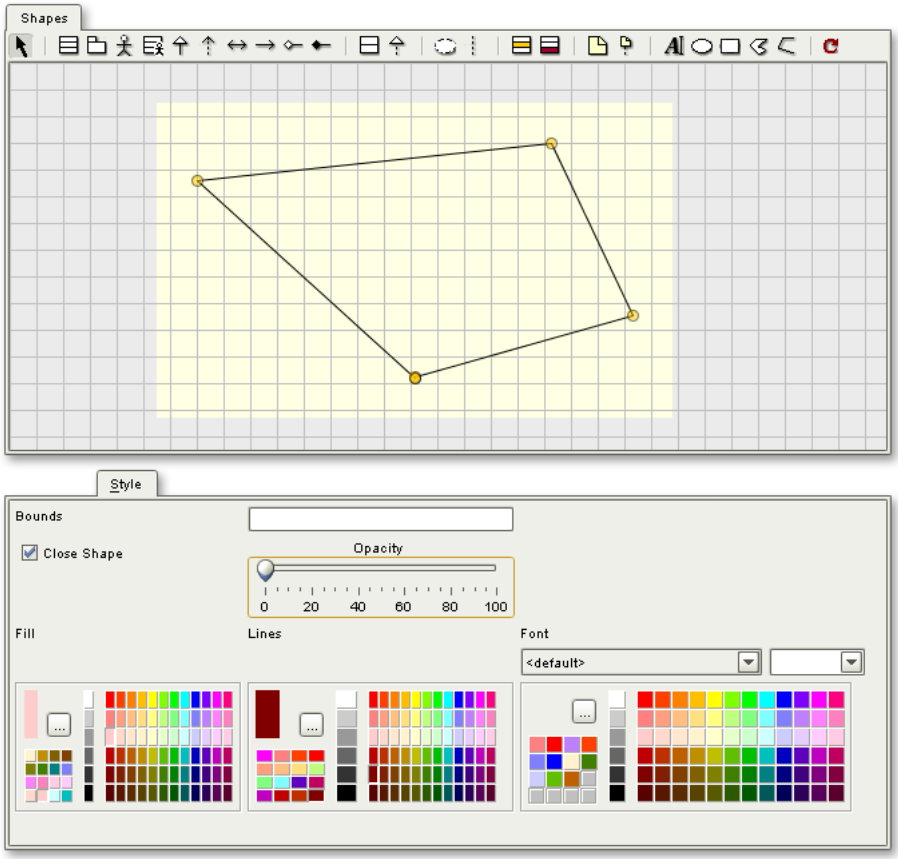


11.5.5.5. Close Shape

Once a shape has been drawn with the line tool, it is possible to close the shape automatically to create a polygon. Select the shape and open the 'style' tab of the Details pane. Check the box titled, 'Close Shape'. The shape can be reopened by unchecking the same box.

Figure 11-11. Open and closed lines

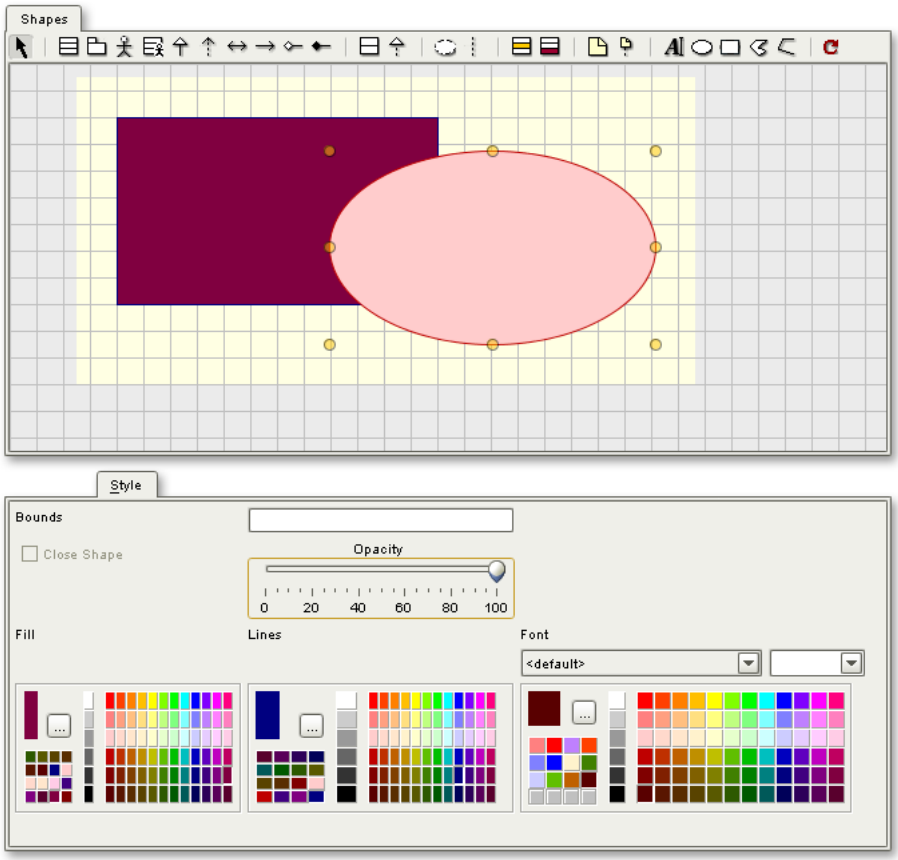


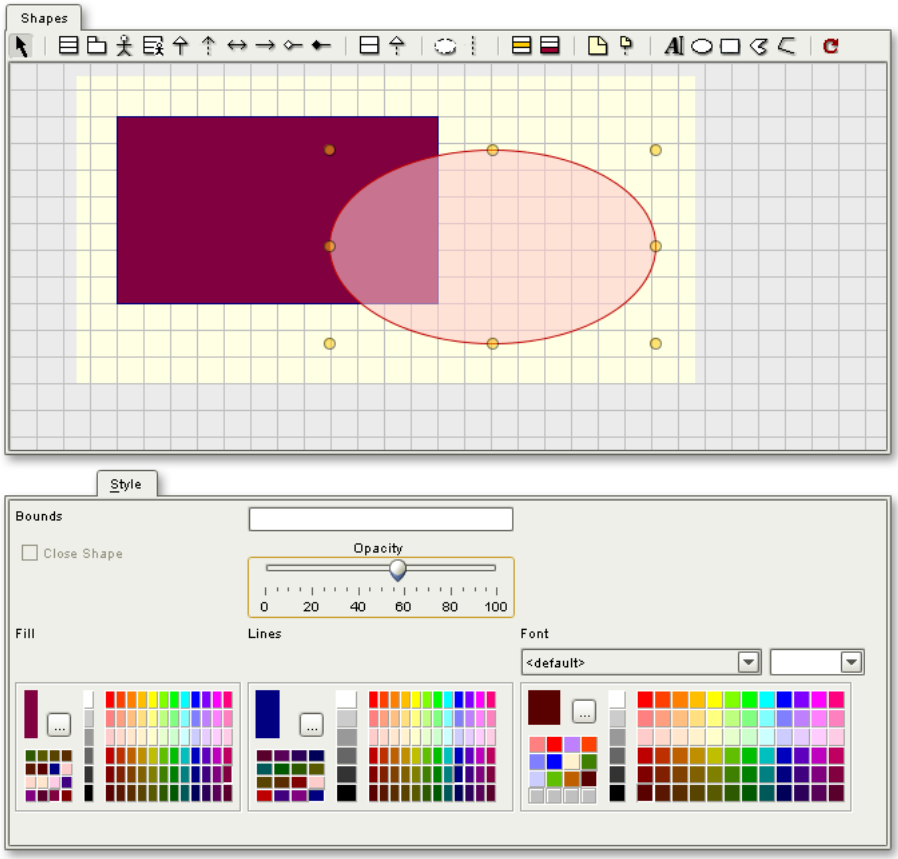


11.5.5.6. Opacity

Fill colors can be applied from the Style tab of the Details pane. It may be advantageous to change the opacity of this fill at times. Fortunately, this is very easily accomplished. Simply select the figure that will have a different opacity and use the slider bar within the ’style’ tab of the Details pane.

Figure 11-12. Changing opacity





11.5.5.7. Waypoints

Once a line or polygon has been created, the shape can be altered by creating and moving a waypoint, much in the same way that connections between elements can be edited. Click on the perimeter and move the resulting gold circle to create an 'elbow'. In this same vein, waypoints can be deleted by selecting and dragging them over an existing waypoint or endpoint.

11.5.5.8. Diagram-specific Tools

The rest of the tools in the toolbar are specific to the current diagram type. They allow the creation of diagram elements and operate similarly to a stamp. With a single click on the icon you get a handle to create one corresponding diagram element. If you double-click, the tool stays selected and you can create a number of diagram elements, one after the other. The cursor changes to a hair cross with which

you can select the position of the new element. To disable this feature just click on the ’select’ tool.

Some tools are only available in a certain context. In Class Diagrams, the tools to create a new attribute or a new operation are only available when a class is selected. Select the desired class and click on the appropriate button to create a new attribute or operation for your class.

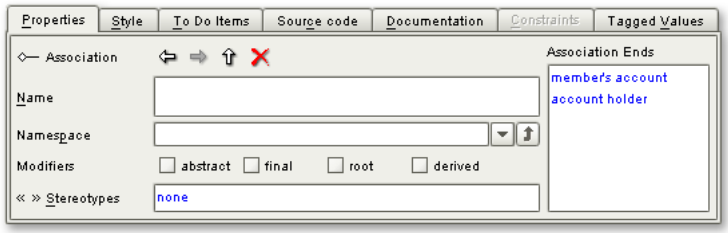
The individual tools are covered in detail in the chapter titled, ’A Walk Through the Diagrams’.

Chapter 12. Element Reference

12.1. Relationships

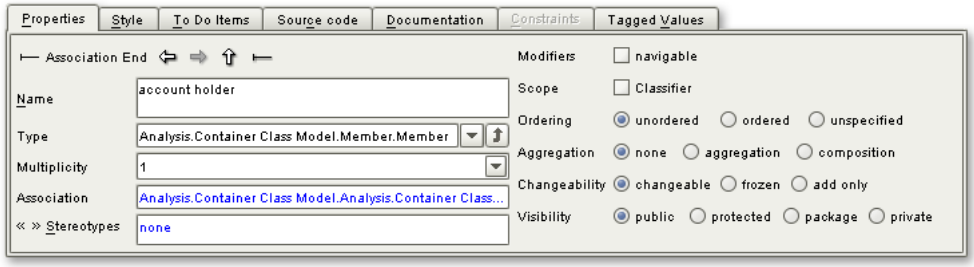
Relationships are very important in UML. They have properties in and of themselves, as well as consisting of other model elements. Every association has two ⊢ association ends that are model elements in their own right, as defined in the UML specification. Figure 9–4 shows the Properties tab for an association, in this case between Account and Member. Notice that there is no stereotype or name for this association, but they could conceivably exist. Also note that the association is part of the `Design.Use Case - Implementation.User Registration` namespace.

Figure 12-1. Properties tab for an association.



An association end can also be given a name, and like an association it doesn’t require one. If an association end does not have its own name, the class name at that end of the association is displayed. Look to the left hand side of Figure 9–4. In this case, both association ends have been named. Like hypertext, they link to the association end properties, not to the class properties.

Figure 12-2. Properties tab for an association end.



Associations can be specialized to an ◇– aggregation or a ◆–composition. To do this, navigate to one of the association ends and change the aggregation type from none to either aggregation or composition. They can also be created directly from the toolbar, using the ◇– 'Create Aggregation' button or the ◆– 'Create Composition' button.

12.1.1. Types of Relationships

- Generalization
- Dependency
- Association
- Directed Association
- Aggregation
- Composition

12.1.2. Navigability

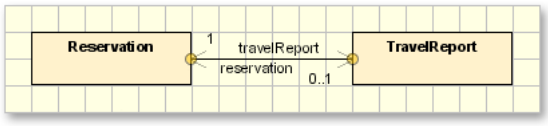
The navigability of associations is similarly changed, using the association ends properties. The check box titled 'navigable', when checked, means *towards the class that this association end points to*. This is a bit counter-intuitive at first, so further explanation is warranted:

Associations can be modeled as navigable in both directions, navigable in only in one direction, or without any navigability. In most cases, navigability is indicated by arrows in the diagrams. The one exception is the default association, an association which is navigable in both directions. In this case arrows are omitted. The navigability of an association occurs at the beginning of the arrow, not at the end.

You can easily navigate to the opposite association end using the navigation button in the Properties tab.

When you first create an association, it is navigable in both directions. The UML standard requires that both arrows are hidden in this case, so it looks just the same as an association with no arrows at all. To distinguish these two cases, the arrows of both its ends show up in grey, if necessary, when you select an association.

Figure 12-3. Highlight hints for associations.



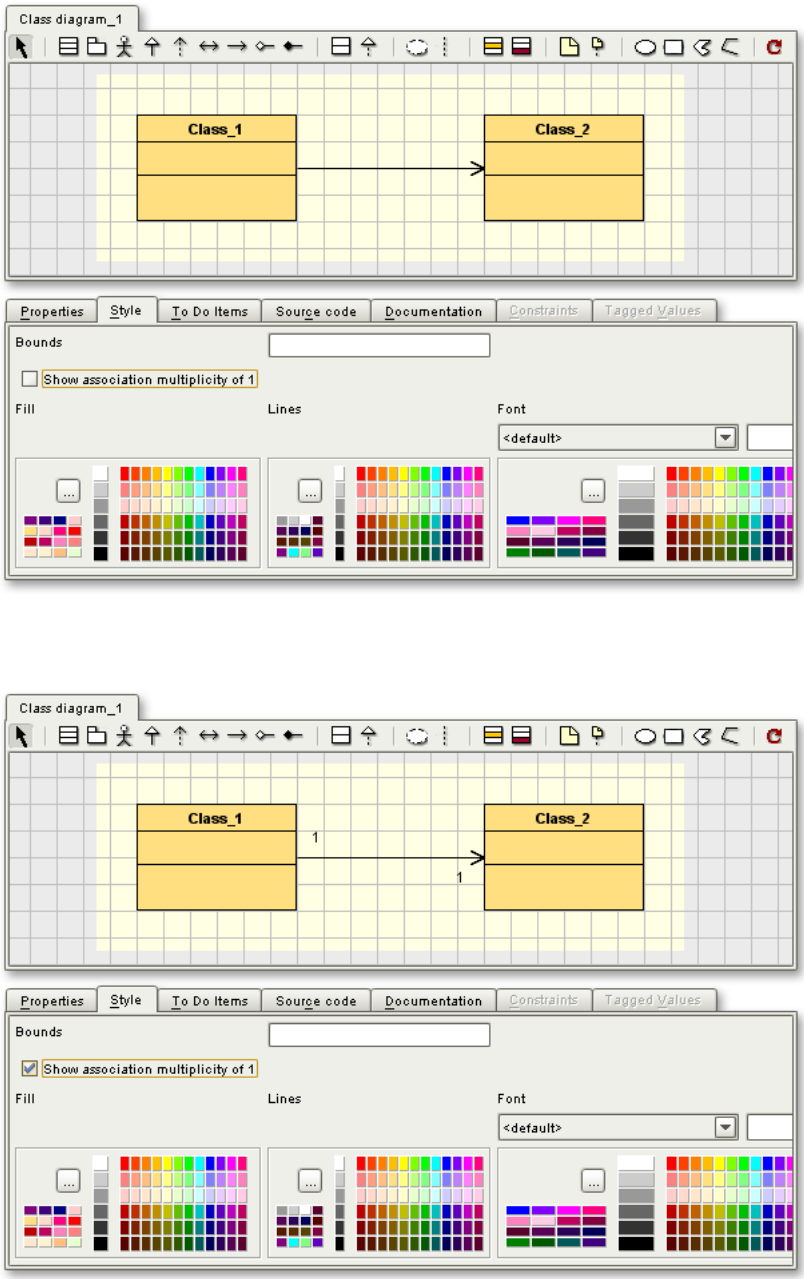
12.1.3. Hiding and Displaying Multiplicity of 1

When a multiplicity of 1 is set, some UML authors recommend hiding the 1, whereas others like to show the 1. To suit your needs, you can set the single multiplicity to be displayed or hidden. This can only be set diagram-wide in order to avoid confusion.

To change the display setting for single multiplicity:

1. Select the diagram where you want to change the setting.
2. Go to the Style tab.
3. Activate or deactivate the ’Show association multiplicity of 1’ check box.

Figure 12-4. Style tab with multiplicity unset and set

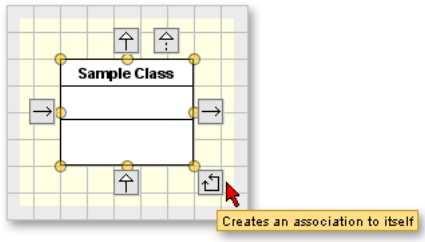


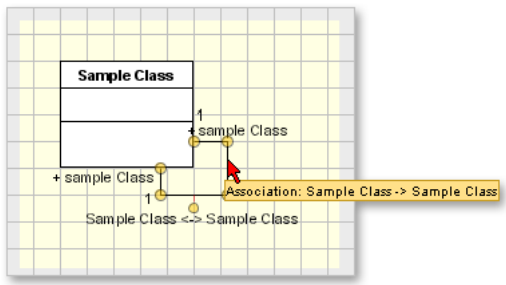
12.1.4. Self-Associations

Associations usually connect two different classes. But they can also be drawn from

one class to itself. Simply use the rapid button in the lower right corner of the class.

Figure 12-5. The rapid button for self-associations





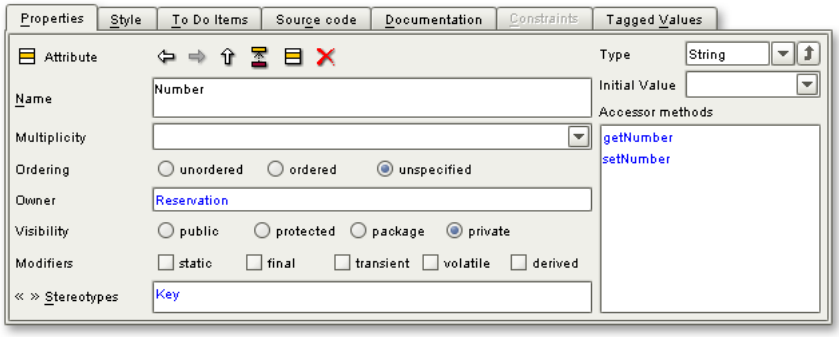
12.2. Classes

Classes are the core elements in an object-oriented model. They are represented by rectangles divided into three sections that display class information in the topmost portion, attribute information in the second, and operation information on the bottom. These compartments can be hidden via the context menu, as can package and stereotype information.

12.2.1. Attributes

Every class can have attributes that express some of its properties. In UML, every attribute has a name and a type. The type can be any other DataType, Class or Interface that is in the model. You can select the type from a combo box of all available types. If the type you need is not in the list, you can create it elsewhere, and then select it from the list.

Figure 12-6. Properties of an attribute.



Attribute Properties

- **Multiplicity** - The multiplicity field determines how many references the class has to this attribute.
- **Ordering** - Can be set to ordered, unordered, or unspecified (default).
- **Owner** - Specifies to which element the attribute belongs.
- **Visibility** - The visibility of an attribute expresses which other classes can access it. The options are:

Public	+	Accessible to all objects
Protected	#	Accessible to instances of the implementing class and its subclasses.
Private	-	Accessible to instances of the implementing class.
Package	~	Accessible to instances of classes within the same package.

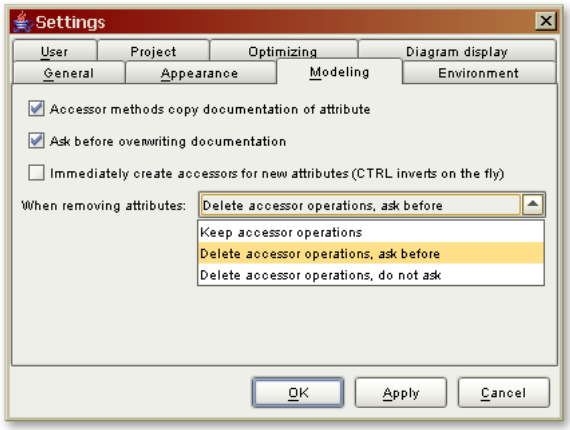
- **Modifiers** - You can also set whether the attribute is write-only by checking the **final** check box. An attribute can also be **static** (class scope instead of instance scope) or **transient** or **volatile**. An initial value can be given as a Java expression.
- **Stereotype** - Stereotypes can be added an removed via a dialog that is accessed by right-clicking the stereotype field and clicking ’Open’.

- **Type** - The Type dropdown sets the type of the attribute.
- **Initial Value** - This optional field gives an initial value to the 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.

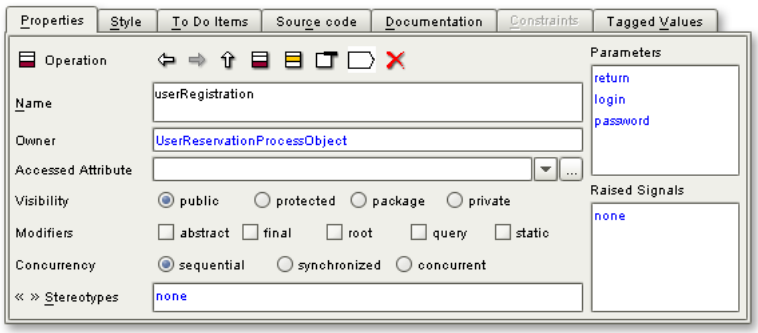
  If you prefer to have only some accessor methods, right-click on one of the entries in the list 'Accessor Methods'. Then, select 'Delete'. The operation will be removed from the list; thus, it will not be marked as accessor any more. You will be asked if you want to remove this relation only or if you want to delete the operation completely. Whether this dialog appears or not is determined by your settings, which can be accessed in Edit-Settings. The 'Modeling' tab is displayed in Figure 9–9. Notice the 'When removing attributes' drop-down list. The default option is to ask before deleting the accessor operation. You can also choose to always keep the operations or to delete them directly without asking. This depends on your preference and your style of working with accessor methods.

Figure 12-7. 'Remove Attributes' setting



12.2.2. Operations

Figure 12-8. Properties of an operation.



For every operation, you can set several UML properties. Among them are visibility, scope (class or instance), and concurrency (with designators like sequential, synchronized, and concurrent). You can set an operation to be final, be a query, abstract, or a root method (with no parent).

The two lists on the right of the Properties tab are used to refine the operation's signature. In the list of parameters, the first parameter return is always there. This

defines a return type for this operation. Similarly, you can add parameters that may be given a name, a type and the modifier ’final’. The final modifier is a special case that we introduced to handle Java.

You can define a constructor by setting the stereotype `<<create>>`. The code generation generates a constructor signature.

Operation Properties

- **Name** - This field gives a name to the operation. This field can also be changed directly in the diagram.
- **Owner** - The ’Owner’ field specifies the element to which the operation belongs.
- **Accessed Attribute** - In the field ’Accessed Attribute’, you can define whether this operation should be marked as an accessor method for an attribute. The primary use of accessor methods is to modify the attribute internally and to control access to the modifications externally. You can choose a modified attribute (if the operation is an accessor method created by Poseidon, an attribute is already selected) or select none if you want to decouple an accessor method from its attribute.
- **Visibility** - Can be set to public, protected, package, or private.
- **Modifiers** - One or more modifiers are set by means of the checkboxes.
- **Concurrency** - Set the concurrency to sequential, synchronized, or concurrent
- **Stereotypes** - Stereotypes can be added an removed via a dialog that is accessed by right-clicking the stereotype field and clicking ’Open’.
- **Parameters** - Lists parameters of the operation. In Poseidon, return types are considered a special type of parameter.
- **Raised Signals** - The last list, ’Raised signals’, is used to define whether this operation throws exceptions.

  Select ’Add...’ from the context menu to insert a new exception.

  To edit this exception, select Open from the context menu, then enter a name and select a type for the thrown exception. As you know, only the type of the exception (not the name) is relevant for code generation. If you need more exception types, simply create the corresponding class in your model (e.g., `MailException` in your package `javax.mail`). Your exception type must end in `...Exception` in order to be visible in the type list of exceptions.

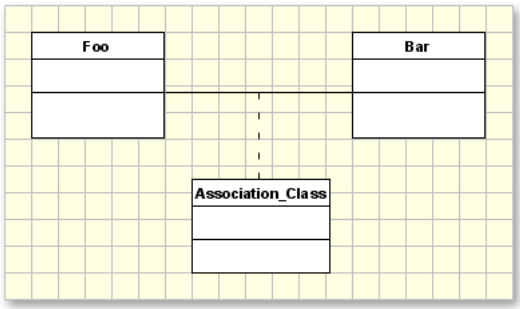
  You may also assign the <<exception>> stereotype to an element and use that element as a raised signal. First, apply the stereotype using the stereotype dialog. Second, add an exception to the desired operation through the ’raised signal’ context menu within the properties tab. Third, select the exception-stereotyped element from the dropdown ’Type’ list.

12.2.3. Association Classes

Poseidon now supports association classes.

To create an association class:

1. Create the two classes to be associated.
2. Select ’Association Class’ from the toolbar.
3. Click on one of the original two classes and drag the mouse to the other class.
4. An association class will be created between the original two classes.



12.3. Interfaces

Interfaces are present in Class, Component, Deployment, and Object diagrams. Poseidon allows you to choose a variety of ways to represent interfaces, the standard being the traditional box notation, which looks like a class with only two compartments (as attributes are not allowed in interfaces). They can also be drawn using lollipop (or ball) notation, and you can toggle back and forth between these two.

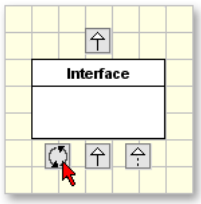
12.3.1. Box Notation

The box notation is just what is sounds like - the Interface is shown as a box with two compartments. The uppermost compartment contains the name, stereotype, and package information, while the bottom compartment displays operations that

belong to the interface. The options available to an interface (such as hiding compartments) are identical to those available to classes.

To create a new interface with box notation, select the Interface button from the toolbar and click in the diagram.

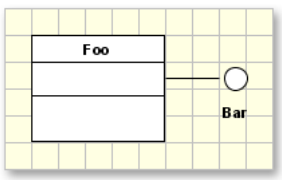
You can toggle between box and lollipop notation using the representation rapid button.



12.3.2. Lollipop Notation

Lollipop notation is a condensed way to represent an interface. The interface itself is drawn as a circle with a solid line connecting it to another element. This indicates that the element to which it is connected 'offers' the interface.

In this example, the class Foo offers Bar as an interface.



You can create the class first and then add the interface using the lollipop rapid button.

Alternatively, you can create the interface with the 'Interface as Circle' toolbar button and connect it to the element by either dragging the lollipop rapid button from the element to the interface, or you can select the lollipop button from the toolbar and drag it from the element to the interface.

The order in which the elements are connected is important to the meaning of the diagram. If you drag the relationship from the interface to the element, you will create a 'realize' relationship, indicating that the interface realizes the element.

12.3.2.1. Sockets

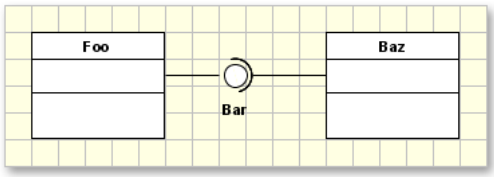
In addition to the compact representations, lollipop notation has an advantage in the

clarity of a diagram. This is because not only can ’offered’ interfaces be modeled, but ’required’ ones as well. These are called ’sockets’ and are drawn with a semi-circle and line to the element requiring the interface.

To create a standalone socket that does not connect to an offered interface, you can either click the socket rapid button from an element, or you can click the socket button in the toolbar and drag it from an element to a blank space in the diagram.

To connect a socket to an offered interface, first create the offered interface as a circle. Then drag the socket rapid button (or socket toolbar button) from the element requiring the interface to the offered interface.

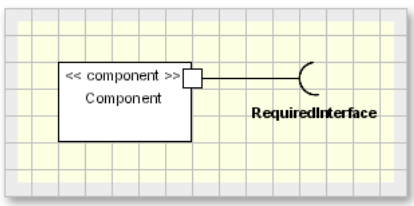
Here we see that class Foo offers the interface (Bar) that class Baz requires.



12.3.3. Ports

Ports provide a way to organize interactions between components and their environment within Component, Object, and Deployment diagrams. They can be uni-directional (only required or only provided interfaces connect to the port), or they can be bi-directional (both required and provided interfaces occur at one port).

Figure 12-9. Uni-Directional Ports



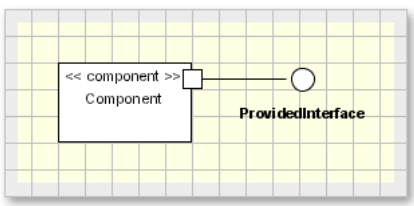
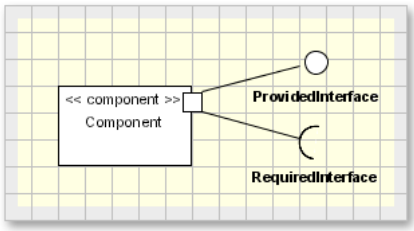


Figure 12-10. Bi-Directional Port



To create a port on a component, first select the component in the Diagram pane, then click the port toolbar button or the port rapid button. This is a bit different than with other element types. You cannot first select the tool from the toolbar and then apply it to the component.

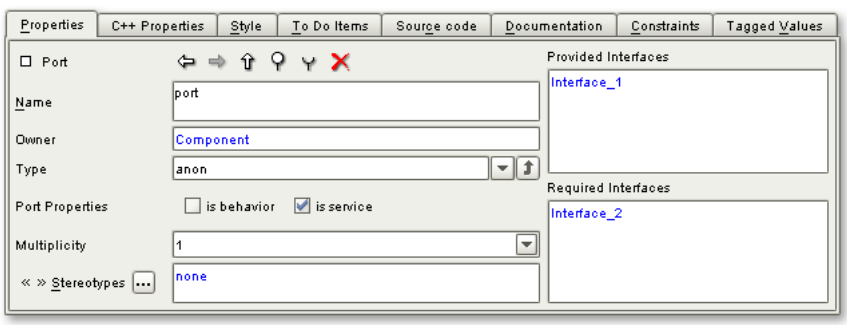
The properties of ports are very similar to those of other elements. The 'Port Properties' entry, however, only appears for ports. These checkboxes describe how the information entering the port will be handled.

- **isService** - When checked, this port provides access to the published functionality of a classifier; when unchecked, this port implements the classifier but is not part of the functionality of the classifier, meaning that it can be altered or deleted.

  The default value is checked.

- **isBehavior** - When checked, signal requests arriving at this port are received by the state machine of the object, rather than by any parts that this object contains.

  The default value is unchecked.



Chapter 13. Using Elements

13.1. Creating New Elements

A new diagram, of course, requires elements in order for it to have significance. There are several ways to add elements to a diagram, as you will see in the next couple of sections.

13.1.1. Diagram Pane Toolbar

The diagram pane toolbar contains buttons to create elements that are specific to that diagram. For example, the button to create an initial state will not appear in a class diagram toolbar. This reduces the amount of buttons that you must deal with at one time.

The create buttons for most elements act as stamps, so that the element is placed wherever you click within a diagram. The exceptions are associations and ports.

Ports are created by first selecting the component that will have a port, then selecting the port tool from the toolbar. At this point the port will be automatically added, you need not click on the component to add the port.

Any sort of relationship needs to exist between two model elements; therefore, both elements of the association must be included in the creation process instead of just stamping a line anywhere. To create a new association element with the toolbar, select the type of association and place the cursor over the first element in the relationship. Click and hold the mouse button, then drag it to the second element in the relationship. Note that for some of the association types, the order in which the elements are connected affects the definition of the association.

Try it Yourself - *Create new elements with the toolbar*

1. Open the class diagram `User Registration: Design-Class Model`.
2. Select the ’create class’ button from the toolbar. The mouse should now appear as a crosshair.

3. Place the crosshair to the right of the class ’User’ and click the mouse button to create the new class.

4. Select the ’generalization’ button from the toolbar. The mouse will again appear as a crosshair.

5. Place the crosshair in the new class, press and hold the mouse button, then drag it to ’User’.

** Note that the order in which they are connected determines the direction of the inheritance.*

6. You are now ready to incorporate the new class into the model. Look through the rest of this guide to learn how to change the name of the class, color-code it, add elements and operations, and more.

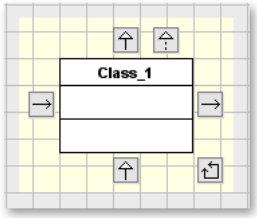
13.1.2. The Rapid Buttons

The toolbar is not the only way to create new diagram elements or associations. Poseidon for UML provides an intelligent shortcut that can speed up the development of a diagram. Select a class and wiggle your mouse near the edge of the class and several additional buttons will appear. They are called Rapid Buttons and are only available if an element is selected.

These rapid buttons can be used in several ways. You can either click on it to create and associate a new corresponding model element with appropriate connection in one step, or keep the mouse button pressed and drag it to an existing model element to create a new association without creating a new class.

Rapid buttons are available for many diagram elements in each of the editors. Here is a class example:

Figure 13-1. Rapid buttons for a class element.

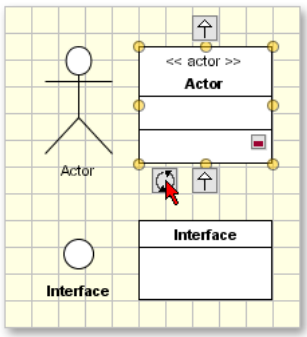


For a class element, the rapid buttons to the left and right represent directed associations, the button on top represents specialization of superclasses, below is the generalization of subclasses, and self-associations are in the bottom right corner.

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.

Rapid buttons for some elements change the way the element is presented. For interface elements, you can toggle between lollipop (or ball) representation and box representation. Similarly, for actor elements you can toggle between stickman and box representation.

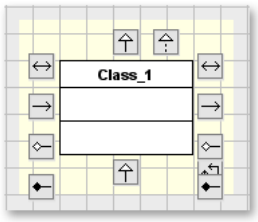
Figure 13-2. Toggled representation of actors and interfaces



The rapid buttons displayed are some of the more commonly used buttons. To display additional buttons as rapid buttons, hold down the ’shift’ button while

rolling over the element with the mouse.

Figure 13-3. Additional rapid buttons for a class element.



- **Attributes and Operations** - You can create new attributes and operations by clicking the rapid button that appears when you hover over the attribute or operation compartment of a class.
- **Delete** - Press Ctrl while hovering over an element to display a rapid button that will not open a confirmation dialog.
- **Go to Sub-Diagrams** - If an element contains diagrams, an additional rapid button will appear in the top right-hand corner that will take you to the sub-diagram.

13.2. Editing Elements

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

13.2.1. Inline Editing Text Values

The diagram drawing area in the Diagram pane not only allows for creating, deleting and moving graphical elements; it is also possible to enter values, such as names, directly into the elements without using a different pane. Exactly which element properties can be modified depends upon the specific element. Most of the elements allow editing of the name of the element at a minimum. For example, selecting a state from within a state diagram and then typing will immediately open a small text editor. When editing is finished, the typed text will replace the previous

text in the navigation tree and Properties tab, as well as in the diagram of the selected state.

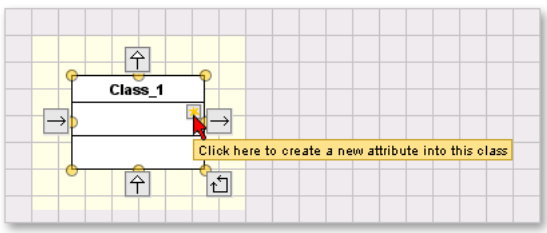
Classes and interfaces offer far more options for editing values than just editing their names. Both of them are constructed of different parts called **compartments**. The first compartment holds the values for the name, the stereotype and the package of the class or interface. You can edit the name of the class as described above; however, stereotypes and packages can only be changed using the Properties tab. The second and third compartments hold the attributes and operations defined for the class or interface (in UML, interfaces can have operations only). Inline editing works the same way here. Select the attribute or operation you want to change and start typing (or double-click on it to open the inline editor). Press Ctrl-Return on the keyboard or click elsewhere in the application to end the editing. Note that in some cases, interfaces may be rendered as a circle in lollipop notation, and therefore will not display compartments.

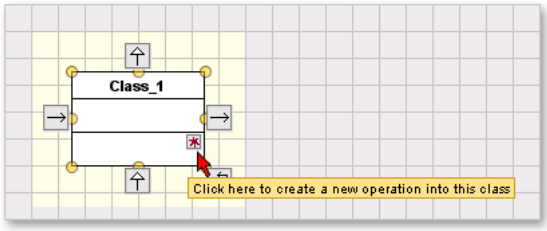
After editing an attribute or operation, you can directly add another attribute or operation without leaving the element by hitting ’return’ on the keyboard instead of Ctrl-Return after editing the first attribute or operation.

You can also create a new attribute or operation with a rapid button by moving the mouse to the right side of the compartment and then clicking on the ’create’ button that appears. As above, Ctrl-Return will end the editing and add the new attribute/operation to the class or interface, and ’return’ on its own will end the editing and create a new attribute or operation..

The attributes and operations compartments in the diagram can be set to invisible for the current diagram via the Context menu, or for the entire model via the ’Settings’ dialog from the ’Edit’ menu.

Figure 13-4. Add a new attribute or operation to a class inline





13.2.2. Editing Via the Details Pane

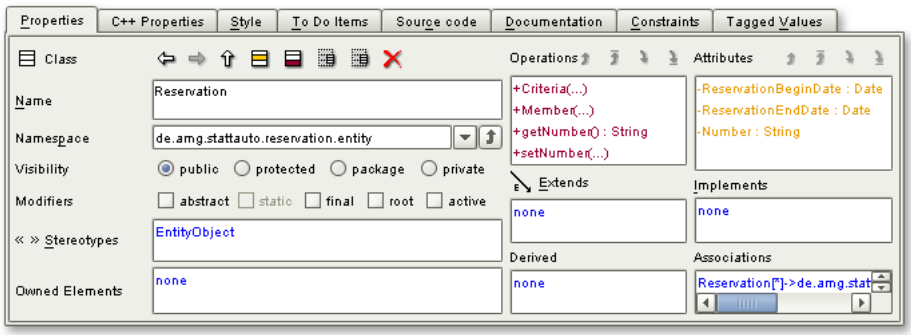
The first tab you will see in the Details pane is the Properties tab.

13.2.2.1. The Properties Tab

There are many modifications that can be made to elements from the Details pane. You can add attributes and operations, rename elements, change namespaces and stereotypes, add colors and borders, and much more. This section will outline some of the most important modifications that can be made. Many of these procedures can be extrapolated to other editing procedures.

Let’s look at a class element, as these are very frequently used elements.

Figure 13-5. Properties tab for a class



The toolbar across the top of the tab contains buttons for navigation between elements, creation buttons, and a delete button. These buttons will change depending on the type of element selected as the current active element.

Below this toolbar are the editable characteristics of the class. The name of the element can be typed directly into the name field with no restrictions. Likewise, Visibility and Modifiers can be directly modified from their checkboxes. Note, however, that these two properties are not displayed in the diagram itself; thus, the changes made will be visible only from the Properties tab (the modifier 'abstract' is the exception to this). The Namespace must be selected from the list of available options. Stereotypes can be applied through the Stereotypes dialog, accessed by right-clicking the Stereotypes field and selecting 'Edit' from the context menu that appears. The Owned Elements section is automatically populated.

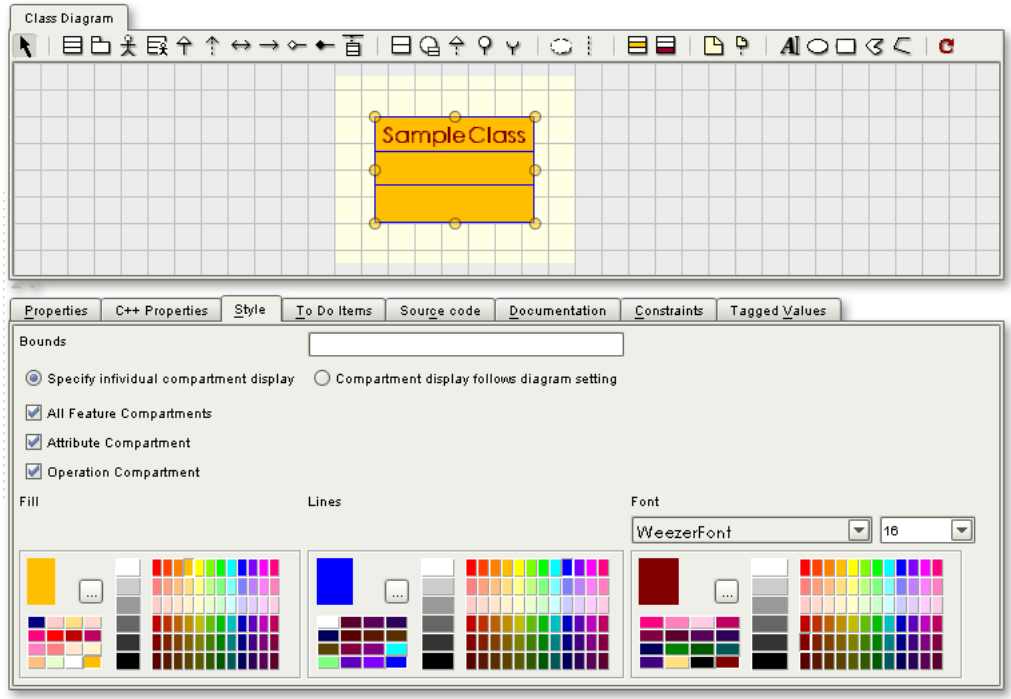
All changes made to the class are propagated throughout the model. For instance, when a namespace is changed, the navigation tree is updated and the class is moved from the original package to the new one that was just selected. This change is also reflected in the Diagram pane: the top compartment of the class will display *(from new_namespace)* in place of *(from old_namespace)*, where old_namespace and new_namespace refer to the original namespace and most recently selected namespace. This easy and convenient mechanism for changing namespaces is provided for nearly all of the elements.

To the left of the editable characteristics are elements which are affiliated with the selected element. In UML, operations and attributes are considered both an elements in their own right as well as a characteristics of a class. As they are elements, they have their own Properties tabs; therefore, to edit the name or any other properties of an operation, for example, we must go to the Properties tab of that operation. That is why it is not editable here. The remaining fields are: `Extends`, `Implements`, `Associations`, and `Derived`. These properties show different relations between the focused class and other model elements.

13.2.2.2. The Style Tab

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

Figure 13-6. Style tab for a class



The Style tab indicates which colors and fonts will be used to display the element. This is very useful when color-coding diagrams or highlighting aspects of the diagram. It is also possible to override diagram-level specifications for compartment visibility. As with the properties tab, not all of the options make sense for every element; therefore, only the appropriate style options are available.

Options for the Style tab:

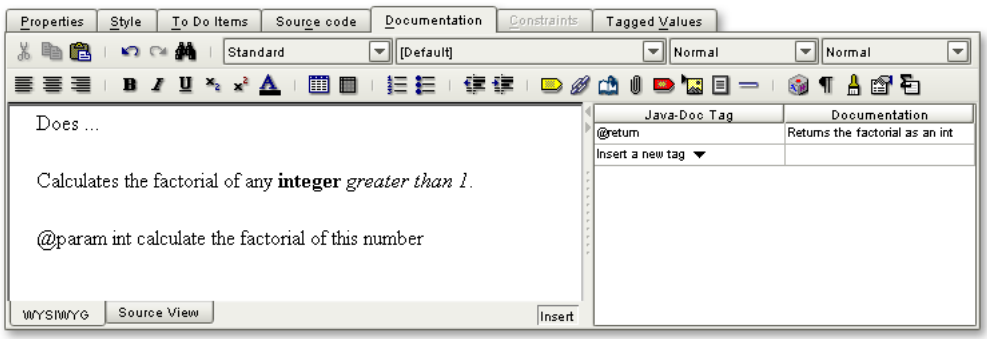
- **Fill** - Determines the background fill color of the element
- **Lines** - Determines the border color of the element
- **Font** - Determines the color and font of the text.
- **Visibility** - Radio buttons and checkboxes determine the visibility of element compartments.

Whereas changes made to an element in the Properties tab are propagated throughout the model, changes made to the style of an element apply to the current diagram only.

13.2.2.3. The Documentation Pane

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 13-7. Editing a method documentation.



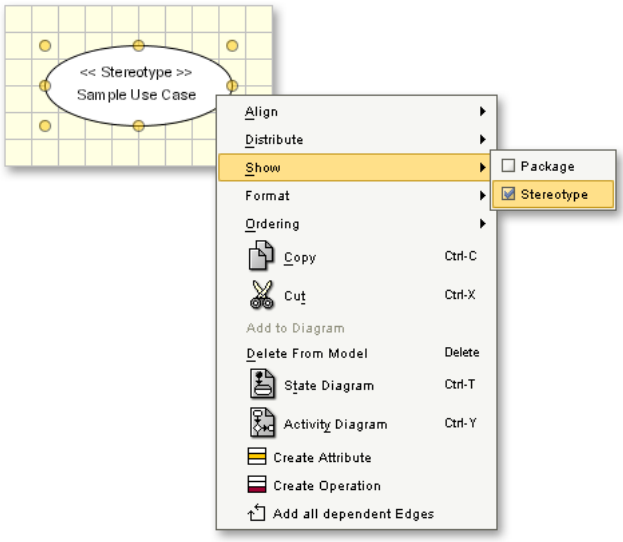
13.2.3. Editing Via the Context Menu

The Context menu can be accessed by right-clicking on an element in a diagram. Entries relevant to the selected element are displayed. Remember that things like attributes and operations are considered elements in their own right; therefore, the context menu for an attribute will be different than that for the class in which it occurs. If you do not see what you expect, be sure that you have selected the proper element to be the active element.

The Show option displays all checked items in the diagram. In the case of a class element, this includes stereotype, package, and compartment options. Unchecked items remain hidden from view.

It is also possible to create things like attributes, operations, and dependent edges when appropriate. These items are listed towards the bottom of the context menu and, once created, are available for editing in the Properties tab.

Figure 13-8. Context menu options for a Use Case



13.2.4. Undo/Redo

Sometimes when working with your models, you might have done something you did not really intend to do. If this happens, the ability to revert your work can be very valuable. Poseidon for UML offers such an undo mechanism. The Undo function is not limited to the last change you made - you can undo all the steps you took prior to that, and you can even redo the things you just undid.

To Undo or Redo actions:

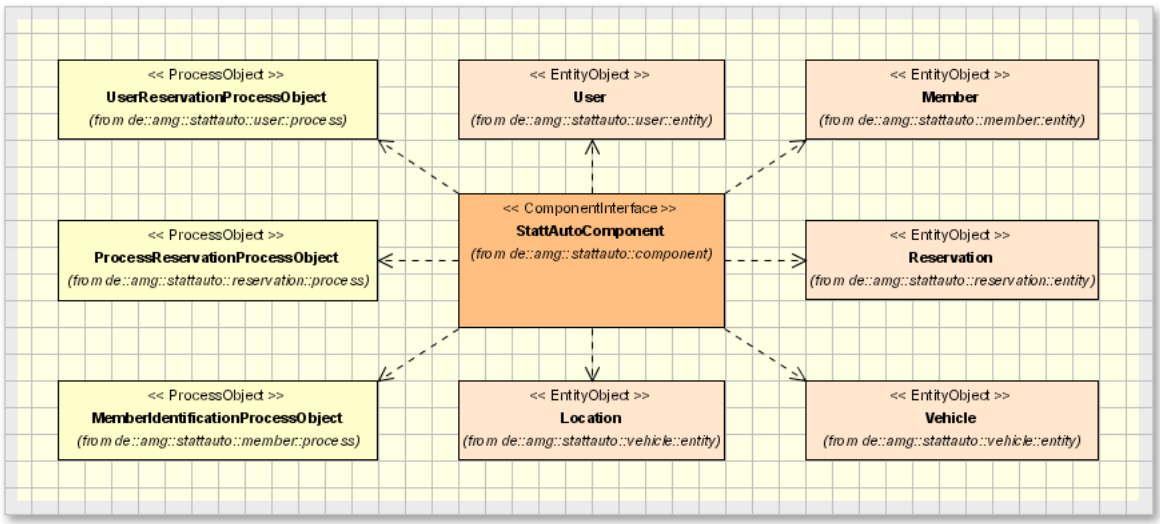
- **Main Menu** - Select Undo or Redo from the Edit menu
- **Main Toolbar** - Click the Undo or Redo button on the main toolbar
- **Quick-Keys** - Use the quick-key Ctrl-Z for undo or Ctrl-W for redo

13.2.5. Stereotypes

One of the general patterns of an architecture is the **Model-View-Controller-Pattern**, or the ***Boundary-Control-Entity-Schema***, as it is often rephrased in the UML community. According to this, an architecture is constructed in three layers.

First, the **Boundary** is responsible for representing information to the user and receiving his interactions. Users of the system interact with this layer only. The next layer, **Control**, contains the rules on how to combine information and how to deal with interaction. It is responsible for transferring control based on the input received from the Boundary layer. And finally, the **Entity** layer holds the data and is responsible for its persistence. To which layer a class belongs is expressed using corresponding stereotypes. You obtain these in the Properties tab of each class. An example for the usage of stereotypes is shown below.

Figure 13-9. 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:

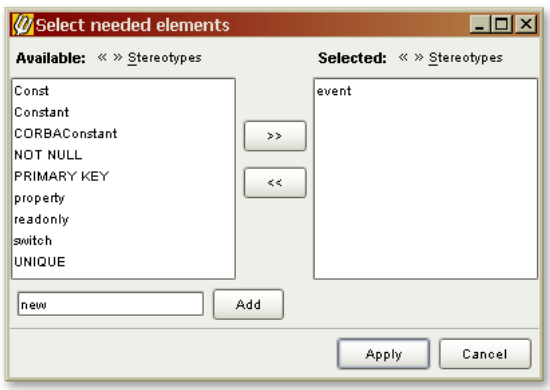
1. Select the element the stereotype applies to from the diagram, Details pane, or Navigation pane.
2. Open the Properties tab for this element in the Details pane.

3. Click the ellipsis button (...) to open the stereotype dialog.

   Note that the previous method to access the dialog (right-click in the stereotype field and select 'Edit' from the menu) still works.

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 13-10. Stereotype dialog



13.2.6. 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 fromse the quick-key Ctrl-Z for undo or Ctrl-W for redo this single model, even if the diagrams are constantly used as a means to change the model. The consequence of this is that modifications to any element within a diagram are applied to the element, not to the diagram. As such, a change made to the element will be seen throughout the entire model.

It then follows that selecting an element and then pressing delete means that the element itself is deleted, meaning that it no longer exists within the model and is removed from all aspects of the model, including other diagrams. Additionally, all connections to other elements, such as associations or inheritances, are completely removed. Note that there is a big difference between deleting an element from a model and removing an element from a diagram.

This leads us to use different terminology with different meanings: You can **delete an element from the model**, which means that the element is removed entirely and is no longer available in the Navigation pane or in any of the diagrams, or you can just **remove its figure from the current diagram** you are working with, leaving the element available to the rest of the model. These are very different things, and different commands are used to achieve them.

To completely remove an element from the model:

- Use the delete button  in the Properties tab
- Select an element or part of an element in the diagram and hit ’delete’ on the keyboard
- Select ’Delete from Model’ in the Context menu
- Use the delete rapid button

To remove an element’s representation from the current diagram:

- Select an element or part of an element in the diagram and use Ctrl-X to cut the item
- Select ’Cut’ from the Context menu

The element, as part of the model, remains untouched in other diagrams and it also remains in the tree in the Navigation pane. For elements that are connected to other elements through, for example, an association or inheritance, removing the first element (e.g. a class) means that the association is no longer valid; therefore, the second element (e.g. the association) is also removed from the diagram, but is likewise still accessible from the navigation pane or other diagrams.

If you want to remove an element but not the connections it has to other elements, you can detach it by selecting the connection and dragging the handle at the end of it to another element *before* you remove the element.

Chapter 14. Generation

UML wouldn’t be worth all the sophisticated work if all it came down to was pretty vector graphics. When analyzing and designing a software system, your final goal will be to generate well-implemented code.

Poseidon for UML provides a very powerful and flexible code generation framework, based on a template mechanism. It is currently used to generate a variety of code types including Java and HTML, but it is flexible enough to generate any kind of programming language, or other output, such as XML.

14.1. Code Generation

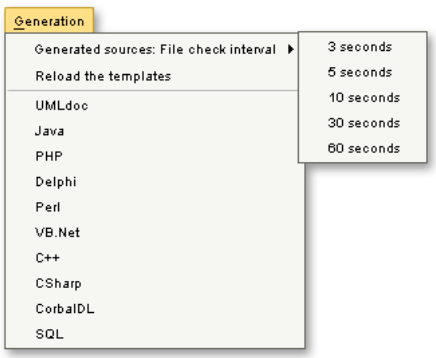
Java code generation is usually based on the classes of a model and other information displayed in the respective Class Diagrams. Additionally, Poseidon can generate setter and getter methods for the fields of each class.

By default, associations between classes in UML are bi-directional; that is, associations allow navigation between classes in both directions. For the common object-oriented programming languages, these need to be transformed into separate uni-directional associations. If one of these is set, the other should be set accordingly. The code for managing bidirectional as well as unidirectional associations is also generated automatically.

14.1.1. Generation Settings

Code and documentation generation are both invoked from the Generation menu.

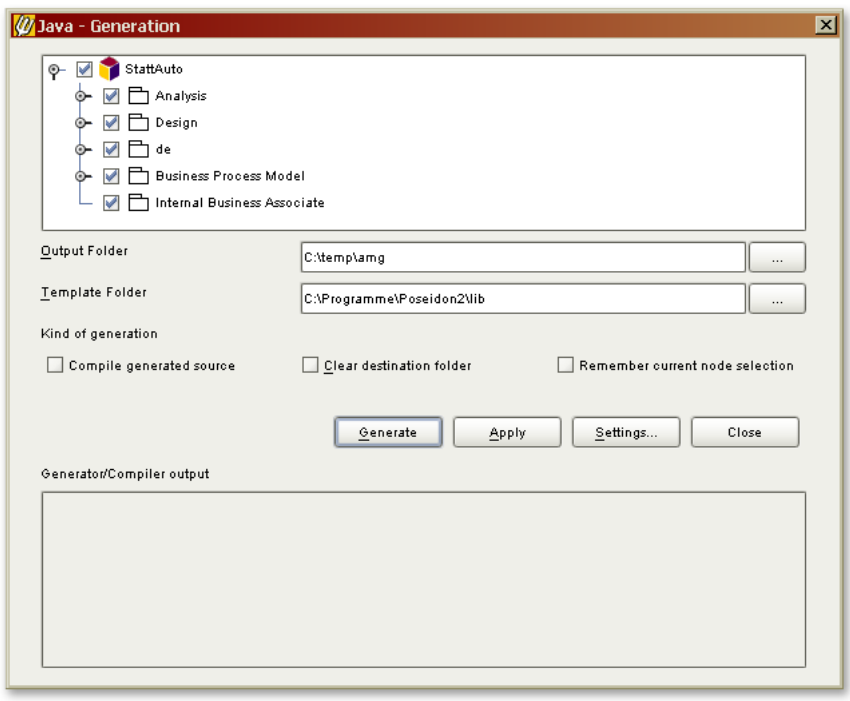
Figure 14-1. Generation menu

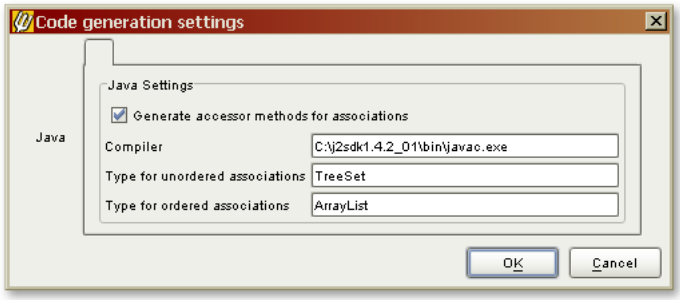


Select the type of generation you would like from the Generation menu and a dialog will appear. Here you can select or deselect model elements from the tree, specify an output and a template folder, and indicate if the destination folder should be cleared. Poseidon can remember your selection of model elements if you enable the checkbox titled, 'Remember current node selection'.

Note that the output and template folders are saved by project and language. This means that output of Java generation can be placed in one folder, and Perl generation from the same project can be placed in another folder. These folder preferences will be saved to the current project, but another project will not recognize these preferences and must have its own preferences set. This is to avoid accidentally overwriting the output from a previous project.

Figure 14-2. Code Generation dialog and settings - Java





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

14.1.2. Reverse Engineering

Software engineers often run into the problem of having to re-engineer an existing project for which only code and no models are available. This is where reverse-engineering comes into play; a tool analyzes the existing code and auto-generates a model and a set of Class diagrams for it.

Poseidon for UML can do this for Java programs, where source code is available and is in a state where it can be compiled without errors. With the corresponding JAR Import function (available only in the Professional and Enterprise editions), it even works with JAR files of compiled Java classes.

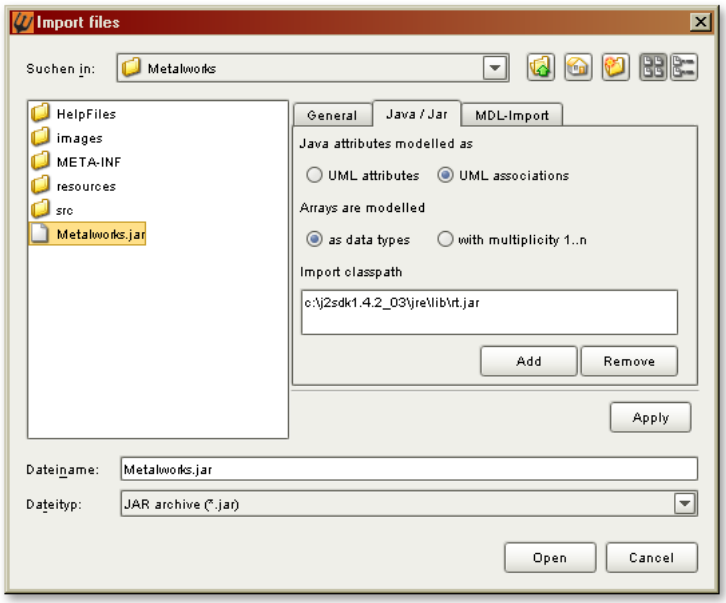
To launch this process, go to the Import Files menu and direct the file chooser to the sources' root package. It will then analyze this as well as all sub-packages. The outcome is a model containing the corresponding packages, all the classes, their complete interface, their associations, as well as one Class Diagram for each package. Note that the path that you select here will be automatically adopted by the generation dialog. The next time you open the code generation dialog, this path will be displayed as output folder by default.

If the imported file uses classes that are part of the JDK, these classes will be created in the model as required, so you may see some apparently empty classes in the package `java` in your model. This is of no concern and is done solely to have a smaller model. But these classes are necessary to have a consistent project. All classes that the imported files use must be present in the model.

Additionally, 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 14-3. 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.

14.1.3. Roundtrip Engineering

Generating code and reverse engineering that same code still does not make round-trip engineering. Reverse-engineering generates a new model from existing code, but it does not by itself reconnect the existing code to an existing model. This feature is only available in the Professional 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

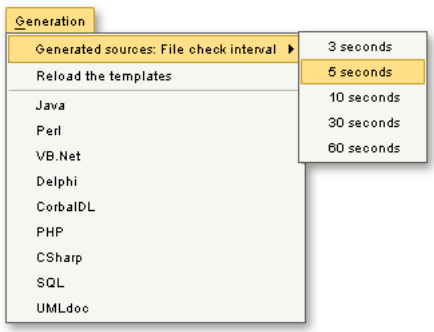
remove methods and member variables. Poseidon keeps track of all changes, and all your project data is in one place - in Poseidon.

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.

To use this feature:

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

Figure 14-4. Select file check interval.



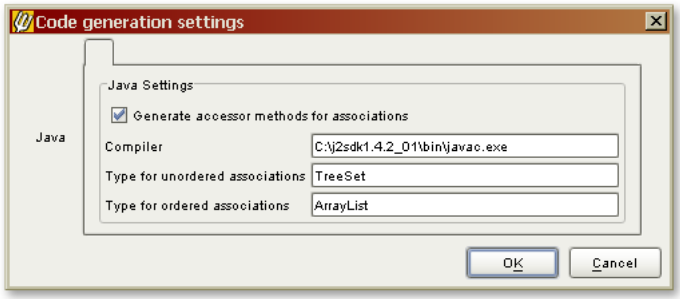
3. Generate the code (the generation window will automatically pop up if you load a model)
4. Use your preferred editor to modify the code (especially the method bodies), modify identifier names, add or remove methods and/or variables
5. Save the file in your editor

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 14-5. 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 therefore 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).

14.1.4. Fine Tuning Code Generation

There are several possibilities to fine-tune the appearance of the generated Java source code. Among them are the creation of accessor methods for attributes, the types for collection attributes, and the list of import statements in the files.

- **Accessor Methods**

  From Poseidon 1.4 on, you can create accessor methods for attributes automatically. This way, you can fine-tune the code so that some attributes have accessors, some not. In previous versions of Poseidon, you could only have setters/getters for all attributes, or for none.

  In Edit - Settings, there is a check box called ’Generate accessor methods for attributes’. Check this box to have accessor methods created for every attribute that is created. If the attribute has a multiplicity of 1..1 or 0..1, two simple `getAttribute()` and `setAttribute()` methods are created. For attributes with a finite multiplicity, an array is generated, and the accessor methods include

`addAttribute()` and `setAttribute()`. For an unbounded multiplicity, a Collection is generated, and the appropriate access methods like `addAttribute()` and `removeAttribute()` are produced.

You can fill the bodies of these access methods according to your business logic. Also, you can hide the display of accessors by setting the check box 'Hide accessor methods' in Edit-Settings-View.

Additionally, you can generate the standard accessor methods for your attributes at code generation time. These will be visible only in the generated code, not in your Poseidon project.

- **Collection Types**

  Poseidon up to version 1.3.1 used the type Vector whenever an association had a multiplicity of ..*.

  From version 1.4 on, the rules are:

  1. Create an attribute of the element's type if the multiplicity is 0..1 or 1..1.
  2. Create a `Collection` type attribute if the multiplicity has an upper bound of *.
  3. Create an array of the element's type if the multiplicity has an upper bound that is not 1 and not * (that is, it is a number).

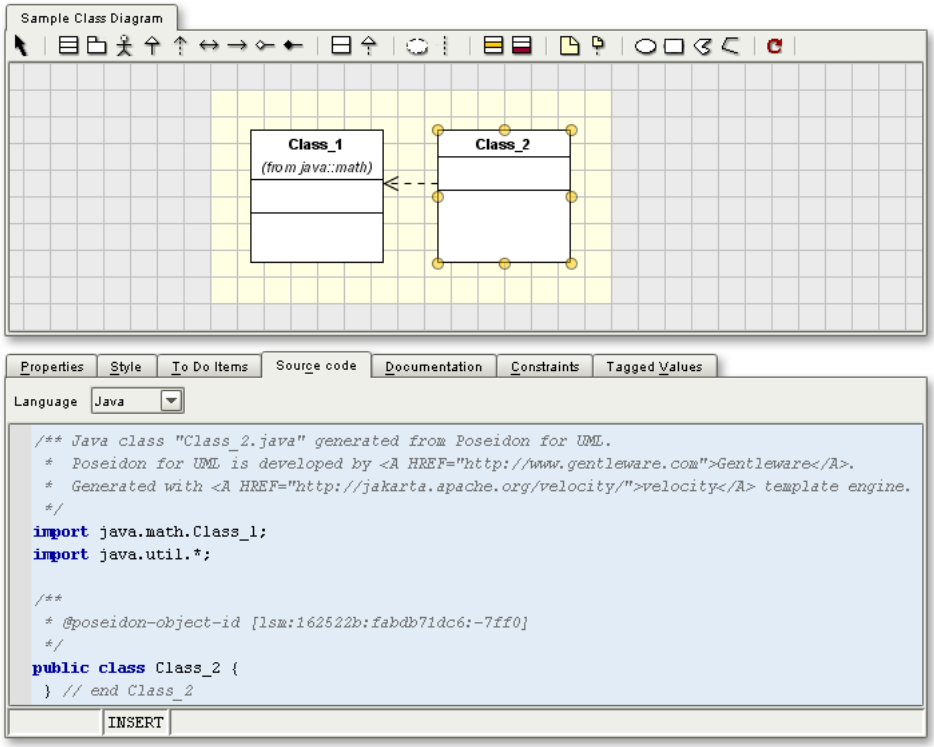
  In `Code Generation-Settings,` you can define what type of collection should be used for Collection types. The default is `ArrayList,` but you can enter any type (e.g., Vector) that implements `Collection.` Accessor methods are programmed against `Collection`.

  This version of Poseidon, you are now able to distinguish between ordered, unordered and sorted attributes, and you will be able to give different kinds of implementation types such as `TreeSet` for unordered, `ArrayList` for ordered and `Vector` for sorted attributes.

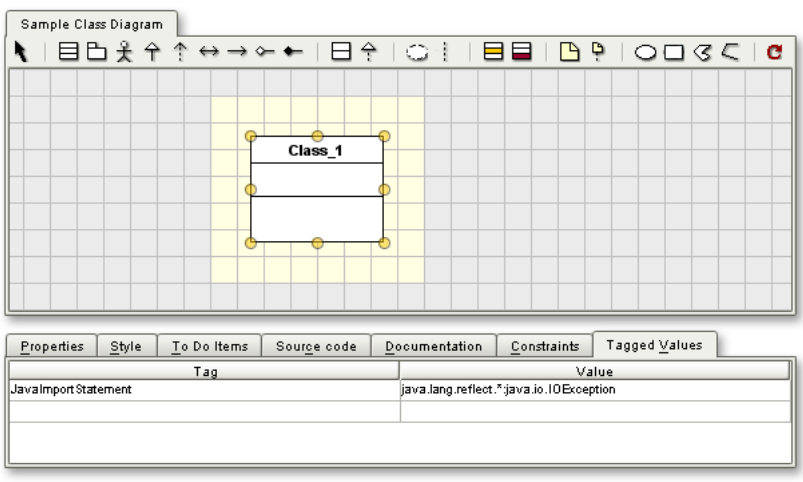
- **Import Statements**

  Import statements can be added to classes in two ways: By drawing dependencies or by entering tagged values.

  The graphical way is to draw a dependency from the class to the class or package that you want to import. An appropriate import statement will be generated: Either `import package.*` or `import package.Class`.



The second way (that does not clutter up your diagrams) is to add a Tagged Value called `JavaImportStatement` to the class. Then enter a number of imports, separated with colons. Qualified names can be given in Java syntax. For example, import `java.lang.reflect.*` and `java.io.IOException` by setting the tagged value `JavaImportStatement` to `java.lang.reflect.*:java.io.IOException`.



Above, you can see the import statement in the Tagged Values tab, and below is the resulting Java code generated by Poseidon.

- **Modifying Templates** (Not available in the Community Edition)

  More advanced customizaton of the generated code is possible if you are using one of the Premium Editions. Modification of the templates that are used for code generation is possible with these editions. We cover this topic briefly in a separate chapter and more deeply in a separate document that is distributed with these editions and online under http://www.gentleware.com?redirect=codegenapi.

- **Javadoc Tags**

  You may not want certain operations to be reverse engineered. Any operations with the Javadoc tag '@poseidon-generated' will be excluded from the reverse engineering process.

14.2. Advanced Code Generation

This chapter describes the code generation functions offered by Poseidon for UML and the options for customizing the code generation templates. Code generation based on standard templates is available in all editions of Poseidon for UML. The standard templates define code generation for Java and HTML. With the Developer and Professional Editions, you have the option of changing the code generation templates to suit your specific requirements. You can even create new templates to generate code for a different programming language such as C#.

14.2.1. Velocity Template Language

Code generation in Poseidon for UML is based on the Velocity Template Language. Velocity is an open source template engine developed as part of the Apache/Jakarta project. Originally designed for use in the development servlet based Web applications, it has also proved to be useful in other areas of application including code generation, text formatting and transformation.

The Velocity Template Language (VTL) supports two types of markup elements: references and directives. Both references and directives can be intermixed freely with the (non-VTL) content of a template, as shown in the examples below.

Since templates have been widely used in the field of Web page generation, we will begin with a simple HTML example. The second example demonstrates the use of VTL to generate Java code.

For further information on Velocity – including complete documentation of the Velocity Template Language – please go to the Velocity Web site at http://jakarta.apache.org/velocity/.

14.2.1.1. References

References are variable elements referring to some entity provided by the context. A reference such as `$userName` or `$userList` can be used to access and store a particular data structure for use within a template; thus, references establish the connection between a template and the context of the Velocity engine.

Within a template it is possible to create a new reference at any time and to assign a value to the new reference. This is done using the `#set` directive (see directives). This means you can add references to the active context as required. If a reference name is used within a template for which no corresponding object or value exists in the active context, the reference name is treated as plain text, i.e. it is output "as is" just like the other (non-VTL) elements of the template.

Every reference must have a unique name. The name (also known as the VTL identifier) begins with a dollar sign `$` followed by a string of characters as described in the following table:

$	dollar sign - the dollar sign must be the first character in the reference name, and it may not occur in any other position.
a-z, A-Z	alphabetic characters - only standard characters are allowed, no accented or diacritical characters. The first character following the dollar sign must always be an alphabetic character.
0-9	numerical characters
-	minus sign (hyphen)

_	underscore

A regular expression describing the reference name syntax would be:

```
$[a-zA-Z][a-zA-Z0-9_/-]*
```

In addition to referencing variables, it is also possible to specify attributes and methods by means of the VTL reference syntax. Using references such as `$item.name` and `$item.price`, you can dynamically insert the attributes associated with the specified object. Likewise, you can access the methods of a referenced object (for example a Java object) using a reference such as `$item.getNameAsString()`. This will return the result of applying the given method to the specified object.

Taking this one step further, you will find that the standard Java templates supplied with Poseidon for UML make extensive use of the following syntax:

```
#set ($name = $currentOp.getNameAsString())
```

Here the reference `$name` is dynamically set to the string returned by the method `$currentOp.getNameAsString`. This use of references to elements of the context establishes a very powerful connection between the templates and the template API.

14.2.1.2. Directives

Directives in VTL are a defined set of commands that can be used for basic control functions within a template. For example you can us the directives to create typical procedural branches (if/else) and loops (foreach).

The current set of VTL directives comprises the following commands:

#set()	function for assigning a value to a reference
#if() #else#elseif()#end	common conditional functions used for branching
#foreach()#end	looping function
#include() #parse()	functions for including code from another template or static resource
#macro()#end	function for defining a reusable set of commands

For complete information on the use of these directives please refer to the Velocity documentation (see http://jakarta.apache.org/velocity/).

14.2.1.3. Comments

Particularly in the case of templates used for code generation it may be advisable to use comments in the templates to explain their use. Comments can be added to a template by means of the following syntax:

## Single line comment ...	The comment continues up to the end of the line. This is comparable to the syntax for single line comments in Java or C beginning with g.
#* Inline or multiline comment *#	The comment continues up to the closing character *#. This is comparable to the syntax for inline and multiline comments in Java or C beginning with /* and ending with */.

The use of comments in VTL is illustrated by the examples below.

14.2.1.4. Examples

Example 14-1. Simple HTML Template

This example uses VTL markup intermixed with HTML code to generate dynamic Web pages based on information retrieved from the context (e.g. from a database).

```
#* This is an
           example of a simple VTL template for generating dynamic HTML
           pages. *# <HTML> <HEAD> <TITLE>Holiday
           Weekend</TITLE> </HEAD> <BODY>
           <B>$roomList.size() rooms available at special holiday
           weekend rates! </B> <BR> Check in for a luxurious
           holiday weekend at these amazing prices.<BR> Choose from:
           #set( $count = 1 ) <TABLE> #foreach( $room in $roomList )
           <TR> <TD>$count)</TD>
           <TD>$room.type</TD> <TD>$room.price</TD>
           </TR> #set( $count = $count + 1 ) #end </TABLE>
           <BR> Call today for a reservation. Toll free number:
           $freePhone </BODY> </HTML>
```

This example makes use of VTL references and directives to generate an HTML page based on data from an external data source, for example a database. The data source is referenced by means of the elements `$roomList` and `$room` (with its attributes `$room.type` and `$room.price`). When this template is applied, the

directives and references are interpreted and the results are inserted into the generated HTML code.

The resulting HTML page might look something like this:

```
<HTML> <HEAD> <TITLE>Holiday
            Weekend</TITLE> </HEAD> <BODY> <B>3 rooms
            available at special holiday weekend rates! </B> <BR>
            Check in for a luxurious holiday weekend at these amazing
            prices.<BR> Choose frome: <TABLE> <TR>
            <TD>1)</TD> <TD>Single Room</TD>
            <TD>$ 100.00</TD> </TR> <TR>
            <TD>2)</TD> <TD>Double Room</TD>
            <TD>$ 150.00</TD> </TR> <TR>
            <TD>3)</TD> <TD>Luxury Suite</TD>
            <TD>$ 250.00</TD> </TR> </TABLE>
            <BR> Call today for a reservation. Toll free number:
            1-800-555-1212 </BODY> </HTML>
```

Example 14-2. Simple Java template

The following example demonstrates the generation of standard Java code and a number of options for changing the format of the generated code by making slight modifications to the template.

Note: The standard Java templates supplied with Poseidon for UML use defined indentation markers to format the code for better reading. The markers are of the format: `$(__)`. These indentation markers are defined as variables that resolve to an empty string. They should never show up in the generated Java code. If you find that the generated code contains such text elements, please ensure that the markers are defined and used correctly in the template.

Below is an excerpt from the template used for generating the class and method declarations.

(A)

```
## Template for standard Java output ## .. snippet
            .. #set ($vis = $currentOp.getVisibilityAsString()) #set ($sta
            = $currentOp.getOwnerScopeAsString()) .. #set ($thrownClause =
            $currentOp.getThrownExceptionsSignature()) #set ($name =
            $currentOp.getNameAsString()) #set ($methodBody =
            $currentOp.getMethodBody()) ..
            ${vis}${static}${final}${synch}${return} ${name}($params)
            $thrownClause { #renderMethodBody($currentOp.getMethodBody()
            $currentOp.hasReturnType()) } ## .. snippet ..
```

One step you could take to modify the Java code generated by this example would be to enter a line break before the "$thrownClause" references in the template so that the thrown exceptions appear in a separate line of the method declaration. In the following example the opening bracket has also been moved to a separate line:

(B)

```
## Template for reformatted Java output ## ..
            snippet .. ${vis}${static}${final}${synch}${return}
            ${name}($params) $thrownClause {
            #renderMethodBody($currentOp.getMethodBody()
            $currentOp.hasReturnType()) } ## .. snippet ..
```

The effects of such a change become clear if we compare a bit of Java code generated on the basis of these simple variations (A and B):

(Java code based on A)

```
public static void main(String[] params) throws
            Exception { doSomething() }
```

(Java code based on B)

```
public static void main(String[] params) throws
            Exception { doSomething() }
```

14.2.2. Working with the Standard Templates

The standard templates supplied with Poseidon for UML can be used to generate Java and HTML code. The generated code is based on Class Diagrams only, but one may want to produce code from deployment diagrams or sequence diagrams. With the Professional Edition you can create your own templates to generate IDL files or C++ code.

The Java code generated on the basis of the standard Java templates is fully Java 2 compliant. The code can make use of all the features supported by Java 2, including exception handling, inner classes, and static initializers.

HTML code generated on the basis of the Standard HTML templates is simple HTML, similar to Javadoc. A separate page is generated for each class in a Class Diagram. As with the Java templates, the Professional Edition of Poseidon for UML allows you to modify the HTML templates to conform with your preferences and requirements.

14.2.3. Code Generation API

For a detailed description of the code generation API, please refer to the online API documentation (http://www.gentleware.com/support/dev/poseidon2-openapi/index.html) (Javadoc) and the separate document describing the code generation framework (http://www.gentleware.com/support/dev/PoseidonCodeGenFramework.html). These files are part of the Developer and Professional distributions in the docs folder.

Also available are two demo plug-ins (http://www.gentleware.com/support/dev/examples.php4) that show the capability of the code generation API and of the Poseidon plug-in API in general. The demo plug-ins are distributed as ready-to-run JAR files, along with the appropriate license keys. Also, the source code is distributed, including an ANT script for building the JARs. You may use these plug-ins as examples and as starting points for your own plug-ins. If you want to create your own plug-ins, please contact plugins@gentleware.com to receive a key for your plug-in.

14.3. Documentation Generation

For generating the corresponding HTML documentation for your model you need the UMLdoc feature, which is available in every edition *except* the Community Edition. Users of any edition, including the Community Edition, can utilitze the UMLdoc Online Service (http://www.umldoc.org). 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, such as comments to your classes or methods, is included in the code. But unlike UMLdoc, Javadoc provides a view of the code only, not of the model. For example, you do not see your diagrams. With the UMLdoc feature you get the same information as with Javadoc, in addition to all diagrams from your model. This includes Class diagrams, Use Case diagrams, sequence diagrams etc. This is valuable information that you would want in your documentation.

14.3.1. 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.

Note that from within Poseidon, the default browser is Netscape. This is not configurable in this version; however, workarounds are available. For instance, in Linux you can create a soft link from Netscape to Mozilla with the command: **ln -s /usr/local/share/mozilla netscape**. Now Poseidon should open external links on your system with Mozilla.

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 14-6. Generated UMLdoc opened in Netscape.

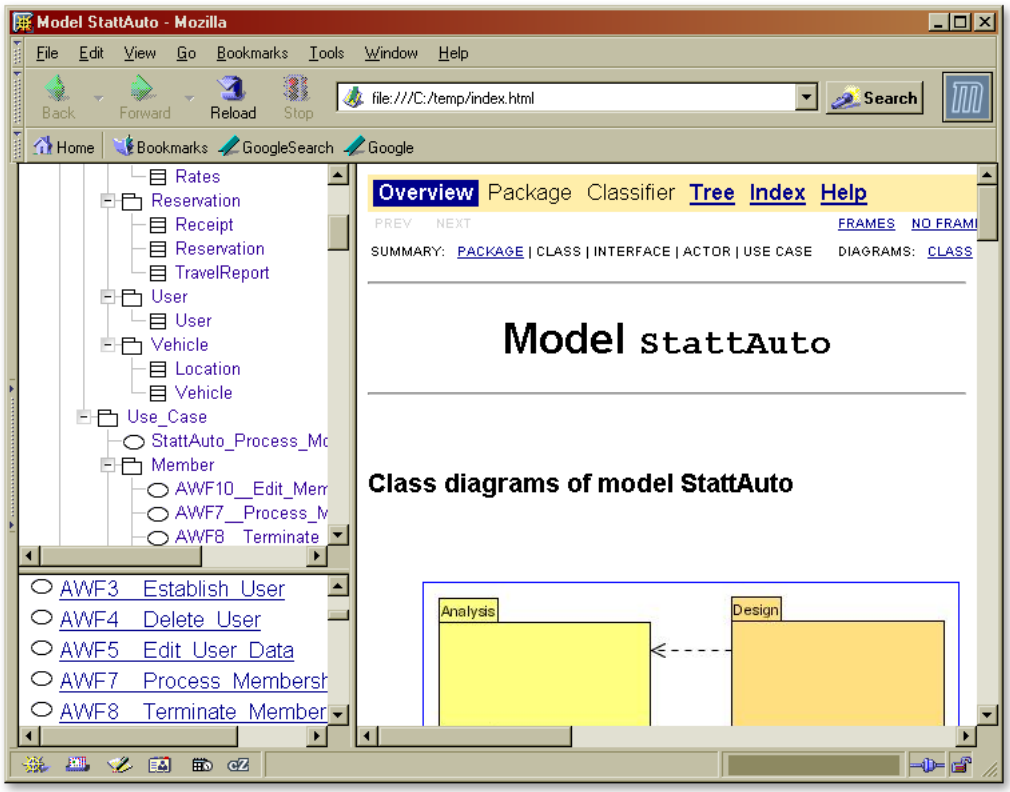
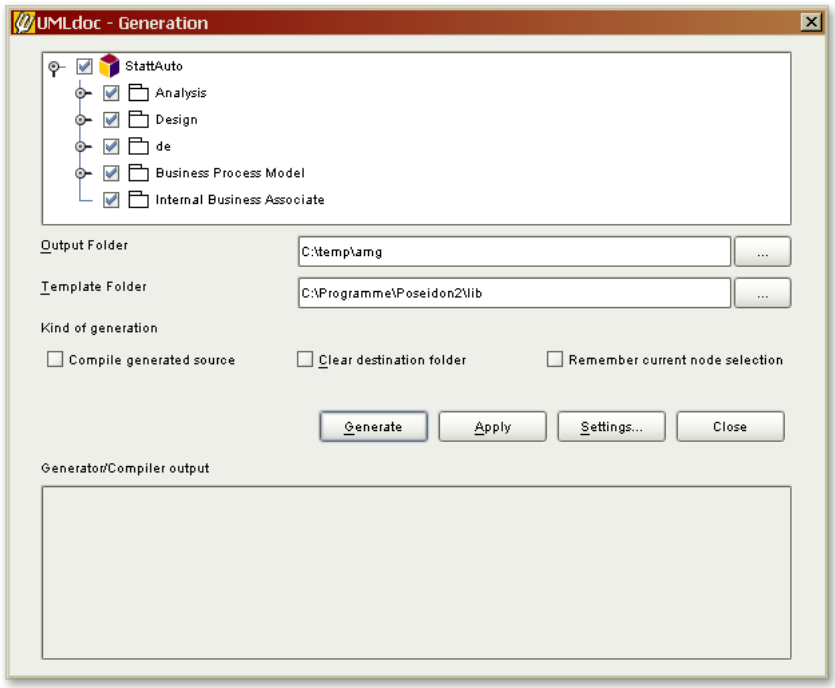
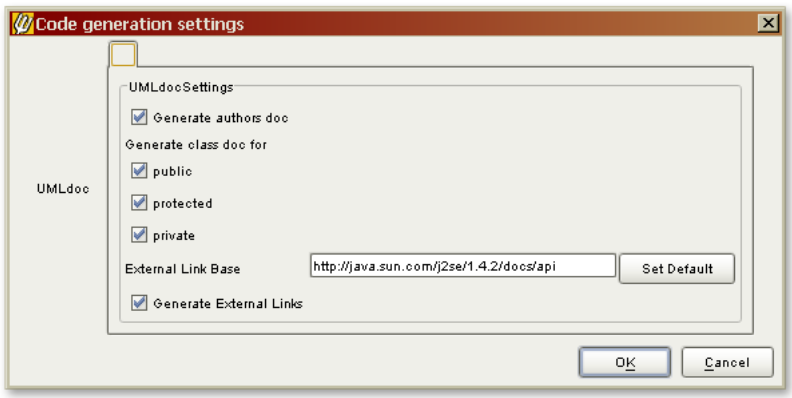


Figure 14-7. Code Generation dialog and settings - UMLdoc





14.3.2. 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.

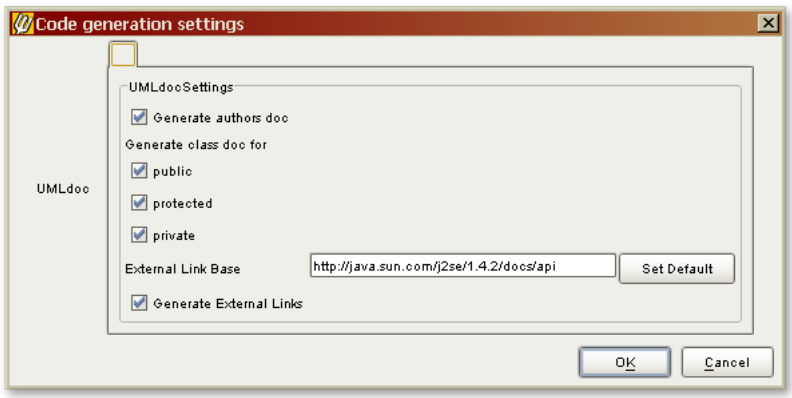
External Link Base

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 14-8. UMLdoc code generation - settings.



14.3.3. 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 15. Plug-Ins

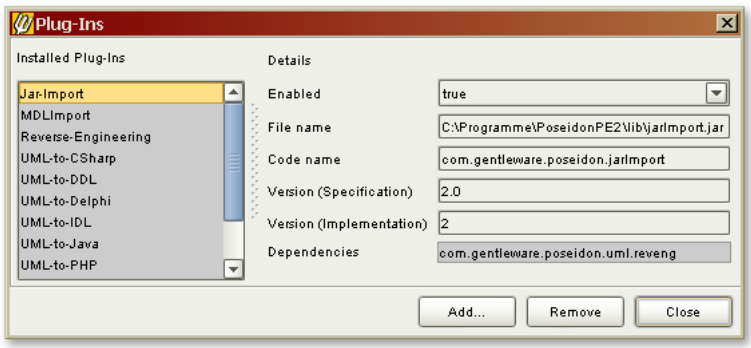
With Poseidon's plug-in interface it is possible to add extended functionality that is well beyond what is implemented in the core product. The Standard Edition of Poseidon for UML comes with this plug-in feature. Development teams from Gentleware AG as well as technology partners are working on plug-ins that meet specific designer and developer needs. The following sections give a brief overview of the most recent plug-ins that are available for shipping (or will be soon). For information on how to install a plug-in, please see the separate documentation (http://www.gentleware.com?redirect=support) available on the Gentleware Web site.

The Professional Edition of Poseidon comes with several options for code generation and one for documentation generation (UMLDoc). Java code generation is the default setting, but you can also choose to generate other types of code. To do this, you have to activate the plug-in that supports the desired language. (Via Plug-Ins -> Plug-Ins Panel). When you do this, a set of stereotypes becomes available that can be used to control the result of the code generation. The next sections describe what stereotypes and tagged values you can use to control the output of code generation.

15.1. The Plug-In Panel

The Plug-In Manager is provides an easy interface to install, manage, and uninstall plug-ins. The left side displays all installed plug-ins, while the right side displays details about the selected plug-in.

The plug-in displayed in the figure below is named JarImport. It contains all of the information used by Poseidon to import archived Java files. This plug-in is standard in the Professional Edition.



Details available in the Plug-in Manager:

- **Enabled** - This dropdown allows you to determine whether or not a plug-in is used by Poseidon.
- **File Name** - Displays the location where the plug-in is installed. This field is not editable.
- **Code Name** - Displays the code name of the plug-in.
- **Version (Specification)** - Displays the version number of the plug-in.
- **Version (Implementation)** - Displays the internal build number of the plug-in.
- **Dependencies** - Lists the plug-ins from which it uses functions.

15.1.1. Installing a New Plug-In

Using Plug-Ins requires three steps. You must first add the license, then install the plug-in, and finally enable the plug-in.

15.1.1.1. Add the Plug-In License

1. Download the plug-in from the Gentleware web site. You must additionally purchase a license key from the Gentleware store (except in the case of beta versions).
2. From Poseidon, open the License Manager from the Help menu.
3. Paste the Serial Number into the ‘New Key/Serial #’ box at the bottom of the License Manager. This number should have arrived via email when you purchased the plug-in.

   Click the ’add’ button.

4. The Serial Number now displays a valid status.

   The Serial Number must be registered in order to receive the Final Key. The Final Key allows you to use an unrestricted version of the plug-in. Failure to register the plug-in will cause the plug-in to cease operation after the grace period expires.

5. Click the ‘register’ button. You can choose to register online (directly from the dialog) or via the website. Once the registration is complete, close the License Manager.

15.1.1.2. Install the Plug-In

1. Check the documentation accompanying your plug-in to determine which directory you should use for installation. For most plug-ins, extract these files into the ‘lib’ directory under the Poseidon installation directory. There are exceptions, however. For example, if you have downloaded the GoVisual autolayout plug-in, extract these files into the ‘lib/ext/’directory.
2. Now that the files are in place, it is time to add the Plug-In to Poseidon.

15.1.1.3. Enable the Plug-In

1. Open the Plug-In Panel (located in the Plug-Ins menu). Click the ‘add’ button.
2. Select the .jar file for the Plug-In from the ‘lib’ directory (or wherever you installed the plug-in, as this is the same file that was unzipped earlier in the process).

   Click the ‘install’ button.

3. Verify that the Plug-In has been installed and is enabled by highlighting the name of the Plug-In from the list of Installed Plug-Ins.

15.2. Removing Plug-Ins

If you decide to no longer use a plug-in, you have the option to disable it from within the Plug-In panel. You can also remove the plug-in from the panel by selecting the plug-in and clicking the ’Remove’ button. This does not remove the files for the plug-in from the Poseidon directory, nor does it remove the license key for the plug-in.

To completely uninstall a plug-in, you must manually delete the files that were added during the installation process, then open the License Manager and remove the License Key.

15.3. Available Plug-Ins

15.3.1. 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. Modeling and coding are not separated anymore.

15.3.3. 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.

15.3.4. MDL Import

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

15.3.4.1. Installing and Using MDL Import

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.

15.3.4.2. Supported Diagrams

This version of the import plug-in reads class, state, activity, usecase, and sequence diagrams. The other diagram types will be incorporated in the next release.

15.3.4.3. Unsupported Features

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

- Poseidon currently supports comments for classes, interfaces, packages, use cases, actors and states, but not for transitions, associations or objects. If a comment is not supported, it is added to the diagram as ordinary text.

- Metaclass: Poseidon does not support meta classes, these classes are imported as ordinary classes.
- Synchronization States: Rose does not discriminate between fork and join states. There is no way of telling how to map synchronization states - this plug-in currently always assumes fork states if the number of outgoing transitions is bigger than one. You are informed about the decision.
- Subsystems: Subsystems are treated as packages - Poseidon does not support subsystems at the moment.

The following features are (at the moment) not being imported at all. You will get a warning after the import is complete that these elements will be missing.

- Destruction Markers
- Swim lanes
- References: MDL files support references to other files (`*.jar` or `*.cab` files, for example). This import tool ignores references, no warning is issued.

Other problems: Some older versions of Rose have a bug in sequence diagrams: Links between objects have a wrong target ID. These links will not be resolved correctly by this plug-in - you will get an error message. Rose does the resolving by name instead of by ID, which seems rather error-prone, so we do not try to do this. Loading and saving the model with a new Rose version like Rose 2000 solves the problem, and the sequence diagram can be correctly be imported.

15.3.4.4. Display Issues

MDL files contain information about the diagram layout. The import plug-in reads the diagram elements coordinates and positions the diagram elements accordingly. A few things should be considered, though. Poseidon uses "smaller" coordinates than Rose. In general, scaling down the coordinates by 40 percent does the job - the diagrams almost look like they did in Rose. You can change the value in the Configuration tab to the right. If you choose 80%, for example, the diagram elements are further apart (but not bigger!) - making it easy to add comments or further elements.

While the coordinates are read from the MDL file, the sizes of diagram elements are dependent on the information being displayed. For example, a classes size depends on the length of the contained methods names and parameters. Long names or lots of parameters may lead to overlapping classes. To solve this, you can either select a higher scaling factor, or (at least for Class Diagrams) you can can edit the display options (select menu item Edit/Settings, and click the tab Diagram display).

Sequence Diagrams

Poseidon performs an automatic layout of sequence diagrams - layout information contained in MDL files is ignored. Objects are currently placed arbitrarily, you might have to re-arrange them and any associated textual information. Apart from that, Rose allows activations to have arbitrary length, while Poseidon calculates the length of activations depending on the stimuli sent. Using the right mouse button, you can force an object to remain activated after the last message was sent.

15.3.4.5. Status

We did extensive testing, and any problems during import should be signaled. But before you use and extend an imported file for production work, you should check your models and diagrams in case some model element was forgotten. If you experience problems or want to request additional features, do not hesitate to contact us at <`http://www.gentleware.com?redirect=contact`>

Chapter 16. Advanced Features

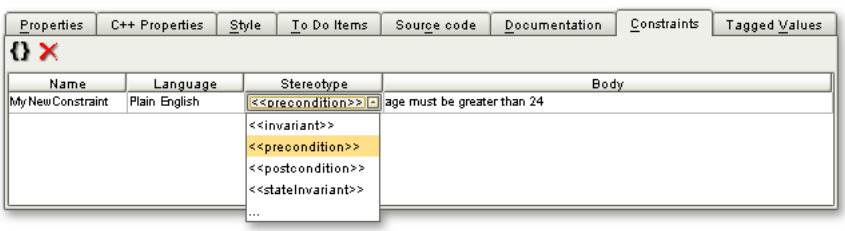
16.1. Constraints with OCL

UML is basically a graphical language. As a graphical language it is very suitable for expressing high-level abstractions for architectures, workflows, processes etc. But for expressing very detailed and fine-grained things like algorithms, equations or constraints, textual languages just tend to be more convenient.

The current UML recognizes this and comes with a supplementary textual language to express constraints. This language is called the Object Constraint Language, or abbreviated OCL. Although Poseidon does not require OCL, nor does it check OCL syntax, there are certainly no restrictions prohibiting you from using OCL.

Since OCL is noted as text, it is simple to support, and many UML tools do it just that way. You can simply enter lines of text in certain fields reserved for constraints. In Poseidon for UML you can do that in the Constraints tab on the Details pane, as shown in the figure below.

Figure 16-1. A Constraints tab.

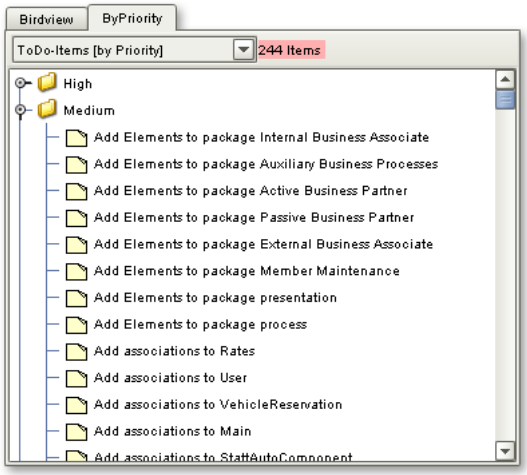


16.2. Critiques

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 contains three priority nodes in the 'by Priority' view.

Figure 16-2. Critiques pane.



Available Views in the Critiques Tab

- **by Decision Type** - The type of action to be taken determines the category of the critique. The available categories are:

  Uncategorized, Behavior, Class Selection, Naming, Storage, Inheritance, Containment, Planned Extensions, State Machines, Design Patterns, Relationships, Instantiation, Modularity, Expected Usage, Methods, Code Generation, and Stereotypes

- **by Diagrams** - Critiques are arranged according to the diagram in which they appear
- **by Knowledge Type** - Critiques are listed according to type. The available types are:

  Correctness, Syntax, Presentation, and Completeness

- **by Offenders** - Lists design elements of the current model and the critiques associated with them

- **by Posters** - Critiques are arranged according to the critic who reported the problem
- **by Priority** - Critiques are categorized in a priority of high, medium, or low

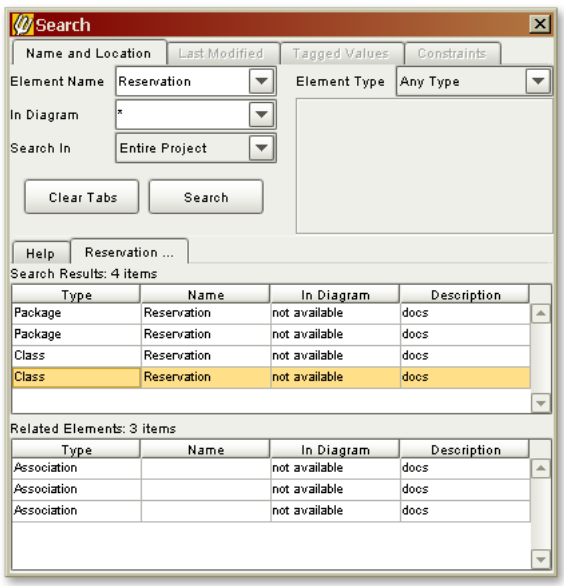
Broadening and improving this feature is part of each development cycle of Poseidon.

16.3. 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 the function key F3.

Type in the name of the element you are looking for (you can also use the asterisk as a wildcard), and specify the type of element you are looking for. If you are looking for a class, this type would be Class.

Figure 16-3. Searching for a class



For each search, a new tab is created so that you can access older search results. You can also restrict the search space to be the result of an earlier search. Selecting

one entry from the results list provokes that `Related Elements` are also shown. Double-click on one entry in the results list whereas effects that the element is selected in the Navigation pane.

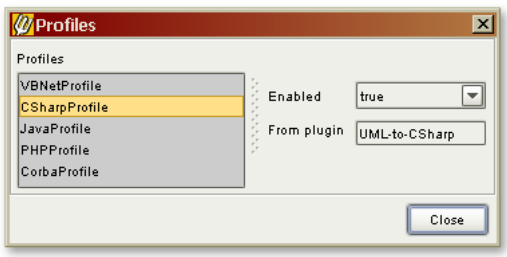
16.4. Profiles

Profiles generically extend the UML through the use of stereotypes that are most often language-specific, provide a common graphical notation and vocabulary, and define a subset ot the UML metamodel (that could possibly be the entire UML metamodel). For example, variable and operation types change based on the profile (and therefore the stereotypes) used. There is a profile associated with each of the language plug-ins, and the profiles that automatically appear in the Profile Manager directly correspond to the set of enabled language-specific plug-ins and are enabled by default. Likewise, if a plug-in is disabled from the Plug-in Manager, the associated profile is automatically disabled and will not appear in the Profile Manager.

It may be advantageous at times to disable these profiles. The Profile Manager displays those profiles that are currently available and allows you to enable and disable them with a simple dropdown menu.with features such as version control and messaging. It also incorporates all of the features of the Professional Edition.

The profile is saved to the project as long as the profile was enabled in the Profile Manager when the project was saved. If the originating plug-in or the profile was disabled at the time of the save, no data related to that profile is saved. Say you have disabled the profile, and then decide to disable the plug-in. If you enable the plugin again, the profile will be automatically enabled. The status of the profile is not saved when the plug-in is disabled.

Figure 16-4. The Profile Manager



Chapter 17. Using The Enterprise Edition

The Enterprise Edition is the high-end version of Poseidon for UML. It is designed for use in highly collaborative development environments, with features such as version control and messaging. It also incorporates all of the features of the Professional Edition.

Before you use the Enterprise Edition for the first time, consult Enterprise Edition Installation Guide for information about how to correctly configure the client application.

17.1. Interface

The collaborative modeling environment is based on a client-server architecture. It is quite unnecessary for those wishing only to model to concern themselves with the details of the server; therefore, this section outlines the new features of the Client application only.

The GUI for the Enterprise Edition is very similar to the Professional Edition, but with some additions. As the Professional Edition features and functionalities are covered elsewhere in this manual, this section is not intended as an exhaustive list of the Enterprise Edition GUI. Rather, it is an addendum containing those features that are exclusive to the Enterprise Edition.

17.1.1. Connection Status

There are two quick ways to determine the connection status to the server.

First, a status icon appears in the lower left-hand corner of the application that states whether the Client application is connected to the Enterprise Server. A tooltip appears with text when you hover over this icon.

Second, the main toolbar will display the five Enterprise-specific buttons that do not appear in the other Editions.

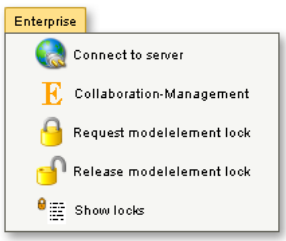
17.1.2. Toolbar

The toolbar of the Enterprise Edition changes depending on the connection to the Poseidon Collaboration Server. When not connected, the toolbar looks identical to the Professional Edition toolbar.

However, functions change while engaged in a collaboration. It no longer makes sense to save a project locally, and to make this difference clear the save button changes. Likewise, functions that are irrelevant while outside of a collaboration such as locking and access to the Collaboration Manager become visible.

 Save to Server
 Collaboration Manager
 Show Locks
 Request Lock
 Release Lock

17.1.3. Menu



- **Connect to Server** - Opens the connection dialog box
- **Collaboration Management** - Opens the Collaboration Management dialog box
- **Request Model Element Lock** - Send a request to the server to make a model element uneditable for other members of the collaboration
- **Release Model Element Lock** - Allows access to a model element for other collaboration members
- **Show Locks** - Displays all locks for the current project.

17.1.4. License Manager

The Enterprise Edition uses a floating license scheme. This means that there are a limited number of people who may connect to the server at one time. Ordinarily,

Poseidon will automatically contact the Floating License Server and attempt to obtain a license. If no license is available on Poseidon startup or when working with Poseidon, a dialog will pop up stating that the license is invalid and that it is trying to request a new license.

This popup provides access to the License Manager. Two tabs are available from here, but as the Enterprise Edition utilizes Floating Licenses, only the Floating License tab is of interest to us at this point.

17.1.4.1. Test Connection

To test the connection to the Floating License Server, specify the server address in the Floating License tab and click 'Test Connection'.

The following results are possible:

- Successfully connected to floating license server with valid license.
- Successfully connected to floating license server but no valid license.
- No floating licenser server available.
- Connection to floating license server failed because of malformed URL.
- Connection to floating license server failed because lookup failed.
- FL_NotReachable=Could not establish a connection to floating license server.

17.2. Modeling with Others

17.2.1. Collaborations

UML exists primarily to facilitate between people. We can look at the symbols and understand the meaning of a much more complex system. The ease with which others can interpret drawings is an essential goal. The UML also provides a powerful mechanism for describing these systems in technical terms, so that a complete semantic and syntactic picture is presented.

Naturally, an environment where people work closely together, regardless of physical location, is an ideal setting for UML to come into play. One question has always lingered - how do we work on the same project with knowledge of what others are doing and without stepping on toes?

Poseidon for UML Enterprise Edition introduces the idea of a collaboration, where teammates have a real-time view of the project and the means to control editing through locks.

The heart of this is found in the Collaboration Manager, where you can create and join collaborations, as well as save projects to the server so that others may have access to them.

In the Collaboration Management Window, you'll see the list of active collaborations and a list of projects that have been saved to the server. To create a new collaboration with a new project, click the "Create Collaboration" button, give the collaboration a name, and wait until the window closes. To re-open a saved project, select it in the bottom list, and click the "Load project"-button.

You can create diagrams and model elements just as if you were working in the standalone version of Poseidon.

17.2.1.1. New Collaboration

1. Click the 'Create Collaboration' button from the toobar
2. In the dialog, give the collaboration a name
3. Wait for the window to close
4. You are now the first member of a project that is hosted on the server. In your toolbar several enterprise-icons will appear, while several standalone-icons (like "create new project") will have been removed.

17.2.1.2. Join Collaboration

If another user has already started a collaboration, you may join that collaboration to work together on the same project. Connecting to the server, and you'll see that users collaboration in the upper part of the collaboration management window. Simply select it, and then click the "Join collaboration"-button. You will now get a copy of that project onto your computer.

1. Open the 'Collaboration Management' window.
2. Select the collaboration to join
3. Click the 'Join Collaboration' button
4. A copy of the project will now be available for you to edit

17.2.1.3. Leave Collaboration

If you decide to end your work, or to create a new project, or to load another project, you have to log off from the current collaboration. Open the Collaboration Management Frame (you can use the "E"-Button in the toolbar), and then select "Leave current collaboration". After that, all work you do is local again, and the buttons for saving and creating new projects locally are back in place. If no other user had joined your collaboration, it will be automatically saved to a file by the server, and removed from the list. To re-enter that project, load it from the list of stored projects.

17.2.2. Projects

Projects can be stored at the server level so that they are accessible to other team members.

17.2.2.1. Load and Start Project

To load and start a project that is stored on the server:

1. Make sure that you are connected to the server and disconnected from any collaborations.
2. Select a project from the lower list in the Collaboration Manager.
3. Click ’Load and Start Selected Project’.
4. The project will open and a new collaboration will be started.

17.2.2.2. Upload Project

To upload project and make it available to others:

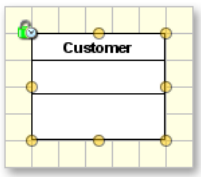
1. Make sure that you are connected to the server and disconnected from any collaborations.
2. Click ’Upload Current Project’ in the Collaboration Manager).
3. The project will now be available to other team members connected to the Enterprise Server. (You may have to close and re-open the Collaboration Manager to see the changes.)

17.2.3. Model Locking and Conflict Checking

It is quite possible that you may require access to a large number of elements, and as only one person may edit an element at a time, you do not want to be concerned with fighting for access with another member of your team. Element locking allows you to guarantee access for yourself, while denying edit priviliges to other team members.

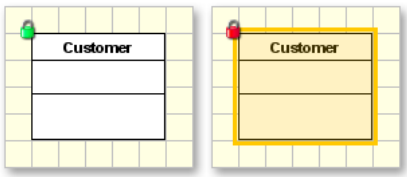
When, for example, a class is locked, no one else may change the class's name, add an association to it, remove an attribute, or delete a state in that class's state diagram. Other users are not even allowed to select a locked element - looking at it as it appears in the diagram must suffice. Additionally, no one else may lock an element that is inside your locked namespace, and likewise you can't lock a namespace that already contains locked elements.

By default, autolocking is activated for the elements you select. This means, when you select an element, it tries to aquire a a temporary lock (which behaves like a normal one). If this is not possible (because of elements locked inside that element) no warning is issued. When you deselect the element, the lock is removed automatically. Autolocking is used to prevent conflicts as early as possible - as mentioned before, a locked element can't even be selected by other users, let alone edited. Autolocking is very useful in most settings, but it can be a hard restriction too. For example, when you select a class, then other users will not even be able to align states that reside in that classes state-diagram. You can deactivate auto-locking in the settings dialog. Note that auto-locking does NOT apply for the model - you can select the model in the tree, and you can lock it by hand, but it will not be locked automatically. Locking is only possible if nothing else inside the namespace is locked (e.g. by another auto-lock), so it is very unlikely that you'd get the model-lock without consulting your co-workers first. And even if you got the models lock, your coworkers might get a bit upset when they notice that they cannot continue working - while you selected the model just to create a few stereotypes. Any element with an autolock will appear to the owner with a green lock and clock icon.

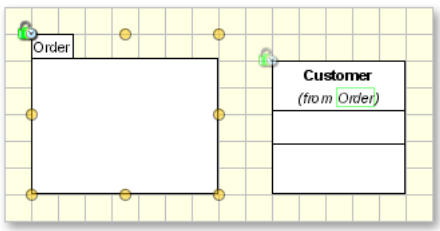


All others will see a red lock and clock. These locks are automatically released when you select a different element.

Namespaces may also be locked manually to reserve certain elements. These must be specifically set and unset while working on the model. To lock a namespace manually, first select the namespace and then press the 'lock' icon in the toolbar. From that point forward, the element will be marked for the lock owner by a green lock in the upper left corner of that element, while all other users will see a red lock.



Elements that are contained within a locked namespace are automatically locked too, so they also get marked. To make it possible to tell the locked and the implicitly locked elements apart, the inner elements get a transparent lock. If you don't want to see the lock-symbols, you can turn them off in the settings window.



Please remember that not all elements are namespaces. Comments are not namespaces, so it is not possible to lock them one by one at the moment.

17.2.3.1. Java-Import

The java-import will need access to most parts of the model. To prevent lots of conflicts with your coworkers during import (which would result in a complete rollback of the import), you have to lock the complete model for yourself before you can start importing. Tell your co-workers to unlock and deselect everything, acquire the lock on the model, perform the import, and remove the lock again.

17.3. Enterprise Server Administration Tool

The Admin Tool enables you to access the running Enterprise Server and perform some administrative actions. You can start the tool by running the admintool.[bat/sh] script. It presents you with the same Connect-to-Server-dialog as the one within the Enterprise Client. After connecting, the main management frame also resembles the Collaborations window from the Enterprise Client. However, there are different buttons here - for instance, you can't start collaborations here, but you can end them.

17.3.1. Collaboration Administration

17.3.1.1. Removing locks

In case some model element locks are not been removed automatically (e.g. when a clients connections crashed), you can use this button to clear all locks immediately. Removing single locks is not supported yet.

17.3.1.2. Renaming collaborations

As within the client, you may rename a running collaboration.

17.3.1.3. Ending collaborations

In case a collaboration was not ended properly (e.g. because a client crashed unexpectedly and the server did not notice this), you may use this button to force it to end.

17.3.2. Project Administration

You can use the button 'Upload Project' to upload a local project file to the server, and likewise you can delete a project with the delete button.

The most interesting feature about the Admin-Tool is its CVS-Support. If your projects directory resides in a CVS-aware directory, you will see more buttons and features in the stored projects section of the admin tool. Read more about this in the next section.

17.3.3. CVS Support

Poseidon’s Admin Tool supports the most basic (and most needed) CVS operations, so it is very easy intuitive to use; however, you should be generally familiar with CVS before you use this plug-in. The following documentation is not meant to be a CVS manual. Please refer to http://www.loria.fr/~molli/cvs/doc/cvs_toc.html#TOC1 for some nice CVS documentation.

Also, the Admin Tool’s CVS support is not a replacement for a fully fledged CVS GUI Client. Please refer to WinCVS (http://www.wincvs.org) for a Windows client, or to (http://www.jcvs.org) for a Java-based Unix/Linux client . The Admin Tool currently only supports PserverConnections. This is the most common usage and will suffice for most user needs. If you need EXT-Connections (e.g. for ssh-tunnelling), please check more advanced CVS Clients.

To use the CVS support from the Admin Tool, you need to have a CVS server running (on linux a cvs server comes along with most distributions, and CVSNT (http://www.cvsnt.org/wiki) is a freely-available Windows CVS server. You must also have a project directory checked out somewhere on your file system.

17.3.3.1. Configuring Your System for CVS Support

As mentioned, you must have created and checked out a directory from your cvs server already, and configure the Enterprise Server to use it as your projects directory (using the setting "collabserver.project.file.location" in the CollabServer.properties file). To access the CVS server, the Admin Tool needs to know the password for the user that performed that initial cvs checkout. This password is stored in a file called ".cvspass", usually located in the user’s home directory. You can either specify that files location, or you may set the encoded password as a separate setting. So, your could add an entry like this (Windows notation),

collabserver.cvs.passFile=C\:\\Dokumente und Einstellungen\\per.GENTLEW\\.cvspass

or instead use

collabserver.cvs.encodedPassword=As84123

(with the latter entry being the encoded password taken from a .cvspass file, not the original one)

On startup, the Enterprise Server displays a few values it reads from the properties files, including those above. It will also notify you it was not able to read a password from the .cvspass file, or when the password could not be used to access the CVS. The CVS support for the Admin Tool will then remain disabled.

When everything is set up properly, restart the Enterprise Server and the Admin Tool. The lower part of the screen will now present you with a CVS view and a few additional buttons for CVS Access.

17.3.3.2. Using the CVS Support

Each file in the current directory is displayed in a color reflecting the file's state in CVS:

- **Black** - File is up-to-date, or there is no CVS available
- **Blue** - File has changed and needs to be committed
- **Green** - File has been newly added and needs to be committed
- **Brown** - File is not known to CVS yet

The following states should not occur for your Poseidon Projects, as the projects are binary and will not get merged, but other files in your CVS directories might look like this:

- **Red** - File was updated, merged, and a conflict was detected.
- **Purple** - File was updated from CVS, local changes have been merged.

Next to the list of files, you can find buttons for performing CVS operations.

Adding a file to the CVS - A file that is not yet known to CVS (for example after it was uploaded) is displayed in brown. Select it, click the add button, enter a comment for it, and then the file is added (but not yet committed) to CVS. Its color changes to green.

Committing a file - A file that has been changed (blue color) or added to CVS (green color) can be committed in order to store the changes in the repository permanently. Just select the file, press the Commit button and enter a comment for that file. The files color changes to black (the up-to-date-color).

Removing a file from the CVS - To delete a file and remove it from the CVS, select it and press the remove button. You will be asked to enter a comment, afterwards the file gets deleted and removed from the CVS. It is still available through the CVS repository, so you can change your mind and restore the file. But you will need to use a more advanced CVS client (or the command line) for this, the Admin Tool does not yet support access to removed files. Note: The regular cvs remove command does not delete a file from the file system, nor does it commit a removed file. To keep things simple, the Poseidon Admin Tool auto-deletes and auto-commits files.

Renaming a file - CVS does not support renaming, so the client first renames the file. Then CVS is told to remove the file with the old name from its repository and add the renamed file to the repository as a new file. In addition to the message you supply, the Admin Tool generates a short message about the old file's name - unfortunately CVS cannot store information about renamed files, and your renamed project has no history except for the message you supply here. Because of the loss of history information, it is not advisable to rename projects just for cosmetic reasons, like from "OurWorkflow" to "Our_WorkFlow". You should only rename when the model inside the file has changed so much that the original filename does not fit the model at all anymore.

Version History - Inside the CVS file view, click the Version button to open the history frame. The history of the selected file will be displayed. Apart from looking at and closing that frame, you can also restore an old version.

Restoring an old revision - CVS does not have a command for restoring an old revision. You may "down-update" to an old version, but that implies that the checked out file is "sticky", and committing changes will create a branch on that old revision of the file. To save you the trouble of sticky files (you can do this by hand if you really want to), the Admin Tool deletes the current version of the file from the file system, checks out the old revision (sticky), renames that file to a temporary file, checks out the latest version of the file (just to get rid of the sticky tag), deletes that file, and renames the temporary file (the old revision) to the original name. Now the old file has been restored and you can modify and/or commit it. As has been checked out from CVS as new, it is not modified yet. Note that he revision will not change; if you restore revision 1.2 of a file that already was at revision 1.6, the restored file will still be 1.6, and then 1.7 once you change and commit it.

Chapter 18. Epilogue

At this point we would like to express our thanks to everyone who, over the years, has contributed to ArgoUML and Poseidon. Without this active community of developers and users, Poseidon would not be what it is today.

Also, we would like to acknowledge the work of all the other open source projects we have made use of. We share with them the intention of developing high-quality software within the open source community. The Poseidon for UML Community Edition, as well as our feedback to the open source of ArgoUML (and to other OS projects) and our activities in the development of improved open standards, are a sustained expression of this support.

And let’s not forget Jason Robbins, who started the quest that led us here.

Poseidon includes open source software by Antlr (Java source reverse engineering), Jakarta’s log4j (logging), Jakarta’s Velocity (Code Generation), Sun’s Netbeans project (the UML repository MDR), TU Dresden (OCL support), Piccolo (diagram rendering), Apache’s batik toolkit (SVG graphics export), and Freehep (Postscript and PDF rendering).

Appendix A. Poseidon C# Code Generation Plug-In Guide

A.1. General Rules

A.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

A.1.2. Additional Stereotypes

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

A.2. Modeling Element Rules

A.2.1. Classes

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

A.2.1.1. Class Signature

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

  1. internal
  2. sealed

A.2.1.2. Class Attributes

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

  1. internal
  2. protected internal
  3. volatile

A.2.1.3. Class Operations

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

  1. internal
  2. protected internal

3. override
4. sealed
5. extern
6. virtual

Everything else will use the checked visibility radio buttons

A.2.2. Interface

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

A.2.2.1. Interface Signature

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

1. internal

A.2.2.2. Interface Members

- All interface members implicitly have public access. It is a compile-time error for interface member declarations to include any modifiers. In particular, interface members cannot be declared with the modifiers abstract, public, protected, internal, private, virtual, override, or static.

Everything else will use the checked visibility radio buttons.

A.2.3. Structure

- Uses the standard UML ’Class’ with the <<*struct*>> stereotype

- Supports single inheritance only

A.2.3.1. Structure Signature

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

- internal

Struct tapes are never abstract and are always implicitly sealed; therefore the ’abstract’ and ’sealed’ modifiers are not permitted in a struct declaration. Since inheritance isn’t supported for structs, the declared accessibility of a struct member cannot be ’protected’ or ’protected internal’.

A.2.3.2. Structure Members

Function members in a struct cannot be abstract or virtual, and the override modifier is allowed only to override methods inherited from the type System.ValueType. A struct may be passed by reference to a function member using a ’ref’ or ’out’ parameter.

Everything else will use the checked visibility radio buttons.

A.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’.

A.2.4.1. Enumeration Signature

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

- internal

Everything else will use the checked visibility radio buttons.

A.2.5. Delegate

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

A.2.5.1. Delegate Signature

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

- internal

Everything else will use the checked visibility radio buttons.

A.2.6. C# Event

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

A.2.7. Operations

There are some translations on the return type of C# operations:

- ’in/out’ parameter direction will be translated to ’ref’
- ’in’ parameter direction will be translated to blank ("")
- ’out’ will be translated to ’out’
- ’root’ will be translated to ’new’

Appendix B. Poseidon CORBA IDL Code Generation Plug-In Guide

B.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

B.2. CORBA Interface

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

B.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

B.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

B.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

B.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

B.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.

Appendix C. Poseidon Delphi Code Generation Plug-In Guide

C.1. Classfiers

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

C.2. Tagged Values

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

C.2.1. Classifier

- Tag = ’uses’ with Value = string that represents unit(s) name to be included in specified unit declaration, separated by comma.

  Description: Handles strings that represent the names of units to be included in specified unit declarations. A ’uses’ tag with a blank value will be defaulted to ’SysUtils’.

  Example: UnitA, UnitB, UnitC

- Tag = ’setvalue’ with Value = string that represents the value of ’Set’, separated by comma.

  Description: Handles the way to input the value of the ’Set’ type.

Example: 1,9

- Tag = ’subrangevalue’ with Value = string that represents the value of ’Sub Range’ separated by comma.

  Description: Handles the way to input the value of the ’Sub Range’ type.

  Example: 1,9

- Tag = ’arrayvalue’ with Value = string that represents the value of ’Array’, separated by comma.

  Description: Handles the way to input the value of the ’Array’ type.

  Example: 1,9

- Tag = ’arraytype’ with value = string that represents the type of ’Array’.

  Description: Handles the way to input the type of the ’Array’ type.

C.2.2. Attribute

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

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

C.2.3. Operation

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

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

- Tag = ’virtual’ with Value = ’true’

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

- Tag =’dynamic’ with Value = ’true’

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

- Tag = ’override’ with Value = ’true’

  Description: Handles the way to set the specified operation into an ’override’ type operation.

- Tag = ’override’ with Value = ’true’

  Description: Handles the way to set the specified operation into an ’override’ type operation.

C.2.4. Exception

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

  Description: Handles the published visibility of ’Exception’.

C.3. Stereotypes

C.3.1. Attribute

- Stereotype = ’Const’

  Description: Handles the way to specify a ’const’ type Attribute.

- Stereotype = ’property’

  Description: This will handle the way to specify a ’property’ type Attribute.

C.3.2. Operation

- Stereotype = ’function’

  Description: Handles the way to specify a ’function’ type Operation.

- Stereotype = ’procedure’

  Description: Handles the way to specify a ’procedure’ type Operation.

C.3.3. Classifier

- Stereotype = ’Enum’

  Description: Handles the way to specify an ’Enumeration’ type Classifier.

- Stereotype = ’Record’

  Description: Handles the way to specify a ’Record’ type Classifier.

- Stereotype = ’Set’

  Description: Handles the way to specify a ’Set’ type Classifier.

- Stereotype = ’SubRange’

  Description: Handles the way to specify a ’Sub Range’ type Classifier.

- Stereotype = ’Array’

  Description: Handles the way to specify an ’Array’ type Classifier.

- Stereotype = ’Exception’

  Description: Handles the way to specify an ’Exception’ type Classifier.

C.4. Modeling Element Rules

C.4.1. Class

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

C.4.2. Interface

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

C.4.3. Enumeration

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

C.4.4. Record

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

C.4.5. Set

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

C.4.6. Sub Range

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

C.4.7. Array

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

C.4.8. Exception

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

C.5. Specific Rules

- An attribute with ’non-1’ multiplicity will generate an Array that is defaulted to type ’int’ with Lower Bound and Upper Bound values based on the specified multiplicity.

  Example: Attribute with multiplicity: 1..2 will generate : Array[1..2] of int;

- A blank value with the ’uses’ Tag, will be defaulted to ’SysUtils’.
- A blank value with the ’setvalue’ Tag will be defaulted to ’a’..’z’
- A blank value with the ’subrangevalue’ Tag will be defaulted to ’a’..’z’
- A blank value with the ’arrayvalue’ Tag will be defaulted to 1..10
- A blank value with the ’arraytype’ Tag will be defaulted to int
- A blank value with the ’procedure’ and ’function’ Tag will be defaulted to procedure

Appendix D. Poseidon PHP4 Code Generation Plug-In Guide

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

D.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

D.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

D.2. PHP4 Class Modeling Rules

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

D.2.1. Class Signature

- There are no visibilities for Class Signature

D.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;
```

D.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, $num); } }
```

  2. Parameter passed by reference

     Tagged value='&' with value='true' in the parameter signature.

     For example:

```
<?php function foo (&$var) { $var++; }
                $a=5; foo ($a); // $a is 6 here ?>
```

3. Function returns a reference

   Tagged value='&' with value='true' in the operation signature.

   For example:

```
<?php function &returnsReference() {
                return $someref; } $newref =& returnsReference();
                ?>
```

Appendix E. Poseidon Perl Code Generation Guide

E.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

E.2. Classes

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

E.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.

E.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.

E.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)

E.6. Aggregation

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

E.7. Inheritance

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

Appendix F. Poseidon SQL DDL Code Generation Plug-In Guide

F.1. Modeling Element Rules

F.1.1. Classes

- Uses the standard UML ’Class’
- Each class is considered as a table.

F.1.2. Attributes

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

F.1.3. Association Ends

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

F.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

F.3. Additional Stereotypes

Stereotypes apply in attributes. The stereotype for allowing NULL values is not

included since it is the default behaviour of columns.

- Primary Key
- Not Null
- Unique

Appendix G. Poseidon VB.Net Code Generation Plug-In Guide

G.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

G.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’

G.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

G.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

G.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

G.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’

G.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’

G.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’

G.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’.

G.10. Visual Basic Events

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

G.11. Attribute & Association Ends

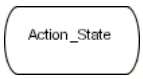
- Supports the tagged value ’Protected Friend’

Glossary

Action

An action is an atomic computation that cannot be terminated externally, and changes the state of the model or returns a value.

Action State

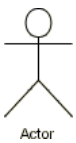


An action state is a simple state in an activity graph representing the execution of an noninterruptible and atomic action that is followed by a transition to another state.

Activation

An activation, also known as focus of control, shows the execution of an operation and the duration of time for that operation.

Actor



An actor is a representation of an entity that interacts with and derives value from the system.

Aggregation

An aggregation relationship is a ’whole-part’ relationship, e.g. a page is a part of a book.

Association

An association is a represention of a semantic relationship between instances of objects.

Association End

An association end contains a reference to a target classifier and defines the participation of the classifier in the association.

Attribute

An attribute is a logical data value of a specified type in a class which is inherent to an object. Each object of the class separately holds a value of the type.

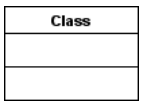
Boundary-Control-Entity-Schema

The boundary-control-entity-schema describes a three layer architecture. The boundary layer is the user interface, control decides what to do with the information gathered from the user interface, and entity holds the data.

Branch

A branch is an element in a state machine where a single trigger leads to more than one possible outcome, each with its own guard condition.

Class



A class is a descriptor for objects that share the same methods, operations, attributes, relationships, and behavior, representing a concept within the system being modeled.

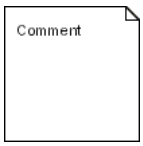
Classifier

A classifier is a model element that describes structural features and behavior. Some classifiers include: class, actor, component, data type, interface, node, and use case.

Collaboration

A collaboration describes a dynamic relationship that exists between objects. Additionally, a ¦ Classifier Role should be associated to the collaboration to illustrate the role an element plays in that collaboration.

Comment



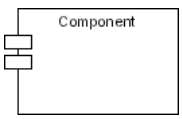
A comment is a textual annotation attached to an element or a collection of elements that has no direct semantics, but may display semantic information.

Previously referred to as a note.

Compartment

A compartment is a division of a symbol, such as a class rectangle divided vertically into smaller rectangles. Each compartment shows the properties of the represented element.

Component



A component is a replaceable, tangible part of a system that realizes of a set of interfaces, including software code, scripts, or command files, run-time objects, documents, databases, etc.

Composition

A composition is a stronger form of aggregation. A part can only be a part of one composite, and the destruction of the whole automatically implies destruction of the parts. Parts with multiplicity that is not fixed can be created after the composite has been created, but once established they live and die with it. Parts can be explicitly removed before the death of the composite.

Constraint

Constraints are expressions that represent semantic conditions or restrictions that are used to limit the use of model elements.

Constructor

A constructor is an operation within the scope of a class that creates and initializes an instance of a class. It may be used as an operation stereotype.

Container

A container is an object that exists to encompass other objects and provide operations to access or iterate over its contents. Examples of containers include arrays, lists, and sets.

Contains

A ’contains’ relationship is used to describe a composition relationship; for example, an airplane contains wings.

Control Flow

Control flow represents the relationship between actions in a sequence as well as between input and output objects, shown with messages attached to associations or as solid arrows between activity symbols.

Dependency

A dependency exists between elements and expresses that elements within one package use elements from the package on which it depends, implying that a change in one element may affect or supply information needed by the other element.

Details Pane

The Details pane is a quadrant of the Poseidon work area, located in the lower right corner, which provides advanced editing and viewing capabilities for all elements.

Descriptor

A descriptor is a model element that describes the commonalities of a set of instances, including their structure, relationships, behavior, constraints, and purpose. Most elements in a model are descriptors.

Diagram

A diagram is a graphical presentation of a compilation of model elements, rendered as a graph of shapes connected by paths. Comprehension resides mainly in the topology, not in the size or placement of the symbols.

Diagram Pane

The Diagram pane is the main working area of Poseidon, where all of the diagrams are displayed.

Drill-down Navigation

Drill-down navigation is a means of moving through a model by moving from element to element via the relationships of those elements.

Element

An element is a broad term with little in the way of specific semantics and refers to an atomic constituent of a model.

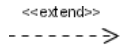
Evaluation Key

An evaluation key is a key granted to a user upon request to allow that user to operate Poseidon for a limited amount of time.

Event

An event is a non-trivial occurrence with a location in time and space.

Extend



An extend relationship exists between an extension use case and a basic use case, and indicates how the behavior of the extension use case can be directly applied to the behavior defined for the base use case. The extension use case incrementally modifies the base use case in a modular way.

Extension Point

An extension point is a named marker that references a location or set of locations within the behavioral sequence for a use case, at which additional behavior can be added.

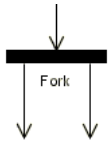
Final Key

A final key is a string provided to Poseidon in order to remove time limits and functionality limits from an evaluation copy of the software.

Final State

A final state is a state within a composite state that, when active, indicates that the activity of the enclosing composite state is complete.

Fork

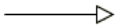


A fork is a complex transition where one source state is replaced by two or more target states, thus increasing the number of active states.

Friend

A friend dependency grants an operation or class permission to use the contents of a class where there would otherwise be insufficient permission.

Generalization



A generalization is a directed relationship between two like elements, where one element is the parent element and the other is the child element. This type of relationship is also referred to as ’kind of’, meaning that the child is a kind of the parent.

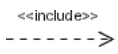
Guard Condition

A guard condition is a boolean expression that must be satisfied in order to enable an associated transition to fire.

Implementation Relations

An implementation relation is a relation that exists only between interfaces and classes.

Include



An include relationship defines a dependency relationship between a source use case and a target use case in which the source use case explicitly incorporates the target use case. The source use case can see and use the target, but neither the source nor the target may access each other’s attributes.

Multiple include relationships may be applied to the same base use case. The same target use case may be included in multiple source use cases.

Inheritance Relations

An inheritance relation allows more specific elements to incorporate structures and behaviors that have been defined by more general elements.

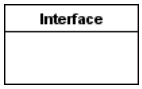
Initial State

An initial state is a syntactic notation indicating the default starting place for an incoming transition to the boundary of a composite state.

Instance

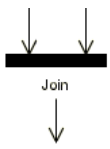
An instance is an individual, concrete entity with its own identity and value. An object is an instance of a class, a link is an instance of an association

Interface



An interface is a named set of operations that characterize the behavior of an element. Interfaces do not have implementations, they lack attributes, states, and associations, they have only operations.

Join



A join is a location in a state machine, activity diagram, or sequence diagram where two or more concurrent threads or states combine into one thread or state.

Label

A label is a notational term for the use of a string on a diagram.

Link

A link is an instance of an association.

Lollipop

A lollipop is a type of notation used to denote an offered interface. It consists of a named circle (the interface) and an relationship drawn as a solid line. This is also known as ball notation.

Merge

A merge is a location in a state machine, activity diagram, or sequence diagram where two or more control paths come together.

Message

A message refers to the transfer of information, such as a signal or operation call, from one object to another with the expectation that activity will result. The receipt of a message instance is normally considered an instance of an event.

Metaclass

A metaclass is class whose instances are classes. Metaclasses are typically used to construct metamodels. A metaclass can be modeled as a stereotype of a class using the keyword metaclass.

Method

A method is an implementation of an operation that specifies the algorithm or procedure.

Model

A model is semantically complete abstraction of a system from a particular viewpoint.

Multiplicity

A multiplicity is a specification of the range of allowable cardinality values. It can be an explicit value, a range of values, or an expression that resolves to one or more values.

Name

A name is a string that is defined within a namespace and is used to identify a model element.

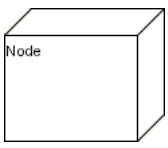
Namespace

A namespace is a part of the model in which names are defined and used, where each name has a unique meaning.

Navigation Pane

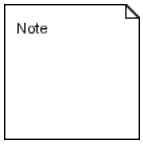
The Navigation pane is located in the top left corner of the Poseidon work area and displays model elements according to pre-determined schemas which can be selected from a dropdown menu.

Node



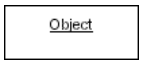
A node is a physical object that exists at runtime and represents a computational resource that executes components. It usually has at least a memory and often processing capability. Nodes can include, but are not limited to, computing devices, human resources, or mechanical processing resources.

Note



The Note element has undergone a name change. See ’Comment’.

Object

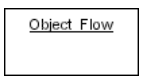


An object is a discrete entity with a well-defined boundary and identity that encapsulates state and behavior, an instance of a class.

Object Constraint Language (OCL)

Object Constraint Language (OCL) is a text language for specifying constraints, writing navigation expressions, boolean expressions, and other queries. It is not intended for writing actions or executable code

Object Flow State



An object flow state represents the existence of an object at a point within a computation. It can also represent the flow of control among operations in target objects.

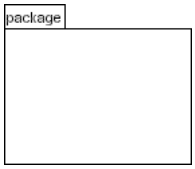
Operation

An operation is the specification of a transformation on the state of an object or a query that returns a value to the caller of the operation.

Overview Pane

The Overview pane is located in the bottom left-hand corner of the Poseidon application and helps the user keep the big picture in mind. It consists of the birdview tab and the critiques tab.

Package



A package, like a file directory, is a general way to put like things together to provide organization. Packages may be nested within other packages.

Parameter

A parameter is the placeholder for an argument that can be changed, passed or returned.

Path

A path is a graphical connection between symbols, usually used to show a relationship.

Plug-in

A Plug-in is a piece of code that extends the capabilities of Poseidon. It may or may not be authored by Gentleware.

Plug-in Key

A Plug-in Key is the string given to Poseidon to activate the Plug-in.

Port

A Port is a connectable element that specifies a set of required and provided interfaces.

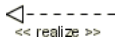
Profile

A profile takes a part of the UML and extends it with stereotypes for a particular purpose.

Project

A project is saved as a zipped .zuml file and contains all information regarding the model, both textual and graphical.

Realization



A realization is the relationship between an element that specifies behavior and one that provides an implementation. A specification describes a behavior or structure without specifying how the behavior will be implemented. An implementation provides the details about how to implement behavior in an effective way.

Relationship

A relationship is reified semantic connection among model elements. Types of relationships include association, generalization, and dependency.

Role

A role is a named slot within an object structure that represents the behavior of an element as it participates in a given context (in contrast to the inherent qualitites of the element).

Socket

A socket is a notation for a required interface. It is denoted as a semi-circle.

Specialization

A specialization produces a more specific description of a model element by adding children. A child element is the specialization of a parent element.

State

A state is a condition or situation during the life of an object during which it satisfies a condition, performs an activity, or waits for an event.

Stereotype

A stereotype characterizes a type of element without specifying its implementation and assists in teh creation of a new model element that is derived from an existing model element.

Synchronization State

A synchronization state is a special state that synchronizes control between two concurrent regions in a state machine.

System

A system is collection of connected units organized to accomplish a purpose. A system can be described by one or more models, possibly from different viewpoints.

Tagged Value

A tagged value is consists of a tag-value pair and is attached to an element to hold some piece of information.

Transition

A transition is a relationship between two states within a state machine where an object in the first state will perform one or more actions and then enter the second state when a certain event occurs and guard conditions are satisfied.

Trigger

A trigger is an event whose occurrence makes a transition eligible to fire.

Type

A type is a declared classifier that the value of an attribute, parameter, or variable must hold. The actual value must be an instance of that type or one of its descendants.

Use Case

A use case defines a piece of behavior of a classifier without revealing its internal structure by describing the behavior of a system from a user's standpoint, providing a functional description of a system and its major processes, and providing a graphical description of users and interactions.

View

A view is a collection of diagrams that describe a particular aspect of the project.

Visibility

Visibility refers to an enumeration whose value determines whether a model element may be seen outside its enclosing namespace.