

# **BOUML**

## **User Manual**

# BOUML

BOUML is a UML tool box allowing you to specify and generate code in C++, Java, Php, Python and IDL.

The main key points of BOUML are :

- it is free
- it runs under Linux, Sun Solaris, MacOS X and Windows
- it allows to program simultaneously in C++, Java, Php, Python and IDL
- thank to a full access to the generated forms, you are the master and you decide what must be generated
- it is extensible, and the external tools (I name them *plug-out* because they are executed outside BOUML) may be developed in C++ or Java, using BOUML for their definition as any other program (clearly I do not like very much language like *Visual Basic*, and I do not understand how this one can be imposed in a UML environment !)
- it is fast and doesn't need a lot of memory to manage several thousands of classes, see [benchmark](#)

To develop BOUML alone at home during the nights and weekends is hard and tiring ... perhaps crazy ! The reasons inciting me to do it are mainly : the pleasure to develop freely a tool and I hope to see it used by many people (I already had this very plea experience with two friends for [Xcoral](#)), and frankly a reaction to the very expensive price of the commercial UML tools.

The graphic part of BOUML is based on **Qt**, many thanks to [Trolltech](#) for this marvelous GUI application framework, clearly BOUML exists thanks to Qt. The implementation of BOUML is compatible with Qt version 2.4 up to (at least) the 3.3, but the releases can't be used because thee releases are compatible with the previous.

The documentation is written with [OpenOffice.org](#) writer, many thanks for this powerful and pleasant office suite. The screen copies are made with [Ksnapshot](#) and sometimes modified by [Le Gimp](#) to have burred areas, thanks to the authors for these practical tools.

Some of the examples given in this documentation spoke about the organs, describing an organ is very easy for me in French ... not in English ! I get the English translation of the words associated to the organs from a [Marya J. Fancey](#) documentation, thanks to her.

## Copyright

BOUML is a free software distributed on the *Internet* (<http://bouml.free.fr/>)

Copyleft 2004-2008 by Bruno Pagès

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the [GNU General Public License](#) for more details.

The complete text of the GNU General Public License can be found in this document appendix.

Copying and distribution of the documentation or a part of the documentation and screenshots is permitted worldwide in any n provided a mention of *Bouml* or <http://bouml.free.fr> However, following the shocking behavior of administrators of Wikipedia, the distribution on all the Wikipedia sites of screenshot any part of the documentation is forbidden without autorisation.

## Apologies :-))

First I apologize for the poor quality of my English, and secondly it is clear that BOUML is not developed following all standard object principles UML help to diffuse, this is a little bit strange for an UML tool...

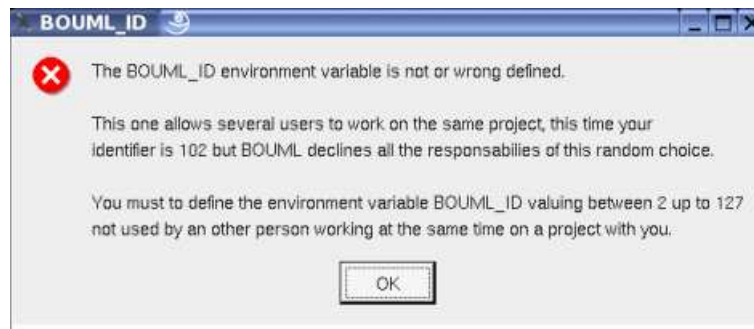
Nevertheless I hope the documentation is enough clear and consistent, and I am sure that BOUML is fast and not glutton in memory !

At least, this document describe BOUML, **not** the Unified Modeling Language.

Next : [Starting](#)

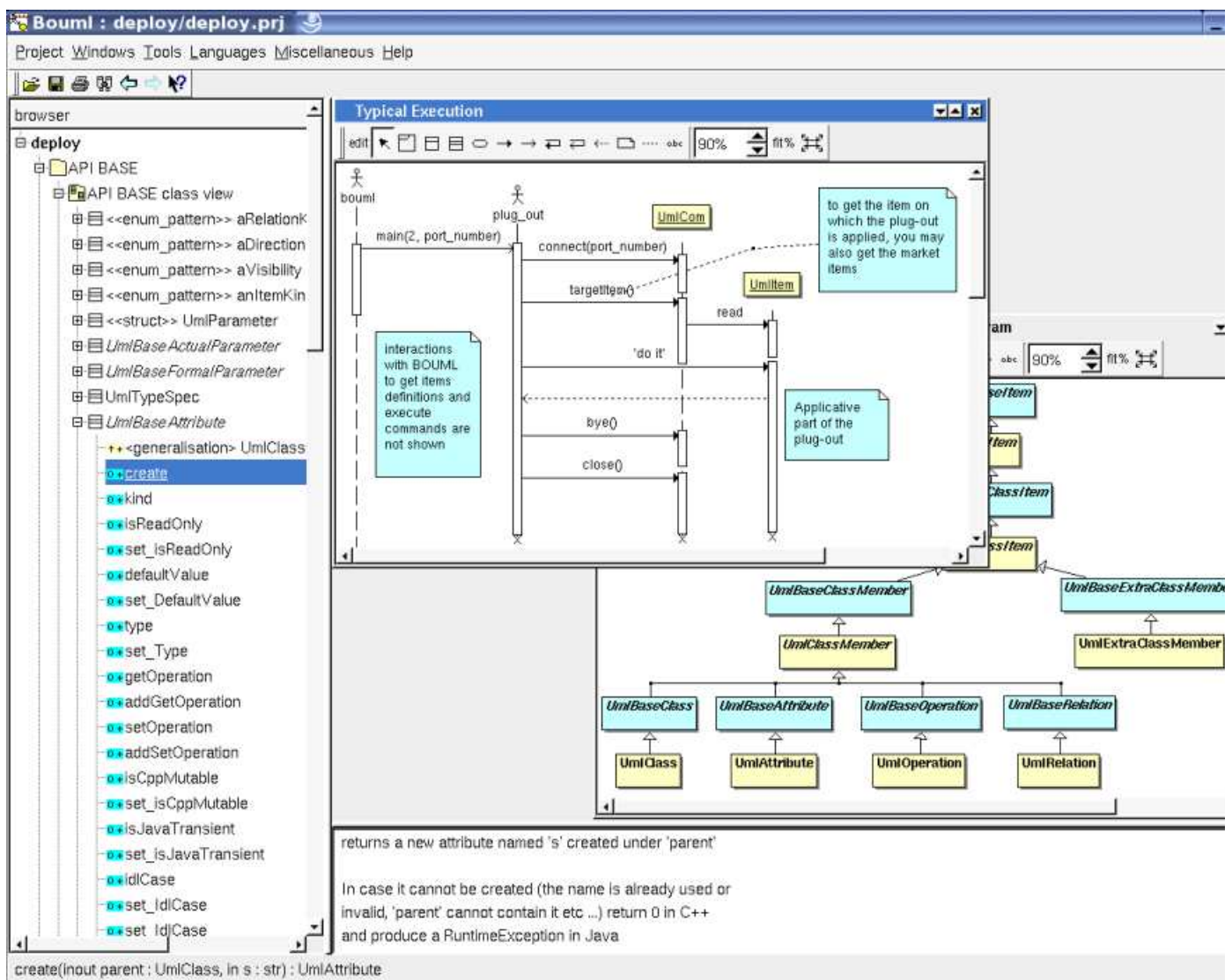
# Starting

When you execute BOUML for the first time, this one ask you to have an own identifier between 2 up to 127 through the environment variable BOUML\_ID ([Multi users considerations](#)) :



I insist : the better way is to exit BOUML after the dialog acknowledge, to define BOUML\_ID then to restart BOUML.

After this annoyance you may use BOUML, [open](#) an already created project, or [create](#) a new one.



The *bouml* window is composed of three parts :

- The left sub-window display a browser presenting your project, the navigation may be done by the mouse or the keyboard arrows. The bold font is used when an item is modifiable, an item is *read-only* when you do not have the write permission for the file(s) supporting it (see [project files](#)). The *system* items supporting the [plug-out's](#) API (for instance *UmlBaseActor*) are also *read-only*.
- The bottom-right sub-window is used to display/modify the comment associated to the current selected item.

- The top-right part is used to display/modify the diagrams, these ones may be maximized or minimized.

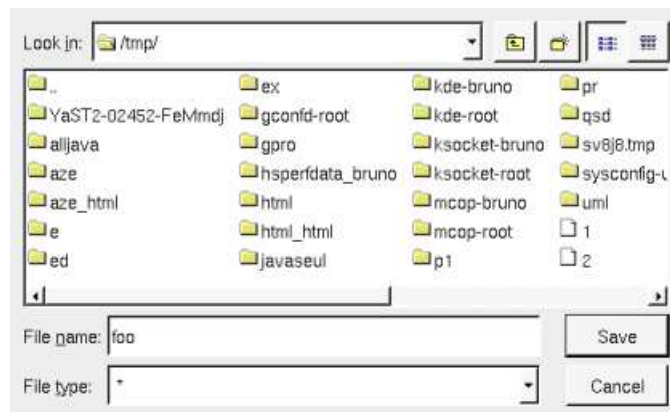
Notes :

- if you have at least a dual monitor configuration the better is to set the environment variable BOUML\_LIMIT\_DESKTOP, see [here](#).
- if you use non ISO\_8859-1 (latin1) characters, for instance Cyrillic, set the environment variable BOUML\_CHARSET, see [here](#).
- it is possible to start a *plug-out* when Bouml is launch, see [here](#)

## Project creation

A project creation is done through the entry *new* of the [project menu](#).

When you create a new project, a file dialog appears (its aspect depend on the used system and window manager) and you have to select the directory where the project will be placed and its name, for instance in */tmp* (probably just for a trial !) with the name *foo*.



In this case BOUML create the directory *foo* under */tmp*, and place some files in */tmp/foo* including *foo.prj* which represents the project (see [project files](#)) :



Do **not** rename or delete the files produced by BOUML nor the directory itself !

## Project loading

You may associate a *.prj* file to *bouml* using your window manager capabilities, call *bouml* through a command line given the *.prj* file (see [project files](#)) in argument, or launch *bouml* then load a project using the [project menu](#) or the open icon, or the historic (the historic is saved in the file *.bouml* placed on your home directory) :



N <u>ew</u>	Ctrl+N
 <u>O</u> pen	Ctrl+O
 <u>S</u> ave	Ctrl+S
Save as...	
 <u>P</u> rint	Ctrl+P
<u>C</u> lose	Ctrl+W
<u>Q</u> uit	Ctrl+Q
/tmp/foo/foo.prj	
/home/bruno/uml/html/html.prj	
/home/bruno/uml/pr/pr.prj	
/home/bruno/uml/singleton/singleton.prj	
/tmp/b/b.prj	
/home/bruno/uml/rose/rose.prj	
/home/bruno/uml/genpro/genpro.prj	
/tmp/es/es.prj	
/home/bruno/uml/empty/empty.prj	
/tmp/b	
/tmp/em/em.prj	

Previous : [bouml](#)

Next : [top level menus](#)

# Top level menus

The menus accessible from the BOUML's window are :

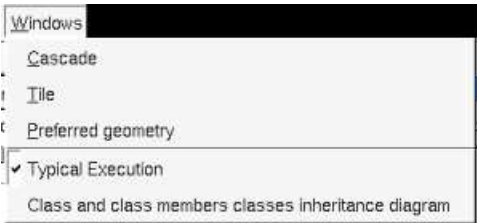
## Project menu :



The *Project* menu allows to create a new project, load a project, save it in place or in a new location, print the current diagram, close the project, or quit *bouml*. The bottom of the menu contains the historic, the upper line correspond to last opened project, for instance the current one. When you try to close a modified project, perhaps to load an other one, BOUML ask for a saving.

If the [environment variable](#) BOUML\_TEMPLATE is defined, you can create a project from the one specified by this variable. The goal is to get the settings from this project rather than to get the default settings I decide to set. This variable must value the absolute path of a project file (extension .prj), and probably this project is empty. In this case, when you choose *create from template* the specified project is load and a *save-as* is automatically done.

## Windows menu :



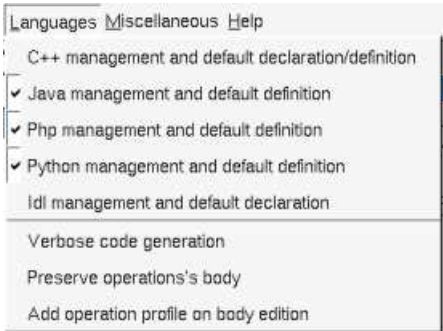
The *Windows* menu allows to select an opened diagram or change the sub-windows disposition.

## Tools menu :



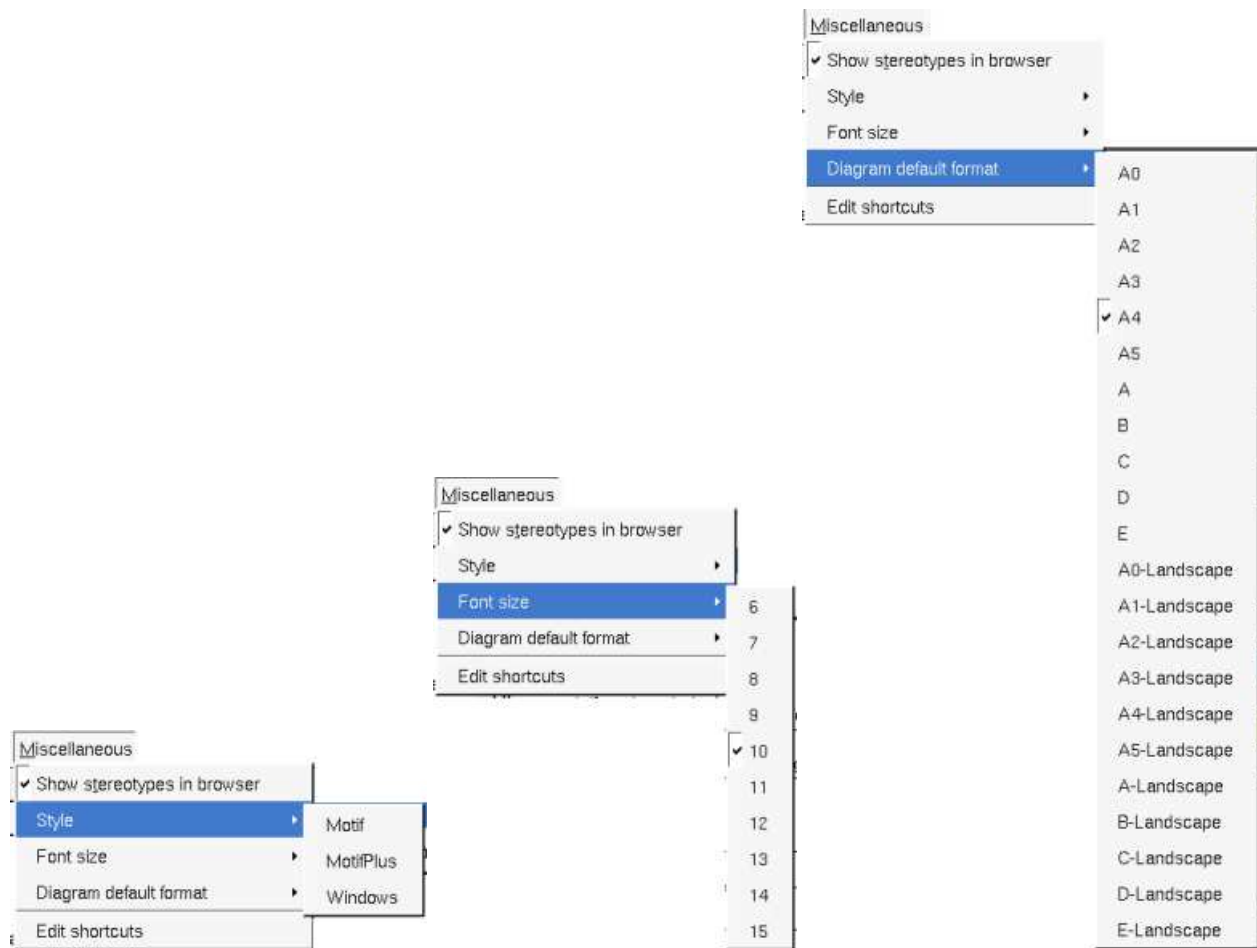
The *Tools* menu allows to (re)open the trace window where the *plug-outs* write their messages, to apply a *plug-out* on the project (independently of the browser item currently selected), to add/remove/modify the known *plug-outs* list (see [plug-out](#)), or to import the *plug-outs* list defined in an other project.

**Language menu :**



The *Languages* menu allows to ask BOUML to produce or not a default definition/declaration (refer to the [C++ generation](#), [Java generation](#), [Php generation](#), [Python generation](#) and [Idl generation](#)) when a browser item is created, and to select for which language(s) the dialogs shown through tabs. This menu also allows to update or not the body of the operation during the cod generation (refer to the [C++ generation](#), [Java generation](#), [Python generation](#) and [Php generation](#)), to produce or not the operation's profile when you edit an operation's body through an external editor (refer to [operation](#) edition).

**Miscellaneous menu :**

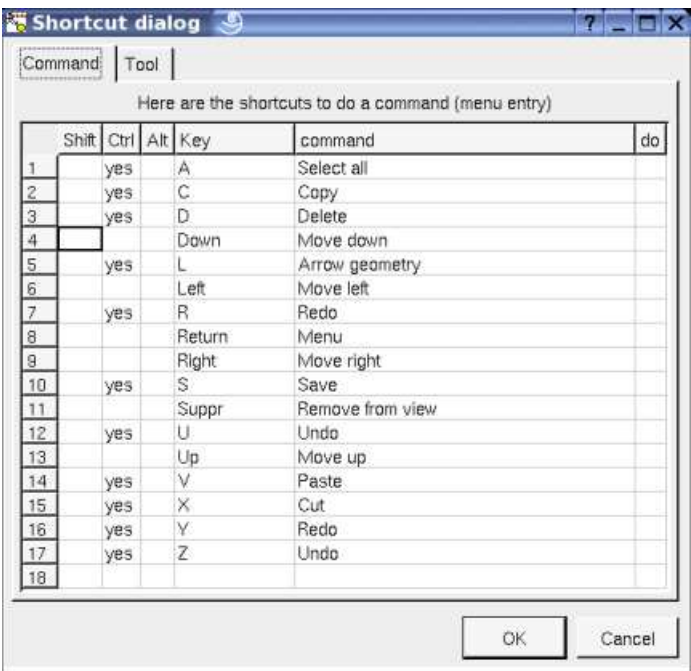


The *Miscellaneous* menu allows to show/hide the stereotypes in the browser, to change the used style, to change the size of the font, to set the default diagram format, and to edit the shortcuts.

Note that sometimes changing the size of the base font does nothing because the desired font is not available. I had introduced this feature mainly because the default size under Linux and Windows is not the same, this may change the relative position of the objects and a relation drawn vertically with one font will be inclined with another. Nevertheless because the fonts under Linux and Windows are not strictly the same, a diagram can't have the same aspect under the two operating systems. This size is memorized in the [file](#) memorizing the project, but BOUML allows to change the font size even when the file is read only (the new font size can't be saved).

Shortcut edition

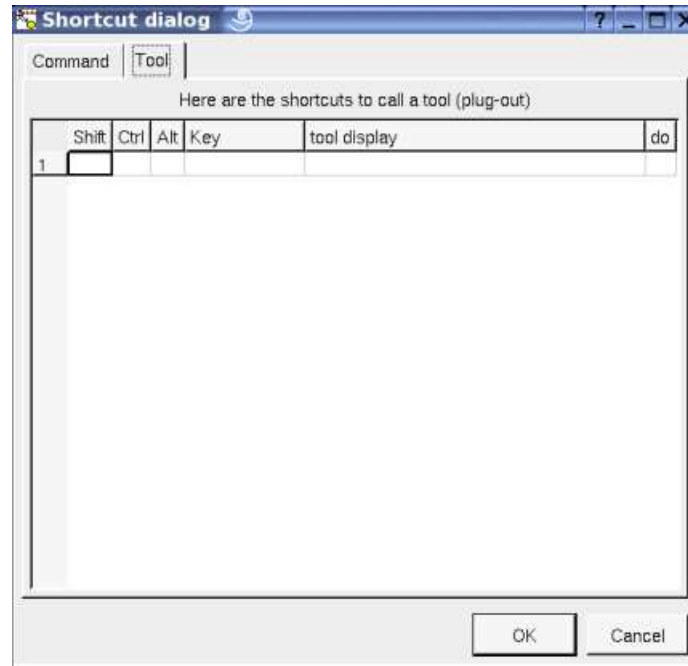
When you select *Edit shortcuts* a dialog appears :



A shortcut is a key and the optional modifiers *shift*, *control* and *alt*. Under MacOS X *control* is replaced by the apple (■) and *alt* is named *option* on the old keyboards.

This first tab allows to define shortcuts to apply a command on the selected element(s). The available commands are proposed on mouse click in the column *command*, their correspond to the menu entries. In the previous picture you see the default shortcuts.

The second tab allows to define shortcuts to apply a *plug-out* :



A *plug-out* is specified by its display specified through the *plug-out editor*. The available *plug-outs* are proposed on a mouse click in the column *tool display*.

## Icons bar



These buttons allow to :

- **open** a project
- **save** the current project
- **print**  
the active diagram, a first menu appears to choose the printer or print in a file and set properties, then a second dialog ask you to print the diagram on 1, 4 or 9 pages
- **search** for  
an element through its name and may be its kins, or to search for a string in the declarations/definitions or operation bodies :



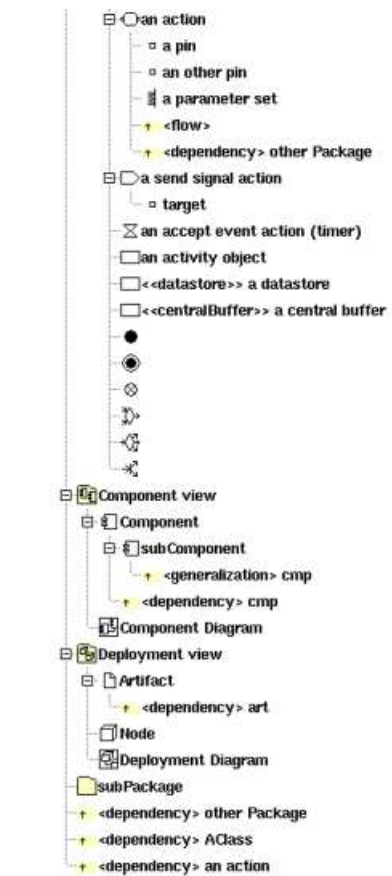
- **navigate** in the historic of the last selected elements in the browser
- **describe**  
the behavior of a button or a menu entry, for that hit the button then the desired button or select a menu entry, a short description will be given. When you maintain the mouse on a button a very small description also appears.

Previous : [starting](#)

Next : [browser items](#)





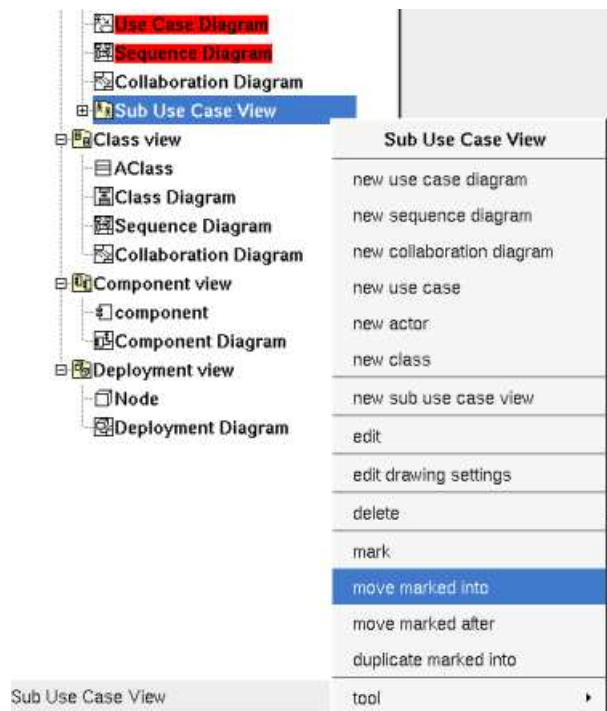


All the browser items may have a description, a stereotype (sometime without special meaning for BOUML, the default stereotype may be defined for some types of item), and [user properties](#). The browser item's stereotype may be showed/hidden through the [Miscellaneous menu](#).


The bold font is used when an item is modifiable, an item is *read-only* when you do not have the write permission for the file(s) supporting it (see [Project files](#)). The *system* items supporting of the *plug-out's* API (for instance *UmlBaseClass*) are also *read-only*, this help you to not modify them to have a chance to remain compatible with the future versions.

When you create a new item, this last is placed at the end of the items of the same container, using the left button of the mouse as for a *drag&drop*. It is possible to move an item to place it in an other container (except for the relations) or to change the items order. The *drag & drop* from the browser to an opened diagram is also possible when it is legal. When you drag an item *it1* on a non-container item *it2*, *it1* will be placed just after *it2* at the same level. When you drag an item *it1* on a container item *it2* which may contain *it1*, *it1* will be the first *it2*'s children, in case *it1* may be moved in *it2* or after *it2*, Bouml ask you to choose.

Multiple selection is not allowed in the browser using the classical way, I consider that it is too simple to lose a multiple selection through a wrong mouse click. I replace classical multiple selection with the possibility to mark browser items through the menu appearing on a right mouse click, or through a *control left mouse click*. Marked items are written on a red background, and may be moved or duplicated (under condition) :



On the example above at the menu appearing on the [use case view](#) propose to mark it, to move the marked items into it, to move the marked items after it (in this case only the order of the items is modified) or to duplicate the marked items into it. The menu is context dependent, for a marked item it proposes to unmark the item, or to unmark all the marked items, and for instance for a [component view](#) it just proposes to mark it because the two marked diagrams cannot be placed into the [component view](#) or after (the marked diagrams cannot be placed into the [package](#) containing the [component view](#)).

When you remove an item from the model, the icon contains a red cross (for instance ) , note that a deleted item is **never** saved when you save your project, but it may be undeleted with or without its children until you close the project. Sometimes it is not possible to delete an item because this one contains read-only items or is referenced by a read-only relation.

A double mouse click on an item allows to edit it except when the item has an associated diagram, in this case the diagram is showed. The other mouse buttons show a menu depending on the item kind.

*control s* is a shortcut to save the project

*control d* and *suppr* are a shortcut to delete the selected item

## Package

A Package (may be the project itself) is first something like a directory and may contain :

- packages
- [use case views](#)
- [class views](#)
- [component views](#)
- [deployment views](#)
- relations

A package may indicate where the code must be produced by the generator for each language, and may specify the C++ namespace Java package / Idl module. A generator producing code for an [artifact](#) looks at the package containing the [deployment view](#) where is the artifact to get these information.

Refer to [package](#) for more information.

## Use case view

A use case view may contain :

- sub use case view
- [use cases](#)
- [classes](#), an actor is just a class having the stereotype *actor*
- [class instances](#)
- [states](#)
- [activities](#)
- [use case diagrams](#)
- [object diagrams](#)
- [sequence diagrams](#)
- [collaboration diagrams](#)

Refer to [use case view](#) for more information.

## **Class view**

A class view is used to design the classes and may contain :

- [classes](#)
- [class instances](#)
- [states](#)
- [activities](#)
- [class diagrams](#)
- [object diagrams](#)
- [sequence diagrams](#)
- [collaboration diagrams](#)

Refer to [class view](#) for more information.

## **Component view**

A component view may contains :

- [components](#)
- [component diagrams](#)

Refer to [component view](#) for more information.

## **Deployment view**

A deployment view may contains :

- [nodes](#)
- [artifact](#)
- [deployment diagrams](#)

Refer to [deployment view](#) for more information.

## **Use case**

A use case is ... a use case, but also a browser container and may contain :

- use cases

- [actors](#)
- [classes](#)
- [class instances](#)
- [states](#)
- [activities](#)
- [relations](#)
- [use case diagrams](#)
- [object diagrams](#)
- [sequence diagrams](#)
- [collaboration diagrams](#)

Refer to [use case](#) for more information.

## Class

A *class* is the support to define a *class* (!), an *interface*, an *enum*, an *union*, a *struct*, an *exception*, or a *valuetype*, depending on the language. The stereotype specify what a class is, the way a stereotype is translated for each language may be specified through the [generation settings](#)

Of course a class is also a browser container and may contain :

- (nested) classes
- [relations](#)
- [attributes](#)
- [operations](#)
- [extra members](#)

The order of the class's sub-items is very important because it is followed by the C++/Java/Idl generators.

Refer to [class](#) for more information.

## Relation

They are two kinds of relation : the [relations between classes](#) whose are considered by the source code generators, and the relations between the other items (inheritance and dependency)

To add a relation you have to use a diagram (obviously the relation drawing may be deleted just after !), it is not possible to add relation through the browser. It is also not possible to move a relation out of the items containing it.

The icon in the browser indicate if a relation between classes is public (+), protected (#) or private (-). The *protected* visibility is translated in *private* in Idl.

A bi-directional relation between classes is supported by two items in the browser.

Generally the stereotype of a relation between classes is used to specify how a relation with a multiplicity different than 1 is translated in each language (refer to [default stereotype](#) and [generation settings](#)), for instance to produce a vector. The *friend* stereotype is a special case for a *dependency* and is used to produce a C++ *friend* declaration.

Refer to [relation](#) for more information.

## Attribute

There is not restriction for the attributes, an attribute's type may be a class or an *int* for instance.

The icon in the browser indicate if the attribute is public (+), protected (#) or private (-). The *protected* visibility is translated in *private* in Idl.

Refer to [attribute](#) for more information.

## Operation

The icon in the browser indicate if the operation is public (+), protected (#) or private (-). The *protected* visibility is translated in *private* in Idl.

Refer to [operation](#) for more information.

## Extra member

An extra member is an alien allowing you to specify directly the generated code for each language. This allows to manage non U items, for instance C++ pre-processor directives, Java attribute initializations etc ...

Refer to [extra member](#) for more information.

## Class instance

A *modeled* class instance, a [sequence diagram](#) , [object diagram](#) or [collaboration diagram](#) may contain *modeled* class instances and *graphical* classes instances (out of the model).

Refer to [class instance](#) for more information.

## State

A *state* represents a *state machine*, a *sub machine*, a *composite state* or a *simple state*

A *state* may contains :

- [state diagrams](#)
- sub-states
- regions
- pseudo states
- transitions
- [actions](#)
- receive signals (an action with the stereotype *receive-signal*)
- send signals (an action with the stereotype *send-signal*)

Refer to [state](#) and for more information.

## Pseudo State

A pseudo state may be :

- final state (for reasons of simplification)
- initial
- ⊞ deep history
- ⊞ shallow history
- ➞ join
- ➞ fork
- junction
- ◇ choice
- entry point
- ⊗ exit point
- ⊗ terminate




A *pseudo state* may contains :

- transitions

Refer to [pseudo state](#) for more information.

## Activity


An activity may contains:

- [activity diagrams](#)
-  parameters
- activity nodes
- expansion and interruptible activity [regions](#)

Refer to [activity](#) for more information.

## Expansion region

An expansion region may contains :

- expansion and interruptible activity sub [regions](#)
-  expansion nodes
- [control nodes](#)
- [action nodes](#)
- [object nodes](#)

Refer to [region](#) for more information

## Interruptible activity region





An interruptible activity region may contains :

- expansion and interruptible activity sub [regions](#)
- [control nodes](#)
- [action nodes](#)
- [object nodes](#)

Refer to [region](#) for more information

## Activity node


An activity node may be :

 [activity action](#) (opaque,  accept event,  accept timer event, read variable value, clear variable value, write variable value, add variable value, remove variable value, call behavior, call operation, send object,  send signal, broadcast signal, unmarshall or value specification)

 [object nodes](#)

[control nodes](#) (  initial,  final,  flow final,  merge,  decision,  fork,  join)

Activity nodes may contain :

- [flow](#)
-  pins (action only)

Refer to [activity action](#), [object node](#) and [control node](#) for more information.

## Component

A component is an autonomous unit with well defined interfaces that is replaceable within its environment.

A component may contains :

- relations
- components

Refer to [component](#) for more information.

## Artifact

Artifacts are basically used to produce source file(s) when the stereotype is source, or to specified how libraries or an execut composed.

A *source* artifact is (potentially) associated to classes, but you may directly give its source(s) contain for instance to define the C++ *main* function. The other artifacts may be associated to artifacts.

Note : In the old release of BOUML (up to the 1.5.1), the behavior of the [artifact](#) was defined at the component level.

An artifact may contains :

- relations

Refer to [artifact](#) for more information.

## Node

To specify how the deployment is made, a node may represent a CPU, a device etc ...

Refer to [node](#) for more information.

## Use case diagram

Refer to [use case diagram](#) for more information.

## Sequence diagram

Refer to [sequence diagram](#) for more information.

## Collaboration diagram

Refer to [collaboration diagram](#) for more information.

## Class diagram

Refer to [class diagram](#) for more information.

## Activity diagram

Refer to [activity diagram](#) for more information

## State diagram

Refer to [state diagram](#) for more information

## Object diagram

Refer to [object diagram](#) for more information

## Component diagram

Refer to [component diagram](#) for more information.

## Deployment diagram

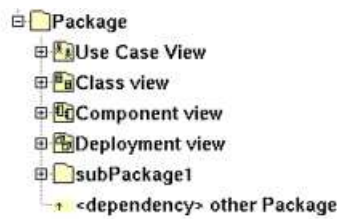
Refer to [deployment diagram](#) for more information.

Previous : [top level menus](#)

Next : [package](#)

# Package

A Package is first something like a directory and may contain other packages, [use case views](#), [class views](#), [component views](#) and [deployment views](#), and dependencies, in any order :



A package allows also to indicate where the code must be produced by the generator for each language, and to specify the C namespace / Java package / Python package / Idl module. A generator producing code for a component looks at the package containing the component view where is the component to get these information.

The *project* is in fact the top level package.

The relation between packages are dependencies and inheritances. Because here the inheritance is very soft, A inherits on B wl contains a class inheriting an other one defined in B, BOUML accept that circular inheritances.

## Menus

The project and package menus appearing with a right mouse click on their representation in the browser are something like supposing them not *read-only* nor deleted :

html
new package
new use case view
new class view
new component view
new deployment view
import project
edit
edit default stereotypes
edit class settings
edit generation settings
edit drawing settings
import
referenced by
mark
generate
reverse
tool

project menu

Package
new package
new use case view
new class view
new component view
new deployment view
import project
edit
edit class settings
edit drawing settings
delete
referenced by
mark
generate
reverse
tool

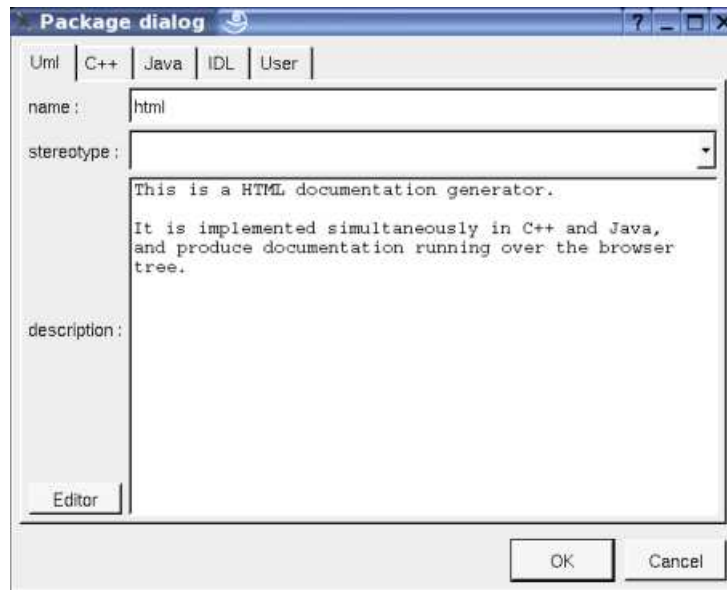
(non project) package menu

### Menu : import project

*import* allows to import a BOUML project into the current package as a standard package with its contain. Note that the imported proje default settings are not imported. It is not possible to import a project formatted by a release of BOUML less than 1.3, then to import an older project you have to reformat it. To do this, load the project to import with a release greater or equal to 1.3, edit the pr package, hit *Ok* (BOUML doesn't check if something is really modified) then save the project.

### Menu : edit

*edit* allows to show/modify the project/packages properties with the following dialog (here Php is not selected in the top level menu *Languages*, so its tab is not visible) :

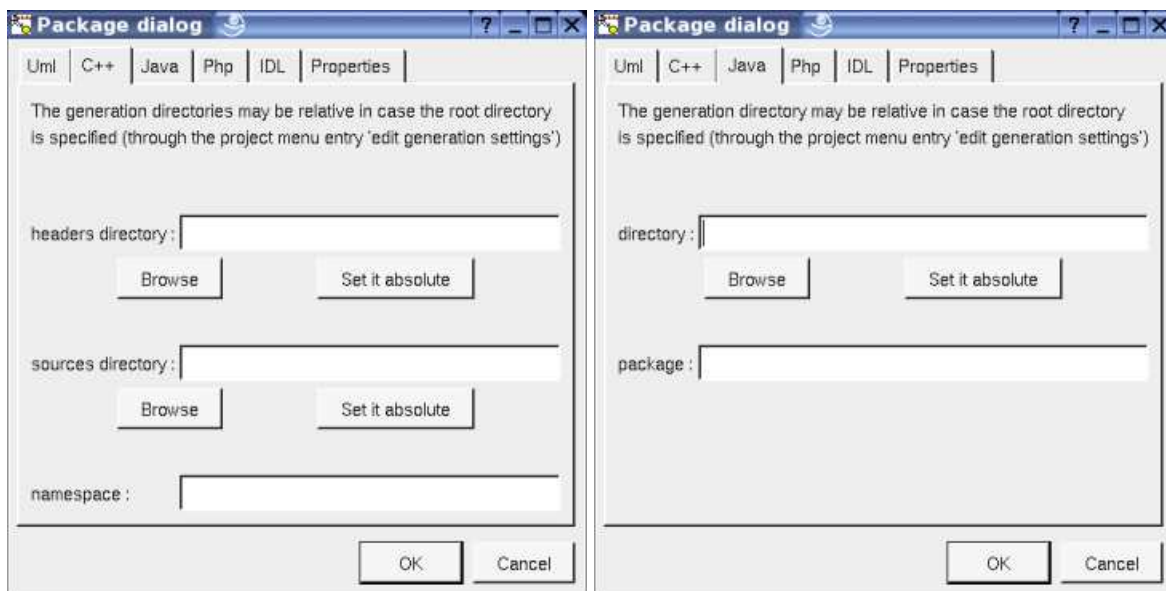


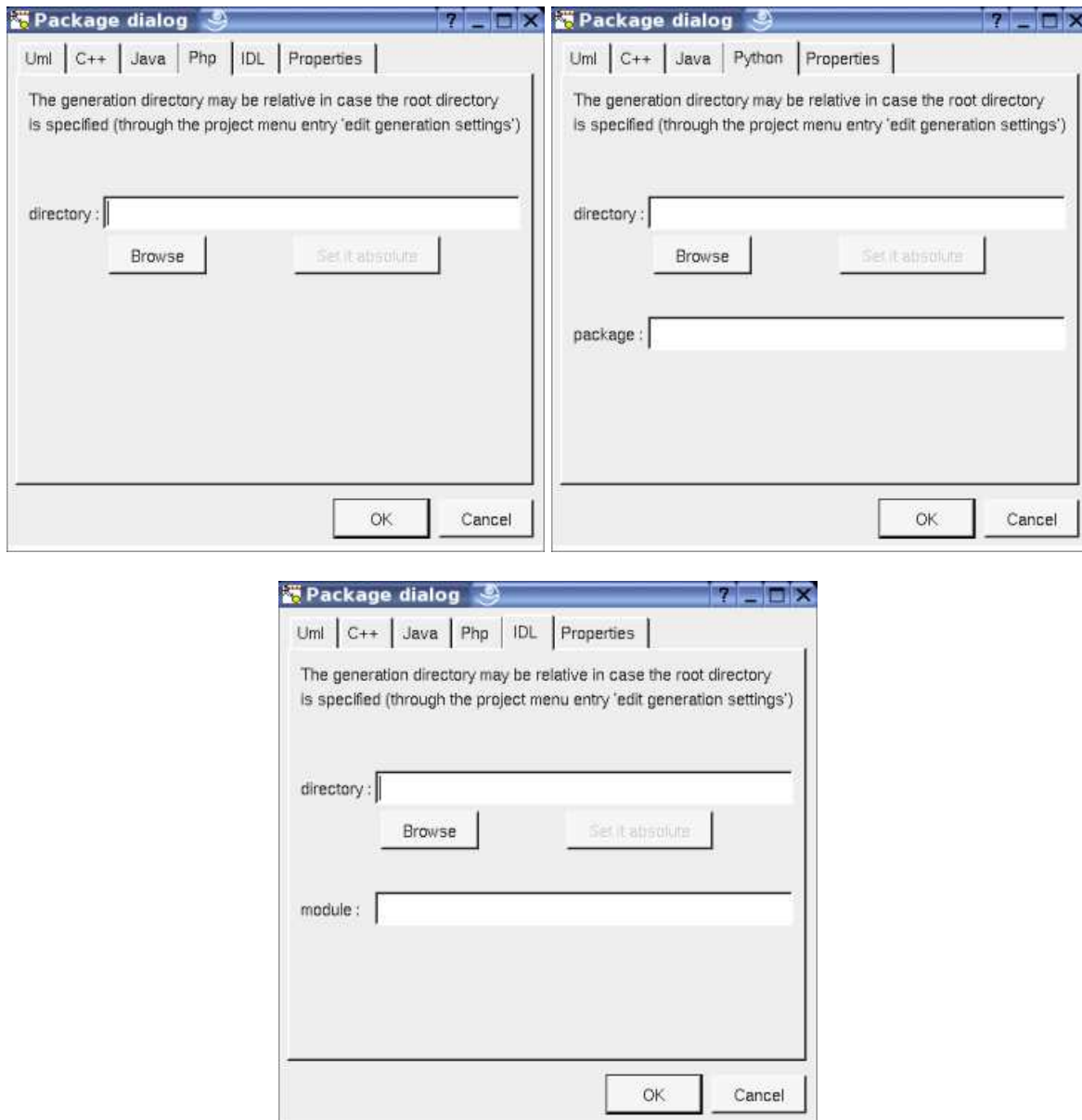
The project's name cannot be changed through this dialog, use *save as* in the [project menu](#).

The *stereotype* of the project and packages doesn't have a special meaning for BOUML.

The *editor* button visible above and associated here to the description, allows to edit the description in an other window, or to call an external editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note that this external editor have to create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel with BOUML which looks each second at the file contents to get the new definition until the dialog is closed (I do not like very much polling but this works even QT isn't compiled with the thread support).

The C++, Java, Php, Python and Idl tabs (visible only if these languages are all set in the *Languages* menu) allow to specify the directory where the generation are made and the *namespace/package/module* :





When a directory specification is empty, the generation is made in the directory specified through the last tab of the [generation settings dialog](#). As it is mentioned in the dialogs, in case a root directory is specified through the [generation settings dialog](#), the directory specifications in the three dialogs above may be relative, the directory(ies) used by the code generators will be the *root* directory collapsed with the appropriate relative directory. This allows to move all the generated files changing only one path.

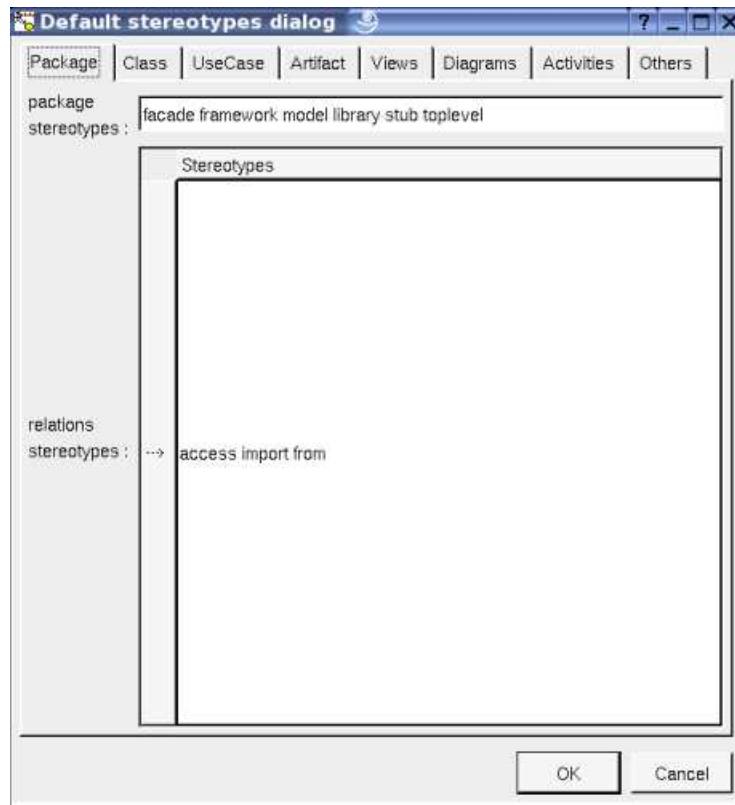
When a *namespace/java* *package/python* *package/module* specification is empty, the generated code will not be placed in a namespace/package/module (!), else you have to specify the full *namespace/package/module* specification : this allows to have a browser's package tree different than the *namespaces/packages/modules* imbrication.

For instance in case the C++ *namespace* is *mynamespace*, the generated code will be placed in the *namespace mynamespace* (!). In case the C++ *namespace* is *mynamespace::subnamespace::subsubnamespace*, the generated code will be placed in the *namespace subsubnamespace* itself nested in the *namespace subnamespace* itself nested in the *namespace mynamespace*. In Java and Python the separator is a “.” rather than “::”. In Idl the separator is “::”.

### **Menu : edit default stereotypes**

This entry is only available in the project's menu. Allows to set a default stereotypes list for some kinds of object, the dialog's associated to the packages is :

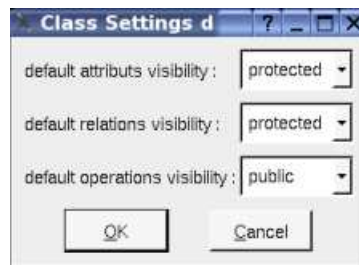




As you can see, it is possible to set a default stereotypes list for the packages and the *dependency* relation starting from a package. In the lists the stereotypes must be separated by a space.

The stereotypes *import* and *from* are specially known by the Python generator and allow to add *import* and *from .. import \** forms.

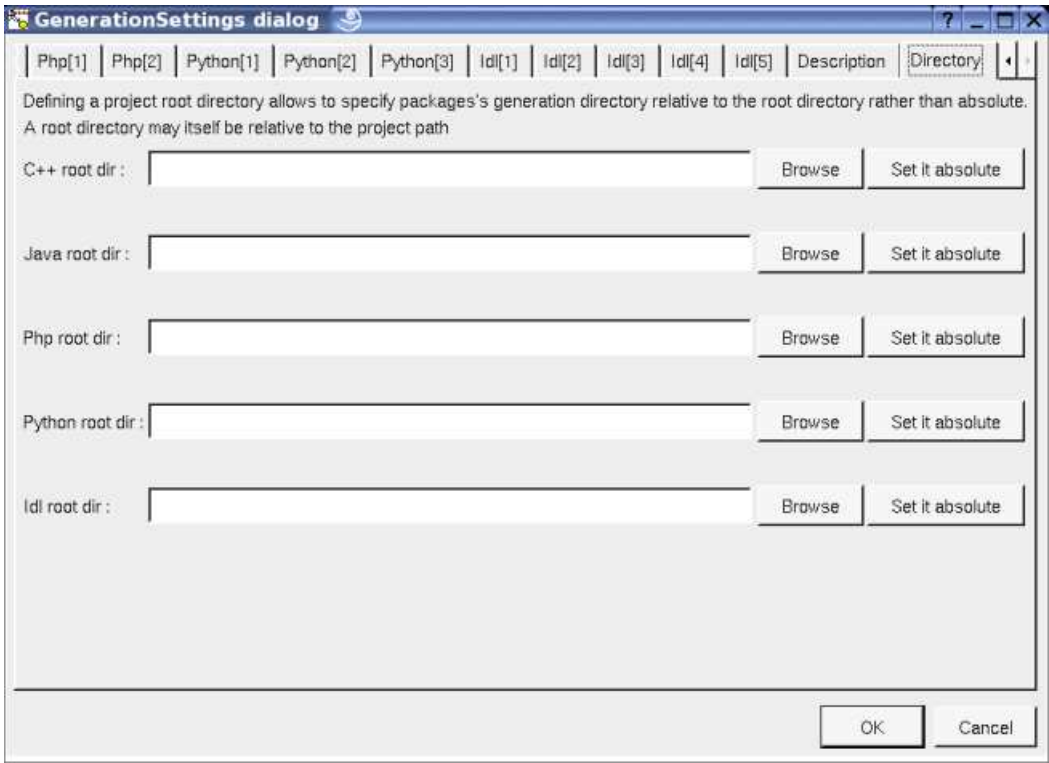
### Menu : edit class settings



As you can see, this setting allows to set a default visibility at the UML level for the attributes, relations and operations. This setting may be redefined in nested packages and class views. At the project level you have to choose between *public*, *protected* and *private*, elsewhere you may also choose *default*, this means that the visibility of the upper level is followed (which itself may be *default* etc ... up to the *project* level). When you create a new package or class view the visibilities are set to *default*.

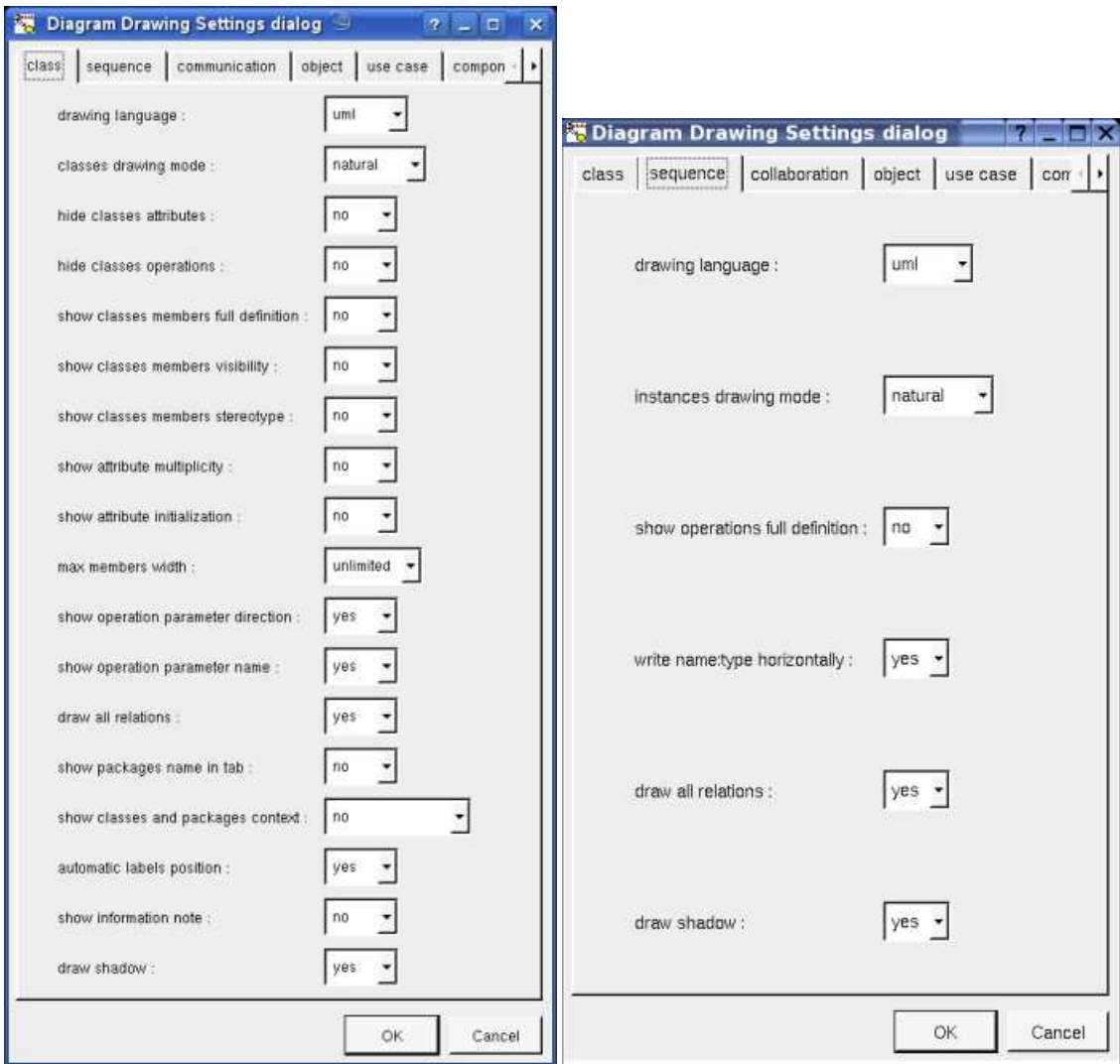
### Menu : edit generation settings

The biggest menu in BOUML, used to specify a default definition/declaration for all the generated forms. This entry is only available in the project's menu, for the package point of view the associated dialog's tab is the last one named *Directory* :



This one allows to specify a *root* directory where the generations will be made, this the possibility to change it at each package level as it was previously said. This *root* directory may be relative to the project. The *browse* buttons call a file browser.

**Menu : edit drawing settings**



**Diagram Drawing Settings dialog**

class | sequence | **collaboration** | object | use case | cor

drawing language : uml

show operations full definition : no

show hierarchical rank : no

write name.type horizontally : yes

show packages name in tab : no

show packages context : no

draw all relations : yes

draw shadow : yes

OK Cancel

**Diagram Drawing Settings dialog**

class | sequence | collaboration | **object** | use case | cor

write name.type horizontally : yes

show packages name in tab : no

show packages context : no

automatic labels position : yes

draw all relations : yes

draw shadow : yes

OK Cancel

**Diagram Drawing Settings dialog**

class | sequence | collaboration | object | **use case** | cor

show packages name in tab : no

show packages context : no

automatic labels position : yes

draw all relations : yes

class drawing mode : actor

draw shadow : yes

OK Cancel

**Diagram Drawing Settings dialog**

sequence | collaboration | object | use case | **component**

show packages name in tab : no

show packages context : no

automatic labels position : yes

draw all relations : yes

draw shadow : yes

draw component as icon : no

show component's required and provided interfaces : no

show component's realizations : no

OK Cancel

**Diagram Drawing Settings dialog**

ase | component | **deployment** | state | activity | color [1]

show packages name in tab : no

show packages context : no

write node instances horizontally : yes

automatic labels position : yes

draw all relations : yes

draw shadow : yes

draw component as icon : no

show component's required and provided interfaces : no

show component's realizations : no

OK Cancel

**Diagram Drawing Settings dialog**

ase | component | deployment | **state** | activity | color [1]

show packages name in tab : no

show packages context : no

automatic labels position : yes

write transition horizontally : yes

show transition definition : no

draw all relations : yes

draw shadow : yes

show state activities : yes

draw state's regions horizontally : yes

drawing language : uml

OK Cancel

**Diagram Drawing Settings dialog**

ase | component | deployment | state | **activity** | color [1]

show packages name in tab : no

show packages context : no

automatic labels position : yes

write flow label horizontally : no

show opaque action definition : no

draw all relations : yes

draw shadow : yes


show information note : yes


drawing language : uml

OK Cancel

**Diagram Drawing Settings dialog**

ase | component | deployment | state | activity | **color [1]** | color [2]


default class color : 

default note color : 


default package color : Transparent


default fragment color : Transparent


default subject color : Transparent


default use case color : 

default duration color : Transparent

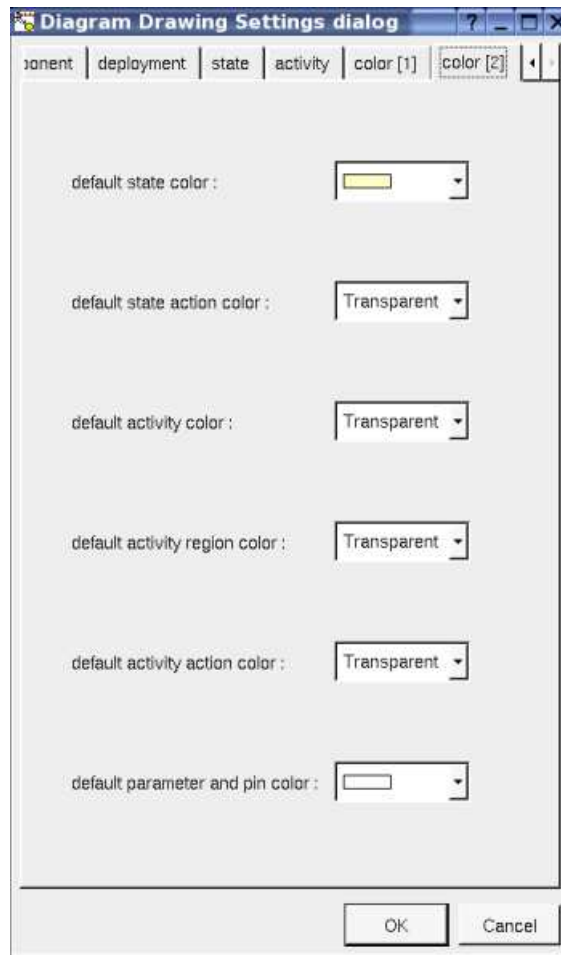
default continuation color : 

default component color : 

default artifact color : 

default node color : 

OK Cancel



This dialog allows to specify how the diagrams must be drawn by default, at the package level you may specify all the drawing settings because a package may contain indirectly all the kinds of diagram. Except at *project* level you may choose *default* for each value, in this case the effective value is specified in the upper level (which itself may specify *default* etc ... up to the *project* level).

Refer to [drawing](#) for the settings concerning the packages.

### **Menu : import**

Allows to import the generation settings (except the specification of the base directories) or the default stereotypes from another project.

### **Menu : delete**

The *delete* entry is only present when the package is not *read-only*. Note that this entry is proposed even the package cannot be deleted because a children at any sub-level is read-only or referenced by a read-only relation, these cases are checked when you ask for the deletion for performance purpose (else the menu may take too much time to appear).

Delete the package and all its children, and all the representation of them in the opened diagrams. After that it is possible to undelete them (from the browser) until you close the project : obviously the deleted items are not saved !

### **Menu : referenced by**

To know who are the packages referencing the current one class through a relation.

### **Menu : mark**

See [mark](#)

### **Menu : generate**

To generate the C++/Java/Idl code of all the sub-items. To generate C++/Java/Idl code for all the project, do it at the project level through the [Tools menu](#). Note that the generators do not re-write a file when it is not necessary, to not change the last modification date of the files for nothing, *make* will appreciate !

Refer to [C++ generator](#), [Java generator](#), [Php generator](#), [Python generator](#) and [Idl generator](#) for more details.

## Menu : reverse

To reverse sources placing the result into the current package.

Refer to [C++ reverse](#), [Java reverse](#), [Java catalog](#) , [Php reverse](#) and [Python reverse](#) for more details.

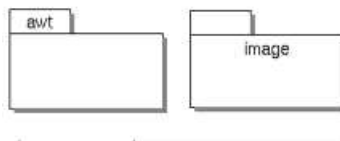
## Menu : tools

The menu entry *tool* is only present in case at least a [plug-out](#) may be applied on the project/package.

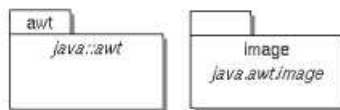
## Drawing

When a package is drawn in a diagram, the available options are :

- *package name in tab* : to indicate if the package's name must be written in the tab which is the small rectangle on the top :



- *show* *classes* *and* *packages* *context* : to indicate if the context where the package is defined must be written, it is not the case just above. The context may be the “UML context” which is the path of the package in the browser (choice followed for *awt* below), or the C++ namespace specified by the package, or the Java package specified by the package (choice followed for *picture* below), or the Python package specified by the package or at least the Idl module :



- *package color* : when you create a new project, the packages are transparent by default.

In all the diagrams, a package may be resized.

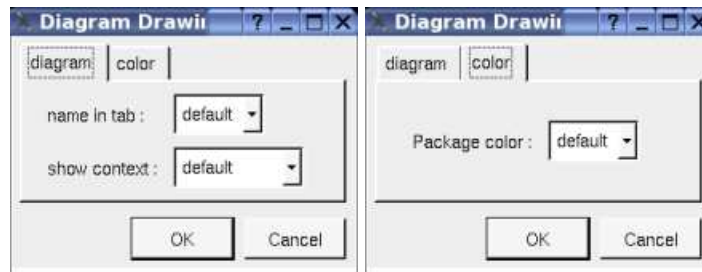
When you move a package in a diagram, the [artifacts](#), [classes](#), [components](#), [deployment nodes](#), sub-packages and [use cases](#) defined in the package (not through an other one) and in collision with the package at the beginning of the move are also moved. If you want also move the connected elements, do a very short move (for instance use the keyboard arrows to do a move and its opposite) then ask for *select linked items* and restart the move. Note : contrarily to the [states](#) when you resize a package the sub-elements are not moved to stay in it.

A right mouse click on a package in a diagram calls the following menu :



The *drawing settings* concerns only the picture for which the menu is called.





*set* allows to automatically open the current diagram when a double mouse click is made on the package representation in a diagram or the browser. After that the only way to edit the package is to choose the *edit* entry in the menu.

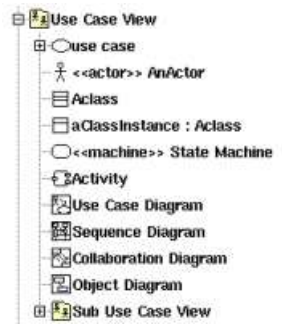
---

Previous : [browser items](#)

Next : [use case views](#)

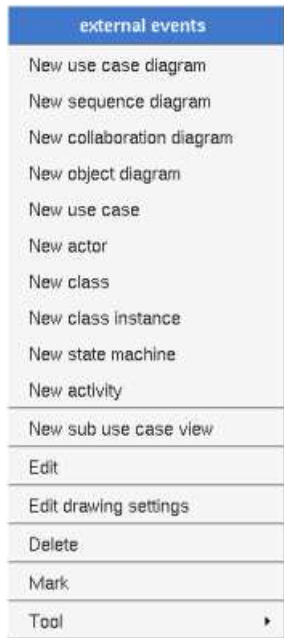
# Use case view

A use case view may contain other use case views, [use cases](#), [actors](#), [classes](#), [class instances](#), [activities](#), [state machines](#), [use case diagrams](#), [object diagrams](#), [sequence diagrams](#) and [collaboration diagrams](#) in any order :



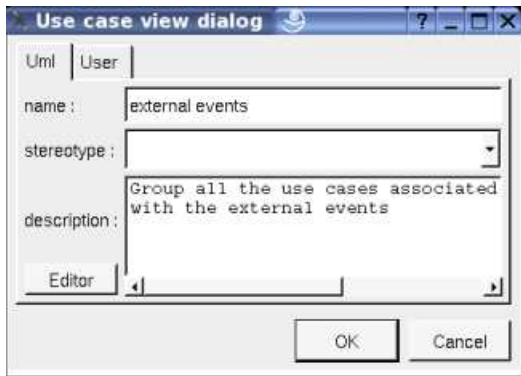
## Menus

The menu appearing with a right mouse click on a use case view in the browser is something like this, supposing it is not *read-only* nor deleted :



### Menu : edit

*edit* allows to show/modify the use case view properties with the following dialog :

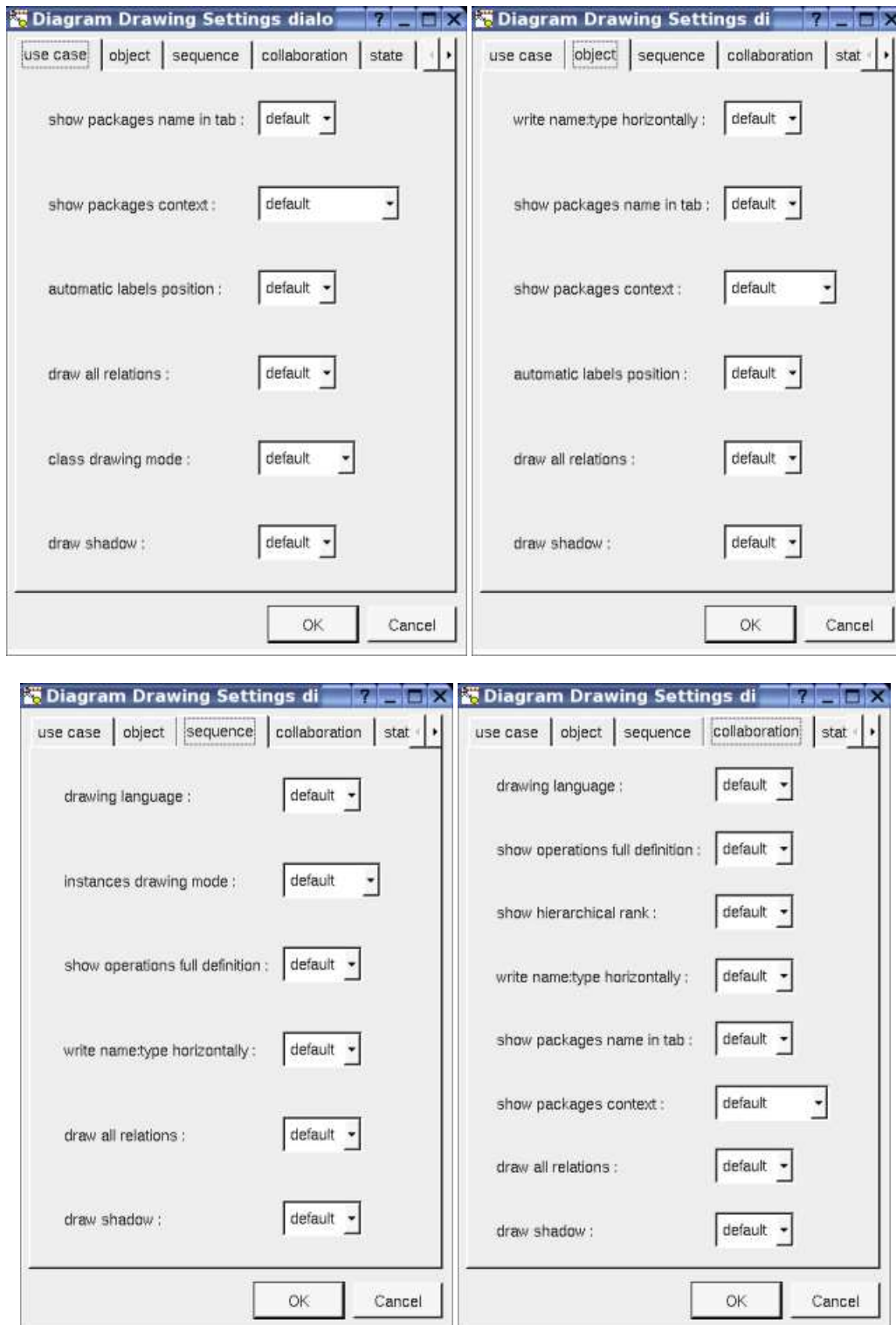


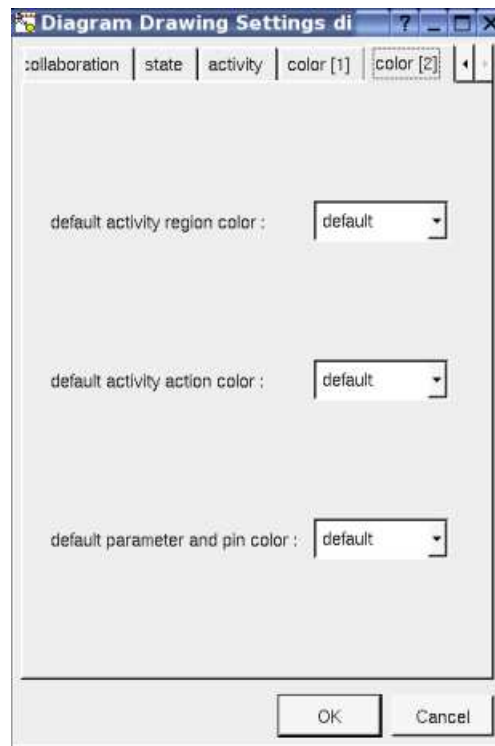
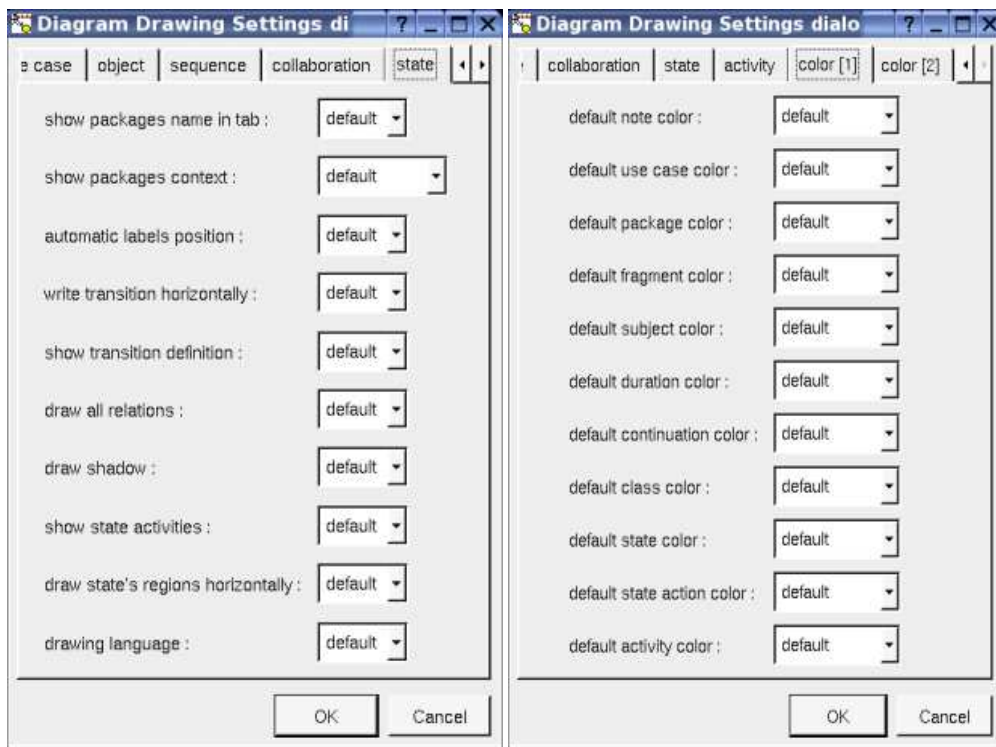
The *stereotype* of a use case view doesn't have a special meaning for BOUML.

The *editor* button visible above and associated here to the description, allows to edit the description in an other window, or to call an external editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note that this external editor have to

create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel with BOUML which looks each second at the file contents to get the new definition until the dialog is closed (I do not like very much polling but this works even QT isn't compiled with the thread support).

## Menu : edit drawing settings





This dialog allows to specify how the sub diagrams must be drawn by default.

### Menu : delete

The *delete* entry is only present when the package is not *read-only*.

Delete the use case view and all its children, and all the representation of them in the opened diagrams. After that it is possible undelete them (from the browser) until you close the project : obviously the deleted items are not saved !

### Menu : mark

See [mark](#)

### Menu : tools

The menu entry *tool* not present above is present in case at least a [plug-out](#) may be applied on the use case view.

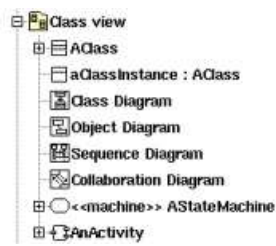
---

Previous : [package](#)

Next : [class view](#)

# Class view

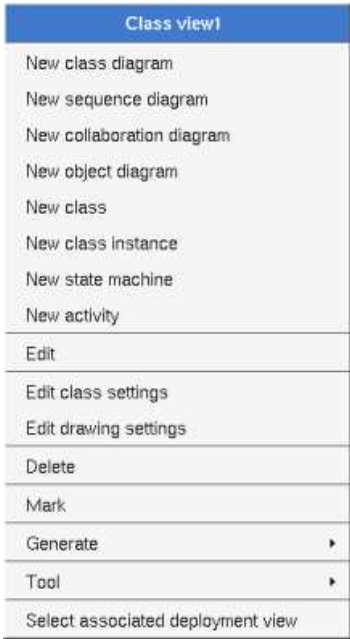
A class view is used to design the classes and may contain [classes](#), [class instances](#), [states](#), [activities](#), [class diagrams](#), [object diagrams](#), [sequence diagrams](#) and [collaboration diagrams](#) in any order :



A class view may have an associated [deployment view](#), in this case the menu of the sub-classes propose a shortcut to create an artifact in the associated deployment view (refer to [class menu](#)).

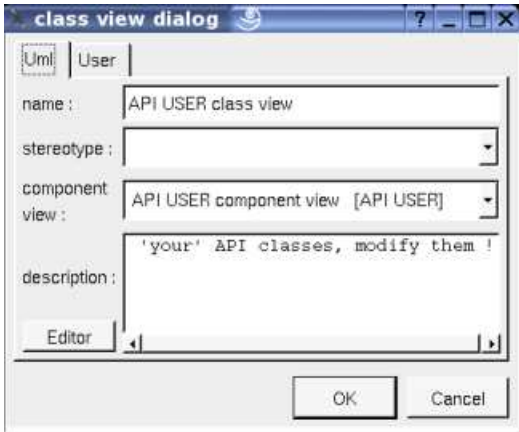
## Menus

The menu appearing with a right mouse click on a class view in the browser is something like this, supposing it is not *read-only* nor deleted :



The menu entry *select associated deployment view* is only present when the class view has an associated deployment view, this feature allows to quickly associate an [artifact](#) to a [class](#) (see [class menu](#)).

### Menu : edit

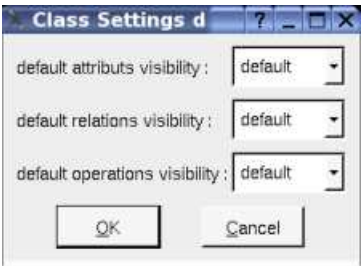


The *stereotype* of a class view doesn't have a special meaning for BOUML.

As you can see, the association with a deployment view is made through this menu.

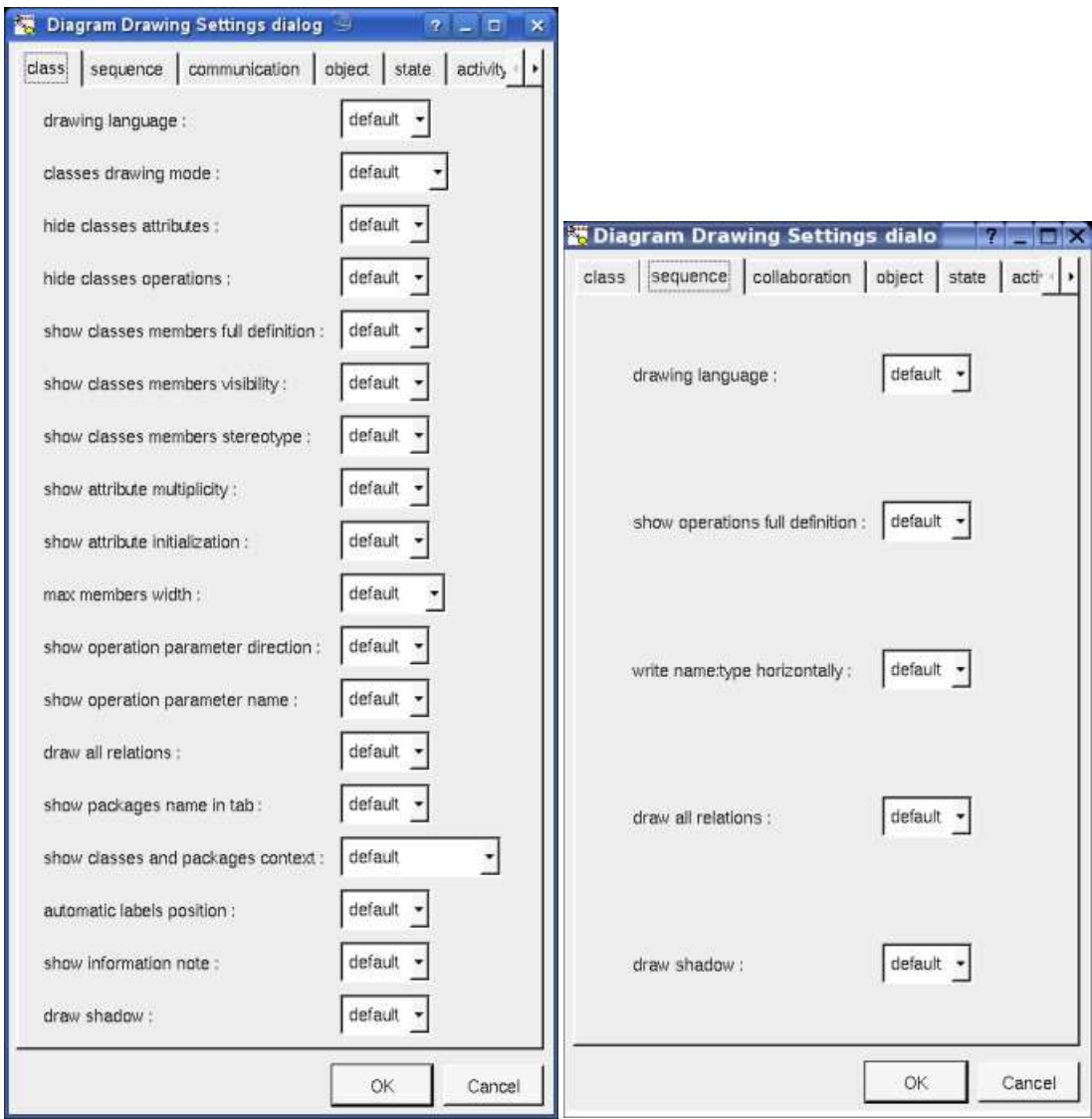
The *editor*  
button visible above and associated here to the description, allows to edit the description in an other window, or to call an external editor (for instance *Xcoral*) specified through the environment variable BOUML\_EDITOR. Note that this external editor have to create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel with BOUML which looks each second at the file contents to get the new definition until the dialog is closed (I do not like very much polling but this works even QT isn't compiled with the thread support).

### Menu : edit class settings



This setting allows to set a default visibility at the UML level for the attributes, relations and operations of the classes defined in the class view. *default* means that the visibility of the upper level is followed (which itself may be *default* etc ... up to the *project* level). When you create a new class view the visibilities are set to *default*.

### Menu : edit drawing settings



**Diagram Drawing Settings dialo** ? \_ □ ×

class | sequence | **collaboration** | object | state | acti...

drawing language : default ▾

show operations full definition : default ▾

show hierarchical rank : default ▾

write name: type horizontally : default ▾

show packages name in tab : default ▾

show packages context : default ▾

draw all relations : default ▾

draw shadow : default ▾

OK Cancel

**Diagram Drawing Settings dialo** ? \_ □ ×

class | sequence | collaboration | **object** | state | acti...

write name: type horizontally : default ▾

show packages name in tab : default ▾

show packages context : default ▾

automatic labels position : default ▾

draw all relations : default ▾

draw shadow : default ▾

OK Cancel

**Diagram Drawing Settings dialo** ? \_ □ ×

class | sequence | collaboration | object | **state** | acti...

show packages name in tab : default ▾

show packages context : default ▾

automatic labels position : default ▾

write transition horizontally : default ▾

show transition definition : default ▾

draw all relations : default ▾

draw shadow : default ▾

show state activities : default ▾

draw state's regions horizontally : default ▾

drawing language : default ▾

OK Cancel

**Diagram Drawing Settings dialo** ? \_ □ ×

s | sequence | collaboration | object | state | **activity**

show packages name in tab : default ▾

show packages context : default ▾

automatic labels position : default ▾

write flow label horizontally : default ▾

show opaque action definition : default ▾

draw all relations : default ▾

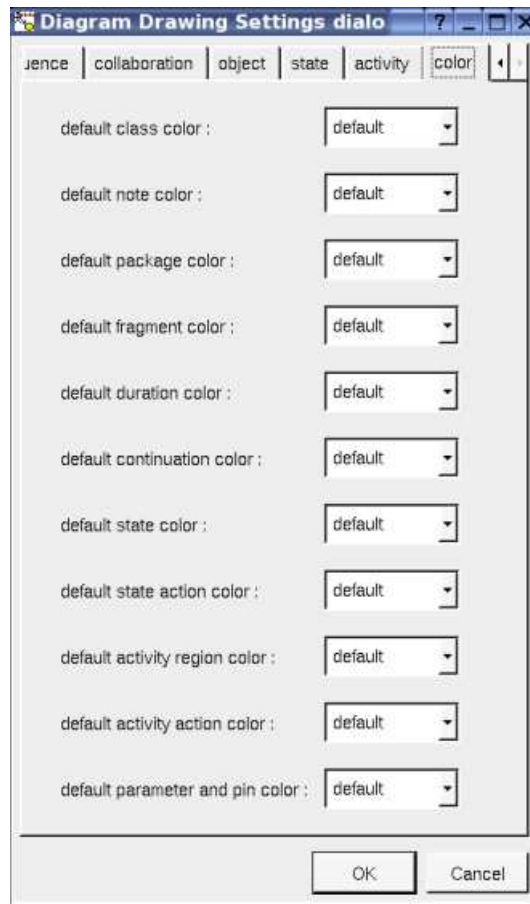
draw shadow : default ▾

show information note : default ▾

drawing language : default ▾

OK Cancel





This dialog allows to specify how the class view's diagrams must be drawn by default.

### **Menu : delete**

The menu entry *delete* is only present when the class view is not *read-only*.

Delete the class view and all its children, and all the representation of them in the opened diagrams. After that it is possible to undelete them (from the browser) until you close the project : obviously the deleted items are not saved !

### **Menu : mark**

See [mark](#)

### **Menu : generate**

To generate the C++/Java/Idl code of all the sub-items. Refer to [C++ generator](#), [Java generator](#) and [Idl generator](#) for more details.

### **Menu : tools**

The menu entry *tool* is present in case at least a [plug-out](#) may be applied on the class view.

### **Menu : select associated deployment view**

Only present when the class view has an associated deployment view, select it as you do this manually, except that you do not have to search it !

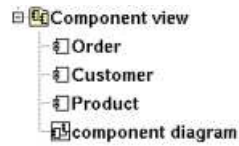
---

Previous : [use case view](#)

Next : [component view](#)

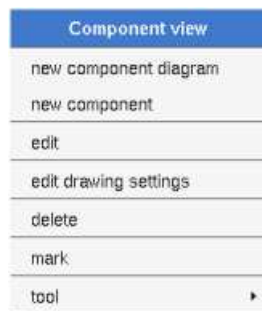
# Component view

A component view may contains [components](#) and [component diagrams](#) in any order :

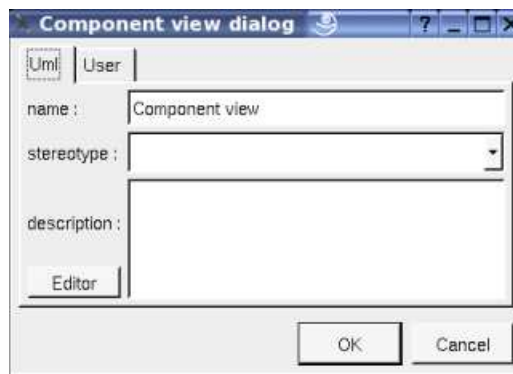


## Menus

The menu appearing with a right mouse click on a component view in the browser is something like this, supposing it is not *read-only* nor deleted :



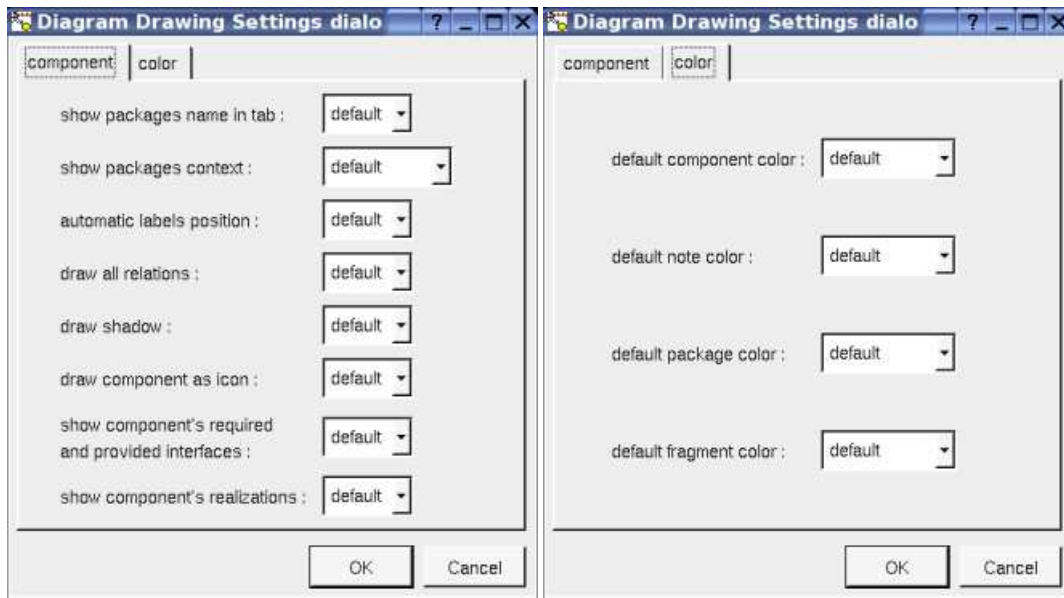
### Menu : edit



The *stereotype* of a component view doesn't have a special meaning for BOUML.

The *editor* button visible above and associated here to the description, allows to edit the description in an other window, or to call an external editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note that this external editor have to create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel wi BOUML which looks each second at the file contents to get the new definition until the dialog is closed (I do not like very much polling but this works even QT isn't compiled with the thread support).

### Menu : edit drawing settings



This dialog allows to specify how the component diagrams must be drawn by default.

### **Menu : delete**

The menu entry *delete* is only present when the class view is not *read-only*.

Delete the component view and all its children, and all the representation of them in the opened diagrams. After that it is possible to undelete them (from the browser) until you close the project : obviously the deleted items are not saved !

### **Menu : mark**

See [mark](#)

### **Menu : tools**

The menu entry *tool* not present above is present in case at least a *plug-out* may be applied on the component view.

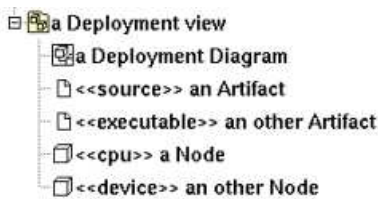
---

Previous : [class view](#)

Next : [deployment view](#)

# Deployment view

A deployment view may contains [nodes](#), [artifacts](#) and [deployment diagrams](#) in any order :



## Menus

The menu appearing with a right mouse click on a deployment view in the browser is something like this, supposing it is not *read-only* nor deleted :



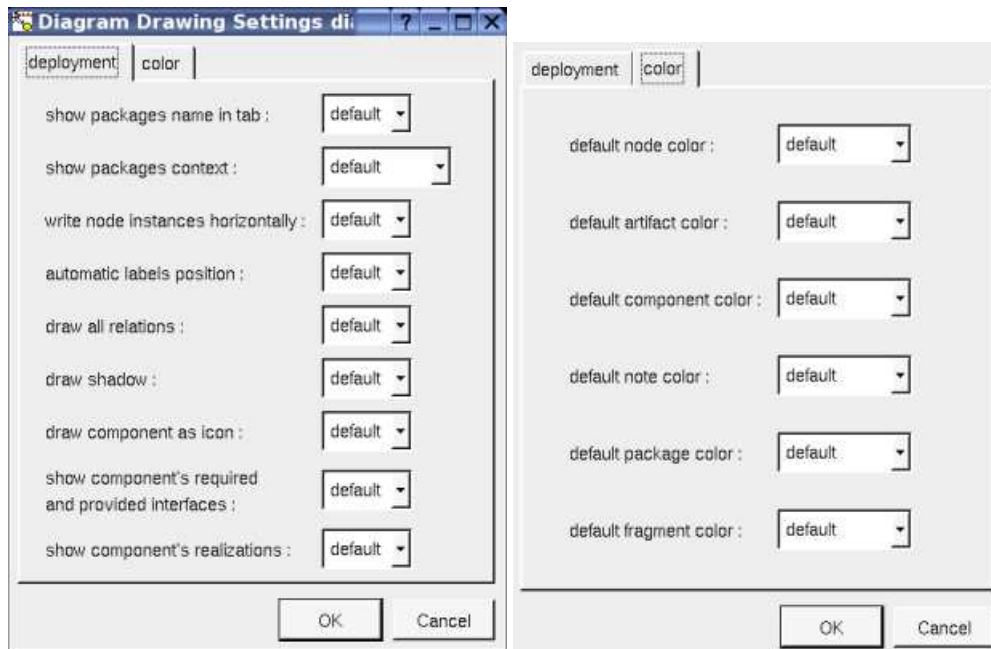
### Menu : edit



The *stereotype* of a deployment view doesn't have a special meaning for BOUML.

The *editor* button visible above and associated here to the description, allows to edit the description in an other window, or to call an external editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note that this external editor have to create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel wi BOUML which looks each second at the file contents to get the new definition until the dialog is closed (I do not like very much polling but this works even QT isn't compiled with the thread support).

### Menu : edit drawing settings



This dialog allows to specify how the deployment diagrams must be drawn by default.

### **Menu : delete**

The menu entry *delete* is only present when the class view is not *read-only*.

Delete the deployment view and all its children, and all the representation of them in the opened diagrams. After that it is possible to undelete them (from the browser) until you close the project : obviously the deleted items are not saved !

### **Menu : mark**

See [mark](#)

### **Menu : tools**

The menu entry *tool* not present above is present in case at least a [plug-out](#) may be applied on the component view. The selected tool is called and applied on the current deployment view.

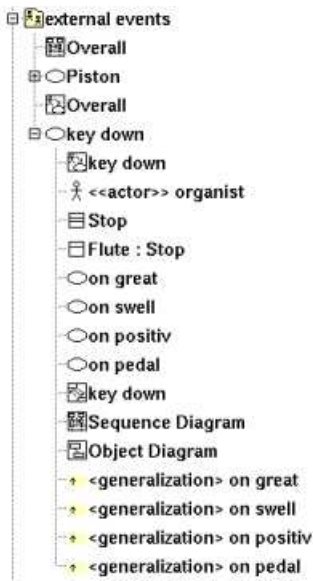
---

Previous : [component view](#)

Next : [use case](#)

# Use case

In the browser, an use case may contain other use cases, actors (classes stereotyped *actor*), [classes](#), [class instances](#), [state machine](#), [activities](#), [use case diagrams](#), [sequence diagrams](#), [collaboration diagrams](#), [object diagrams](#), and the relations, in any order :



Define a use case nested in an other one in the browser is a practical way when a use case is refined into sub use cases.

The relations between use cases presented in the browser are the inheritance and dependency.

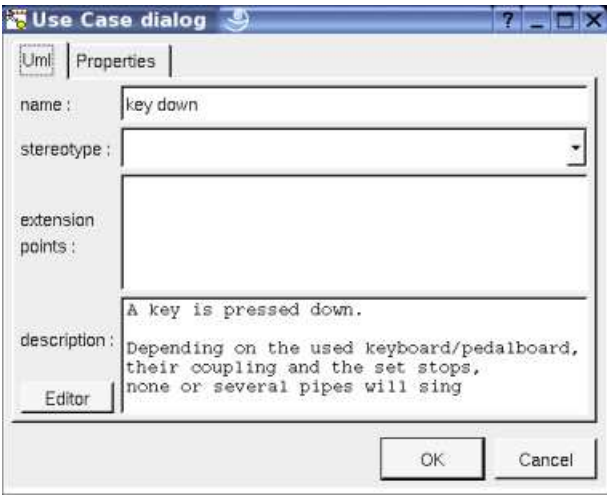
## Menus

The menu appearing with a right mouse click on a use case in the browser are something like this, supposing it is not *read-only* nor deleted :

key down
New use case diagram
New sequence diagram
New collaboration diagram
New object diagram
New use case
New actor
New class
New class instance
New state machine
New activity
Edit
Edit drawing settings
Delete
Referenced by
Mark
Tool

### Menu : edit

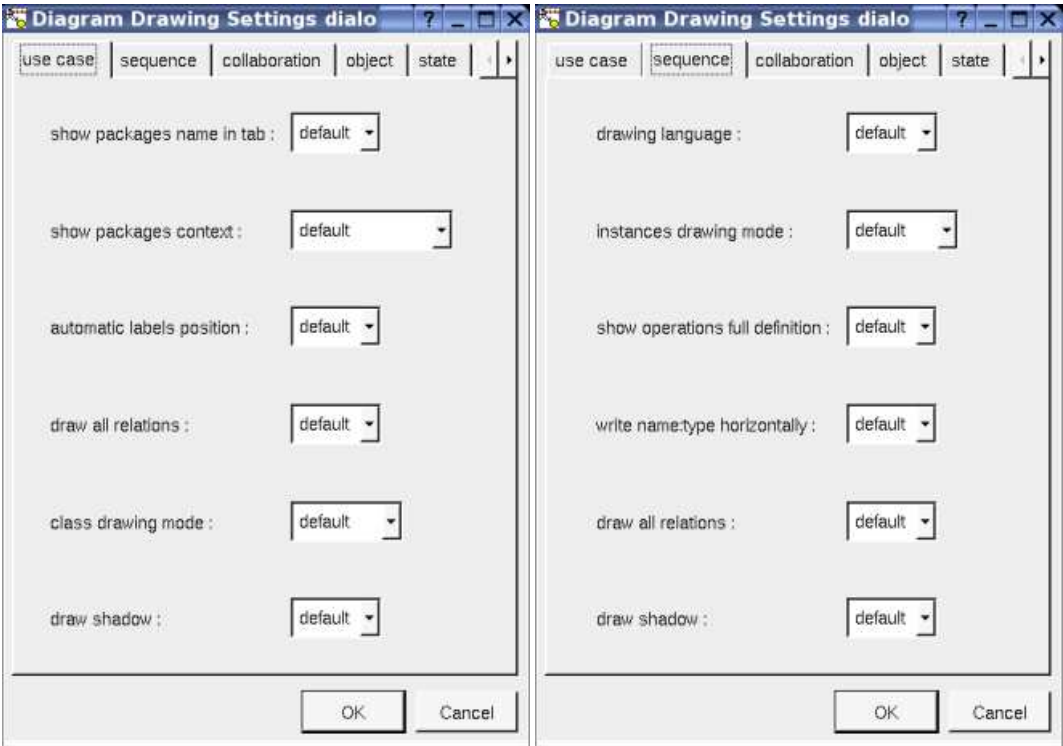
*edit* allows to show/modify the use case properties with the following dialog :

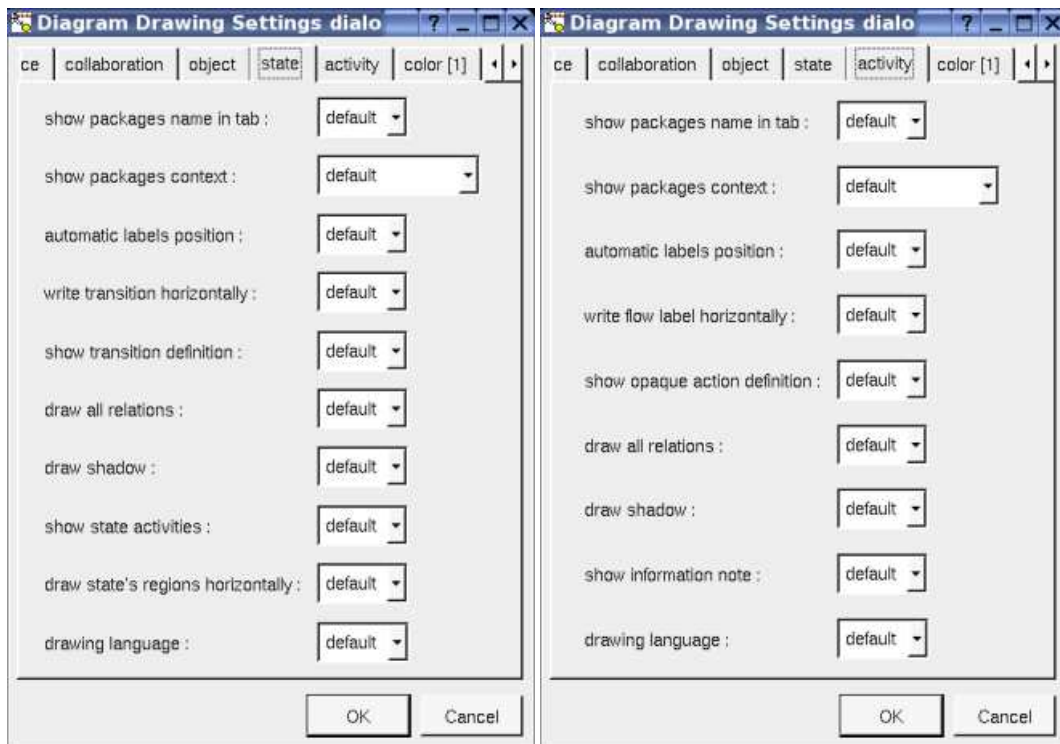
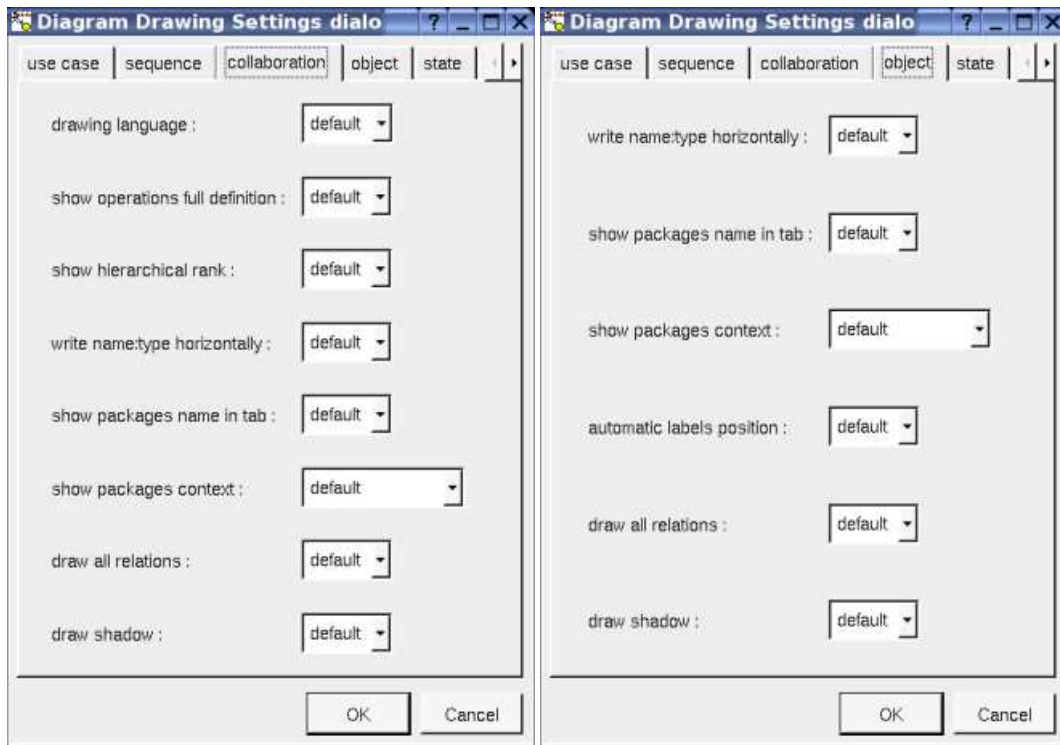


The *stereotype* of a use cases doesn't have a special meaning for BOUML.

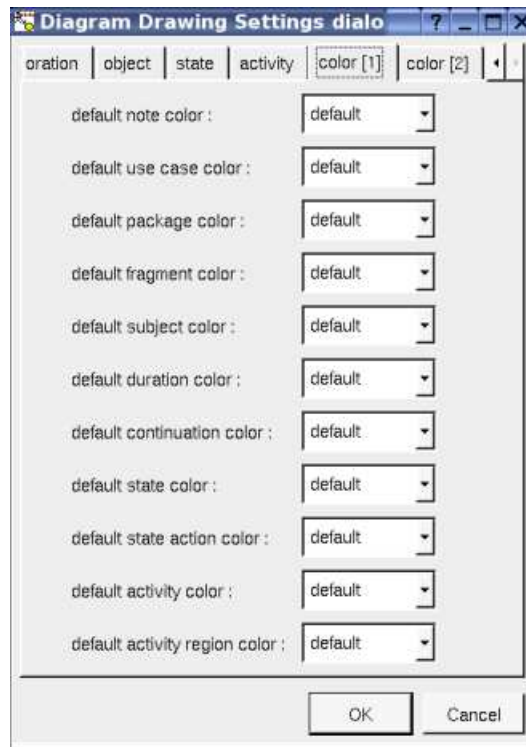
The *editor* button visible above and associated here to the description, allows to edit the description in an other window, or to call an external editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note that this external editor have to create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel with BOUML which looks each second at the file contents to get the new definition until the dialog is closed (I do not like very much polling but this works even QT isn't compiled with the thread support).

**Menu : edit drawing settings**









This dialog allows to specify how the sub diagrams must be drawn by default.

Refer to [drawing](#), for the settings concerning the use cases.

### **Menu : delete**

The *delete* entry is only present when the use case is not *read-only*.

Delete the use case and all its children, and all the representation of them in the opened diagrams. After that it is possible to undel them (from the browser) until you close the project : obviously the deleted items are not saved !

### **Menu : referenced by**

To know who reference the use case through a dependency or an inheritance, show a dialog, for instance :



### **Menu : mark**

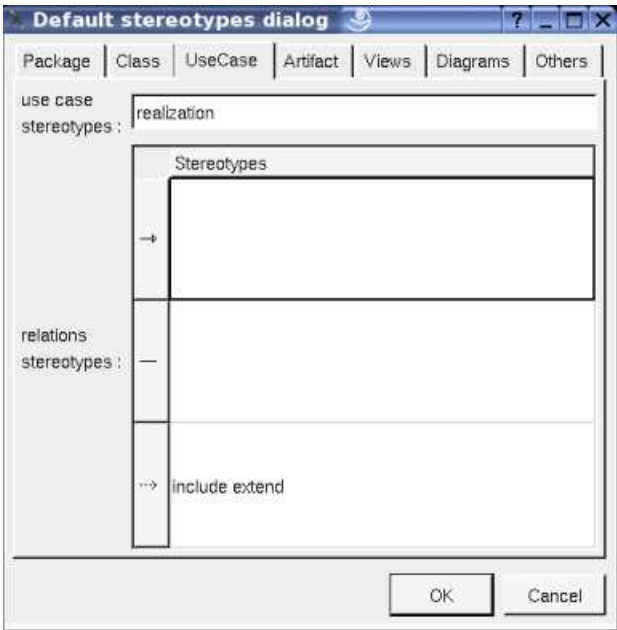
See [mark](#)

### **Menu : tools**

The menu entry *tool* is only present in case at least a [plug-out](#) may be applied on the use case. The selected tool is called and applied on the current use case.

## **Default stereotypes**

The [dialog](#) allowing to set the default stereotypes has a tab reserved for the use cases :



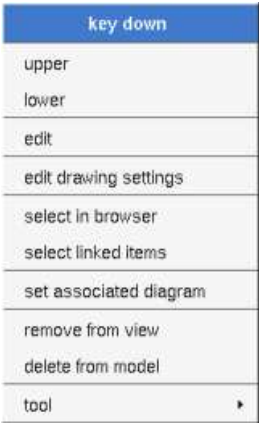
UML propose a simple line for the relation between use cases and actors/classes. I propose also an arrow because I prefe representation.

## Drawing

A use case is drawn in a diagram with a sizable ellipse, the available options are :

- *use case color*
- *automatic* : this concerns the relation's labels, by default BOUML automatically move the labels associated to a relation accordingly to its extremities. If you do not like how BOUML place the labels, set this option to *false* and you will be the master (note that the *labels default position* entry of the relation's menus in the diagrams remains available to help you when you are lost).

A right mouse click on a use case in a diagram calls the following menu :



The *drawing settings* concerns only the picture for which the menu is called.

*select linked items* select all the items connected with the current one through any line

*set* *associated* *diagram*  
allows to automatically open the current diagram when a double mouse click is made on the use case representation in a diagram or the browser. After that the only way to edit the use case is to choose the *edit* entry in the menu.

Previous : [deployment view](#)

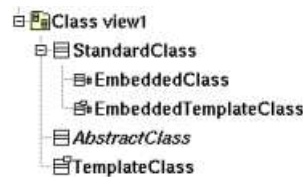
Next : [class](#)

# Class

A class may contain nested classes, [relations](#), [attributes](#), [operations](#), [extra members](#) and dependencies on [packages](#). The order of the class's sub-items is very important because it is followed by the C++/Java/Php/python/Idl generators.



The representation of a class in the browser indicates if the class is abstract or is a template class, furthermore a nested class icon has a reduced size :



A *class* is the support to define all kinds of class depending on the languages, the stereotype specify what a class is. The way a stereotype translated for each language is configured and is specified through the [generation settings](#). You can set a default list of stereotypes for the classes and their relations through the [edit default stereotype](#) entry of the [project's menu](#).

Some stereotypes have a special meaning for BOUML :

- *typedef* : a class having this stereotype may have an associated base type used to construct the typedef. It is not possible to add a relation/attribute/operation to a class stereotyped *typedef*, nor to define it as a *template* (but it may instantiate a *template*). This stereotype doesn't have sense in Java.
- *enum* : a class having this stereotype is an *enum*, need at least the JDK 5 in Java. Because a Java JDK 5 *enum* may have attributes, relations and operations, it is possible to add these kinds of members to a class having the stereotype *enum*, but default definitions are empty in C++ and Idl. It is not possible to define or instantiate a *template*, they are not yet managed in Java. In Php an *enum* produces a class having only *const* variables
- *enum\_pattern* : a class having this stereotype is an *enum* even in Java for which a special definition compatible with JDK release less than 5 is produced. It is not possible to add a relation/operation to a class stereotyped *enum* nor to define or instantiate a *template*.
- *union* : a class having this stereotype cannot instantiate a *template*
- *actor* : just to change the representation, not relevant for the code generators

Obviously because you may change the stereotype at any time, it is possible to define for instance a class with an operation and after that to change its stereotype to *enum\_pattern* ... In this case BOUML does nothing, for instance it does not delete itself the operation to help you to come back to a more appropriate stereotype without undesirable modifications.

The extend of the previous stereotypes include C++ and Java and Idl both. This means for instance that you cannot add an operation to an *enum\_pattern* independently of the translation of this stereotype in C++/Java/Php/Python/Idl specified through the [generation settings](#). But it is also possible to have stereotypes with an extend limited to each language through the definition of new stereotype. For instance, if you define the stereotype *eiu* which is translated by *enum* in C++, *interface* in Java, *class* in Php and Python and *union* in IDL through the [generation settings](#), you will have exactly that you want : magic ! isn't it ?

The stereotypes known at each language level by BOUML are :

- **C++** : *class* (or empty stereotype), *enum*, *struct*, *typedef* and *union*
- **Java** : *class* (or empty stereotype), *enum*, *enum\_pattern*, *interface* and *@interface*
- **Php** : *class* (or empty stereotype), *enum*, *interface*

- **Python** : *class* (or empty stereotype), *enum* (produce also a class)
- **Idl** : *enum*, *exception*, *interface*, *struct*, *typedef*, *union* and *valuetype*

The class's properties will be exposed below.

## Menus

The class menu appearing with a right mouse click on its representation in the browser depend on the stereotype and the [artifact](#) association, it is something like these, supposing the class not *read-only* nor deleted (see also [menu in a diagram](#)) :

StandardClass	Enum	EnumPattern
add attribute	add item	add item
add operation	add attribute	edit
add nested class	add operation	duplicate
add extra member	add extra member	delete
edit	edit	create source artifact
duplicate	duplicate	referenced by
delete	delete	mark
create source artifact	select associated artifact	generate ▶
referenced by	referenced by	tool ▶
mark	mark	
generate ▶	generate ▶	
tool ▶	tool ▶	

### Menu : add ...

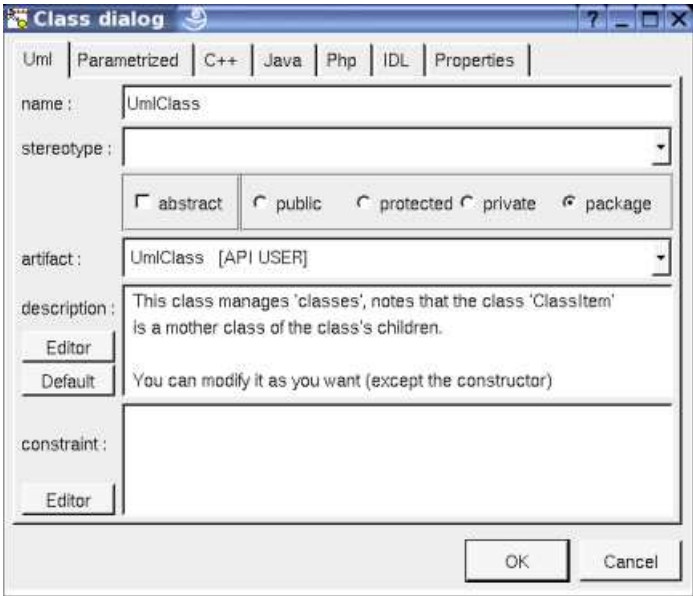
The first entries of the menu allowing to add something may not be present depending of the class's stereotype.

### Menu : edit

*edit* allows to show/modify the class properties. In case the class is read-only, the fields of the dialog are also read-only. Note th consistency purpose you cannot edit a class while any other edition is made, and while a class is edited all the other editions impossible.

#### Class dialog, tab Uml

The tab *Uml* is a global tab, independent of the language :



The proposed *stereotypes* are the default one specified through the [Default stereotypes dialog](#) more the current one (empty here). It is possible to choose into the list or to give a new one, or to empty it.

*abstract* allows to declare that the class is an abstract class (not instanciable).

The [artifact](#) used to produce the class source code must be chosen in the proposed list or the field may be empty. Obviously a nested class can have an associated [artifact](#), but a nested class has a visibility.

The *description* will be generated in the class source code (in case the *comment* or *description* macro appears in the class definition) as a comment.

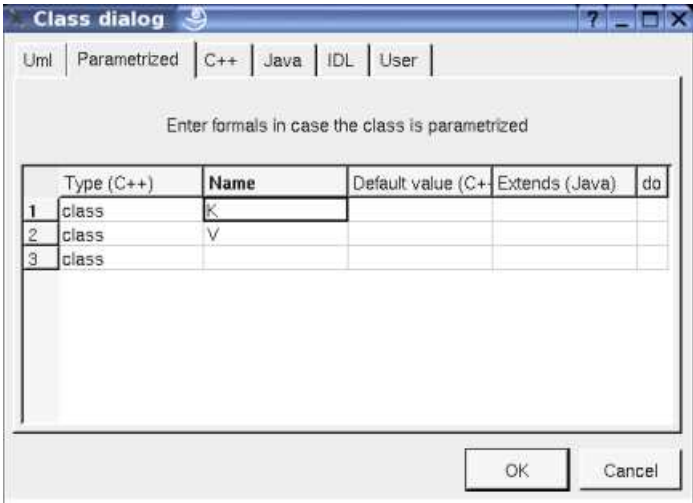
The [editor](#) button visible above and associated here to the description, allows to edit the description in an other window, or to call an external editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note that this external editor [have to create an own window](#), its parameter is the pathname of the file containing the description, its execution is done in parallel with BOUML which looks each second at the file contents to get the new definition until the dialog is closed (I do not like very much polling but this works even QT isn't compiled with the thread support).

The [default](#) button visible above associated to the description allows to set the description with a default content specified through the *generation settings*. Useful to generate comments compatible *Java Doc* or *Doxygen* for instance.

### Class dialog, tab Parametrized

The *parametrized* tab allows to specify the formal of a *template / generic* class.

For instance with :



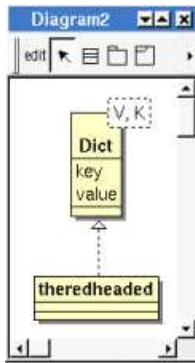
if the class have the default proposed class definition, the C++ class definition (named *Dict*) is :

```
template<class V, class K = string> class Dict ...
```

### Class dialog, tab Instantiate

The *instantiate* tab allows to specify the actual of a class inheriting templates classes. It appears only when the edited class inherits or realizes at least a template class. Obviously, a template class may inherit another ones.

For instance with this definition of *Dict* :



Enter formal in case the class is parametrized					
	Type (C++)	Name	Default value (C++)	Extends (Java)	do
1	class	V	string		
2	class	K	string		
3	class				
4	class				

OK Cancel

if the class have the default proposed class definition, the C++ class definition of *theredheaded* is :

```
class theredheaded : public Dict<string, string> ...
```

the Java class definition is :

```
class theredheaded extends Dict<String, String> ...
```

When the class *theredheaded* is edited only the *actuals* are editable in the tab *Instantiate*, both for C++ and Java.

Supposing you have a template class *vector* and you want to define a vector or boolean, obviously the right definition is not :

```
class vectorOfBoolean : public vector<bool> {...}
```

the right definition is :

```
template<> class vector<bool> {...}
```

To do this you just have to define the class named *vector<bool>*

### Class dialog, tab C++

This tab allows to give the C++ definition of the class, it is visible only when C++ is set through the *Languages* menu.

Template definition :

☐ external

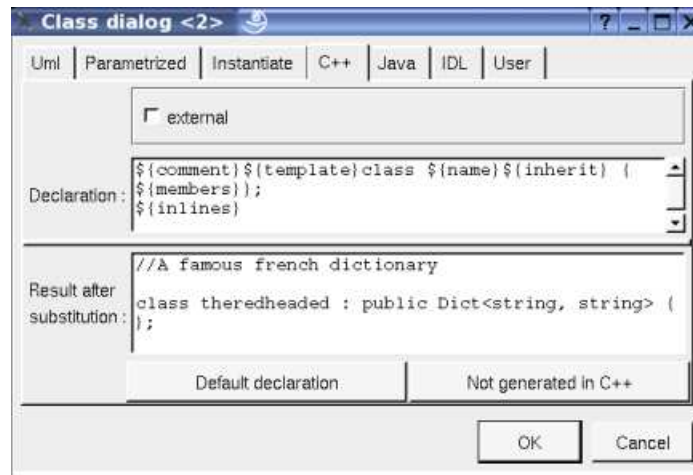
Declaration: `${comment} ${template} class ${name} ${inherit} {  
${members}};  
${inlines}`

Result after substitution: `template<class V, class K = string> class Dict {  
..key  
..value  
};`

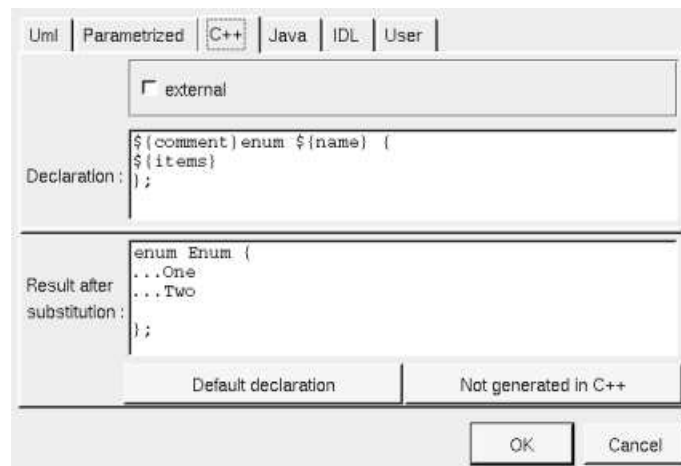
Default declaration Not generated in C++

OK Cancel

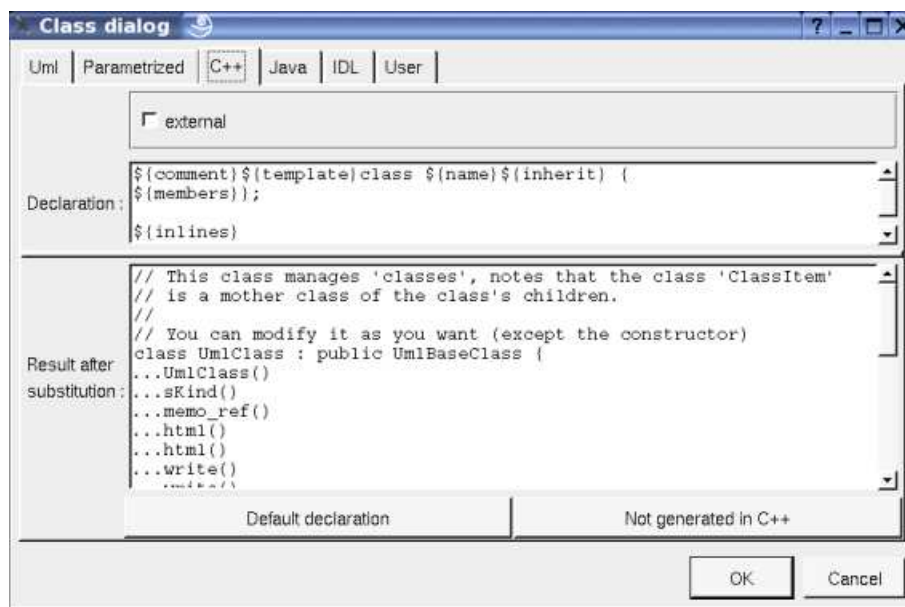
Template instantiation (of course a class may be both a template definition and a template instantiation) :



Enum (class with the stereotype *enum* or *enum\_pattern*) :



Standard class :



In BOUML the generated code is obtained by the substitution of macros in a text, only the **Declaration** part is editable, the other one help you to see what will be generated for C++ (supposing you do not modify the C++ code generator !).

When you push the button **default declaration**, the form specified through the [generation settings](#) depending on the class stereotype (and of course the language !) is proposed, this last may be modified as you want, even to produce illegal source code. For instance if you do not want to follow the inheritance specified by the class's relations, you just have to replace *\${inherit}* by anything you want.

When you do not want to have this class defined in C++ (but it must be defined in Java or Idl, so the class must have an [associated artifact](#)), empties the declaration manually or using the button **not generated in C++** (an other way is to empty the C++ definition of the [artifact](#)).



The *external*  
 check box must be used when the class must not be defined in C++ by BOUML, but you want to specify the name of the class (to not follow the *Uml/Java/Idl* name) and/or the included files and/or *using* forms or any other forms needed to use it.

*template* produce a template declaration in case the class is a template class, else an empty string.

*members* produce the declaration and definition of the class members ([relations](#), [attributes](#), [operations](#) and [extra members](#)) following the browser order and repartition in the header and source files.

*inline* produce the definition of the [operations](#) and [extra members](#) declared *inline*, may be an empty string.

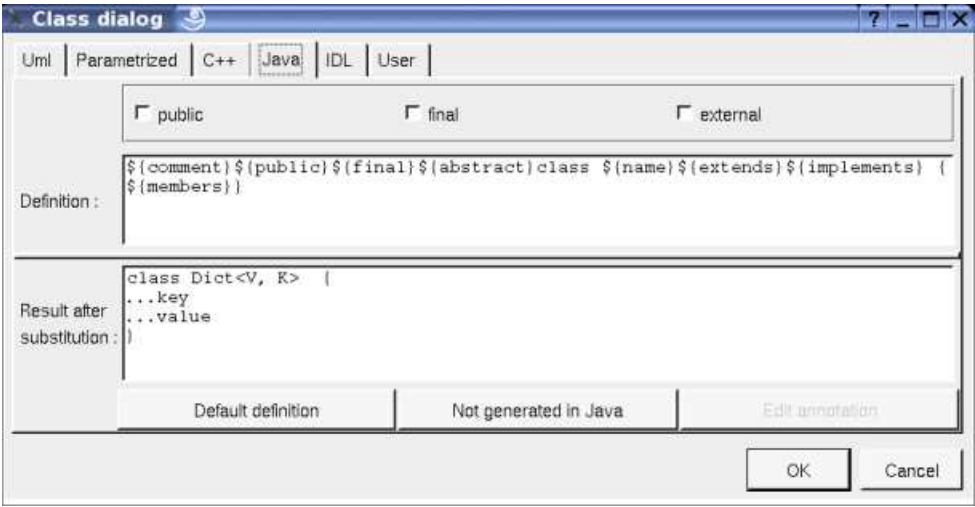
Refer to the [generation settings](#) for more details.

The forms  
 are replaced by the value of the corresponding user property if it is defined for the class or at an upper level, else it is not substituted. @{property}

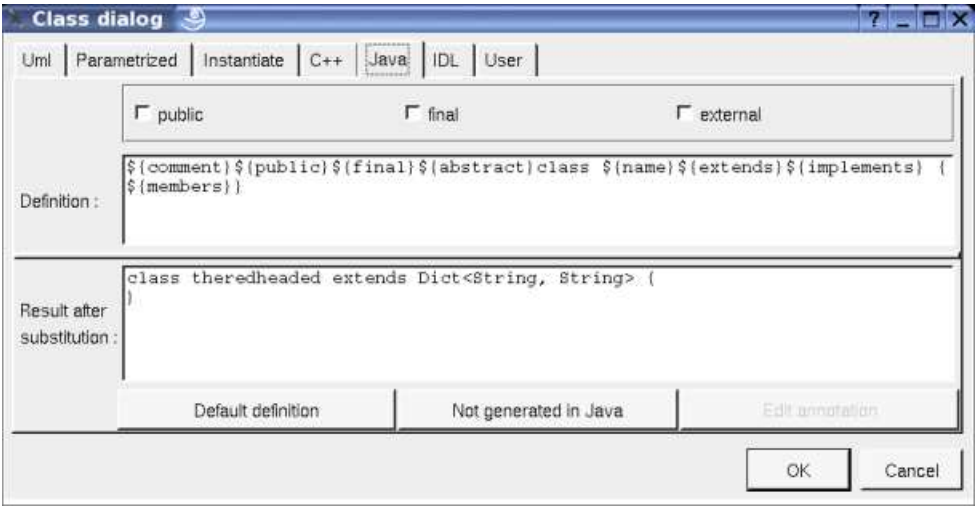
### Class dialog, tab Java

This tab allows to give the Java definition of the class, it is visible only when Java is set through the *Languages* menu

Generic definition :

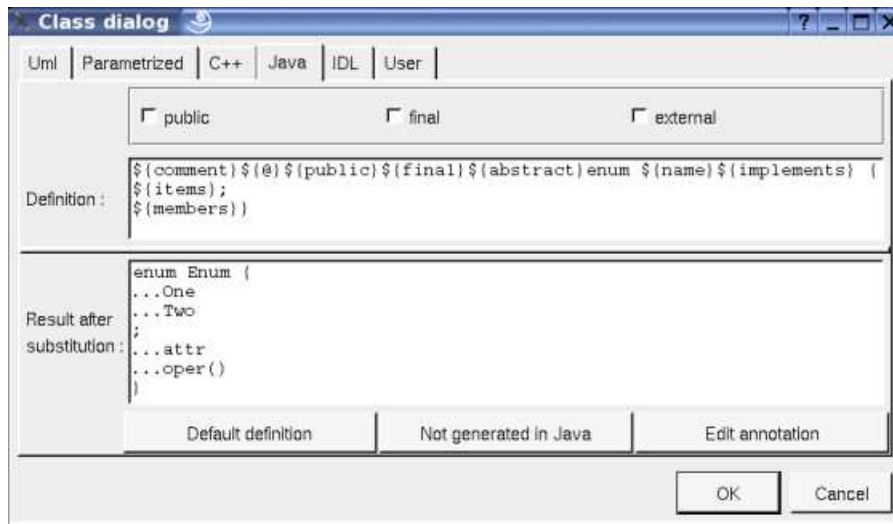


Generic instantiation :

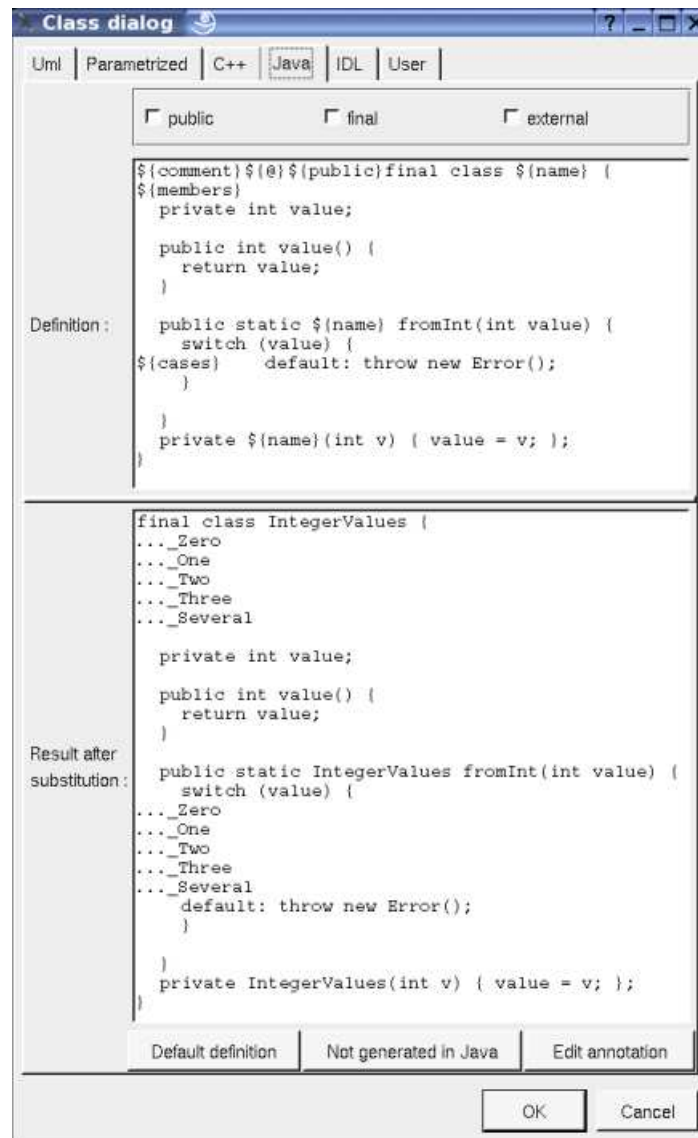


For a JDK 5 *enum* (class with the stereotype *enum*) :

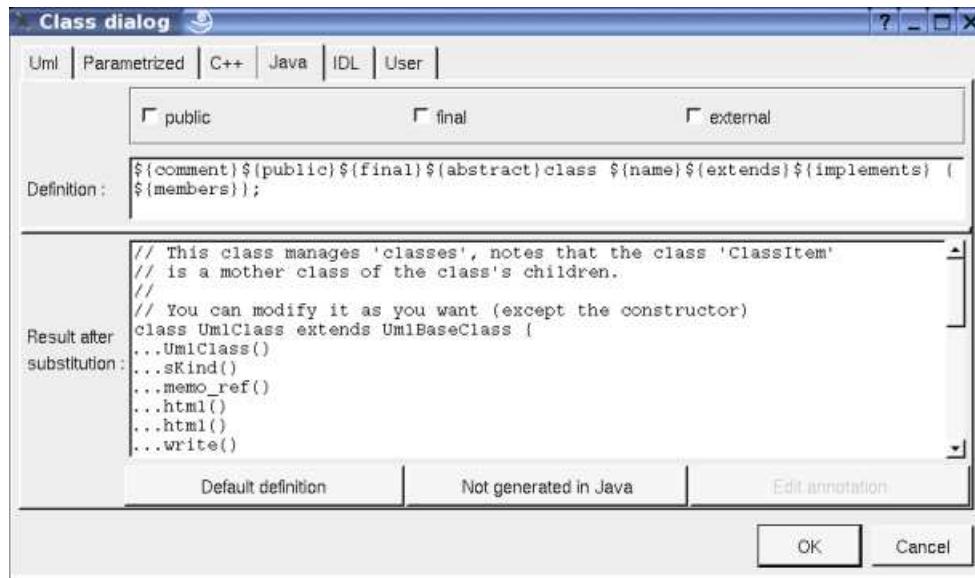




For a class with the Java stereotype *enum\_pattern* :



Standard class :



`${public}` produce an empty string when the check box **public** is not checked, else produce **public** (!)

`${final}` produce an empty string when the check box **final** is not checked, else produce **final** (!!)

The **external** check box must be used when the class must not be defined in Java by BOUML, but you want to specify the name of the class (to not follow the *Uml/C++/Idl* name).

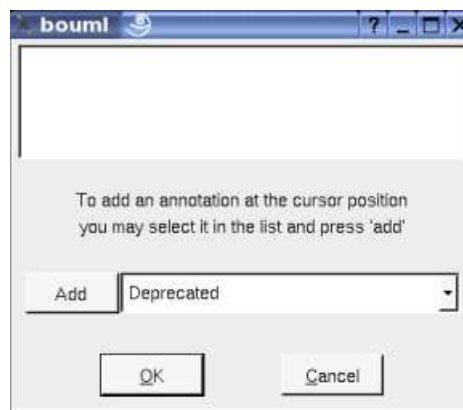
`${members}` produce the declaration of the class members ([relations](#), [attributes](#), [operations](#) and [extra members](#)) following the browser order, doesn't produce the enumeration items of an *enum*. In an *enum*, an attribute is not an enumeration item when its stereotype is *attribute*.

`${items}` produces the enumeration items of an *enum*. an attribute is an enumeration item when its stereotype is not *attribute*.

`${extend}` is replaced by the class inheritance

`${implement}` is replaced by the interface inheritance

`${@}` produces the annotations, when it is present in the definition the button *Edit annotation* is active, a specific dialog allows you to enter the annotations proposing the default annotations (*Deprecated ...*) and the ones defined through the classes stereotyped *@interface* :

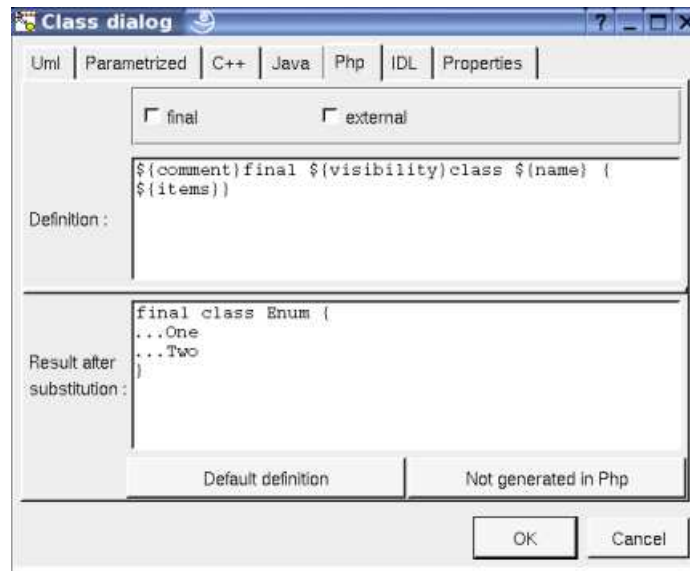


The **forms** `@{property}` are replaced by the value of the corresponding user property if it is defined for the class or at an upper level, else it is not substituted.

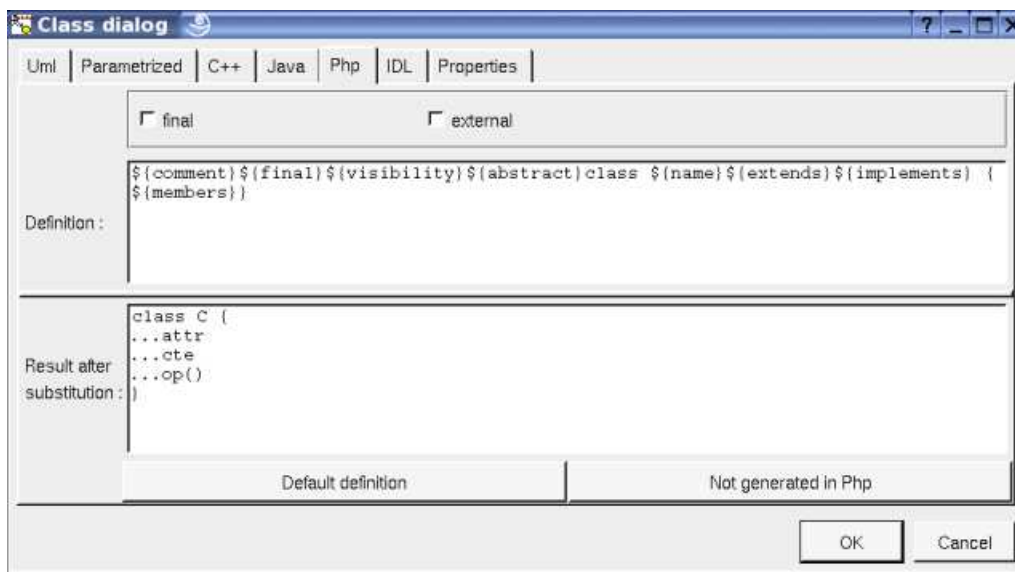
### Class dialog, tab Php

This tab allows to give the Php definition of the class, it is visible only when Php is set through the *Languages* menu

For an *enum* (class with the stereotype *enum*) :



Standard class :



`${visibility}` produce an empty string when the visibility except if this one is *package*. If you want to generate Php 4, set the visibility to *package*

`${final}` produce an empty string when the check box **final** is not checked, else produce **final** (!!)

`${abstract}` produce an empty string when the check box **abstract** is not checked, else produce **abstract** (!!)

The **external** check box must be used when the class must not be defined in Php by BOUML, but you want to specify the name of the class (to follow the *Uml/C++/Php/Python/Idl* name).

`${members}` produce the declaration of the class members ([relations](#), [attributes](#), [operations](#) and [extra members](#)) following the browser order

`${items}` produces the enumeration items of an *enum*. other members are not produced

`${extend}` is replaced by the class inheritance

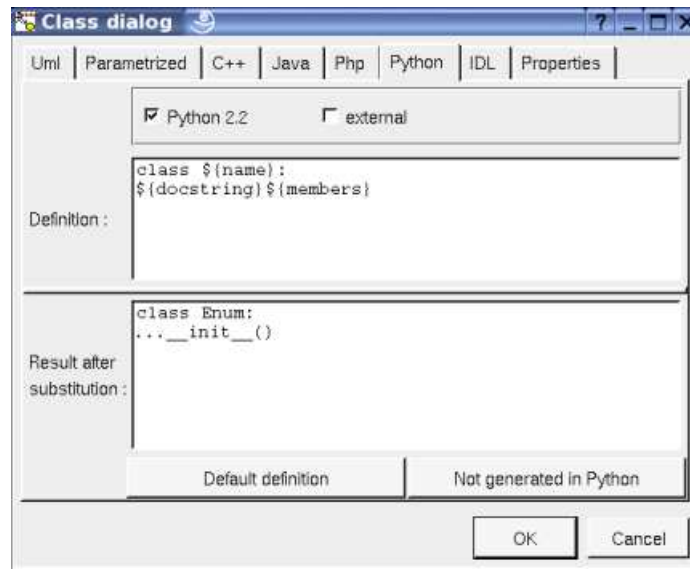
`${implement}` is replaced by the interface inheritance

The forms `@{property}` are replaced by the value of the corresponding user property if it is defined for the class or at an upper level, else it is not substituted.

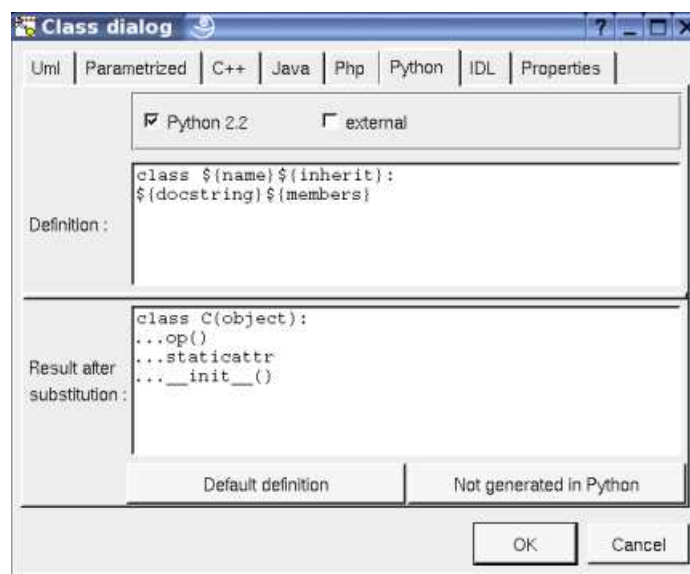
### Class dialog, tab Python

This tab allows to give the Php definition of the class, it is visible only when Php is set through the *Languages* menu

For an *enum* (class with the stereotype *enum*), recall the instance attributes are produced in the operation `__init__` :



Standard class :



The **Python** 2.2 check box allows to ask for classes introduced by Python 2.2, the default value of this toggle is the one specified by the [generation settings](#)

The **external** check box must be used when the class must not be defined in Python by BOUML, but you want to specify the name of the class (to not follow the *Uml/C++/Php/Python/Idl* name) on the first line (*\${name}* by default), may be followed by *import* forms whose will be produced by the code generator.

*\${members}* produce the declaration of the class members ([relations](#), [attributes](#), [operations](#), [extra members](#) and sub classes) following the browser order. Note that the instance attributes are produced at the beginning of the body of the operation `__init__`.

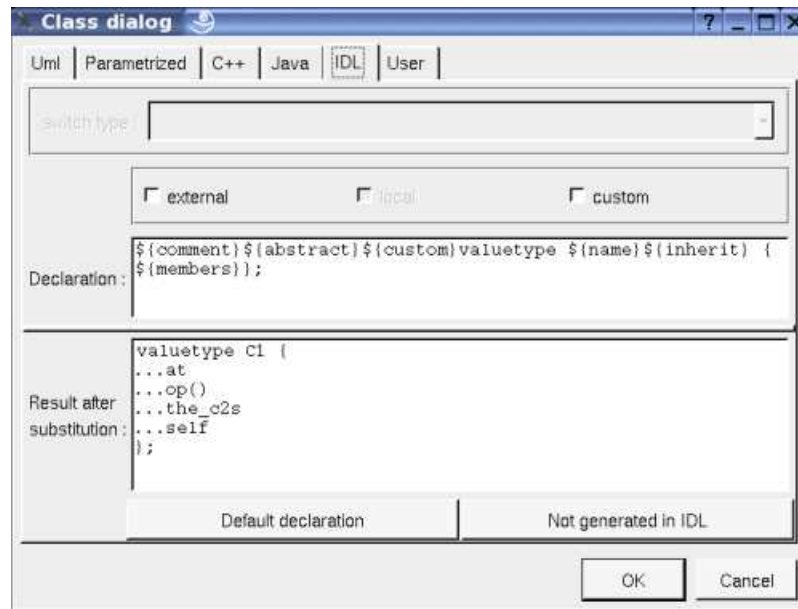
*\${inherit}* is replaced by the inheritance, a class without inheritance inherits of *object* if the toggle *Python 2.2* is set

*\${docstring}* is replaced by the description of the class placed between `"""` and followed by a newline. If the description is empty nothing produced. You can replace it by *\${comment}* or *\${description}* if you prefer.

The *forms* *@{property}* are replaced by the value of the corresponding user property if it is defined for the class or at an upper level, else it is not substituted.

### Class dialog, tab Idl

This tab allows to give the Idl definition of the class, it is visible only when Idl is set through the *Languages* menu



`${local}` produce an empty string when the check box **local** is not checked, else produce *local* (!), this one is only available when the stereotype of the class is *interface* in Idl (see [generation settings](#)).

`${custom}` produce an empty string when the check box **custom** is not checked, else produce *custom* (!), custom is available when the stereotype of the class is *valuetype* in Idl (see [generation settings](#)).

The **external** check box must be used when the class must not be defined in Idl by BOUML, but you want to specify the name of the class (to follow the *Uml/C++/Java* name) and/or the included files needed to use it. Not yet implemented in the Idl generator.

`${members}` produce the declaration of the class members ([relations](#), [attributes](#), [operations](#) and [extra members](#)) following the browser order.

Refer to the [generation settings](#) for more details.

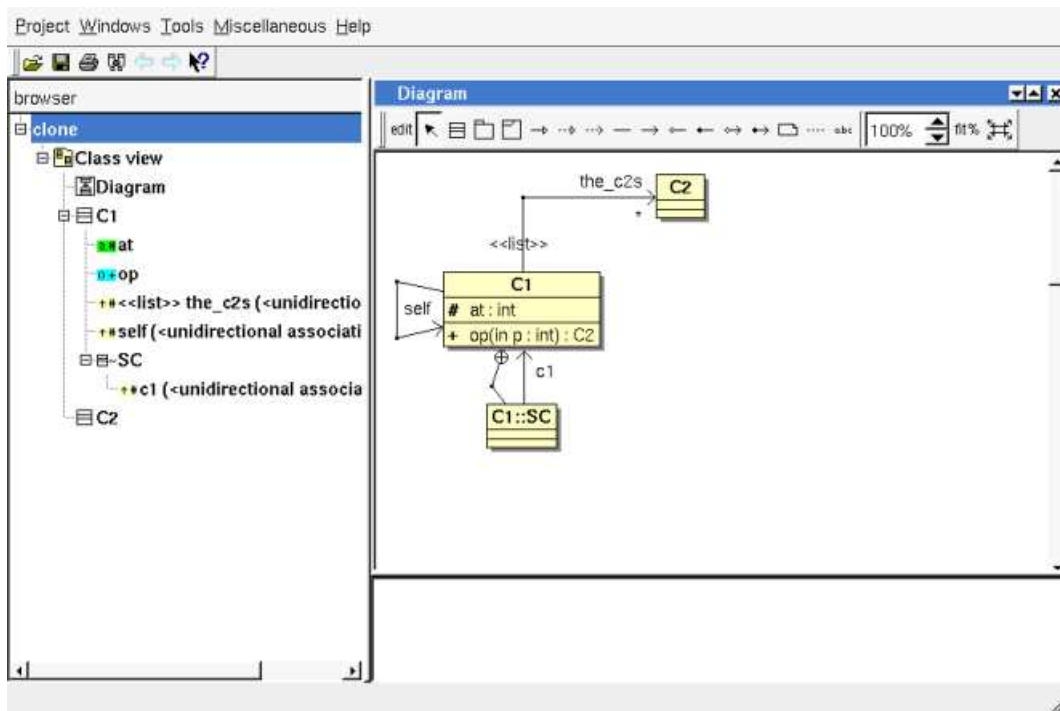
The forms `@{property}` are replaced by the value of the corresponding user property if it is defined for the class or at an upper level, else it is not substituted.

## **Menu : duplicate**

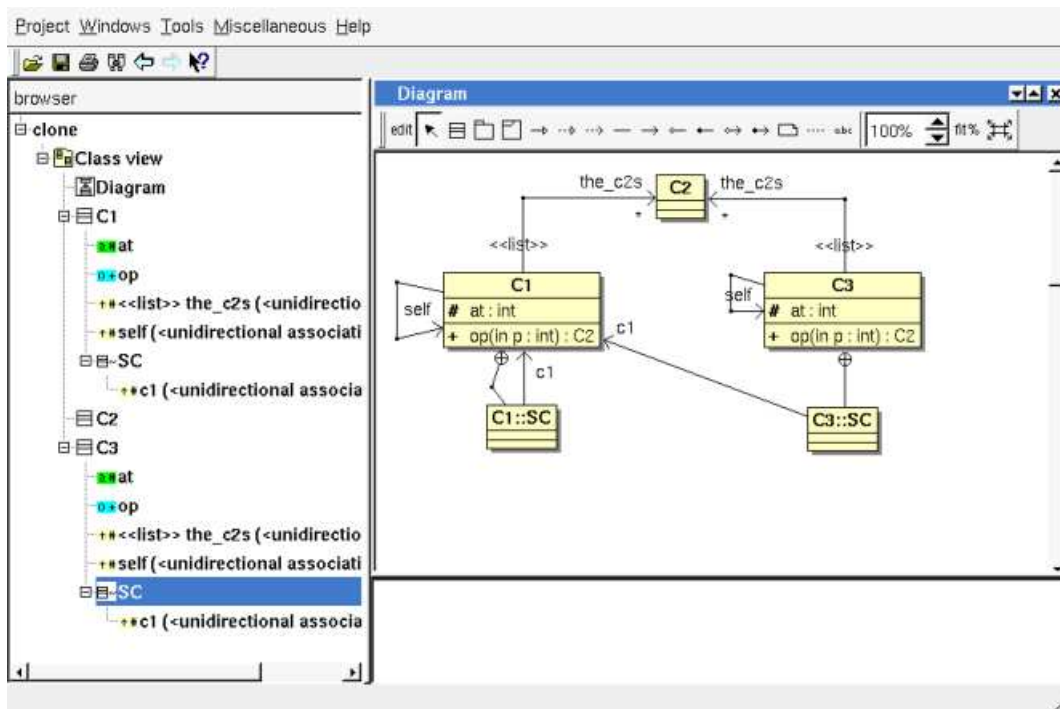
The menu entry *duplicate* clone the class and all its children, including the nested classes.

The self relations (a self relation is a relation between a class and itself) are specially managed : they became a self relation to the clone.

For instance, if the initial situation is the following :



After the duplication of *C1* into the new class *C3* :



Note that the duplication of the relation *c1* of *C3::SC* is still a relation to *C1*, because only the self relation are specially managed, not all the relation to the duplicated class (the context of the inner class(es) SC is visible thanks to the *drawing setting* “show context” and also to the inner class relation).

### Menu : create source artifact

The menu entry *create source artifact* is present when the class does not have an associated artifact, but the class view containing the class have an [associated](#) deployment view. Choosing this entry a new artifact having the name of the current class and the stereotype *source* is created in the deployment view associated to the class view containing the current class.

To quickly create is needed the associated *deployment view* and associate an *artifact* to all the classes of a class view use the [plug-out deploy](#).

### Menu : select associated artifact

This menu entry is present when the class have an associated artifact, to quickly select this one, the current class may also be quickly





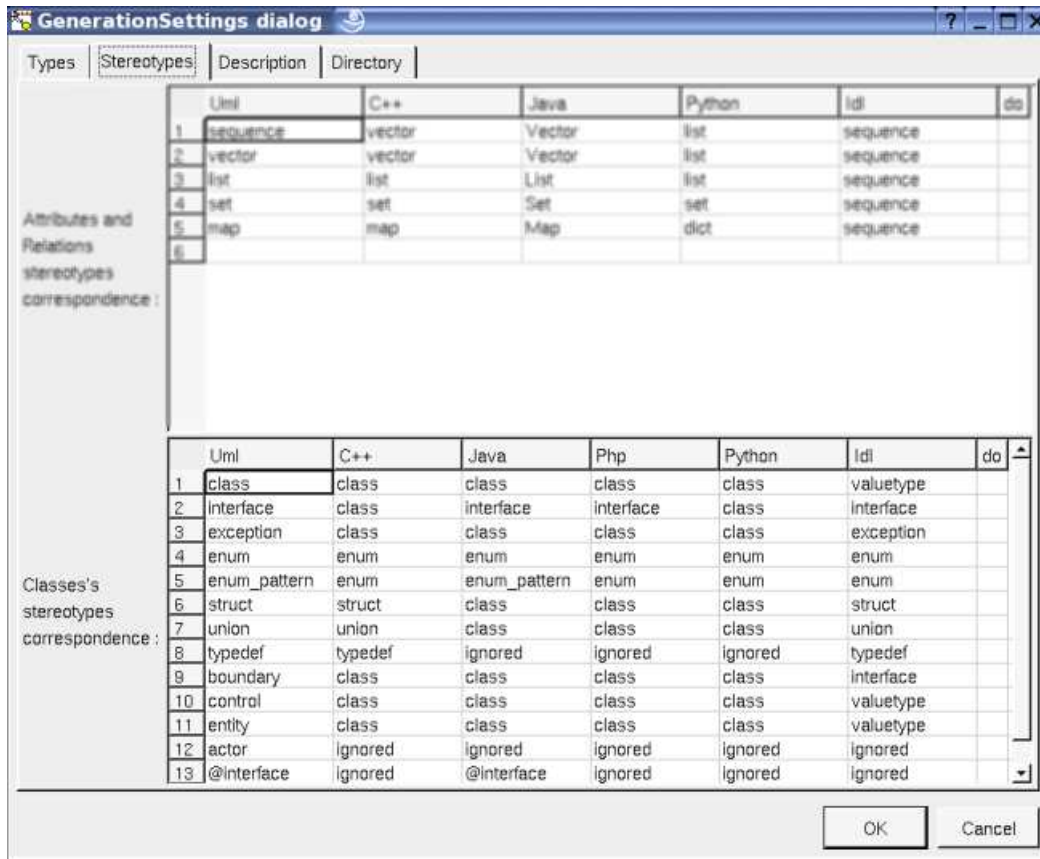
The code generators does not distinguish the generalization and the realization, even they do not have the same meaning in UML.

## Generation settings

This very important dialog allows to specify many default definitions concerning the classes, more details will be given in [C++ generator](#), [Java generator](#), [Php generator](#), [Python generator](#) and [Idl generator](#).

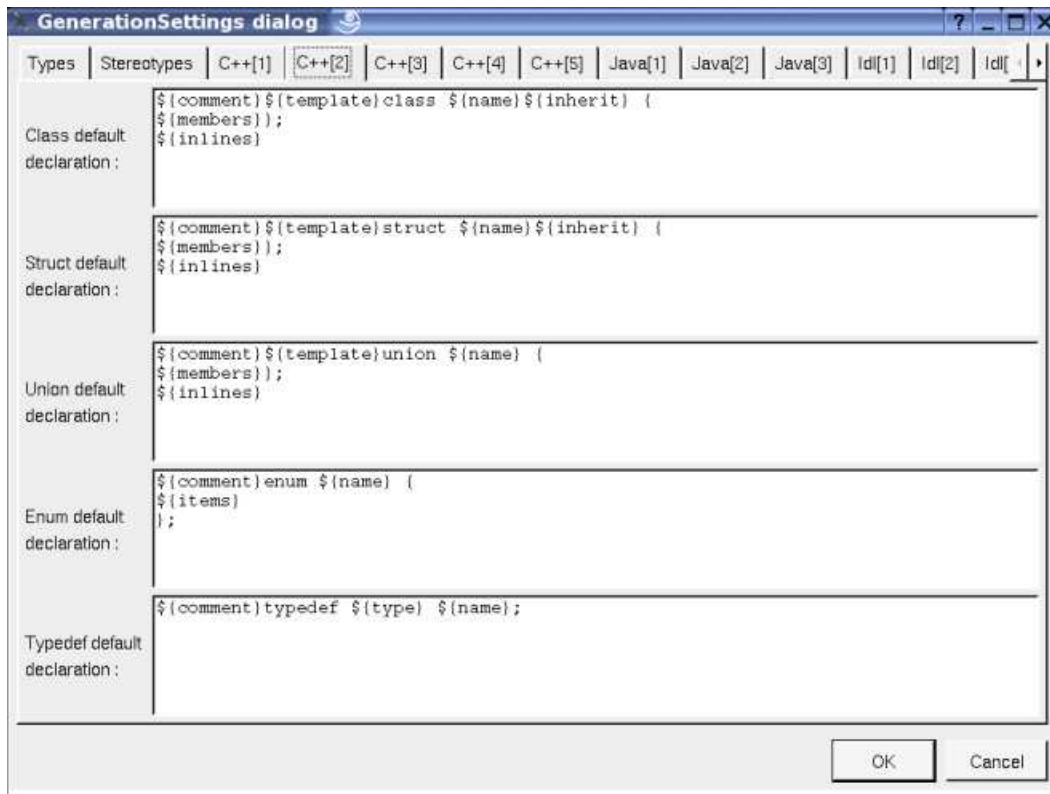
The tabs associated to a given language are only visible when the corresponding language is set through the *Languages* menu.

The tab *Stereotypes* allows to specify how the class's stereotypes are translated in each language :

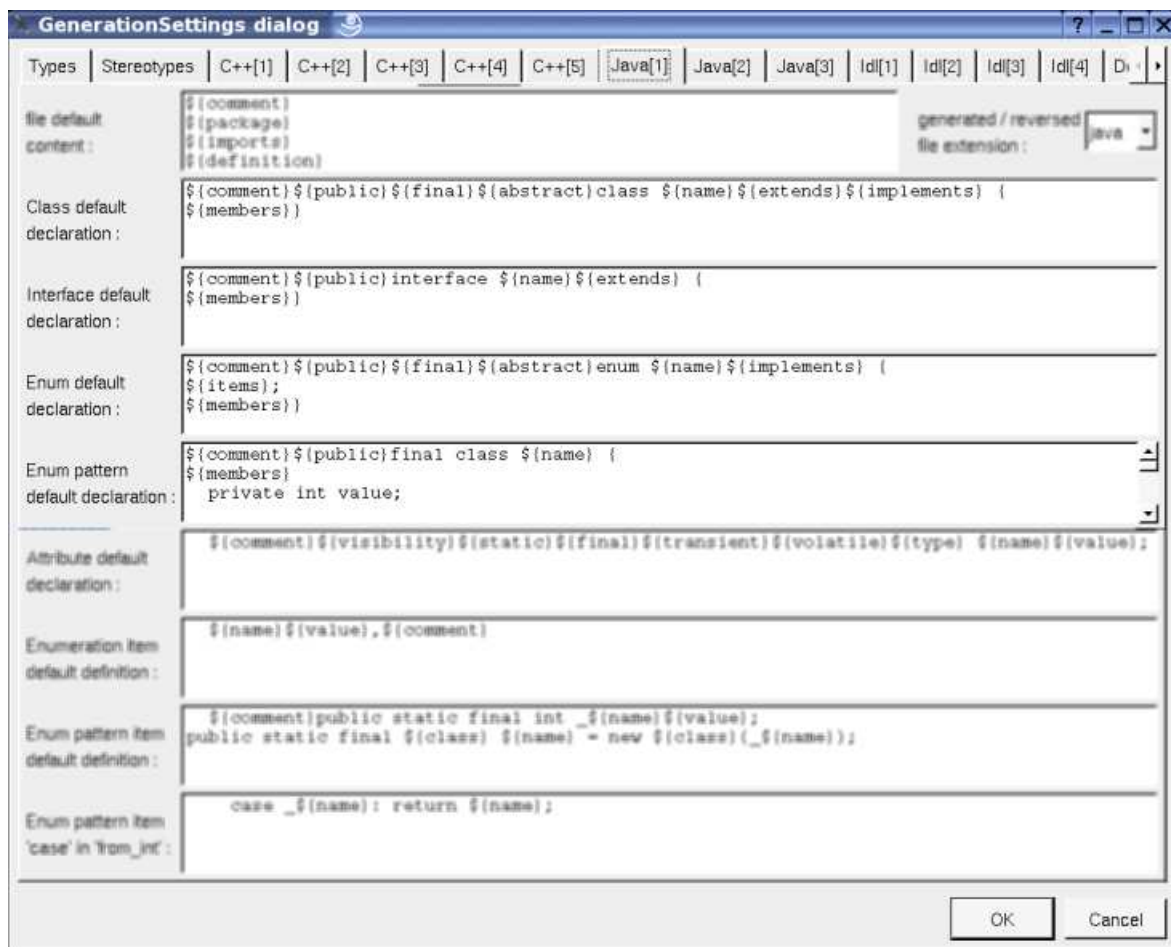


The second C++ tab allows to define the default C++ class definition depending on the stereotype :

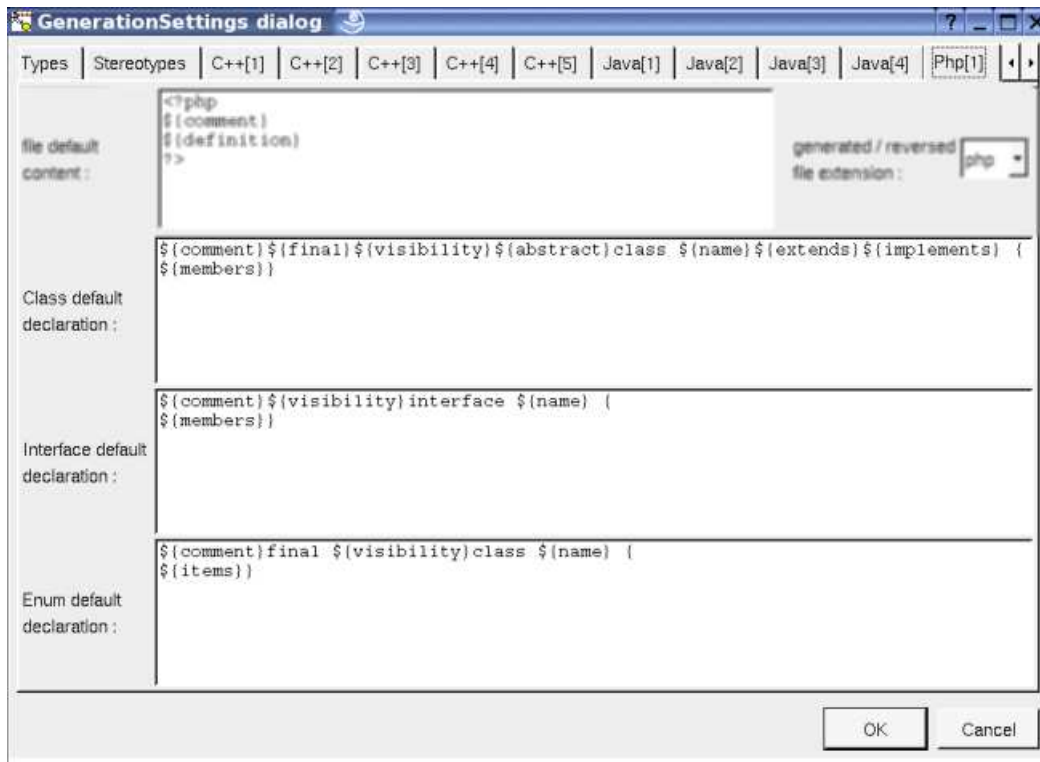




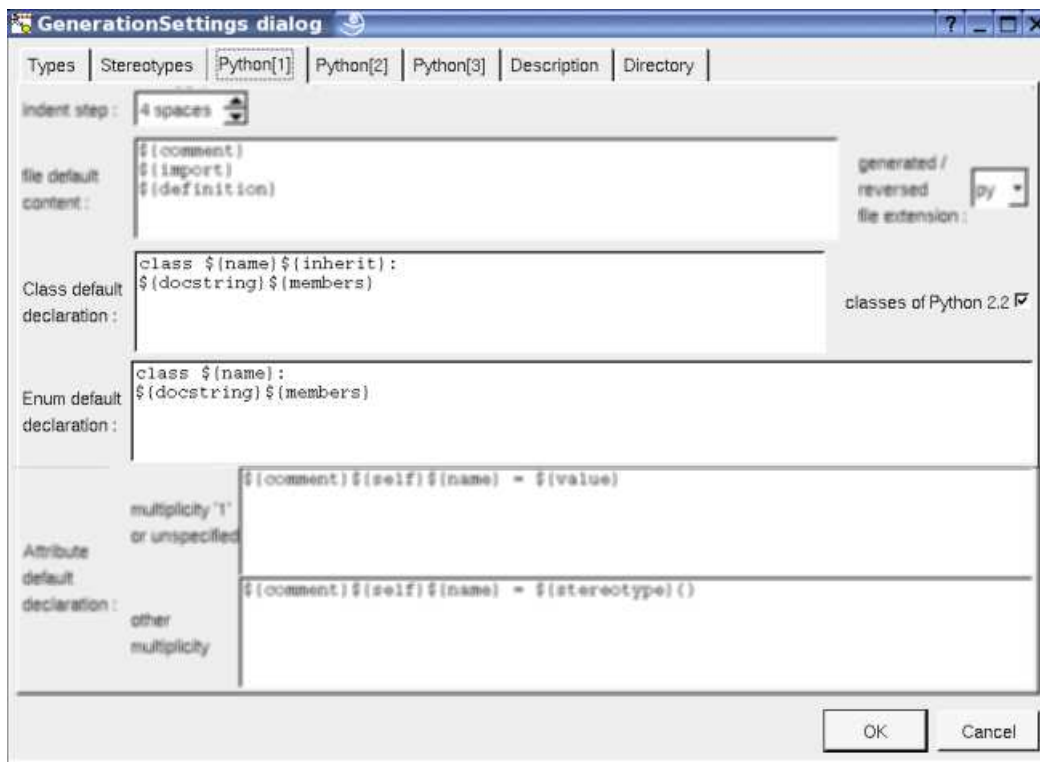
The first *Java* tab allows to define the default Java class definition depending on the stereotype :



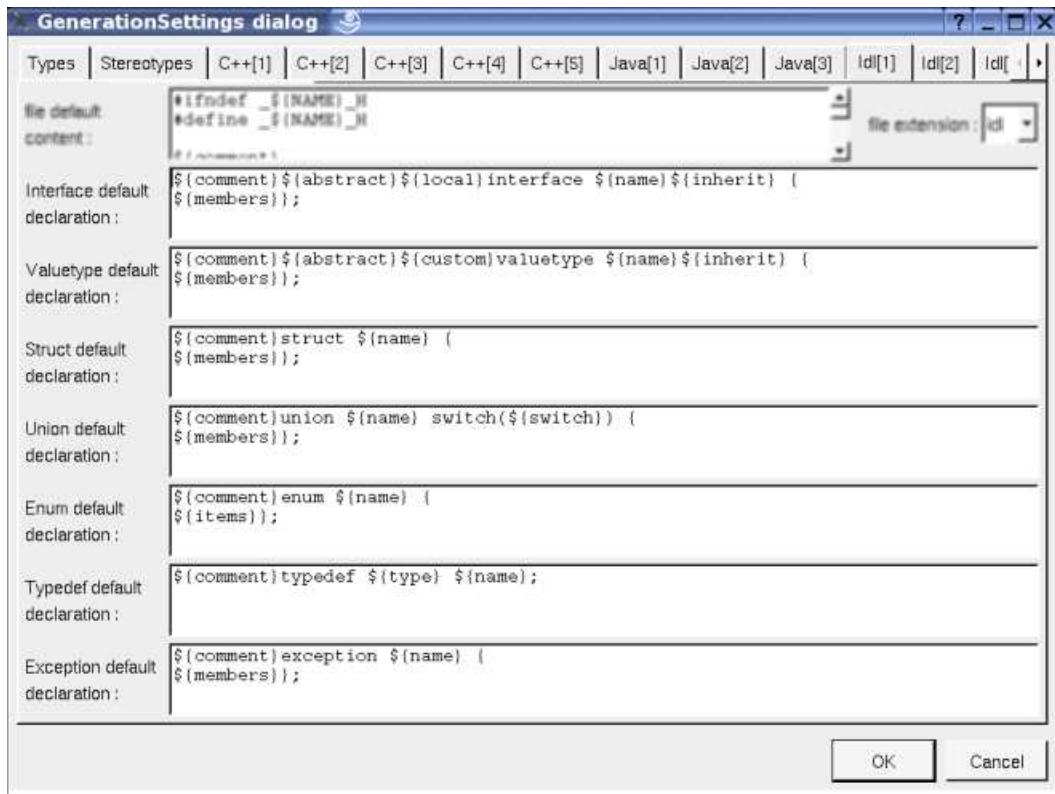
The first *Php* tab allows to define the default Php class definition depending on the stereotype :



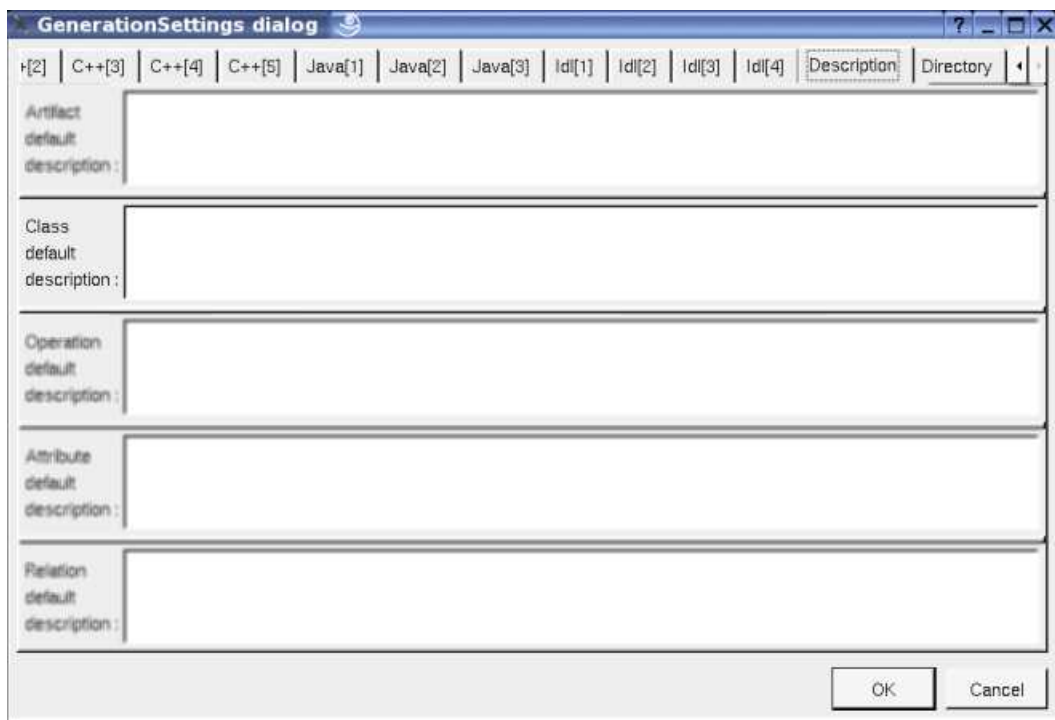
The first *Python* tab allows to define the default Python class definition depending on the stereotype, and to use *Python 2.2* classes or not by default :



The first *Idl* tab allows to define the default Idl class definition depending on the stereotype :

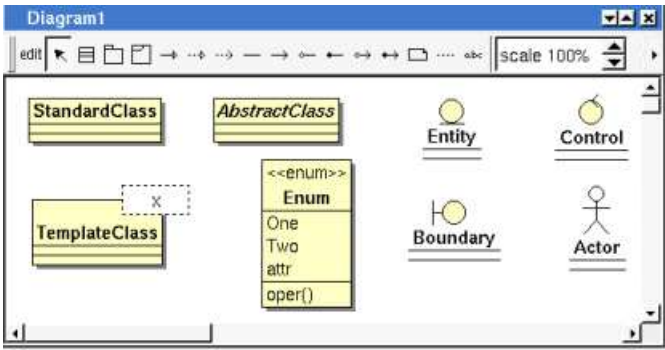


The tab *Description* allows to set a default description (the default description is used when you hit the button *default* in the description part of a class of the class dialog) :



## Drawing

A class is drawn in a diagram as a rectangle or an icon depending on the class's stereotype and the [drawing settings](#) :



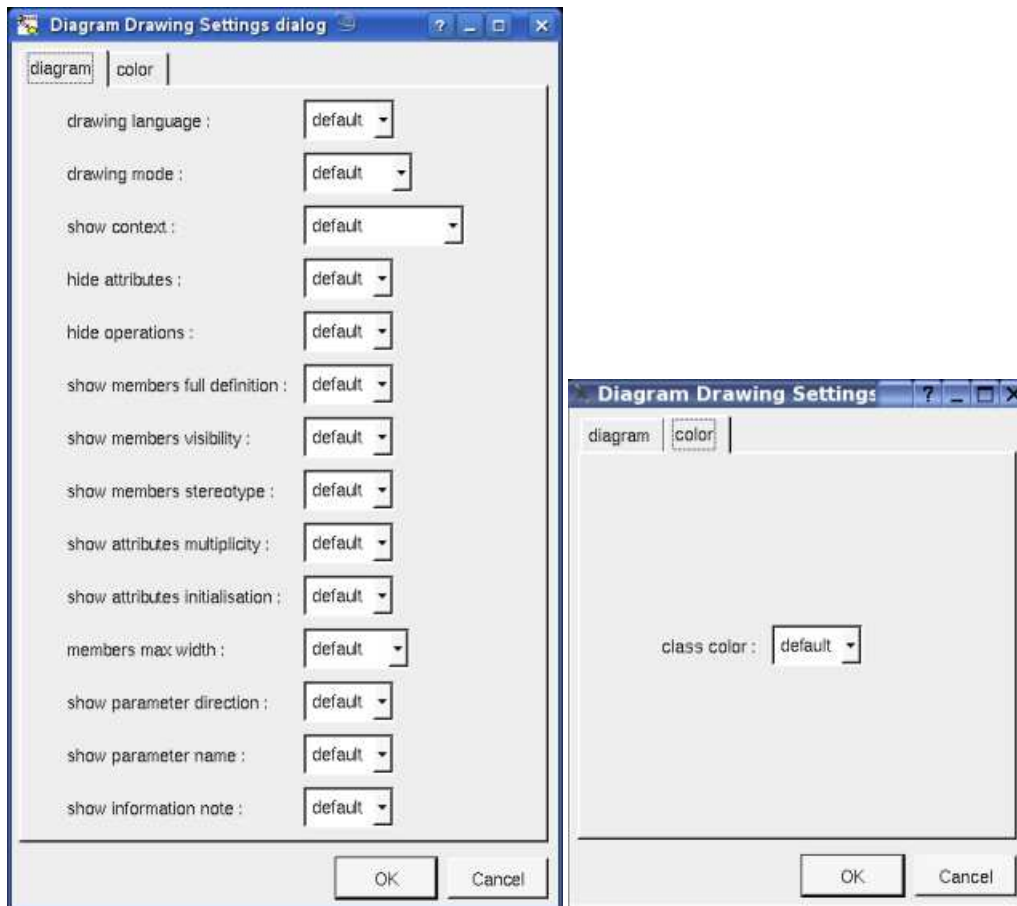
A right mouse click on a class in a diagram calls the following menu (supposing the class editable) :

StandardClass
upper
lower
edit drawing settings
individual attribut visibility
individual operation visibility
edit class
add attribut
edit attribut
add operation
edit operation
select in browser
set associated diagram
remove from view
delete from model
generate
tool

**Menu : edit drawing settings**

These *drawing settings* concerns only the picture for which the menu is called.

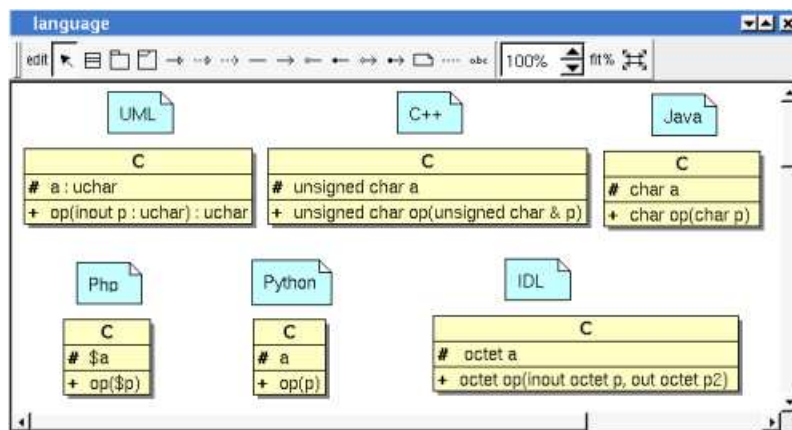
A *settings* valuing *default* indicates that the setting specified in the upper level (here the diagram) must be followed, obviously this one may also be *default* ... up to the project level. When you add a class in a diagram, all the settings are set to *default*.



### drawing language :

Allows to choose how the operations and attributes of the class must be written in case *show full members definition* is true (at the class level or an upper level), this may hide an operation or an attribute when it is not defined for the chosen language.

For instance, the class *C* has the attribute *a* having the type *uchar* (supposed translated *unsigned char* in C++, *char* in Java and *octet* in Idl), the operation *op* returning an *uchar* and having the *input* parameter *p* having the type *uchar* and only in Idl a second *out* parameter *p2* of the same type. The *C* picture is, depending on the *drawing language* :



### drawing mode :

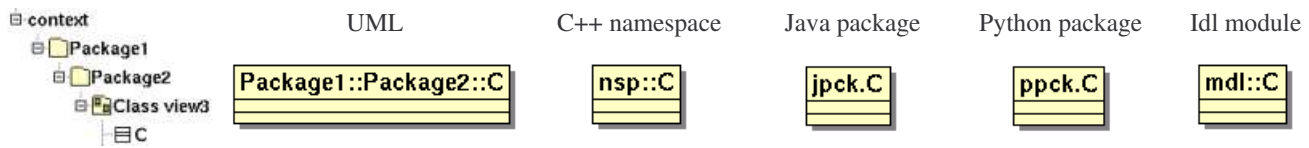
Allows to draw the class using a rectangle or an icon :

- class* : to draw a rectangle whatever the stereotype of the class
- control* : to draw the *control* icon whatever the stereotype of the class
- boundary* : to draw the *boundary* icon whatever the stereotype of the class
- entity* : to draw the *entity* icon whatever the stereotype of the class
- actor* : to an actor whatever the stereotype of the class
- natural* : the draw the class depending on its the stereotype

### show context :

To indicate if the context where the class is defined must be written, and if yes, how. The context may be the “UML context” which is the path of the class in the browser, or the C++ *namespace*, or the Java *package* or at least the Idl *module*. In case the class has an associated artifact, the *namespace/package/module* is the one specified by the [package](#) containing the artifact, else by the package containing the class itself. Obviously nothing is written in case the *namespace/package/module* is not specified.

For instance, the package *Package2* specify the C++ *namespace nsp*, the Java *package jpck*, the python package *ppck* and the Idl *module mdl*. The C picture is, depending on the *show context* :



#### hide attributes :

To hide or not the attributes, it is also possible to [specify](#) the visibility for each one.

#### hide operations :

To hide or not the operations, it is also possible to [specify](#) the visibility for each one.

#### show full members definition :

To write all the attribute/operation definitions, or just their names. See also [drawing language](#).

#### show members visibility :

To write or not the visibility

#### show attributes multiplicity :

To write or not the multiplicity of the attributes when the drawing language is UML and you ask to show for full members definition

#### show attributes initialization :

To write or not the default value of the attributes when you ask to show for full members definition

#### member max width :

To limit the width (in characters) used to produce the definition of the operations and attributes. When the width is greater that the max width, the string is cut and ... is added. Doesn't take into account the stereotype.

#### show parameter direction :

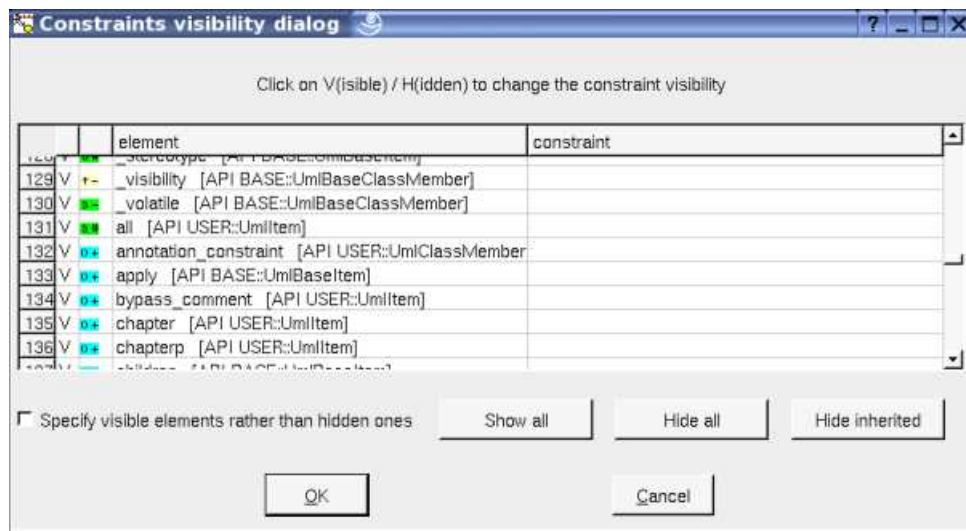
To write or not the direction of the operation's parameters

#### show parameter name :

To write or not the name of the operation's parameters

#### show information note :

To show or not the constraints as a note. This note is able to contain the constraints of the class itself, the constraints of its member and the inherited constraints. To choose which constraint must be written, edit the node and a dedicated dialog will appears :

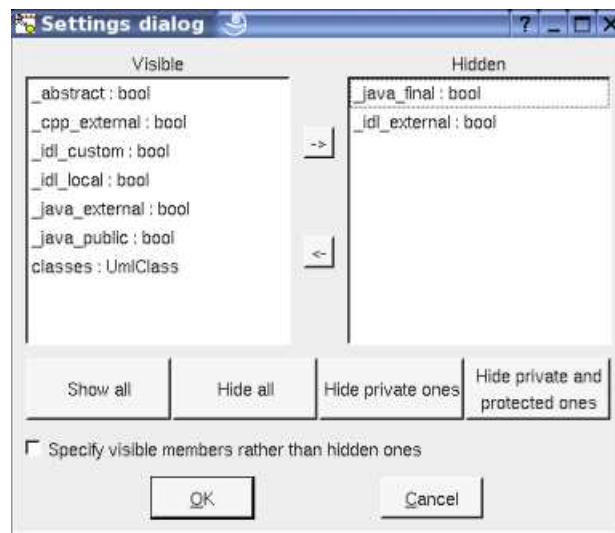


### class color :

To specify the fill color.

### Menu : individual attribute visibility :

To specify the hidden / written attributes of the class through the following dialog :

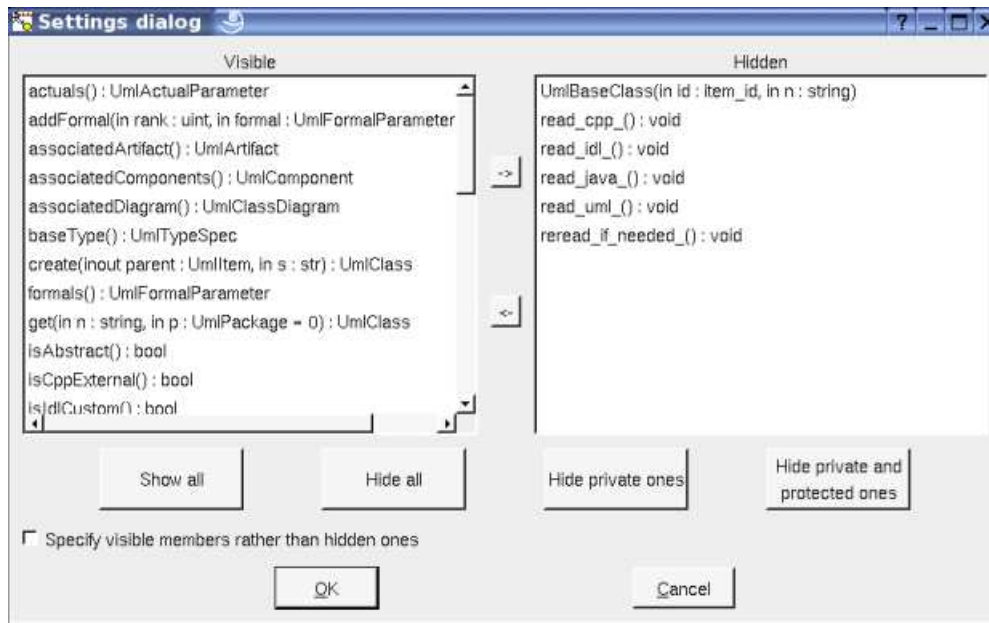


If you set *specify visible members rather than hidden ones*, when you will add new attributes these ones will not be added in the class picture.

### Menu : individual operation visibility :

To specify the hidden / written operations of the class through the following dialog :





If you set *specify visible members rather than hidden ones*, when you will add new operations these ones will not be added in the class picture.

### **Menu : set associated diagram**

*set* *associated* *diagram*  
allows to automatically open the current diagram when a double mouse click is made on the class representation in a diagram browser. After that the only way to edit the class is to choose the *edit* entry in the menu.

### **Menu tools :**

Appears only when at least one *plug-out* is associated to the classes. To apply a *plug-out* on the class.

---

Previous : [use case](#)

Next : [relation](#)



# Relation

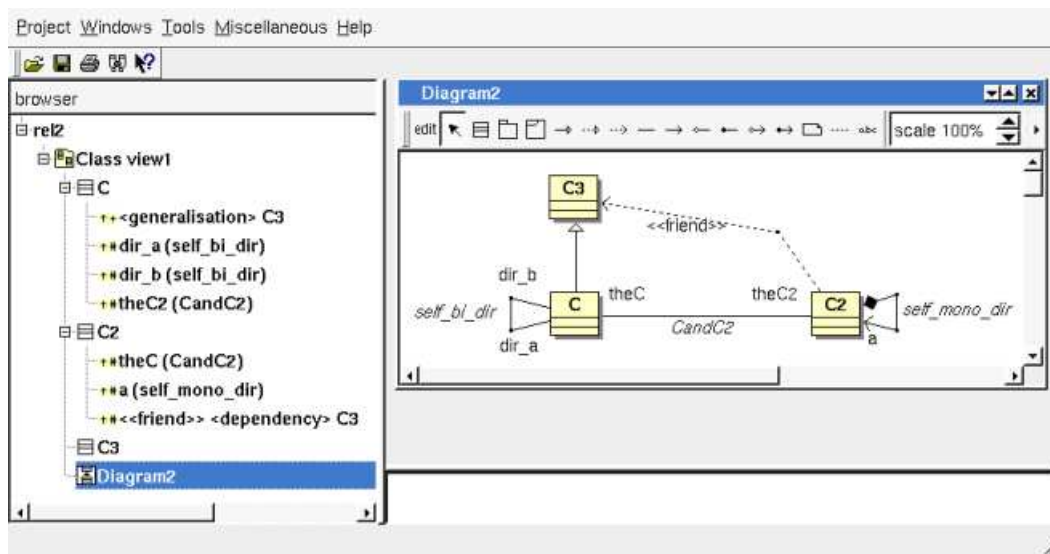
We consider in this chapter only the relations between classes. To add a relation to a class you have to use a class diagram (obviously the diagram or the relation drawing may be deleted just after !), it is not possible to add a relation through the browser. It is also possible to move a relation into an other class or elsewhere.

The icon in the browser indicates if the relation is public (+), protected (#), private (-) or package (~). The *protected* and *package* visibilities are translated in *private* in Idl. The *package* visibility is translated in *private* in C++, and doesn't produce a visibility in Php (use this case to produce Php 4 code). A class relation is underlined.

A bi-directional relation is supported by two relation items in the browser.

A relation have a name, a stereotype, may have one or two roles and multiplicity (one by direction). Note that it is not possible specify a stereotype for each direction of a bi directional relation, frankly I avow that the main reason is to simplify as it is possible the drawing of the relations in the diagrams with the associated labels.

In the browser are written the name of the relations (by default its kind) between parenthesis, the role(s) of the association aggregations, the end class of the generalization, realization and dependency. The stereotype may be may be showed/hidden through the [Miscellaneous menu](#). The name of a relation is not used by the code generators.



Generally the relation's stereotype is used to specify how a relation with a multiplicity different than 1 is translated in each language, for instance to produce a vector. The *friend* stereotype is a special case for a *dependency* and is used to produce a C++ *friend* declaration.

There are several kinds of relation :

- *generalization* : to produce an inheritance
- *realization* : produce also an inheritance, the code generators do not distinguish a *realization* and a *generalization*.
- *dependency* : only the *dependency* having the stereotype friend are considered by the C++ code generator.
- *unidirectional* *association*  
: to define a directional relation, the code generators will produce an attribute having the role name. In C++ the default generated code use pointer(s).
- *directional aggregation* : to define a directional relation, the code generators do not distinguish association and aggregation.
- *directional composition (directional aggregation by value)* : to define a directional composition, the code generators will produce an attribute having the role name. In C++ the default generated code does not use pointer(s).
- *association*  
: to define a bi-directional relation, the code generators will produce two attributes whose names are the roles's name.
- *aggregation*  
: to define a bi-directional aggregation, the code generators will produce two attributes whose names are the roles's name. This kind of relation may be considered to be a shortcut to define two (directional) aggregations.
- *composition* (aggregation by value)  
: to define a bi-directional composition, the code generators will produce two attributes whose names are the roles's name. This

kind of relation may considered to be a shortcut to define two (directional) aggregations by value.

By default, the example above produce the following codes (the visibility may be replaced by *var* to be compatible with Php 4) :

<b>C++</b> <pre>class C : public C3 { protected:     C * dir_a;     C * dir_b;     C2 * theC2; };  class C2 { protected:     C * theC;     C2 a; friend class C3; };</pre>	<b>Java</b> <pre>class C extends C3 { protected C dir_a; protected C dir_b; protected C2 theC2; }  class C2 { protected C theC; protected C2 a; }</pre>	<b>Php</b> <pre>class C extends C3 { protected \$dir_a; protected \$dir_b; protected \$theC2; }  class C2 { protected \$theC; protected \$a; }</pre>	<b>Idl</b> <pre>valuetype C : C3 {     private C dir_a;     private C dir_b;     private C2 theC2; };  valuetype C2 {     private C theC;     private C2 a; };</pre>
---	--	---	---

The multiplicity and stereotype may be used to produce the appropriate code with a natural and intuitive way. For instance the *C*'s relation *theC2* with the stereotype *vector* and multiplicity *\** will produce (depending on the [generation settings](#)) :

<b>C++</b> <pre>vector&lt;C2 *&gt; theC2;</pre>	<b>Java</b> <pre>protected Vector theC2;</pre>	<b>Idl</b> <pre>private sequence&lt;C2&gt; theC2;</pre>
--	---	--

With the stereotype *vector* and the multiplicity 12, the *C*'s relation *theC2* will produce (depending on the [generation settings](#)) :

<b>C++</b> <pre>C2 * theC2[12];</pre>	<b>Java</b> <pre>protected C2[] theC2;</pre>	<b>Idl</b> <pre>private sequence&lt;C2,12&gt; theC2;</pre>
--	---	---

Refer to the [C++ Generator](#), [Java generator](#), [Php Generator](#), [Python Generator](#) and [Idl generator](#) for additional information.

## Menus

The relation menu appearing with a right mouse click on its representation in the browser is something like these, supposing it is not *read-only* nor deleted (see also [menu in a diagram](#)) :



### Menu : edit

*edit* allows to show/modify the relations properties. In case the relation is read-only, the fields of the dialog are also read-only.

**Relation dialog, tab Uml**

The tab *Uml* is a global tab, independent of the language :

Here the two roles are editable because the relation is bi-directional, and because it is an association or aggregation.

The default **name** of a relation is its kind between < and >, for instance <association>, the default name does not appears in a diagram.

It is possible to change the kind of the relation through the combo box **type**, but it is not possible to reverse a relation, the role A is always attached to the start class used to draw the relation, and the role B is only available when the relation is bi-directional.

The proposed **stereotypes** are the default ones specified through the [Default stereotypes dialog](#) more the current one (empty here). It is possible to choose into the list or to give a new one, or to empty it. The stereotype and the multiplicity are used both (may be with the stereotype of the class(es) in Idl) to compute the default declaration.

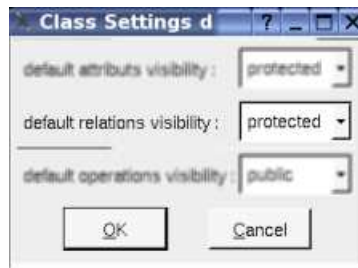
**Association** allows to define a *class association*, to produce it in the generated code use the keyword `${association}`

By default, the names of the generated attributes are the **role names**.

The **multiplicity** is used to compute the default declaration and the generated code for each language, the proposed ones are not the alone know by code generators, you may also use a number or a number between [] (they are not distinguished by BOUML) to specify a vector, or a sequence or numbers each between [] to specify a multi dimensioned array.

The **initial** **value** give the default value of the generated attribute(s), in C++ it is taken into account only when the relation is a read only class relation. By default the generated form is exactly the one you give, this means that you have to give the '=' when it must be generated, this allows to give the arguments of a constructor.

The **visibility** may be chosen with the *radio buttons*, the relations have a default visibility set through the [class settings](#) :



By default the *description* is used to produce comments by the code generators.

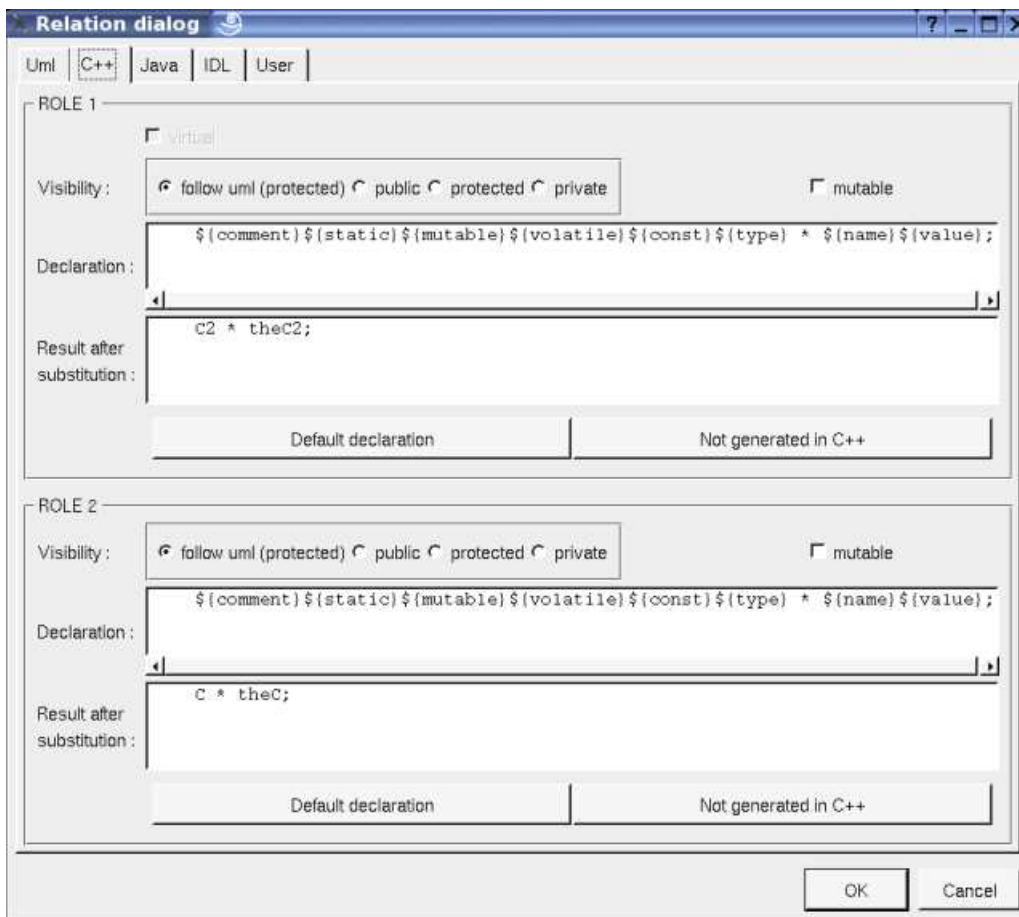
The *editor* button visible above and associated here to the description and initial value, allows to edit the description or initial value in an window, or to call an external editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note that this external editor have to create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel with BOUML which looks each second at the file contents to get the new definition until the dialog is closed (I do not like very much this polling but this works even QT isn't compiled with the thread support).

The *default* button visible above associated to the description allows to set the description with a default contain specified through the *generation settings*. Useful to generate comments compatible Java Doc or Doxygen for instance.

### Relation dialog, tab C++

This tab allows to give the C++ definition of the relation, it is visible only when C++ is set through the *Languages* menu

For an association or an aggregation without multiplicity :



for a generalization, realization or a dependency :

In BOUML the generated code is obtained by the substitution of macros in a text, only the **Declaration** part is editable, the other one help you to see what will be generated for C++ (supposing you do not modify the C++ code generator !).

When you push the button **default declaration**, the form specified through the [generation settings](#) depending on the stereotype and multiplicity (and of course the language !) is proposed, this last may be modified as you want, even to produce illegal source code.

When you do not want to have this relation defined in C++, empties the declaration manually or using the button **not generated in C++**.

It is possible to follow the **visibility** as it is set in the UML Tab or to change it.

`${static}` produce an empty string when the relation is not an class relation (see the UML tab), else produce *static*

`${const}` produce an empty string when the relation is not read only (see the UML tab), else produce *const*

`${type}` is replaced by the class pointed by the relation (see the UML tab)

`${name}` is replaced by the relation's role name (see the UML tab)

`${stereotype}` is replaced by the stereotype translated in C++ (first tab of the *generation setting* dialog)

`${multiplicity}` is replaced by the multiplicity (see the UML tab)

`${h_value}` is replaced in the header file by the initial value of the relation (see the UML tab)

`${mutable}` produce an empty string when the check box **mutable** is not checked, else produce *mutable*

`${volatile}` produce an empty string when the check box **volatile** is not checked (see the UML tab), else produce *volatile*

`${association}` produce the association class

The `@{property}` forms are replaced by the value of the corresponding user property if it is defined for the class or at an upper level, else it is not substituted.

When the check box **virtual** is checked the inheritance is *virtual*.

Refer to the [generation settings](#) and the [C++ Generator](#) for more details and explanation of the other keywords.

#### Relation dialog, tab Java

This tab allows to give the Java definition of the relation, it is visible only when Java is set through the *Languages* menu

For an association or an aggregation without multiplicity :

The screenshot shows the 'Relation dialog' window with the 'Java' tab active. It displays the configuration for two roles, 'ROLE 1' and 'ROLE 2'. For each role, there is a 'transient' checkbox, a 'Declaration' field containing a macro template, a 'Result after substitution' field showing the resulting Java code, and three buttons: 'Default declaration', 'Not generated in Java', and 'Edit annotation'. The 'OK' and 'Cancel' buttons are at the bottom right.

In BOUML the generated code is obtained by the substitution of macros in a text, only the **Declaration** part is editable, the other one help you to see what will be generated for Java (supposing you do not modify the Java code generator !).

When you push the button **default declaration**, the form specified through the [generation settings](#) depending on the stereotype and multiplicity (and of course the language !) is proposed, this last may be modified as you want, even to produce illegal source code.

When you do not want to have this relation defined in Java, empties the declaration manually or using the button **not generated in Java**.

It is possible to follow the **visibility** as it is set in the UML Tab or to change it replacing `${visibility}` by the desired one.

`${static}` produce an empty string when the relation is not an class relation (see the UML tab), else produce *static*

`${final}` produce an empty string when the relation is not read only (see the UML tab), else produce *final*

`${volatile}` produce an empty string when the check box **volatile** is not checked (see the UML tab), else produce *volatile*

`${transient}` produce an empty string when check box **transient** is not checked, else produce *transient*

`${type}` is replaced by the class pointed by the relation (see the UML tab)

`${stereotype}` is replaced by the stereotype translated in Java (first tab of the *generation setting* dialog)

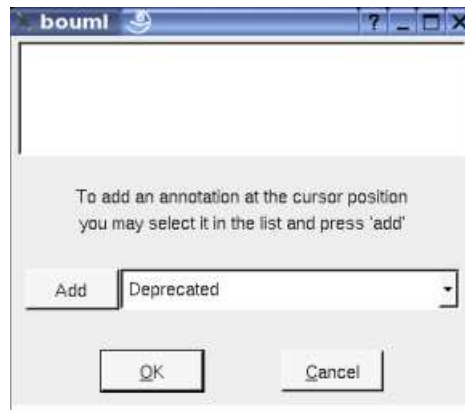
`${multiplicity}` is replaced by the multiplicity, a form [123] become []. (see the UML tab)

`${name}` is replaced by the relation's role name (see the UML tab)

`${value}` is replaced by the initial value of the relation (see the UML tab)

`${association}` produce the association class

`#{@}` produces the annotations, when it is present in the definition the button *Edit annotation* is active, a specific dialog allows you to enter the annotations proposing the default annotations (*Deprecated ...*) and the ones defined through the classes stereotyped *@interface* :

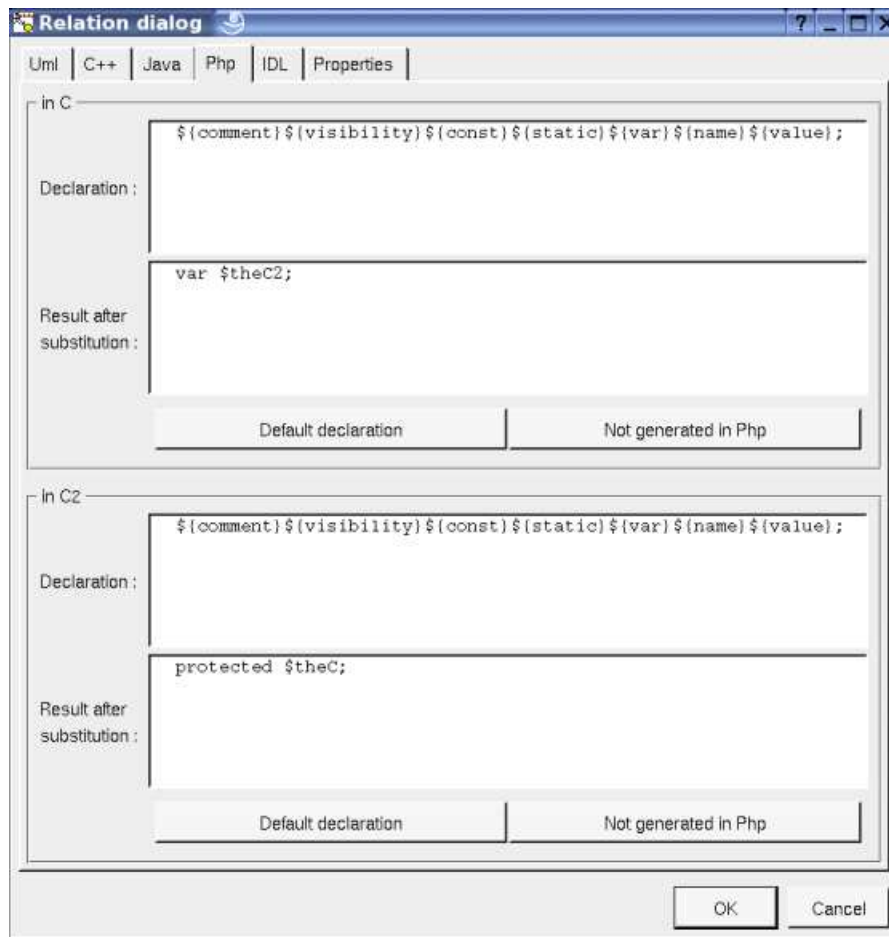


The forms @{property}  
are replaced by the value of the corresponding user property if it is defined for the class or at an upper level, else it is not substituted.

Refer to the [generation settings](#) and [Java generator](#) for more details and explanation of the other keywords.

### Relation dialog, tab Php

This tab allows to give the Php definition of the relation, it is visible only when Php is set through the *Languages* menu



In BOUML the generated code is obtained by the substitution of macros in a text, only the **Declaration** part is editable, the other one help you to see what will be generated for Php (supposing you do not modify the Php code generator !).

When you push the button **default declaration**, the form specified through the [generation settings](#) depending on the stereotype and multiplicity (and of course the language !) is proposed, this last may be modified as you want, even to produce illegal source code.

When you do not want to have this relation defined in Php, empties the declaration manually or using the button **not generated in Php**.

It is possible to follow the **visibility** as it is set in the UML Tab or to change it replacing `$(visibility)` by the desired one, to not generate the visibility set it to *package*

`$(static)` produce an empty string when the relation is not an class relation (see the UML tab), else produce *static*



`${const}` produce an empty string when the relation is not read only (see the UML tab), else produce `const`

`${var}` produce `var` when the relation is not read only nor static(see the UML tab) and the visibility is not `package`, else produce an empty string

`${name}` is replaced by the relation's role name (see the UML tab)

`${value}` is replaced by the initial value of the relation (see the UML tab)

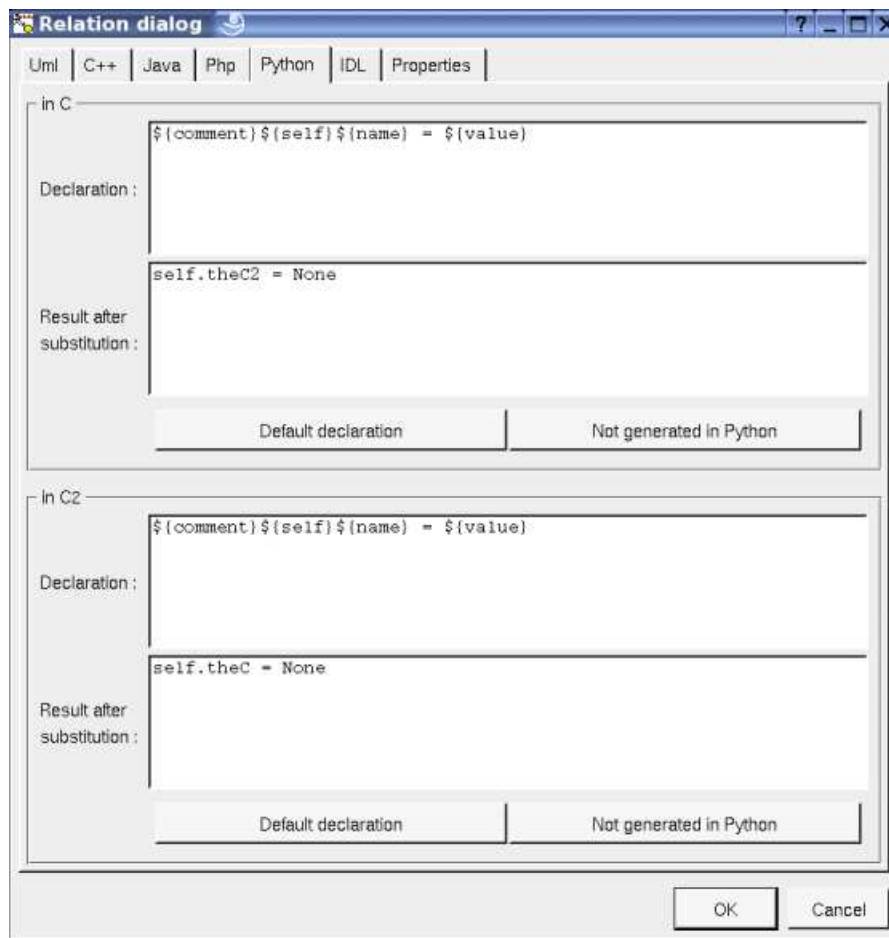
The `@{property}` forms are replaced by the value of the corresponding user property if it is defined for the class or at an upper level, else it is not substituted.

Refer to the [generation settings](#) and [Php generator](#) for more details and explanation of the other keywords.

### Relation dialog, tab Python

This tab allows to give the Python definition of the relation, it is visible only when Python is set through the *Languages* menu

For an association or an aggregation without multiplicity or having the multiplicity 1:



In BOUML the generated code is obtained by the substitution of macros in a text, only the **Declaration** part is editable, the other one help you to see what will be generated for Java (supposing you do not modify the Java code generator !).

When you push the button **default declaration**, the form specified through the [generation settings](#) depending on the stereotype and multiplicity (and of course the language !) is proposed, this last may be modified as you want, even to produce illegal source code.

When you do not want to have this relation defined in Java, empties the declaration manually or using the button **not generated in Python**.

The instance relations are automatically produced at the beginning of the body of the operation `__init__` (this operation is produced for each class even when you don't define it). The static relation are produced inside the class definition.

`${comment}`

produces a comment from the UML description, adding a # at the beginning of each line, following the current indent. A new line is produced if needed at the end to not have something else generated on the same line after the comment.

`${description}` is like for `${comment}` but without adding #.



`#{self}` inside the operation `__init__` this produces the name of the first parameter followed by a dot, else nothing.

`#{type}` produces the type of the relation, it is used by default when the relation is a composition to create an instance of this type

`#{multiplicity}` produces the multiplicity of the relation, probably used inside a comment

`#{stereotype}` is replaced by the [translation in Python](#) of the relation's stereotype (see the UML tab)

`#{name}` is replaced by the role name (see the UML tab)

`#{value}` is replaced by the initial value of the relation (see UML tab) if it is not empty, else *None*

The forms `@{property}` are replaced by the value of the corresponding user property if it is defined for the class or at an upper level, else it is not substituted.

Refer to the [generation settings](#) and [Python generator](#) for more details and explanation of the other keywords.

## Relation dialog, tab Idl

This tab allows to give the Idl definition of the relation, it is visible only when Idl is set through the *Languages* menu

For an association or an aggregation without multiplicity :

The screenshot shows the 'Relation dialog' window with the 'IDL' tab selected. The window is divided into two main sections, 'ROLE 1' and 'ROLE 2'. Each section contains a 'Declaration' field and a 'Result after substitution' field. The 'Declaration' field for both roles contains the macro substitution: `#{comment}#{visibility}#{type} #{name};`. The 'Result after substitution' field for ROLE 1 shows the generated code: `private C2 theC2;`, and for ROLE 2, it shows `private C theC;`. Below each 'Result after substitution' field are two buttons: 'Default declaration' and 'Not generated in Idl'. The 'Default declaration' button is highlighted. At the bottom of the dialog are 'OK' and 'Cancel' buttons.

In BOUML the generated code is obtained by the substitution of macros in a text, only the **Declaration** part is editable, the other one help you to see what will be generated for Idl (supposing you do not modify the Idl code generator !).

When you push the button **default declaration**, the form specified through the [generation settings](#) depending on the stereotype of the class, the stereotype and multiplicity of the relation (and of course the language !) is proposed, this last may be modified as you want, even to produce illegal source code.

When you do not want to have this relation defined in Idl, empties the declaration manually or using the button **not generated in Idl**

It is possible to follow the **visibility** as it is set in the UML Tab or to change it replacing `#{visibility}` by the desired one.

`#{type}` is replaced by the class pointed by the relation (see the UML tab)

*\$(name)* is replaced by the relation's role name (see the UML tab)

The **check** **box** *truncatable*  
is available when the relation is an attribute or a generalization or a realization, and allows to produce the Idl keyword *truncatable*.

The *forms* *@{property}*  
are replaced by the value of the corresponding user property if it is defined for the class or at an upper level, else it is not substituted.

Refer to the [generation settings](#) and [Idl generator](#) for more details and explanation of the other keywords.

**Menu : duplicate**

The **menu** **entry** *duplicate*  
clone the relation in the two directions when the relation is bi-directional. The name of the role(s) is empty in the new relation(s).

**Menu : add ...**

These entries allow to produce get and set operations on the relation, contrarily to the similar operations made by hand, these two ones are linked to the associated relation to be updated, or deleted when the relations is deleted. The default name of the operations and their characteristics are set through the [generation settings](#).

By default, the get and set operations generated for the *C*'s relation *theC2* are :

<b>C++</b> <pre>inline const C2 * C::get_theC2() const {     return theC2; }  void C::set_theC2(C2 * value) {     theC2 = value; }</pre>	<b>Java</b> <pre>public final C2 get_theC2() {     return theC2; }  public void set_theC2(C2 value) {     theC2 = value; }</pre>	<b>Php</b> <pre>final public function getTheC2() {     return \$this-&gt;theC2; }  public function setTheC2(\$value) {     \$this-&gt;theC2 = \$value; }</pre>
---	---	---

<b>Python</b> <pre>def getTheC2(self):     return self.theC2  def setTheC2(self, value):     self.theC2 = value</pre>	<b>Idl</b> <pre>C2 get_theC2();  void set_theC2(C2 value);</pre>
--	---

**Menu : delete**

The menu entry *delete* is only present when the relation is not *read-only*.

Delete the relation, associated get and set operations when they exist, and all the representation of them in the opened diagrams. After that it is possible to undelete them (from the browser) until you close the project : obviously the deleted items are not saved !

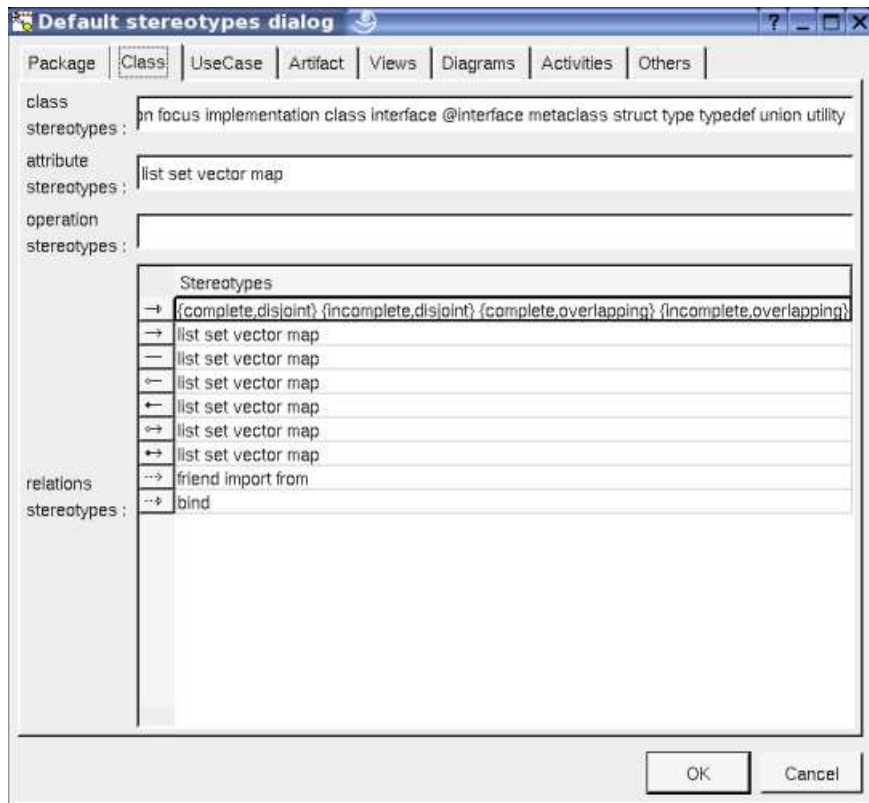
**Menu : mark**

See [mark](#)

---

## Default stereotypes

The [dialog](#) allowing to set the default stereotypes has a tab reserved for the relations :

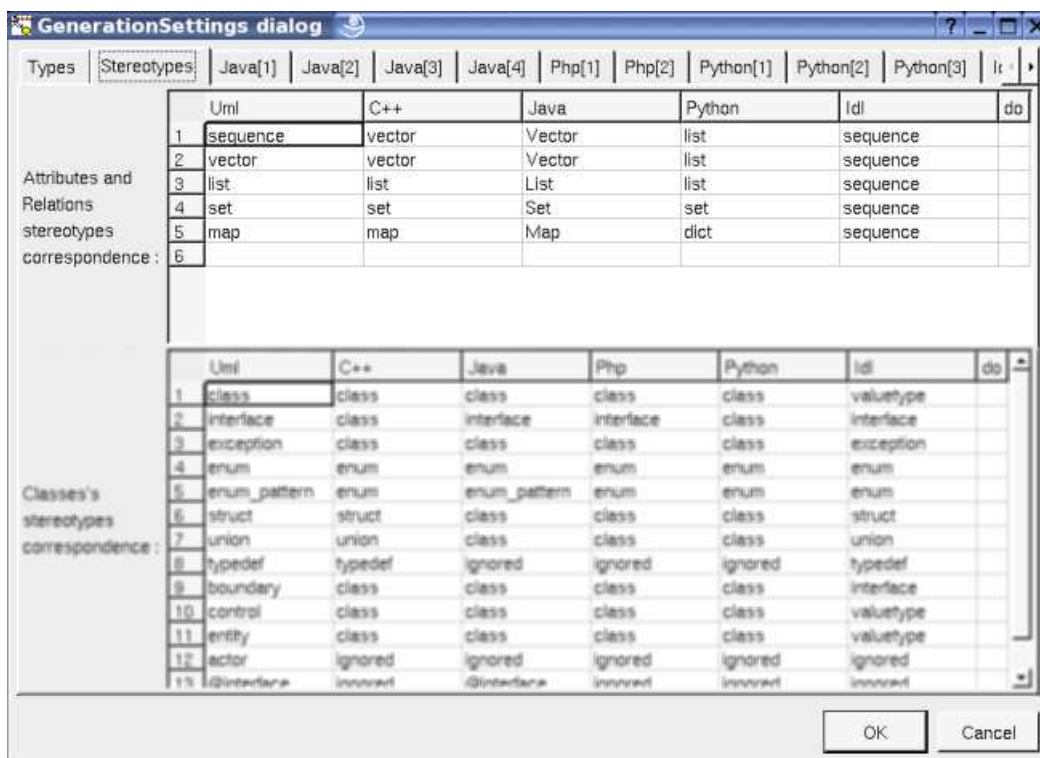


As you can see it is possible to specify a default stereotypes list for each kind of relation. Associated to a multiplicity the stereotype is used by the code generators to produce the desired forms, the translation in each language of the UML stereotype is specified through the [generation settings](#).

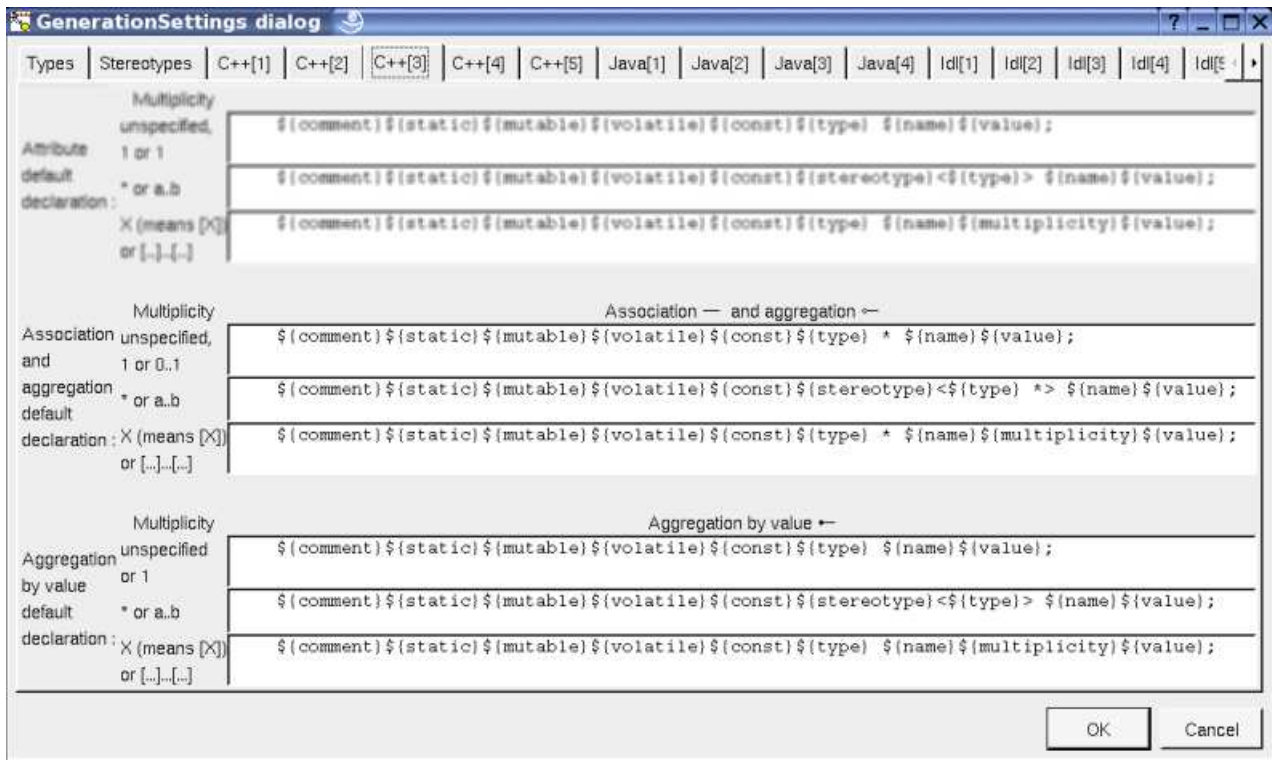
## Generation settings

This very important dialog allows to specify many default definitions concerning the relations, more details will be given in [C++ generator](#), [Java generator](#), [Python generator](#) and [Idl generator](#).

The tab *Stereotypes* allows to specify how the relation's stereotypes are translated in each language except Php :



The third C++ tab allows to specify the default C++ generated code for the relations :



The image shows the 'GenerationSettings dialog' with the 'C++[3]' tab selected. The dialog is divided into three main sections: 'Multiplicity', 'Association and aggregation', and 'Aggregation by value'. Each section contains a table with columns for 'Multiplicity', 'Association and aggregation', and 'Aggregation by value'. The 'Multiplicity' section shows templates for attribute, association, and aggregation. The 'Association and aggregation' section shows templates for association and aggregation. The 'Aggregation by value' section shows templates for aggregation by value. The 'C++[3]' tab is highlighted in the top tab bar.

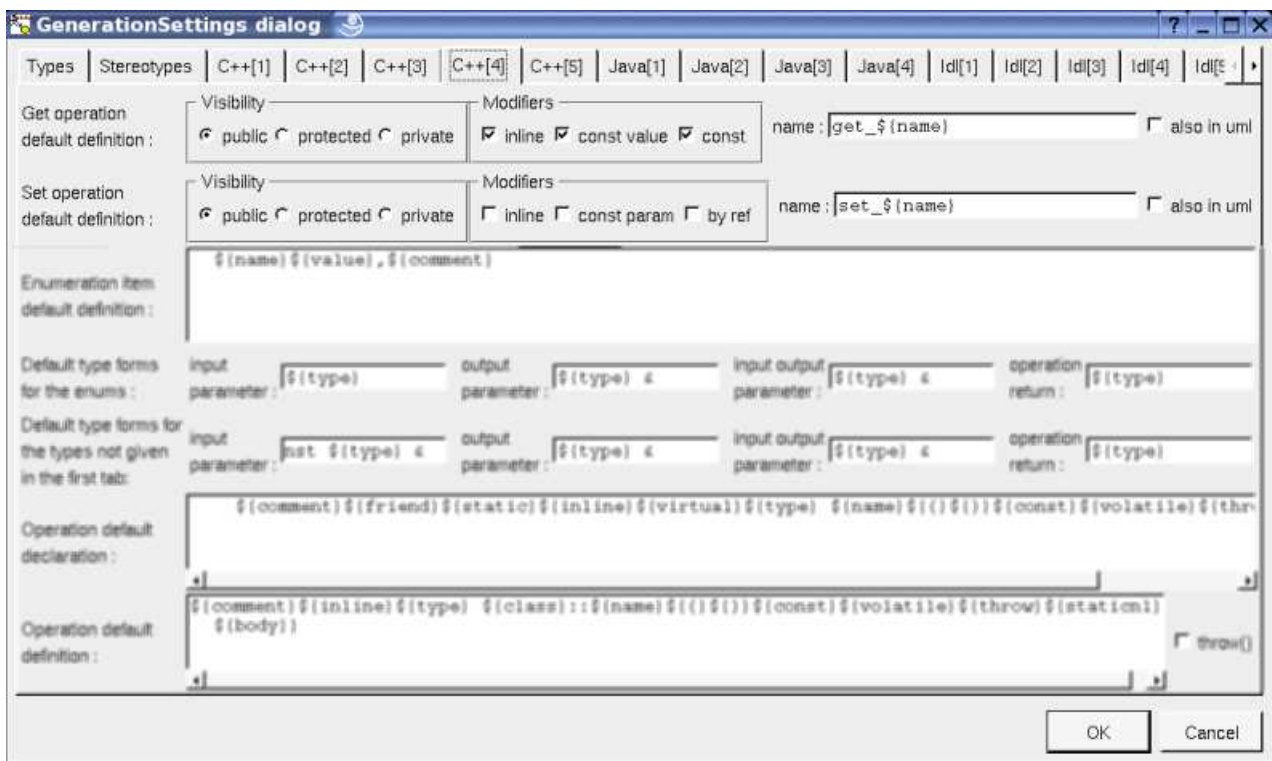
Multiplicity	Association and aggregation	Aggregation by value
unspecified, 1 or 1	unspecified, 1 or 0..1	unspecified or 1
default: * or a..b	default: * or a..b	default: * or a..b
declaration: X (means [X]) or [...]..[...]	declaration: X (means [X]) or [...]..[...]	declaration: X (means [X]) or [...]..[...]

Templates shown in the dialog:

- Multiplicity: `{comment}{static}{mutable}{volatile}{const}{type} {name}{value};`
- Association: `{comment}{static}{mutable}{volatile}{const}{stereotype}<{type}> {name}{value};`
- Aggregation: `{comment}{static}{mutable}{volatile}{const}{stereotype}<{type}> {name}{value};`
- Aggregation by value: `{comment}{static}{mutable}{volatile}{const}{type} {name}{value};`

Buttons: OK, Cancel

The fourth C++ tab allows to specify the get and set operations specificities in C++ :



The image shows the 'GenerationSettings dialog' with the 'C++[4]' tab selected. The dialog is divided into several sections: 'Get operation', 'Set operation', 'Enumeration item', 'Default type forms for the enums', 'Default type forms for the types not given in the first tab', 'Operation default declaration', and 'Operation default definition'. The 'Get operation' and 'Set operation' sections have 'Visibility' and 'Modifiers' tabs. The 'Enumeration item' section has a 'default definition' field. The 'Default type forms' sections have 'input parameter', 'output parameter', 'input output parameter', and 'operation return' fields. The 'Operation default declaration' and 'Operation default definition' sections have text areas for code templates. The 'C++[4]' tab is highlighted in the top tab bar.

Sections and fields:

- Get operation:** Visibility (public, protected, private), Modifiers (inline, const value, const), name: `get_{name}`, also in uml.
- Set operation:** Visibility (public, protected, private), Modifiers (inline, const param, by ref), name: `set_{name}`, also in uml.
- Enumeration item:** default definition: `{name}{value}, {comment}`
- Default type forms for the enums:** input parameter: `{type}`, output parameter: `{type} &`, input output parameter: `{type} &`, operation return: `{type}`
- Default type forms for the types not given in the first tab:** input parameter: `just {type} &`, output parameter: `{type} &`, input output parameter: `{type} &`, operation return: `{type}`
- Operation default declaration:** `{comment}{friend}{static}{inline}{virtual}{type} {name}{()}{const}{volatile}{throw}`
- Operation default definition:** `{comment}{inline}{type} {class}:: {name}{()}{const}{volatile}{throw}{static}{body}`

Buttons: OK, Cancel

The form in from of *also in uml* allows to specify the name of the operation in C++, the same rule is followed in UML if *also in uml* is set.

The second Java tab allows to specify the default Java generated code for the relations in Java :

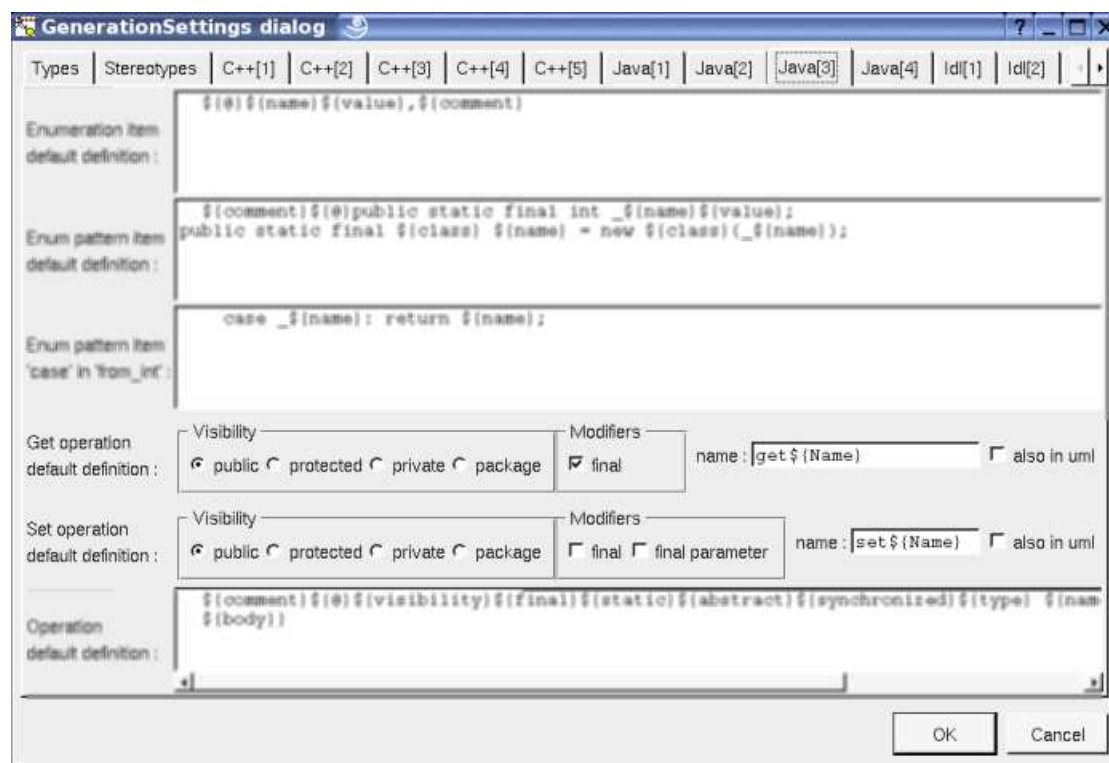


The screenshot shows the 'GenerationSettings dialog' with the 'Java[2]' tab selected. The dialog has a tabbed interface with tabs for Types, Stereotypes, C++[1] through C++[5], Java[1] through Java[4], Id[1] through Id[5], Description, and Dir. The 'Java[2]' tab contains three sections for default definitions:

- Attribute default declaration:**
  - multiplicity '1' or unspecified: `$(comment)$(@)$(visibility)$(static)$(final)$(transient)$(volatile)$(type) $(name)$(value);`
  - multiplicity '\*' or 'a..b': `$(comment)$(@)$(visibility)$(static)$(final)$(transient)$(volatile)$(stereotype) $(name)$(value);`
  - X (probably a number): `$(comment)$(@)$(visibility)$(static)$(final)$(transient)$(volatile)$(type)$(multiplicity) $(name)$(val`
- Association and aggregation default declaration:**
  - multiplicity '1' or unspecified: `$(comment)$(@)$(visibility)$(static)$(final)$(transient)$(volatile)$(type) $(name)$(value);`
  - multiplicity '\*' or 'a..b': `$(comment)$(@)$(visibility)$(static)$(final)$(transient)$(volatile)$(stereotype) $(name)$(value);`
  - X (probably a number): `$(comment)$(@)$(visibility)$(static)$(final)$(transient)$(volatile)$(type)$(multiplicity) $(name)$(val`

At the bottom right are 'OK' and 'Car' buttons.

The third Java tab allows to specify the get and set operations specificities in Java :



The screenshot shows the 'GenerationSettings dialog' with the 'Java[3]' tab selected. The dialog contains several sections for default definitions and operation specificities:

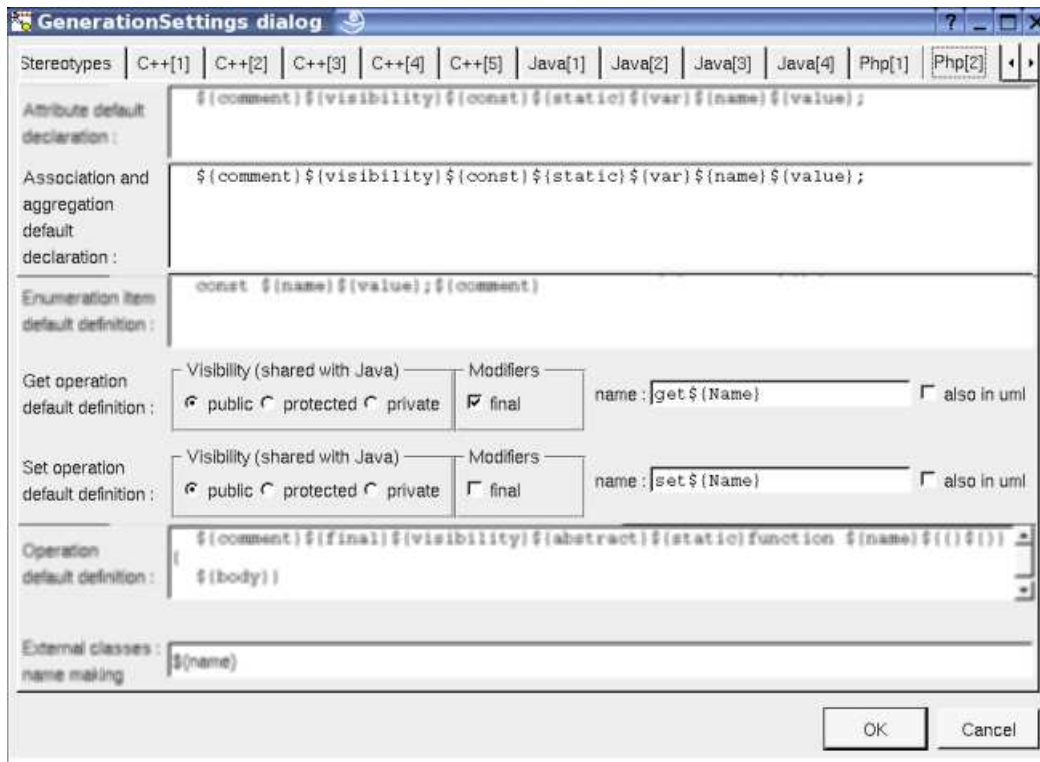
- Enumeration item default definition:** `$(@)$(name)$(value),$(comment)`
- Enum pattern item default definition:** `$(comment)$(@)public static final int _$(name)$(value);  
public static final $(class) $(name) = new $(class)(_$(name));`
- Enum pattern item 'case' in 'from\_int':** `case _$(name): return $(name);`
- Get operation default definition:**
  - Visibility: ☒ public ☐ protected ☐ private ☐ package
  - Modifiers: ☒ final
  - name:  ☐ also in uml
- Set operation default definition:**
  - Visibility: ☒ public ☐ protected ☐ private ☐ package
  - Modifiers: ☐ final ☐ final parameter
  - name:  ☐ also in uml
- Operation default definition:** `$(comment)$(@)$(visibility)$(final)$(static)$(abstract)$(synchronized)$(type) $(name)$(body)`

At the bottom right are 'OK' and 'Cancel' buttons.

The form in from of *also in uml* allows to specify the name of the operation in Java, the same rule is followed in UML if *also in uml* is set.

The second Php tab allows to specify the default Php generated code for the relations and to specify the get and set operation specificities in Php :





**GenerationSettings dialog**

Stereotypes | C++[1] | C++[2] | C++[3] | C++[4] | C++[5] | Java[1] | Java[2] | Java[3] | Java[4] | Php[1] | **Php[2]** | »

Attribute default declaration : `${comment}${visibility}${const}${static}${var}${name}${value};`

Association and aggregation default declaration : `${comment}${visibility}${const}${static}${var}${name}${value};`

Enumeration item default definition : `const ${name}${value};${comment}`

Get operation default definition :  
 Visibility (shared with Java) : ☒ public ☐ protected ☐ private  
 Modifiers : ☒ final  
 name : `get ${Name}` ☐ also in uml

Set operation default definition :  
 Visibility (shared with Java) : ☒ public ☐ protected ☐ private  
 Modifiers : ☐ final  
 name : `set ${Name}` ☐ also in uml

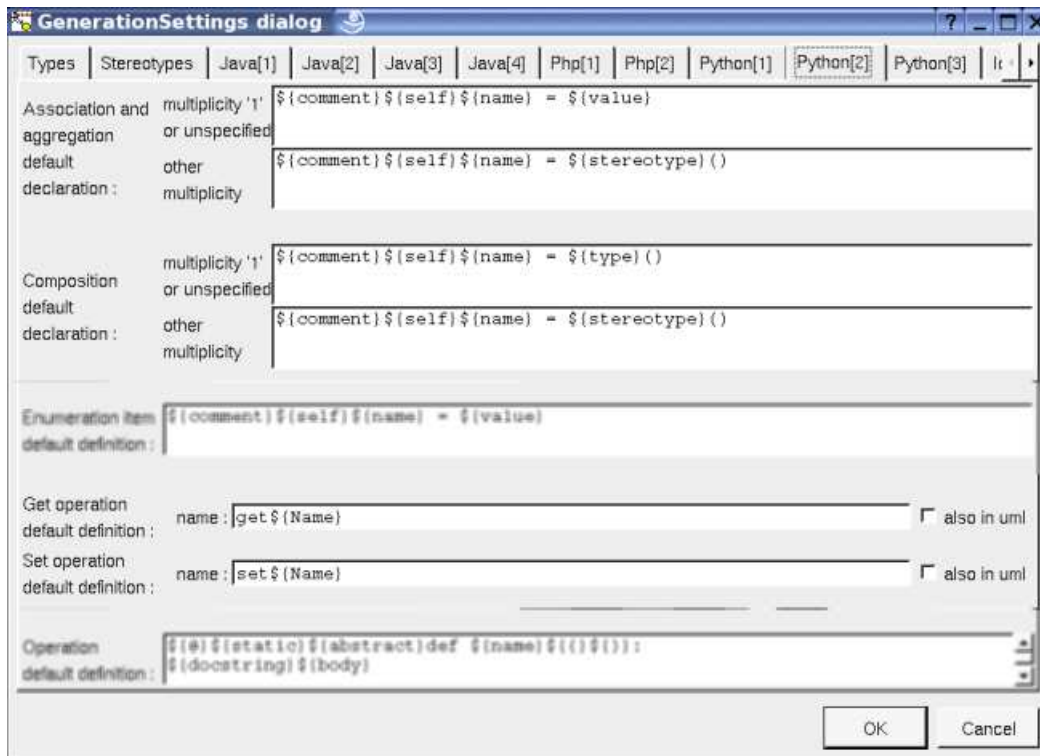
Operation default definition : `${comment}${final}${visibility}${abstract}${static}function ${name}${()}${()};  
{  
 ${body}  
}`

External classes :  
 name making : `${name}`

OK Cancel

The form in from of *also in uml* allows to specify the name of the operation in Php, the same rule is followed in UML if *also in uml* is set.

The second Python tab allows to specify the default Python generated code for the relations and to specify the get and set c specificities in Python :



**GenerationSettings dialog**

Types | Stereotypes | Java[1] | Java[2] | Java[3] | Java[4] | Php[1] | Php[2] | **Python[1]** | **Python[2]** | Python[3] | »

Association and aggregation default declaration :  
 multiplicity '1' or unspecified : `${comment}${self}${name} = ${value}`  
 other multiplicity : `${comment}${self}${name} = ${stereotype}()`

Composition default declaration :  
 multiplicity '1' or unspecified : `${comment}${self}${name} = ${type}()`  
 other multiplicity : `${comment}${self}${name} = ${stereotype}()`

Enumeration item default definition : `${comment}${self}${name} = ${value}`

Get operation default definition :  
 name : `get ${Name}` ☐ also in uml

Set operation default definition :  
 name : `set ${Name}` ☐ also in uml

Operation default definition : `[@${static}${abstract}def ${name}${()}${()};  
 ${docstring}${body}`

OK Cancel

The form in from of *also in uml* allows to specify the name of the operation in Python, the same rule is followed in UML if *also in uml* is set.

The third Idl tab allows to specify the default Idl generated code for the relations :

Types	Stereotypes	C++[1]	C++[2]	C++[3]	C++[4]	C++[5]	Java[1]	Java[2]	Java[3]	Java[4]	Idl[1]	Idl[2]	Idl[3]	Idl[4]
Association and aggregation default declaration :	multiplicity '1' or unspecified	<code>\${comment} \${readonly} \${attribute} \${type} \${name};</code>												
	multiplicity '*' or 'a..b'	<code>\${comment} \${readonly} \${attribute} \${stereotype} &lt;\${type}&gt; \${name};</code>												
	X (probably a number)	<code>\${comment} \${readonly} \${attribute} \${stereotype} &lt;\${type}, \${multiplicity}&gt; \${name};</code>												
Association and aggregation default declaration in valuetype :	multiplicity '1' or unspecified	<code>\${comment} \${visibility} \${type} \${name};</code>												
	multiplicity '*' or 'a..b'	<code>\${comment} \${visibility} \${stereotype} &lt;\${type}&gt; \${name};</code>												
	X (probably a number)	<code>\${comment} \${visibility} \${stereotype} &lt;\${type}, \${multiplicity}&gt; \${name};</code>												
Association and aggregation default declaration in union :	multiplicity '1' or unspecified	<code>\${comment} case \${case} : \${readonly} \${type} \${name};</code>												
	multiplicity '*' or 'a..b'	<code>\${comment} case \${case} : \${readonly} \${stereotype} &lt;\${type}&gt; \${name};</code>												
	X (probably a number)	<code>\${comment} case \${case} : \${readonly} \${stereotype} &lt;\${type}, \${multiplicity}&gt; \${name};</code>												

OK Cancel

At least the third Idl tab allows to specify the get and set operations specificities in Idl :

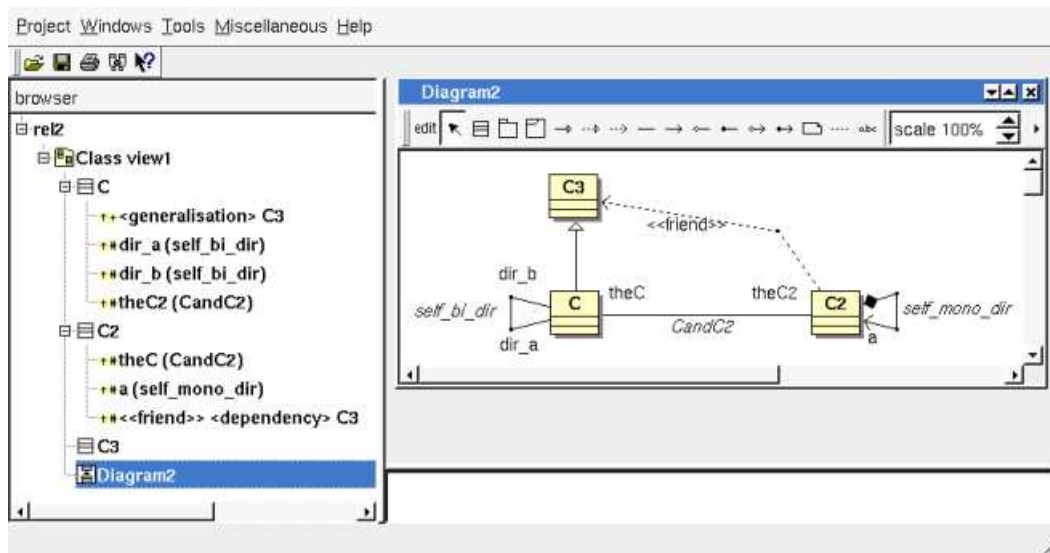
C++[1]	C++[2]	C++[3]	C++[4]	C++[5]	Java[1]	Java[2]	Java[3]	Java[4]	Idl[1]	Idl[2]	Idl[3]	Idl[4]
Union item default declaration :	multiplicity '1' or unspecified	<code>\${comment} case \${case} : \${readonly} \${type} \${name};</code>										
	multiplicity '*' or 'a..b'	<code>\${comment} case \${case} : \${readonly} \${stereotype} &lt;\${type}&gt; \${name};</code>										
	X (probably a number)	<code>\${comment} case \${case} : \${readonly} \${stereotype} &lt;\${type}, \${multiplicity}&gt; \${name};</code>										
Enumeration item default declaration :	<code>\${name}, \${comment}</code>											
Get operation default definition :	name : <code>get_\${name}</code>											<input type="checkbox"/> also in uml
Set operation default definition :	<input type="checkbox"/> oneway name : <code>set_\${name}</code>											<input type="checkbox"/> also in uml
Operation default declaration :	<code>\${comment} \${oneway} \${type} \${name} \${()} \${raisesn} \${raises};</code>											

OK Cancel

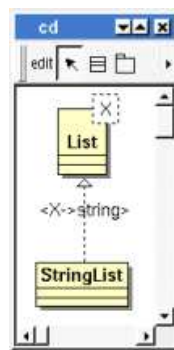
The form in from of *also in uml* allows to specify the name of the operation in Idl, the same rule is followed in UML if *also in uml* is set.

## Drawing

A relation is drawn in a diagram with a line may be with an arrow, and the associated labels : relation's name, stereotypes, role(s) and multiplicity(ies) :



When a class inherits a template class the *actuals* are indicated as a label of the *realization* :



To add a new relation between classes, select the appropriate relation through the buttons on the top of the diagram sub window, click on the start class with the left mouse button, move the mouse, each time the mouse button is raised a new line break is introduced, at least click on the end class. To abort a relation construction press the right mouse button or do a double click with the left mouse button.

A line may be broken, during its construction or after clicking on the line with the left mouse button and moving the mouse with mouse button still pushed. To remove a line break, a double click on the point with the left mouse button is enough, or use the line break menu of the point using the right mouse button.

By default the lines go to the center of their extremities, to decenter a line click near the desired extremity and move the mouse down. To come back to a center line, use the menu geometry

A right mouse click on a relation in a diagram or a double click with left mouse button calls the following menu (supposing the relation editable) :

CandC2
edit
select in browser
select labels
labels default position
geometry (Ctrl+I)
remove from view
delete from model
tool

### Menu : edit

To edit the relation

### Menu : select labels

To select the labels (relation's name, stereotypes, role(s) and multiplicity(ies)) associated to the relation. Useful when you are lost after many label movings.



**Menu : labels default position**

To place the labels (relation's name, stereotypes, role(s) and multiplicity(ies)) associated to the relation in the default position. Useful when you are lost after many label movings.

By default when you move a class or a relation point break or edit the relation, the associated labels are moved to their default position, this may be irritating. To ask BOUML to not move the associated labels on the relations in a diagram, use the [drawing settings](#) of the diagrams.

**Menu : geometry**

Allows to choose one of the line geometry or to recenter a decentered line :



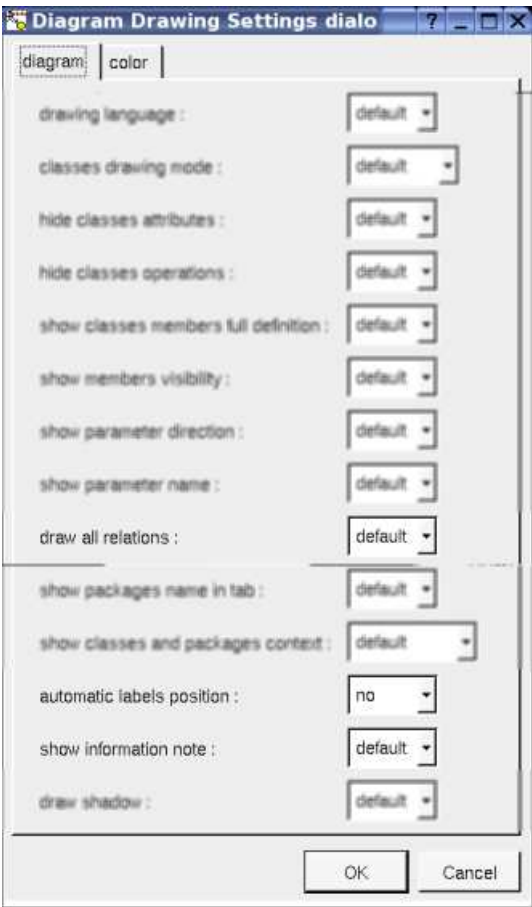
Note : if you manually move the central line of the last two geometries this one stop to be automatically updated when you move one of the two extremities of the relation.

**Menu : tools**

Appears only when at least one [plug-out](#) is associated to the relation. To apply a *plug-out* on the relation.

**Drawing settings**

The drawing settings associated to the relations in a diagram may be set through the the diagram itself or one of its parent, for instance :



A settings valuing default indicates that the setting specified in the upper level (here the diagram) must be followed, obviously this one may also be default ... up to the project level.

**draw all relations :**

To automatically draw or not the new relations, obviously a relation will be added in a diagram only when the start and end classes (may be the same) are drawn.

**class diagram automatic labels position :**

To ask BOUML to move or to not move the associated labels on the relations in a diagram when the relation is edited or the associated class pictures moved etc ...

**show information note :**

TO ask Bouml to write the relation's constraint in the note attached to the class (except if this setting is set to false on the class, or when you ask for hide at least relation's constraint).

---

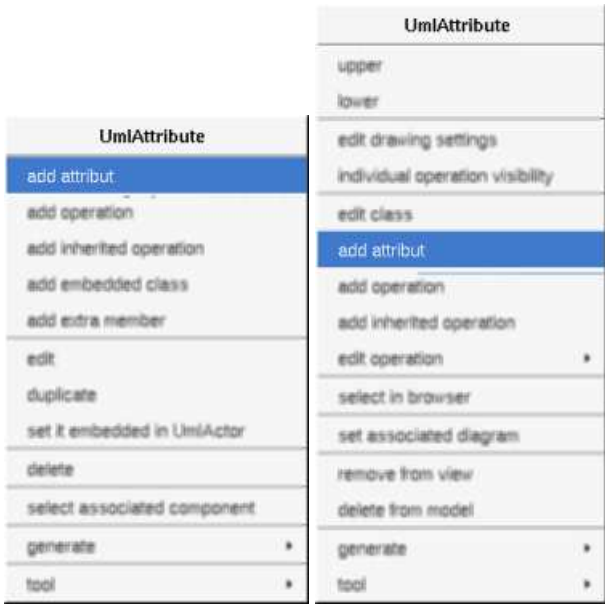
Previous : [class](#)

Next : [attribute](#)

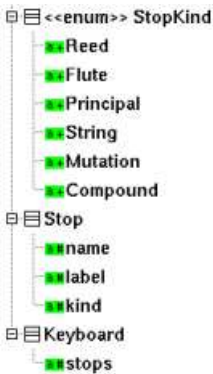
# Attribute

Contrarily to the relations, there is no restriction concerning the types of the attributes, whose may be a class or an *int* for instance.

An attribute is created through the class menu called from the [browser](#) or a class [picture](#) in a class diagram :



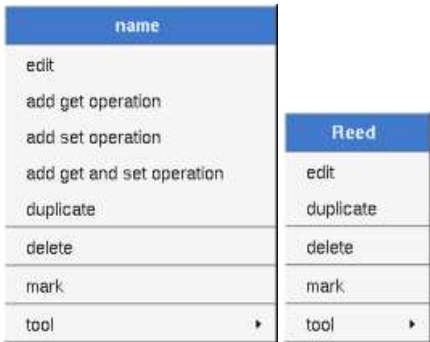
The icon in the browser indicates if an attribute is public (+), protected (#), private (-) or package (~). The *protected* and *package* visibilities are translated in *private* in Idl. The *package* visibility is translated in *private* in C++. A class attribute is underlined. It is possible to move an attribute from a class into an other using the *drag and drop* or the marked items capabilities, see [browser items](#).



In a class having the stereotype *enum*, the attributes are renames *item* in the menus.

## Menus

The attribute menu appearing with a right mouse click on its representation in the browser is something like these, supposing it is not *read-only* nor deleted (the second one is associated to an enumeration's item) :



[Menu](#) : edit

*edit* allows to show/modify the properties of the attribute. In case the attribute is read-only, the fields of the dialog are also read-only.

### Attribute dialog, tab Uml

The tab *Uml* is a global tab, independent of the language.

For a standard attribute :

Attribute dialog

Uml | C++ | Java | Php | Idl | Properties

class : Stop [Organ]

name : name

stereotype :

type : string

multiplicity :

Initial value : Editor

☐ public ☒ protected ☐ private ☐ package ☐ class attribute ☐ volatile ☐ read-only

description : the name of the stop.  
Reeds are : trumpet, cromorne, ...  
Flutes are : bourdon, rhorflote, ...  
Principal are : montre, prestant, ...  
Strings are : viola, unda maris, ...  
Mutation are : nazard, thierce, ...  
Compound are : mixture, cornet, ...

Editor

Default

constraint : Editor

OK Cancel

For an enumeration's item :

Enum item dialog

Uml | C++ | Java | Php | Idl | Properties

class : StopKind [Organ]

name : Reed

value : Editor

description : All the organ's pipes are not flue pipes,  
the reed pipes have a metal tongue (the reed)  
resting against a shallot.  
Editor

Default

constraint : Editor

OK Cancel

The proposed **types** for a standard attribute are the *non class* types defined in the first tab of the [generation settings](#), more all the classes defined in the project (with their localization to distinguish synonymous classes). You are not limited to the proposed list, and any form (even invalid for the target language(s)) may be given. The button *type* shows a menu proposing :

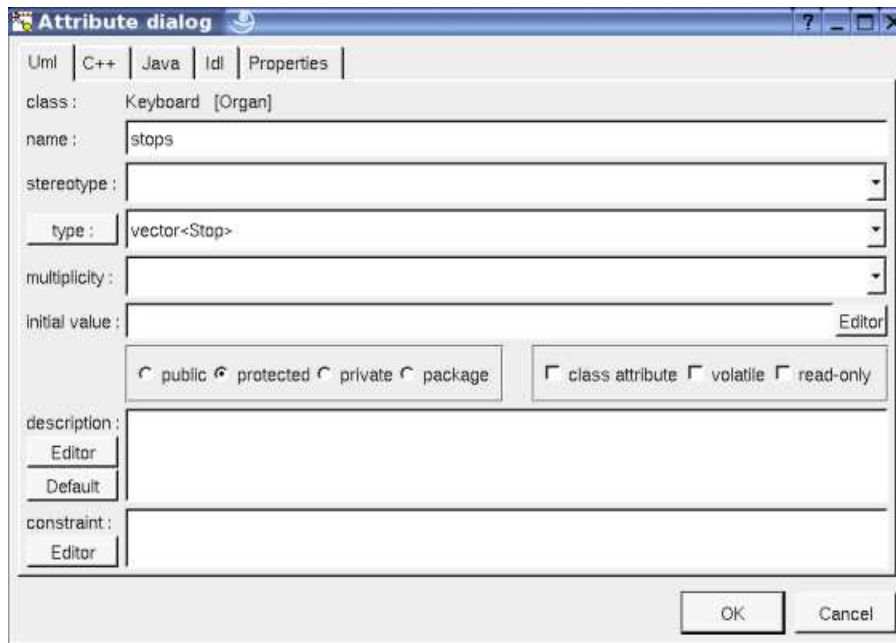
- if the current type of the attribute is a class : to select this class in the browser
- if the attribute is not read-only and if a class is selected in the browser : to set the type to be this class
- if the view containing the class containing the attribute is not read-only : to create a new class and to set the attribute type to it

Nevertheless

I

**strongly**

recommend you to not add type modifier (like \*, & etc ...) with a class name in the UML tab because BOUML differentiate a reference to a class of the model and any other forms. Supposing you also have the class attribute *stops* defined (for C++) like this :



the C++ generated code will be the desired one, but if you rename the class *Stop*, the type of the attribute *stops* will **not** be updated. Furthermore the C++ code generator will not see the usage of the type *Stop* and will not automatically generate class declaration and *#include* form for it.

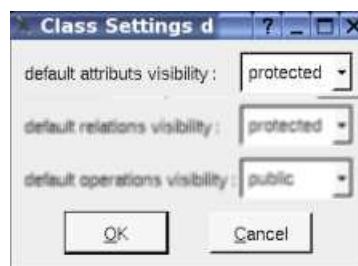
The good way may be :

- to say that *stops* is a *Stop* and modify the [attribute definition](#) in C++ to have a *vector*
- to set the stereotype to vector and the multiplicity to \*
- but the right way here is to use a relation with the stereotype *vector* and the multiplicity \* because *Stop* is a class

The proposed **stereotypes** are the default ones specified through the [Default stereotypes dialog](#) more the current one (empty here). It is possible to choose into the list or to give a new one, or to empty it. The stereotype of an attribute does not have special meaning for BOUML.

The **initial value** give the default value of the generated attribute(s), in C++ it is taken into account only when the attribute is a class attribute enumeration item. By default the generated form is exactly the one you give, this means that you have to give the '=' when it generated, this allows to give the arguments of a constructor.

The **visibility** may be chosen with the *radio buttons*, the attributes have a default visibility set through the [class settings](#) :



By default the **description** is used to produce comments by the code generators.

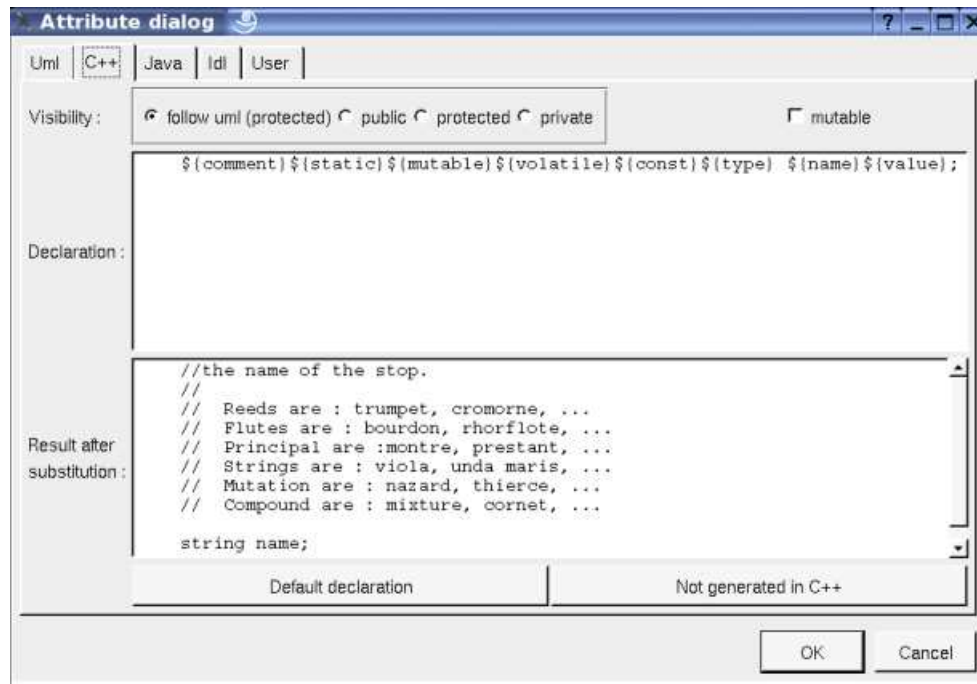
The **editor** button visible above and associated here to the description and initial value, allows to edit the description or initial value in an window, or to call an external editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note that this external editor have to create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel with BOUML which looks each second at the file contents to get the new definition until the dialog is closed (I do not like very much this polling but this works even QT isn't compiled with the thread support).

The **default** button visible above associated to the description allows to set the description with a default contain specified through the *generation settings*. Useful to generate comments compatible Java Doc or Doxygen for instance.

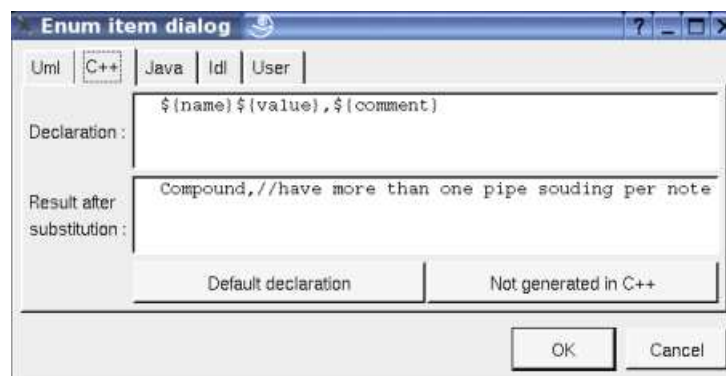
**Attribute dialog, tab C++**

This tab allows to give the C++ definition of the attribute, it is visible only when C++ is set through the menu *Languages*

For a standard attribute :



For an enumeration's item :



In BOUML the generated code is obtained by the substitution of macros in a text, only the **Declaration** part is editable, the other one help you to see what will be generated for C++ (supposing you do not modify the C++ code generator !).

When you push the button **default declaration**, the form specified through the [generation settings](#) is proposed, this last may be modified as you want, even to produce illegal source code.

When you do not want to have this attribute defined in C++, empties the declaration manually or using the button **not generated in C++**.

It is possible to follow the **visibility** as it is set in the UML Tab or to change it.

**\${static}** produce an empty string when the attribute is not a class attribute (see the UML tab), else produce *static*

**\${const}** produce an empty string when the attribute is not read only (see the UML tab), else produce *const*

**\${type}** is replaced by the type of the attribute (see the UML tab)

**\${stereotype}** is replaced by the [translation in C++](#) of the attribute's stereotype (see the UML tab)

**\${multiplicity}**  
is replaced by the multiplicity of the attribute (see the UML tab), must be used in case the multiplicity is a vector or array dimensioning ([ and ] are added when they are not present).

**\${name}** is replaced by the attribute's name (see the UML tab)

**\${value}**  
is replaced in the source file by the initial value of the attribute (see the UML tab) when this one is not an enumeration item

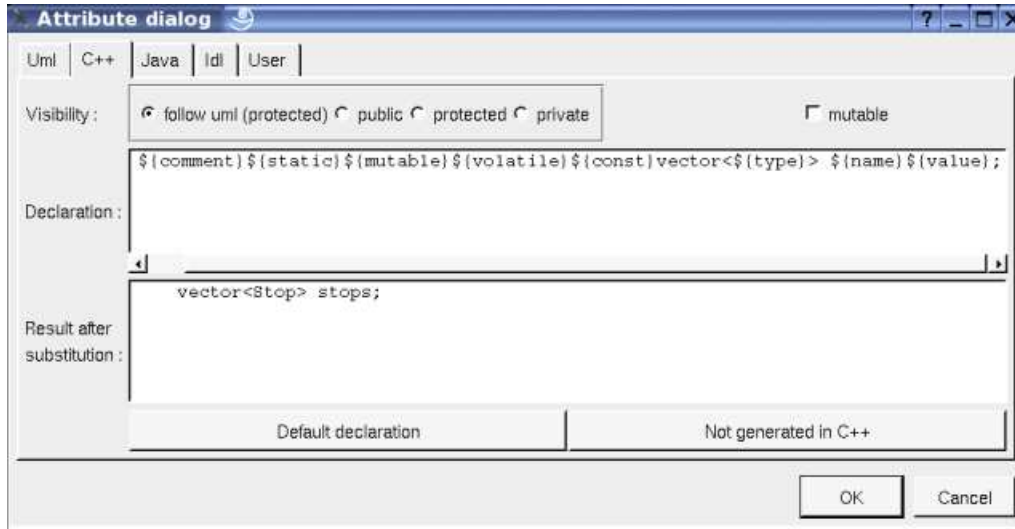
enumeration items the value is of course generated in the header file.

$\{h\_value\}$  is replaced in the header file by the initial value of the attribute (see the UML tab).  $\{h\_value\}$  must be used when you want to set the value of a *const* attribute.

$\{mutable\}$  produce an empty string when the check box *mutable* is not checked, else produce *mutable*

$\{volatile\}$  produce an empty string when the check box *volatile* is not checked, else produce *volatile*

The definition of the attribute *stops* may be :

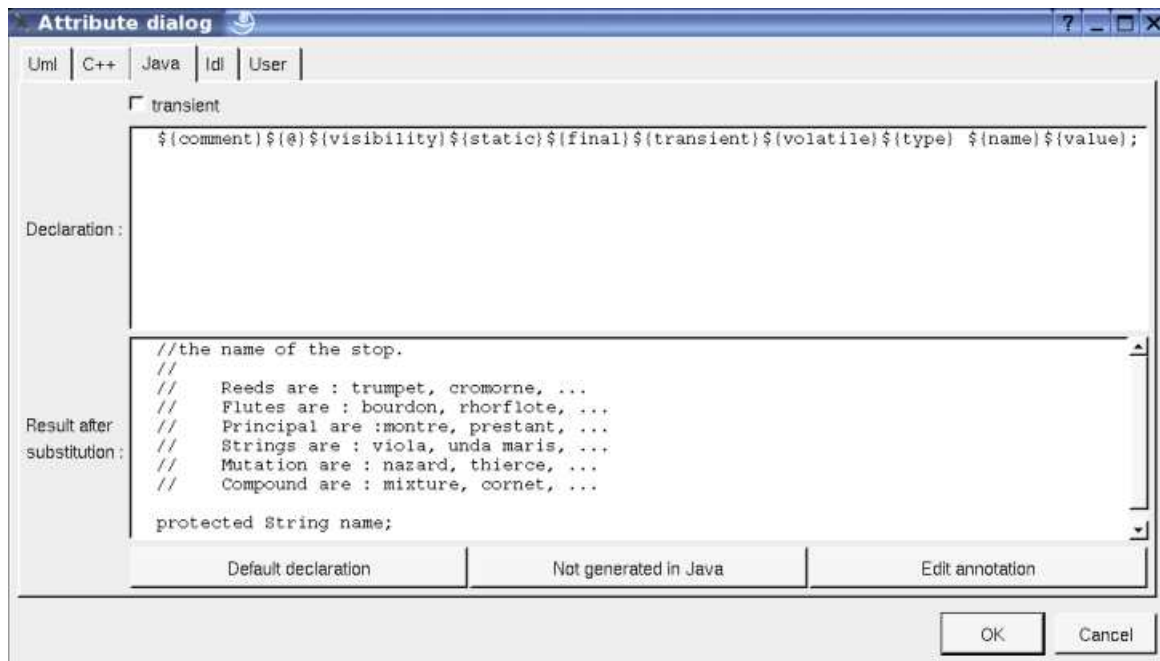


Refer to the [generation settings](#) and the [C++ Generator](#) for more details and explanation of the other keywords.

### Attribute dialog, tab Java

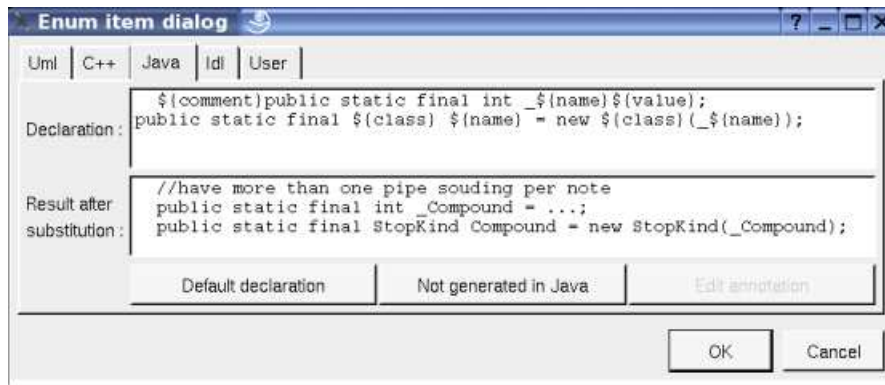
This tab allows to give the Java definition of the attribute, it is visible only when Java is set through the menu *Languages*

For a standard attribute (as you can see *string* is automatically translated into *String* thanks to the [generation settings](#)):



For an enumeration's item :





In BOUML the generated code is obtained by the substitution of macros in a text, only the **Declaration** part is editable, the other one help you to see what will be generated for Java (supposing you do not modify the Java code generator !).

When you push the button **default declaration**, the form specified through the [generation settings](#) is proposed, this last may be modified as you want, even to produce illegal source code.

When you do not want to have this attribute defined in Java, empties the declaration manually or using the button **not generated in Java**.

It is possible to follow the **visibility** as it is set in the UML Tab or to change it replacing `${visibility}` by the desired one.

`${static}` produce an empty string when the attribute is not an class attribute (see the UML tab), else produce *static*

`${final}` produce an empty string when the attribute is not read only (see the UML tab), else produce *final*

`${transient}` produce an empty string when check box **transient** is not checked, else produce *transient*

`${volatile}` produce an empty string when the check box **volatile** is not checked, else produce *volatile*

`${type}` is replaced by the class pointed by the attribute (see the UML tab)

`${stereotype}` is replaced by the [translation in Java](#) of the attribute's stereotype (see the UML tab)

`${multiplicity}`

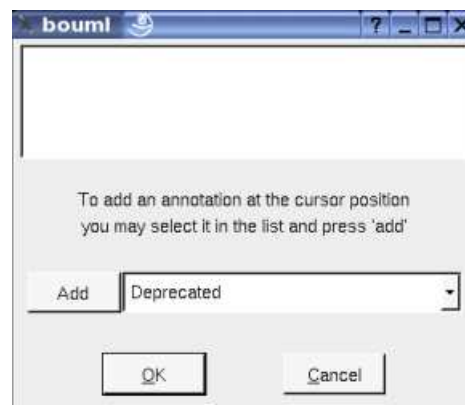
is replaced by the multiplicity of the attribute (see the UML tab), must be used in case the multiplicity is a vector or array dimensioning ( [ and ] are added when they are not present).

`${name}` is replaced by the attribute's name (see the UML tab)

`${value}`

is replaced by the initial value of the attribute (see the UML tab), may be used to specify the value of an enumeration's item for instance

`${@}` produces the annotations, when it is present in the definition the button *Edit annotation* is active, a specific dialog allows you to enter the annotations proposing the default annotations (*Deprecated ...*) and the ones defined through the classes stereotyped *@interface* :



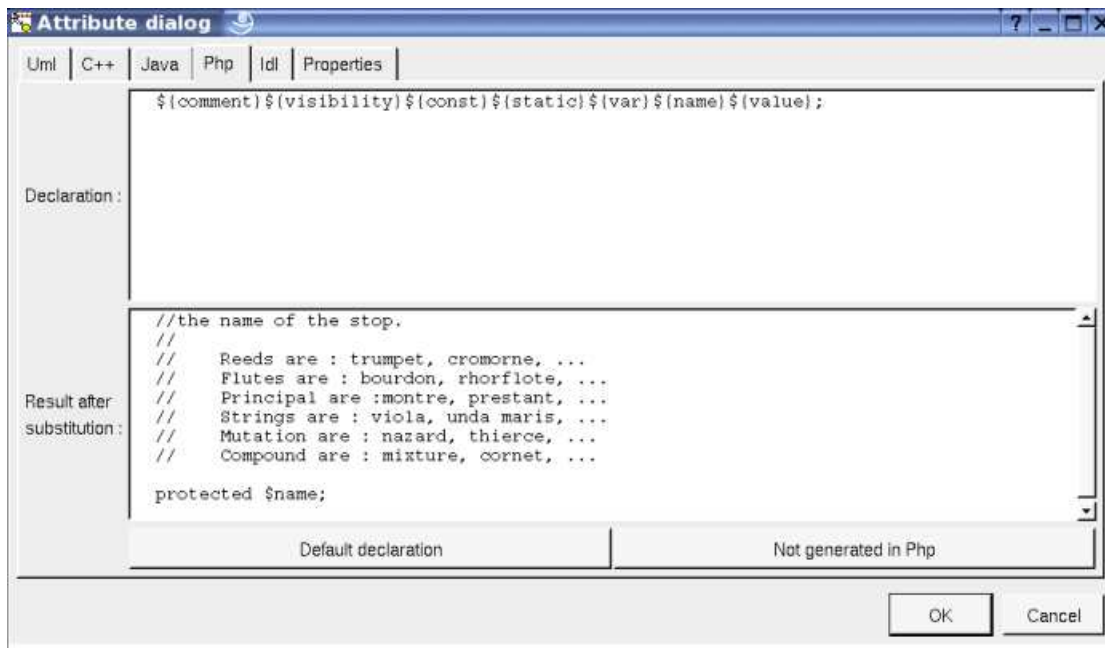
Refer to the [generation settings](#) and [Java generator](#) for more details and explanation of the other keywords.

### Attribute dialog, tab Php

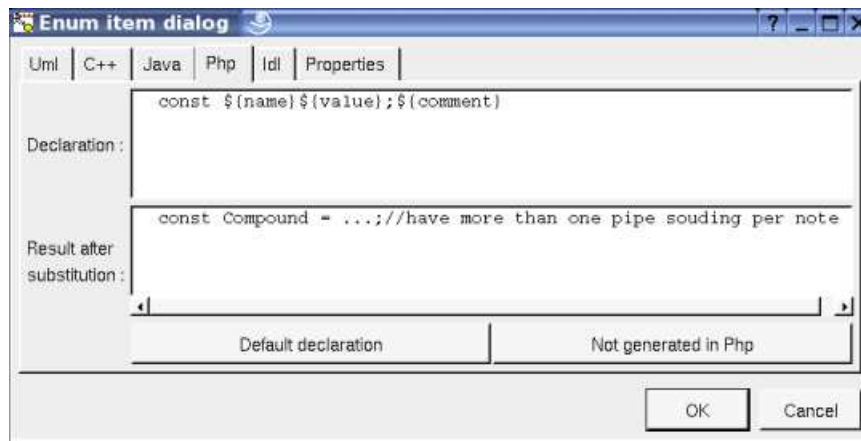
This tab allows to give the Php definition of the attribute, it is visible only when Php is set through the menu *Languages*



For a standard attribute :



For an enumeration's item (of course in the generated code ... is replaced by the rank of the item in the enum) :



In BOUML the generated code is obtained by the substitution of macros in a text, only the **Declaration** part is editable, the other one help you to see what will be generated for Php (supposing you do not modify the Php code generator !).

When you push the button **default declaration**, the form specified through the [generation settings](#) depending on the stereotype and multiplicity (and of course the language !) is proposed, this last may be modified as you want, even to produce illegal source code.

When you do not want to have this attribute defined in Php, empties the declaration manually or using the button **not generated in Php**.

It is possible to follow the **visibility** as it is set in the UML Tab or to change it replacing `${visibility}` by the desired one, to not generate the visibility set it to *package*

`${static}` produce an empty string when the attribute is not an class attribute (see the UML tab), else produce *static*

`${const}` produce an empty string when the attribute is not read only (see the UML tab), else produce *const*

`${var}` produce *var* when the attribute is not read only nor static(see the UML tab) and the visibility is not *package*, else produce an empty string

`${name}` is replaced by the attribute's name (see the UML tab)

`${value}` is replaced by the initial value of the attribute (see the UML tab)

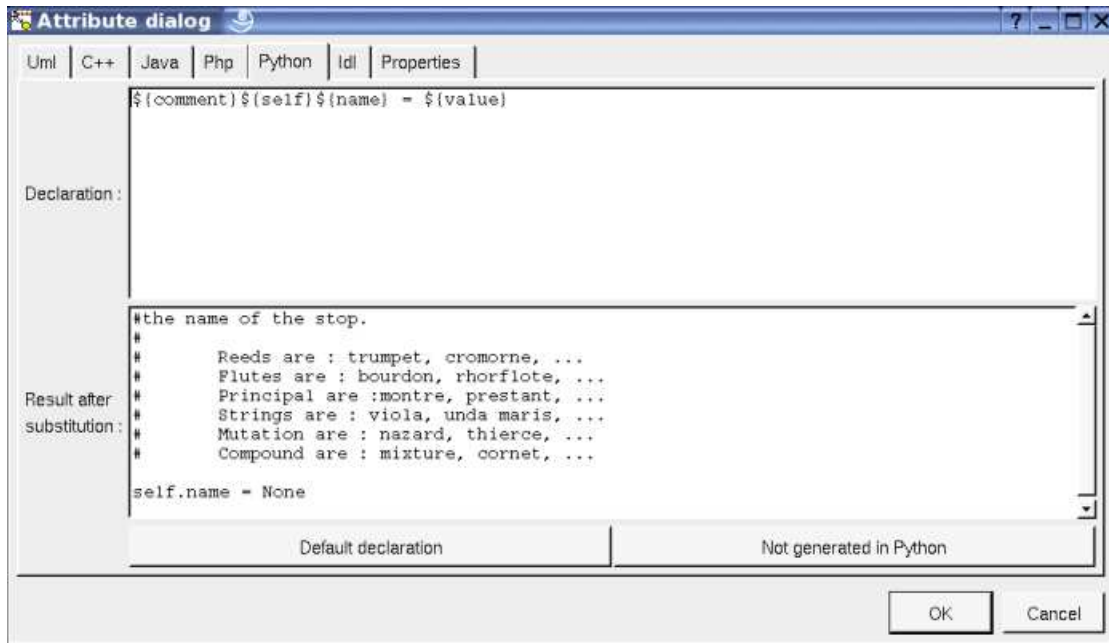
The forms `@{property}` are replaced by the value of the corresponding user property if it is defined for the class or at an upper level, else it is not substituted.

Refer to the [generation settings](#) and [Php generator](#) for more details and explanation of the other keywords.

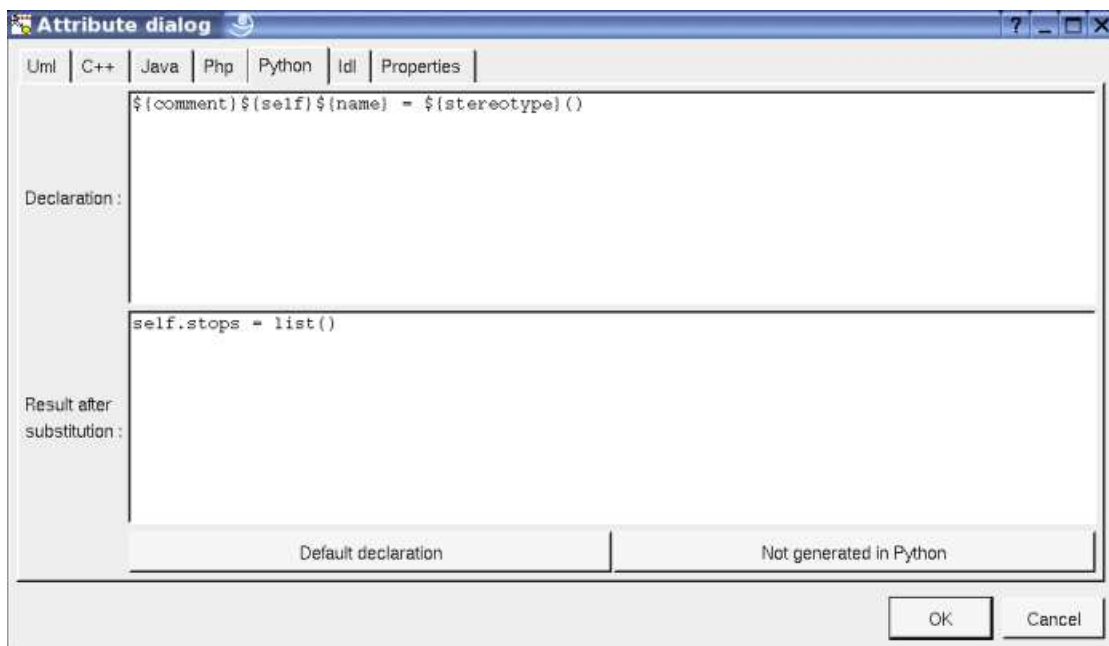
## Attribute dialog, tab Python

This tab allows to give the Python definition of the attribute, it is visible only when Python is set through the menu *Languages*

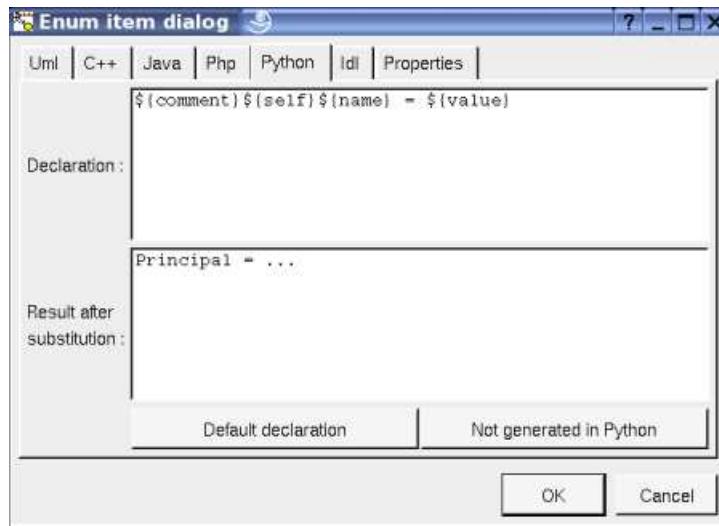
For a standard attribute without multiplicity or with the multiplicity 1 :



For a standard attribute with a multiplicity not equal to 1 :



For an enumeration's item (of course in the generated code ... is replaced by the rank of the item in the enum) :



In BOUML the generated code is obtained by the substitution of macros in a text, only the **Declaration** part is editable, the other one help you to see what will be generated for Python (supposing you do not modify the Python code generator !).

When you push the button **default declaration**, the form specified through the [generation settings](#) depending on the stereotype and multiplicity (and of course the language !) is proposed, this last may be modified as you want, even to produce illegal source code.

When you do not want to have this attribute defined in Python, empties the declaration manually or using the button **not generated in Python**.

The instance attributes are automatically produced at the beginning of the body of the operation `__init__` (this operation is produced for each class even when you don't define it). The static attributes are produced inside the class definition.

`$(comment)`

produces a comment from the UML description, adding a # at the beginning of each line, following the current indent. A new line is produced if needed at the end to not have something else generated on the same line after the comment.

`$(description)` is like for `$(comment)` but without adding #.

`$(self)` inside the operation `__init__` this produces the name of the first parameter followed by a dot, else nothing.

`$(type)` produces the type of the attribute, probably used inside a comment

`$(multiplicity)` produces the multiplicity of the attribute, probably used inside a comment

`$(stereotype)` is replaced by the [translation in Python](#) of the attribute's stereotype (see the UML tab)

`$(name)` is replaced by the attribute's name (see the UML tab)

`$(value)` is replaced by the initial value of the attribute (see the UML tab) if it is not empty, else *None*

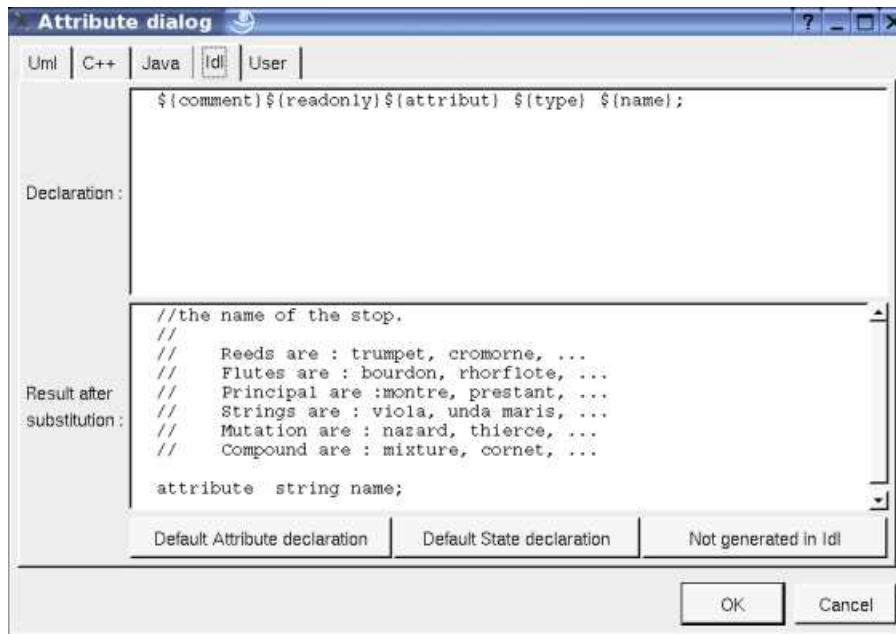
The `@{property}` forms are replaced by the value of the corresponding user property if it is defined for the class or at an upper level, else it is not substituted.

Refer to the [generation settings](#) and [Python generator](#) for more details and explanation of the other keywords.

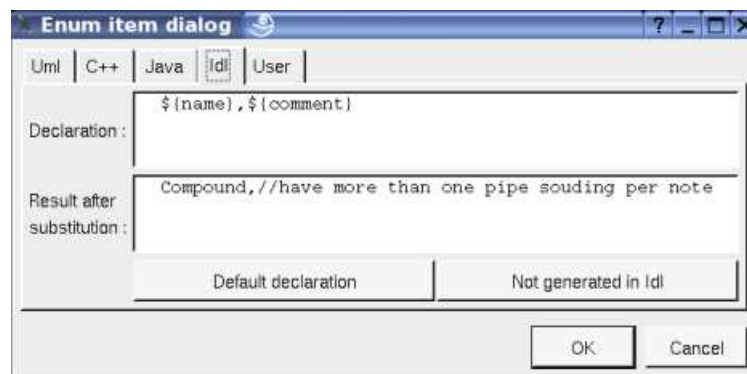
### Attribute dialog, tab Idl

This tab allows to give the Idl definition of the attribute, it is visible only when Idl is set through the menu *Languages*

For a standard attribute (here of a *valuetype*) :



For an enumeration's item :



In BOUML the generated code is obtained by the substitution of macros in a text, only the **Declaration** part is editable, the other one help you to see what will be generated for Idl (supposing you do not modify the Idl code generator !).

When you push the button **default declaration**, the form specified through the [generation settings](#) is proposed, this last may be modified as you want, even to produce illegal source code.

When you do not want to have this attribute defined in Idl, empties the declaration manually or using the button **not generated in Idl**

It is possible to follow the **visibility** as it is set in the UML Tab or to change it replacing `${visibility}` by the desired one.

`${type}` is replaced by the class pointed by the attribute (see the UML tab)

`${name}` is replaced by the attribute's name (see the UML tab)

`${stereotype}` is replaced by the [translation in Idl](#) of the attribute's stereotype (see the UML tab)

`${multiplicity}` is replaced by the multiplicity of the attribute (see the UML tab)

Refer to the [generation settings](#) and [Idl generator](#) for more details and explanation of the other keywords.

## **Menu : duplicate**

The menu entry *duplicate* clone the attribute.

## **Menu : add ...**

These entries allow to produce get and set operations on the attribute, contrarily to the similar operations made by hand, these two ones are linked to the associated attribute to be updated, or deleted when the attributes is deleted. The default name of the operator and their characteristics are set through the [generation settings](#).

By default, the get and set operations generated for the *Stop*'s attribute *name* are :

**C++**

```
inline const string Stop::get_name() const {
    return name;
}

void Stop::set_name(string new_value) {
    name = new_value;
}
```

**Java**

```
public final String get_name() {
    return value;
}

public void set_name(String new_value) {
    name = new_value;
}
```

**Php**

```
final public function getName()
{
    return $this->name;
}

public function setName($value)
{
    $this->name = $value;
}
```

**Idl**

```
string get_name();

void set_name(string new_value);
```

**Menu : delete**

The menu entry *delete* is only present when the attribute is not *read-only*.

Delete the attribute, associated get and set operations when they exist, and all the representation of them in the opened diagrams. After that it is possible to undelete them (from the browser) until you close the project : obviously the deleted items are not saved !

**Menu : mark**

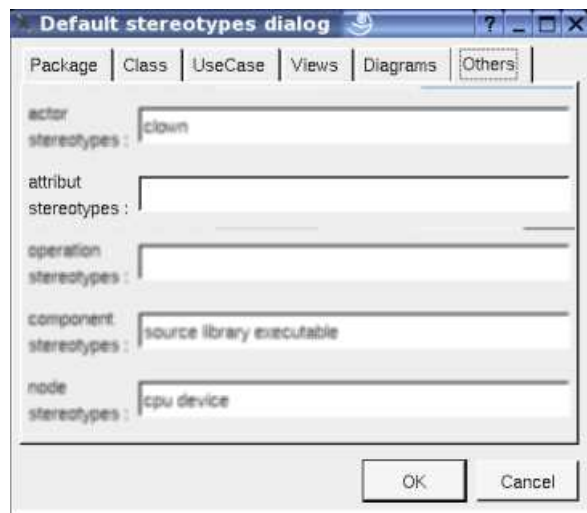
See [mark](#)

**Menu : tool**

Appears only when at least one [plug-out](#) is associated to the attributes. To apply a *plug-out* on the attribute.

## Default stereotypes

The [dialog](#) allowing to set the default stereotypes has a tab associated to the attributes :

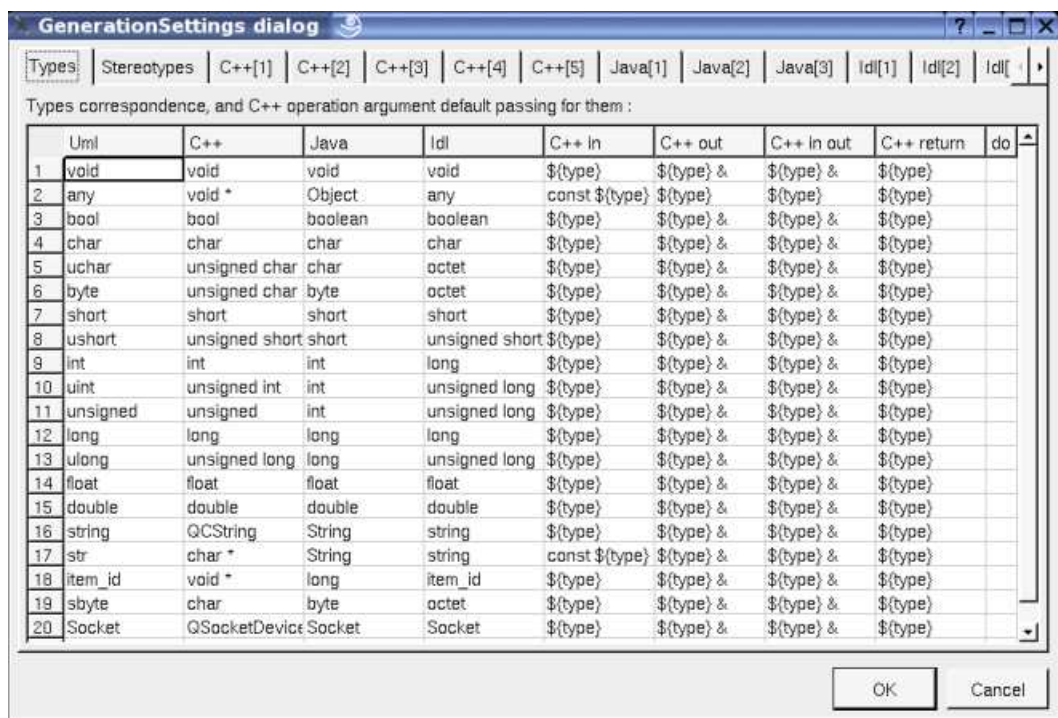


The stereotype of an attribute does not have a special meaning for BOUML.

## Generation settings

This very important dialog allows to specify many default definitions concerning the attributes, more details will be given in [C++ generator](#), [Java generator](#) and [Idl generator](#).

The first tab allows to specify how the types not supported by an UML class are translated in each language (except Php and Python) :



The third C++ tab allows to set the default definition of an attribute in C++, only visible when C++ is set through the menu *Languages* :



**GenerationSettings dialog**

Types | Stereotypes | C++[1] | C++[2] | **C++[3]** | C++[4] | C++[5] | Java[1] | Java[2] | Java[3] | Java[4] | Idl[1] | Idl[2] | Idl[3] | Idl[4] | Idl[5]

Multiplicity	unspecified,	<code>\$(comment)\$(static)\$(mutable)\$(volatile)\$(const)\$(type) \$(name)\$(value);</code>
Attribute	1 or 1	<code>\$(comment)\$(static)\$(mutable)\$(volatile)\$(const)\$(stereotype)&lt;\$(type)&gt; \$(name)\$(value);</code>
default	* or a.b	<code>\$(comment)\$(static)\$(mutable)\$(volatile)\$(const)\$(type) \$(name)\$(multiplicity)\$(value);</code>
declaration	X (means [X]) or [...]-[...]	<code>\$(comment)\$(static)\$(mutable)\$(volatile)\$(const)\$(type) \$(name)\$(multiplicity)\$(value);</code>

---

Association — and aggregation —

Multiplicity	unspecified,	<code>\$(comment)\$(static)\$(mutable)\$(volatile)\$(const)\$(type) * \$(name)\$(value);</code>
Association	1 or 0..1	<code>\$(comment)\$(static)\$(mutable)\$(volatile)\$(const)\$(stereotype)&lt;\$(type)&gt; * \$(name)\$(value);</code>
and	* or a.b	<code>\$(comment)\$(static)\$(mutable)\$(volatile)\$(const)\$(type) * \$(name)\$(multiplicity)\$(value);</code>
aggregation	* or a.b	<code>\$(comment)\$(static)\$(mutable)\$(volatile)\$(const)\$(type) * \$(name)\$(multiplicity)\$(value);</code>
default		
declaration	X (means [X]) or [...]-[...]	

---

Aggregation by value —

Multiplicity	unspecified	<code>\$(comment)\$(static)\$(mutable)\$(volatile)\$(const)\$(type) \$(name)\$(value);</code>
Aggregation	or 1	<code>\$(comment)\$(static)\$(mutable)\$(volatile)\$(const)\$(stereotype)&lt;\$(type)&gt; \$(name)\$(value);</code>
by value	* or a.b	<code>\$(comment)\$(static)\$(mutable)\$(volatile)\$(const)\$(type) \$(name)\$(multiplicity)\$(value);</code>
default		
declaration	X (means [X]) or [...]-[...]	

OK Cancel

The fourth C++ tab allows to set the default definition of an enumeration's item in C++ and to specify the get and set operation specificities in C++, only visible when C++ is set through the menu *Languages* :

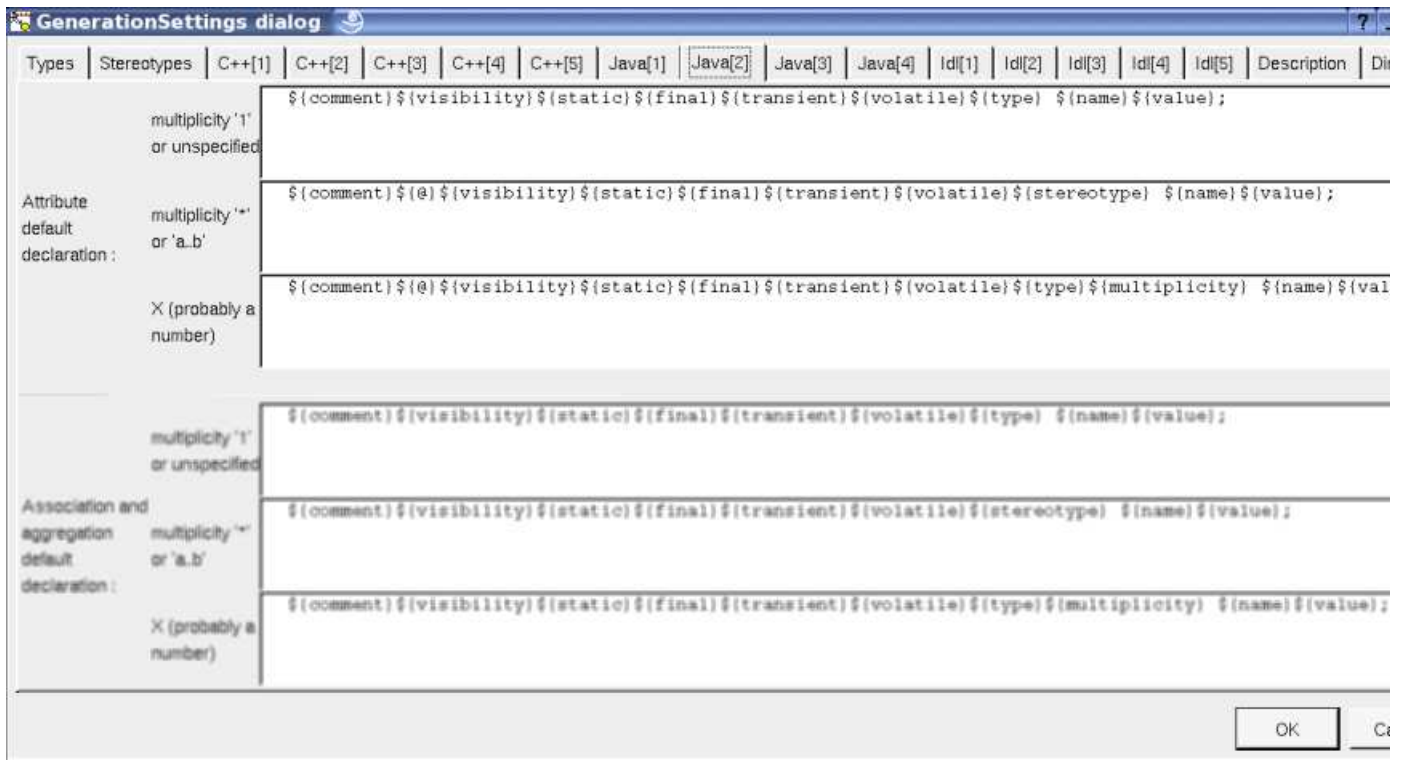
**GenerationSettings dialog**

Types | Stereotypes | C++[1] | C++[2] | C++[3] | **C++[4]** | C++[5] | Java[1] | Java[2] | Java[3] | Java[4] | Idl[1] | Idl[2] | Idl[3] | Idl[4] | Idl[5]

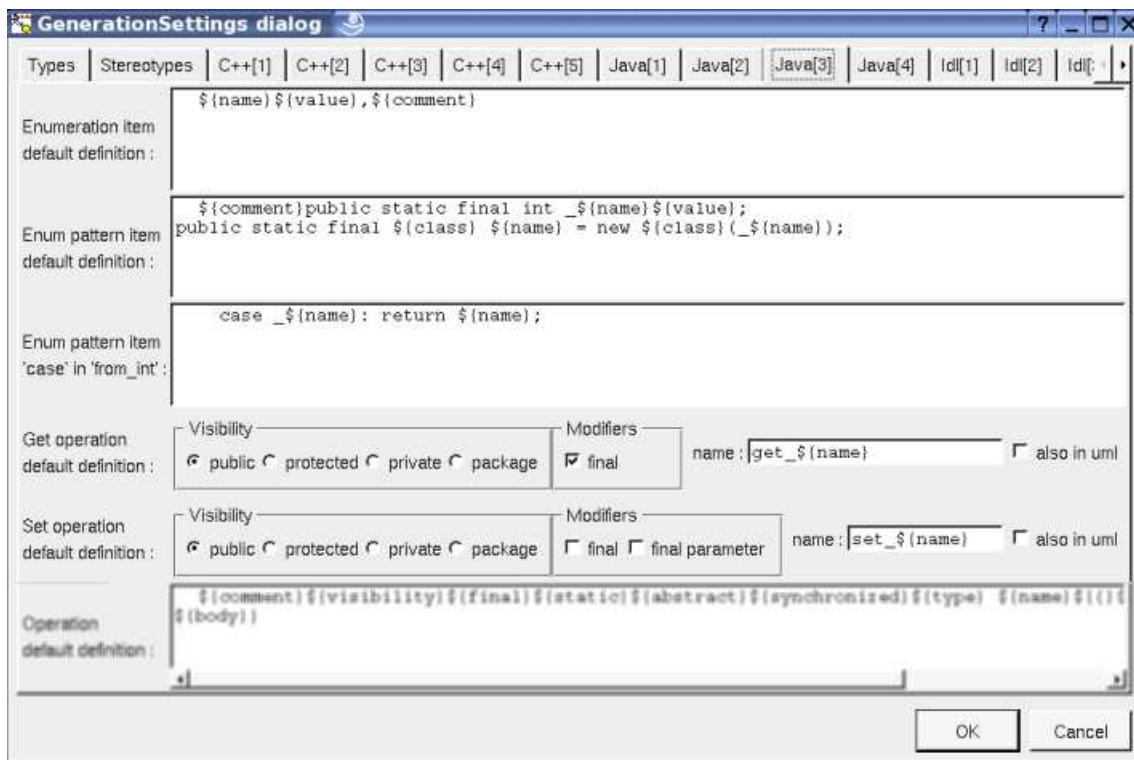
Get operation default definition :	Visibility : <input checked="" type="radio"/> public <input type="radio"/> protected <input type="radio"/> private Modifiers : <input checked="" type="checkbox"/> inline <input checked="" type="checkbox"/> const value <input checked="" type="checkbox"/> const name : <code>get_\$(name)</code> <input type="checkbox"/> also in uml
Set operation default definition :	Visibility : <input checked="" type="radio"/> public <input type="radio"/> protected <input type="radio"/> private Modifiers : <input type="checkbox"/> inline <input type="checkbox"/> const param <input type="checkbox"/> by ref name : <code>set_\$(name)</code> <input type="checkbox"/> also in uml
Enumeration item default definition :	<code>\$(name)\$(value), \$(comment)</code>
Default type forms for the enums :	input parameter : <code>\$(type)</code> output parameter : <code>\$(type) &amp;</code> input output parameter : <code>\$(type) &amp;</code> operation return : <code>\$(type)</code>
Default type forms for the types not given in the first tab :	input parameter : <code>const \$(type) &amp;</code> output parameter : <code>\$(type) &amp;</code> input output parameter : <code>\$(type) &amp;</code> operation return : <code>\$(type)</code>
Operation default declaration :	<code>\$(comment)\$(friend)\$(static)\$(virtual)\$(type) \$(name)\$(())\$(const)\$(volatile)\$(throw)\$(abstr</code>
Operation default definition :	<code>\$(inline)\$(type) \$(class)::\$(name)\$(())\$(const)\$(volatile)\$(throw)\$(static){</code> <code>\$(body)}</code> <input type="checkbox"/> throw()

OK Cancel

The second Java tab allows to set the default definition of an attribute in Java, only visible when Java is set through the menu *Languages* :

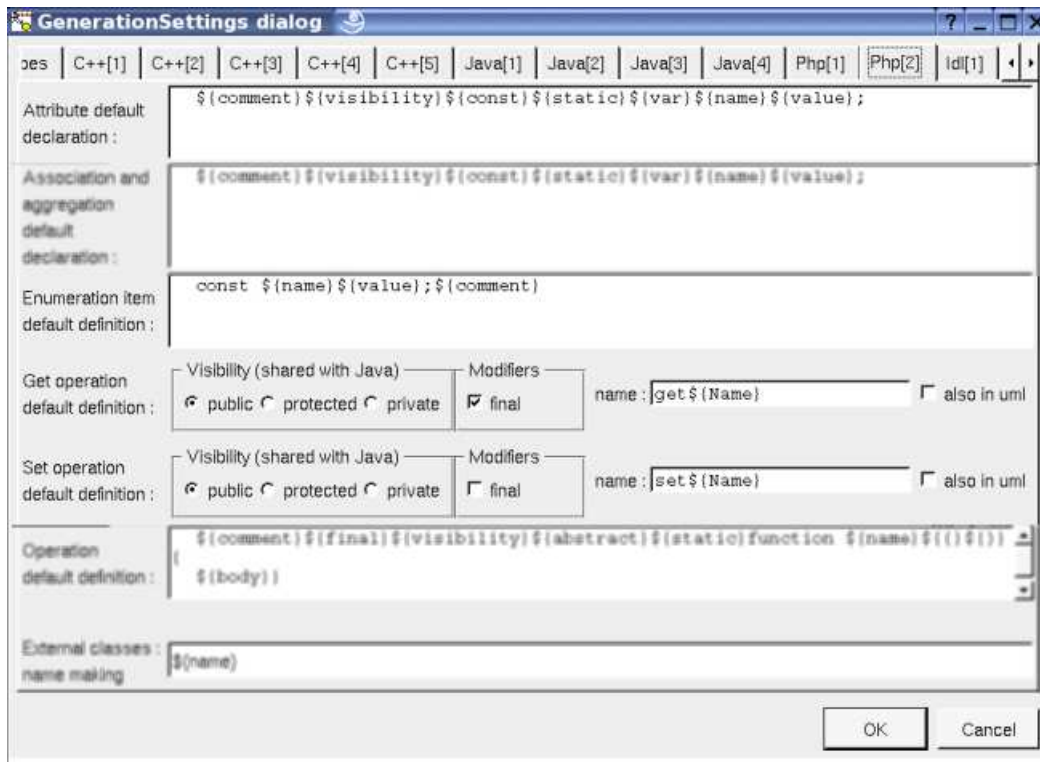


The third Java tab allows to set the default definition of an enumeration's item in Java, to set the get and set operations specificities in Java, only visible when Java is set through the menu *Languages* :

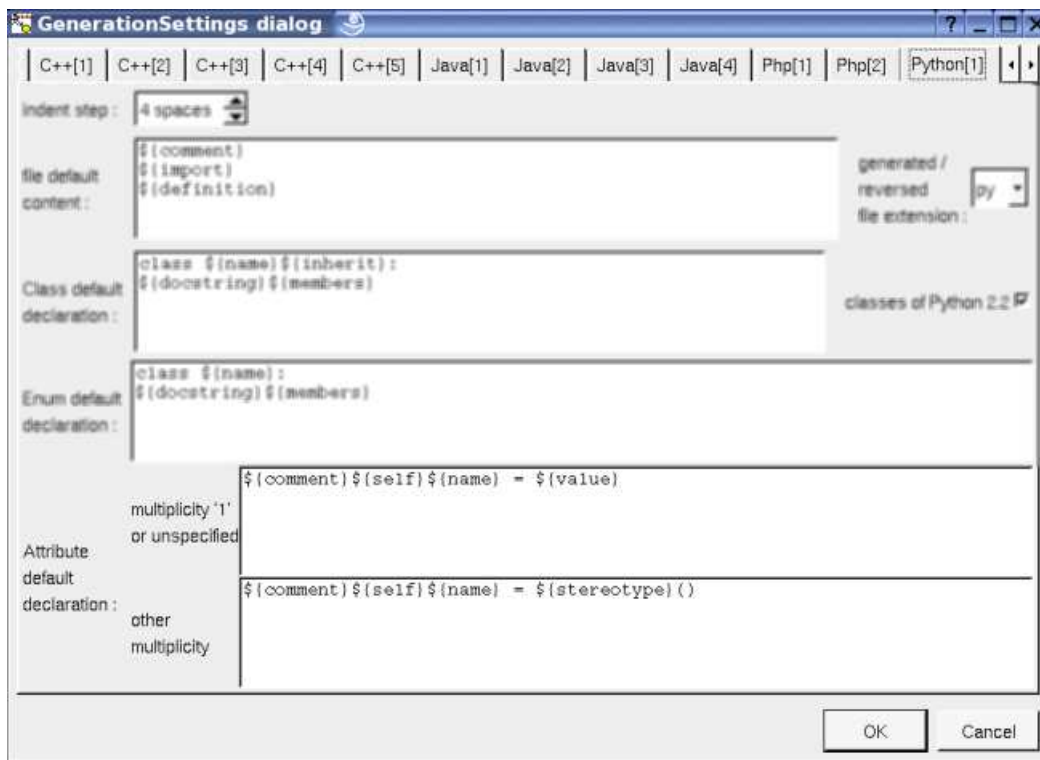


The second Php tab allows to set the default definition of an attribute, to set the default definition of an enumeration's item, and to set the get and set operations specificities in Php, only visible when Php is set through the menu *Languages* :

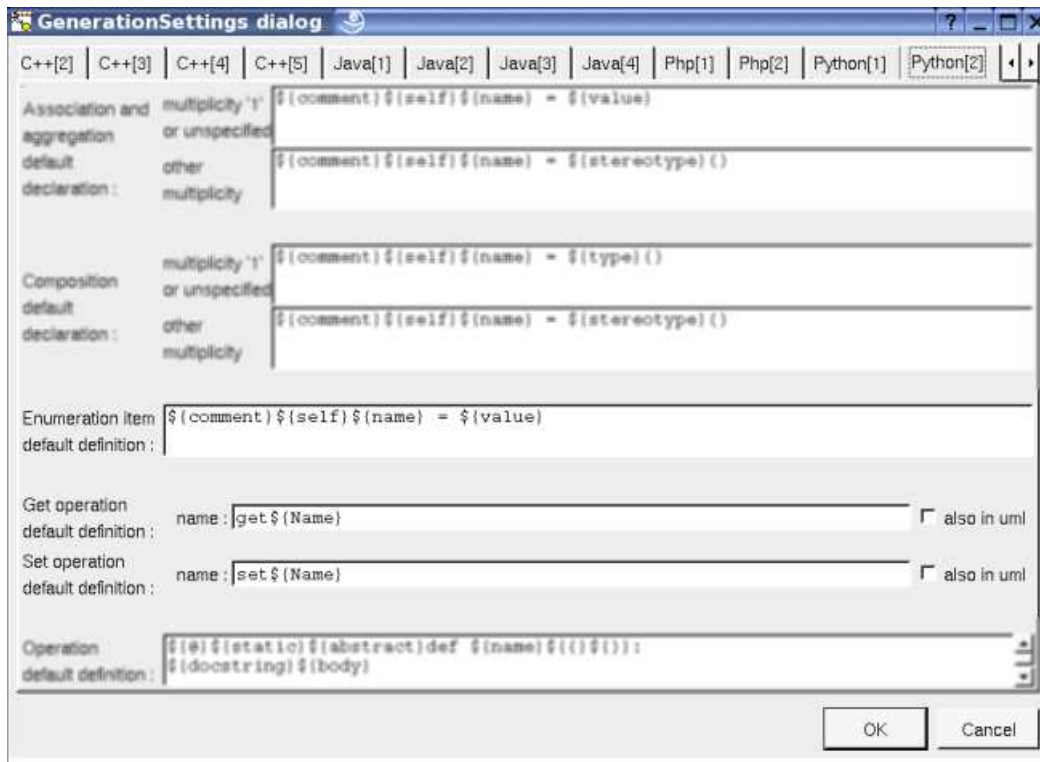




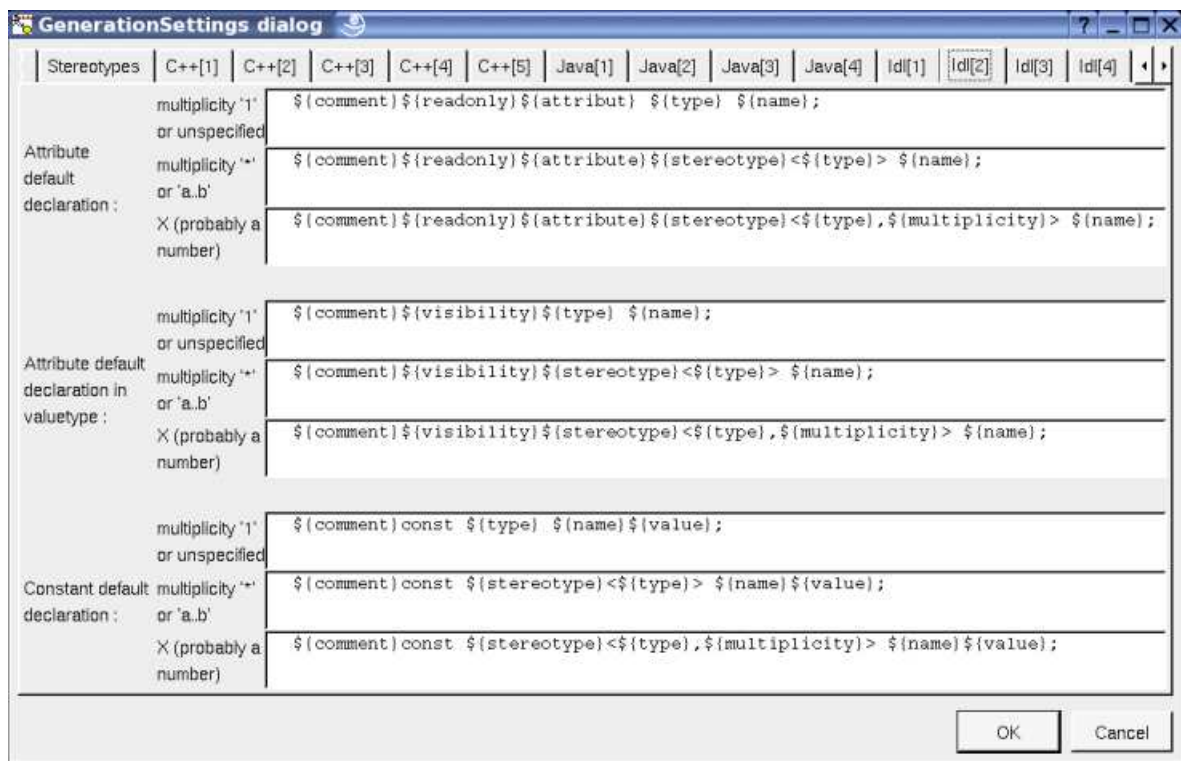
The first Python tab allows to set the default attribute definition for Python depending on the multiplicity, only visible when Python is set through the menu *Languages* :



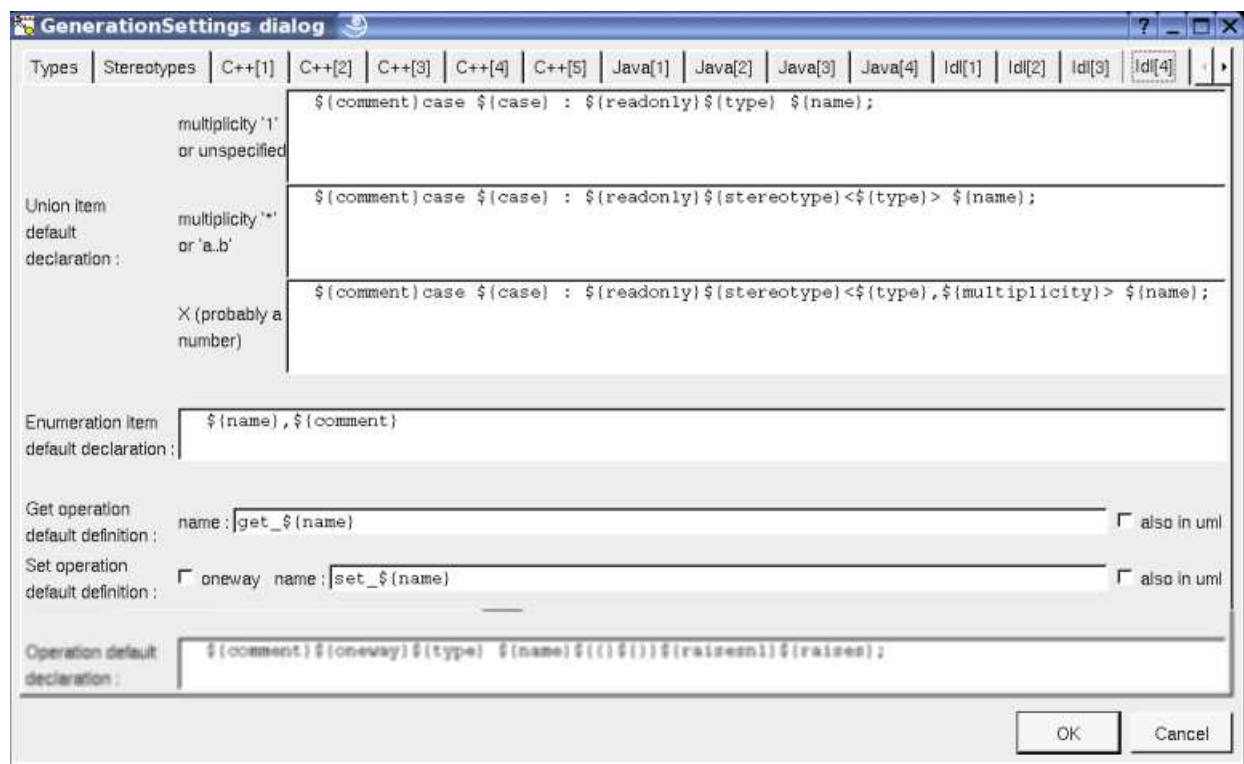
The second Python tab allows to set the default enum item definition for Python and the get and set specificities for Idl, only visible when Python is set through the menu *Languages* :



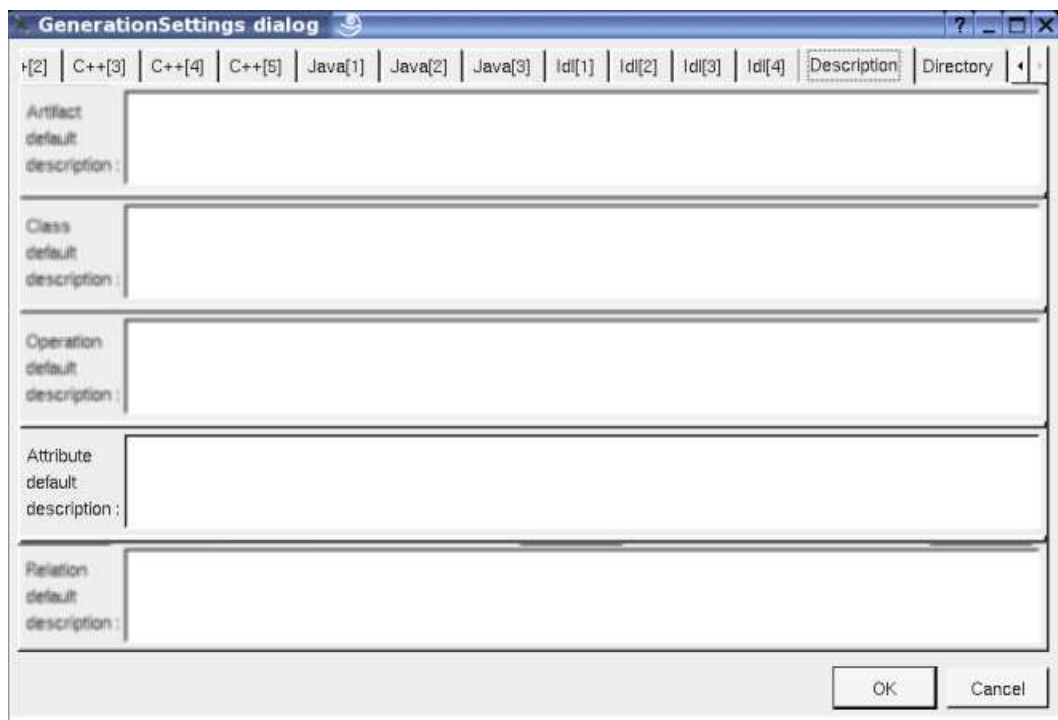
The second Idl tab allows to set the default attribute definition for Idl, only visible when Idl is set through the menu *Languages* :



The third Idl tab allows to set the default definition of an attribute in a union, the enumeration item default definition and the get and set specificities for Idl, only visible when Idl is set through the menu *Languages* :

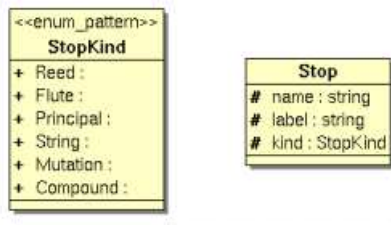


The tab *Description* allows to set a default description :



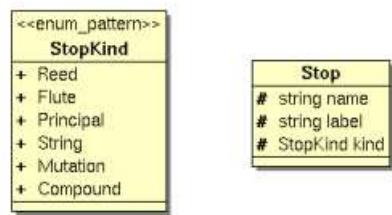
## Drawing

The attributes are only drawn with their class in a class diagram (here the drawing language is UML, the full member definitions and the visibility are shown) :

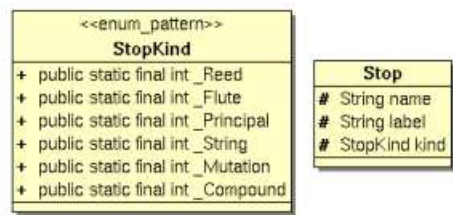


The way an attribute is written or not in its class picture depend on the [class drawing settings](#), [drawing language](#), [hide attributes](#), [show member visibility](#), [show attribute multiplicity](#), [show attribute initialization](#), [member max width](#) and [show full members definition](#), and also on the [class menu](#) entry [individual attribute visibility](#).

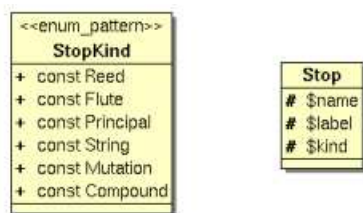
Changing the [drawing language](#) from UML (the default) to C++ or IDL :



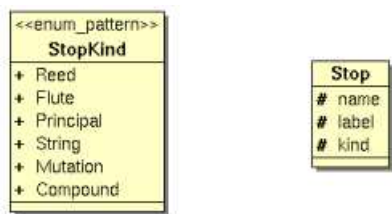
Changing the [drawing language](#) to Java:



Changing the [drawing language](#) to Php :



Changing the [drawing language](#) to Python :



Previous : [relation](#)

Next : [operation](#)

# Operation

BOUML only manage class's operations and *friend* operations, not standard C or Php and Python functions (even it is possible to define a C, Php and Python function through an [extra member](#) or in an artifact definition etc ...). An operation may have a stereotype, but it does not have a special meaning for BOUML.

Contrarily to several UML toolbox, the body of the operations is fully managed by BOUML and saved among the project data (see [project files](#)). However you can also enter the bodies out of BOUML thanks to the option *Preserve operations's bodies* of the *Language* (see [top level menu](#)).

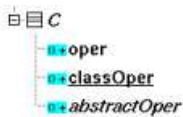
An operation is created through the class menu called from the [browser](#) or a class [picture](#) in a class diagram :

UmlOperation	UmlOperation
add attribut	upper
add operation	lower
add inherited operation	edit drawing settings
add embedded class	individual attribut visibility
add extra member	individual operation visibility
edit	edit class
duplicate	add attribut
set it embedded in UmlNode	edit attribut
delete	add operation
select associated component	add inherited operation
referenced by	edit operation
mark	select in browser
generate	select linked items
tool	set associated diagram
	remove from view
	delete from model
	generate
	tool

The icon in the browser indicate if the operation is public (+), protected (#) , private (-) or package (~). The *protected* and *package* visibilities are translated in *private* in Idl. The *package* visibility is translated in *private* in C++, and produce nothing in Php allowing you to generate Php 4 code. Note that BOUML doesn't check the validity of the (re)refinition of an inherited operation (for instance the visibility is not checked, the operation may be *final* in Java etc ...), but some operations are not part of the proposed operations list : the constructors, destructors, and the operations already defined in the current class for the UML point of view (the 'UML' definition of an operation is the one written under the browser window when the operation is selected in the browser).

The operation named \_\_init\_\_ has a special management in Python, it is used to place the initializations or the instance attributes and relations. The python generator add the definition of this operation when it doesn't exist.

A class operation is underlined, an abstract one is written in *italic*. It is possible to move an operation from a class into an other, except for the *get* and *set* operations associated to a [relation](#) or an [attribute](#).



## Menus

The operation menu appearing with a right mouse click on its representation in the browser is something like these, supposing it is not *read-only* nor deleted :

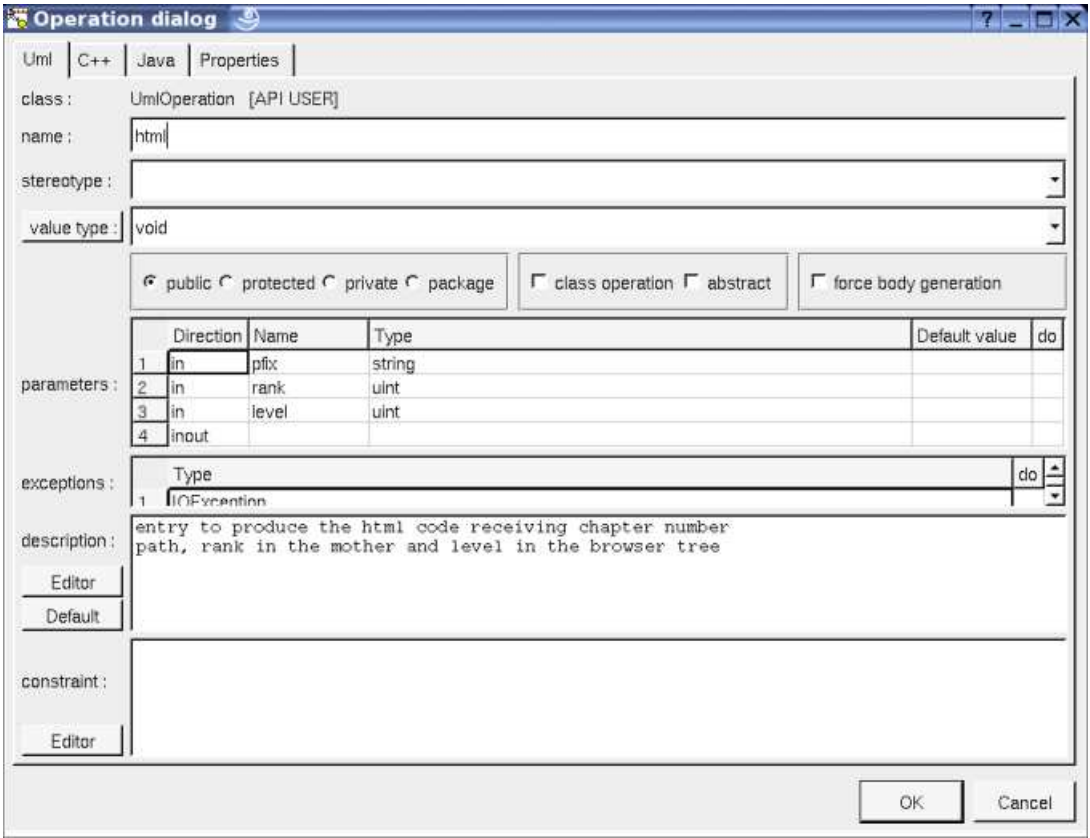


**Menu : edit**

*edit*  
allows to show/modify the operation properties including the body. In case the operation is read-only, the fields of the dialog read-only.

**Operation dialog, tab Uml**

The tab *Uml* is a global tab, independent of the language :



The *name* may contains spaces to define C++ operator for instance, and a class may have several operations having the same name. A constructor have the name of its class.

The proposed type for the *value type* etc ... are the *non class* types defined in the first tab of the [generation settings](#), more all the classes defined in the project (with their localization to distinguish synonymous classes). You are not limited to the proposed list, and any form (even invalid for the target language(s)) may be given. The button *value type*: shows a menu proposing :

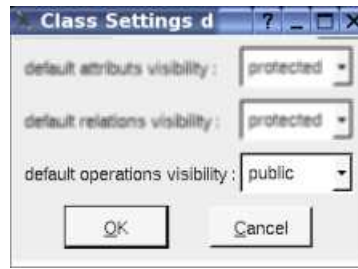
- if the current value type is a class : to select this class in the browser
- if the operation is not read-only and if a class is selected in the browser : to set the value type to be this class
- if the view containing the class containing the operation is not read-only : to create a new class and to set the return type to it

Nevertheless I **strongly** recommend you to not add type modifier (like \*, & etc ...) with a class name in the UML tab because BOUML differentiate a reference to a class of the model and any other forms, see [type considerations](#).

The proposed *stereotypes* are the default ones specified through the [Default stereotypes dialog](#) more the current one (empty here). It is possible to choose into the list or to give a new one, or to empty it. The stereotype of an operation does not have special meaning for BOUML.

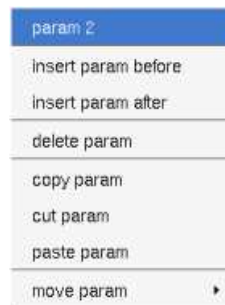


The *visibility* may be chosen with the *radio buttons*, the operations have a default visibility set through the [class settings](#) :



**Force body generation** allows to generate the body of the operation even if *preserve operations's body* is set in the menu [Languages](#). This option is used by the [state machine generator](#) to produce the body of the generated operations in all the cases.

The *parameters* have obviously a *direction* (but I known a commercial UML tool which does not manage it !), a *name*, a *type* (following the considerations about the return value type) and a *default value* (unfortunately this one is not yet managed by the code generators !). The last column (*do*) of the parameters table allows to insert/delete/move a parameter :



The *exceptions* produced by the operation may be specified (even for C++), these types follow the considerations about the return value type. The last column (*do*) of the exception table allows to insert/delete/move an exception.

By default the *description* is used to produce comments by the code generators.

The *editor* button visible above and associated here to the description, allows to edit the description in an other window, or to call an external editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note that this external editor have to create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel with BOUML which looks each second at the file contents to get the new definition until the dialog is closed (I do not like very much polling but this works even QT isn't compiled with the thread support).

The *default* button visible above associated to the description allows to set the description with a default content specified through the *generation settings*. Useful to generate comments compatible Java Doc or Doxygen for instance.

### Operation dialog, tab C++

This tab allows to give the C++ declaration and definition of the operation, it is visible only if C++ is set through the *Language* menu

Operation dialog

Uml | C++ | Java | Properties

☒ follow uml (public) ☐ public ☐ protected ☐ private

☐ const ☐ volatile ☐ friend ☒ virtual ☐ inline

Declaration:

`${comment}${static}${virtual}${type} ${name}${(}${t0} ${p0}, ${t1} ${p1}, ${t2} ${p2}${})}${const}${volatile}${abstract};`

Result after substitution:

`//entry to produce the html code receiving chapter number  
//path, rank in the mother and level in the browser tree  
virtual void html(QCString pfix, unsigned int rank, unsigned int level);`

Definition:

`$(inline)$(type) $(class)::$(name)$(())$(t0), $(t1), $(t2)$(())$(const)$(volatile)$(staticn){  
$(body)}`

Result after substitution:

`void UmlOperation::html(QCString, unsigned int, unsigned int) {  
$(body)}`

☒ contextual body indent

Edit body

Default declaration From definition Not generated in C++ Edit parameters

Default definition From declaration Not generated in C++ Edit parameters

OK Cancel

In BOUML the generated code is obtained by the substitution of macros in a text, only the **Declaration** and **Definition** part are editable, the other one help you to see what will be generated for C++ (supposing you do not modify the C++ code generator !). You can see above that the definition doesn't contains the name of the parameters whose aren't used to avoid C++ compiler warnings.

When you push the button **Default declaration** or **Default definition**, the form specified through the [generation settings](#) is proposed adding all the operation parameters given in the UML tab without any modifier. These forms may be modified as you want, even produce illegal source code, but more probably to add type modifiers etc ...

The buttons **From definition** and **From declaration** allow to produce the declaration/definition from the definition/declaration getting the specified list of parameters and return type. For instance if I modify the operation's declaration like this (the first parameter is modified and the second and third removed) :

Operation dialog

Uml | C++ | Java | Properties

☒ follow uml (public) ☐ public ☐ protected ☐ private

☐ const ☐ volatile ☐ friend ☒ virtual ☐ inline

Declaration:

`${comment}${static}${virtual}${type} ${name}${(}${t0} & ${p0}${})}${const}${volatile}${abstract};`

Result after substitution:

`//entry to produce the html code receiving chapter number  
//path, rank in the mother and level in the browser tree  
virtual void html(QCString & pfix);`

Definition:

`$(inline)$(type) $(class)::$(name)$(())$(t0), $(t1), $(t2)$(())$(const)$(volatile)$(staticn){  
$(body)}`

Result after substitution:

`void UmlOperation::html(QCString, unsigned int, unsigned int) {  
$(body)}`

☒ contextual body indent

Edit body

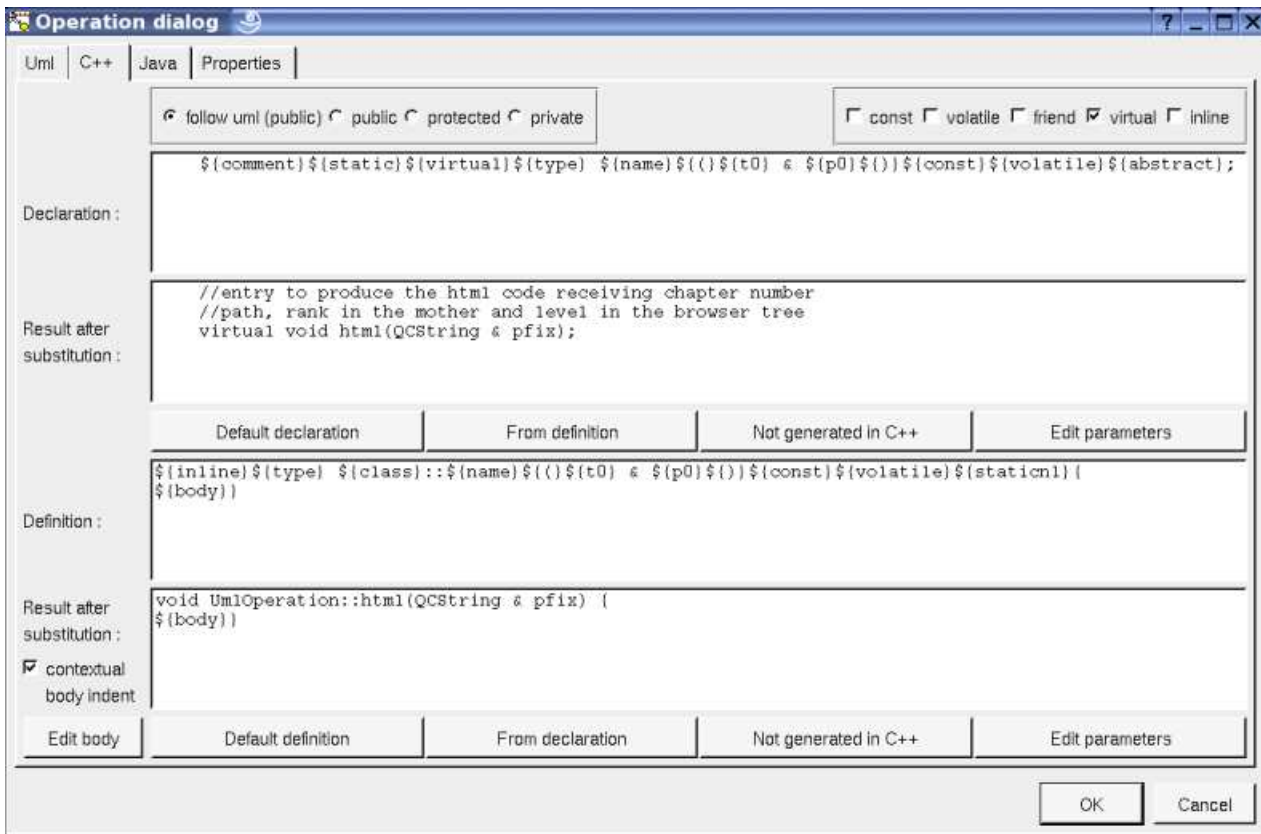
Default declaration From definition Not generated in C++ Edit parameters

Default definition From declaration Not generated in C++ Edit parameters

OK Cancel

and I press the **From declaration** button, the modifications are reported in the operation's definition :





When you do not want to have this operation defined in C++, empties the declaration/definition manually or using the button *not generated in C++*.

It is possible to follow the **visibility** as it is set in the UML Tab or to change it.

*static* produce an empty string when the operation is not an class operation (see the UML tab), else produce *static*

*const* produce an empty string when the check box **const** is not checked, else produce *const*

*volatile* produce an empty string when the check box **volatile** is not checked, else produce *volatile*

*friend* produce an empty string when the check box **friend** is not checked, else produce *friend*

*virtual* produce an empty string when the check box **virtual** is not checked, else produce *virtual*

*inline* produce an empty string when the check box **inline** is not checked, else produce *inline*

*type* is replaced by the value type (see the UML tab)

*abstract* produce the string = 0 when the operation is abstract (see the UML tab), else an empty string.

*class* is replaced by the name of the class containing the operation.

*staticnl* produce a line break when the operation is static, else an empty string. In case you do not like this notation, change the [generation settings](#) to remove this macro.

*{}* and *()* produce ( and ), but there are also a mark for BOUML to find the parameters list

*<n>* and *<n>* produce the type and the name of each parameter (count from 0), this allows you to remove a parameter, add type modifiers etc ...

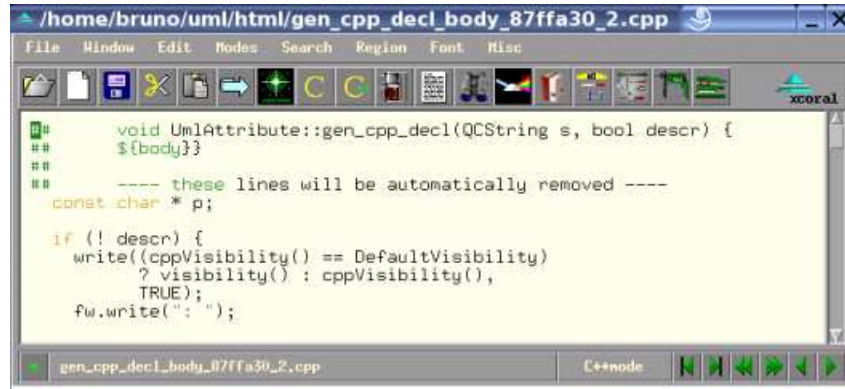
*throw* is replaced by the form *throw (...)* when at least an exception is defined in the UML tab. It is also possible to produce *throw()* when there is no exception depending on the [generation settings](#).

*body* is replaced by the body of the operation, this macro may also be replaced by the body itself (BOUML use this way for the *get* and *set* operations associated to a relation/attribute). The usage of *body* has a great advantage : when you hit **Default definition** or **From declaration** the body is not cleared ! At least BOUML share the declaration and definition forms of the operations (and other objects) to minimize the needed memory size, *body* help for this ! When *contextual body indent* is toggled the indentation of the keyword *body* is added to the indentation of the operation at the beginning of each line except the ones starting by a '#' or when the previous one ends by '\'. Since release 3.4 *contextual body indent* is not set when an operation is produced by a reverse to not alter the original indent.

The forms @{property}  
are replaced by the value of the corresponding user property if it is defined for the class or at an upper level, else it is not substituted.

A *template operation* is defined with an empty declaration (allowing Bouml to detect this case) and placing the actuals between `#{class}` and `::` in the definition part.

The **edit body** button is active when the definition contains `#{body}`, to edit it through the editor you want define the [environment variable](#) BOUML\_EDITOR, furthermore, when you set *add operation profile on body edition* through the *miscellaneous* menu the form defining the operation in the editor is added at the beginning of the edited file.



### Operation dialog, tab Java

This tab allows to give the Java definition of the operation, it is visible only if Java is set through the *Language* menu



In BOUML the generated code is obtained by the substitution of macros in a text, only the **Definition** part is editable, the other one help you to see what will be generated for Java (supposing you do not modify the Java code generator !).

When you push the button **Default definition**, the form specified through the [generation settings](#) is proposed adding all the operation parameters given in the UML tab. These forms may be modified as you want, even to produce illegal source code !

When you do not want to have this operation defined in Java, empties the definition manually or using the button **not generated in Java**.

`#{visibility}` produce the visibility (see the UML tab)

`#{final}` produce an empty string when the check box **final** is not checked, else produce *final*

`#{static}` produce an empty string when the operation is not a class operation (see the UML tab), else produce *static*

`#{abstract}` produce *abstract* when the operation is abstract (see the UML tab), else an empty string.

`#{synchronized}` produce an empty string when the check box **synchronized** is not checked, else produce *synchronized*

`#{type}` is replaced by the value type (see the UML tab)

`#{name}` is replaced by the name of the operation.

`#{()` and `#{()}` produce ( and ), but there are also a mark for BOUML to find the parameters list

`#{t<n>}` and `#{p<n>}`  
produce the type and the name of each parameter (count from 0), this allows you to remove a parameter, add type modifiers etc ...

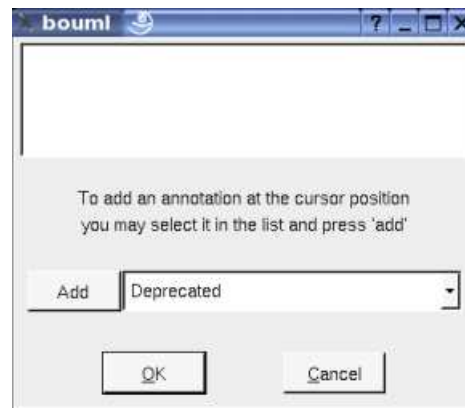
`#{throws}` is replaced by the form *throw (...)* when at least an exception is defined in the UML tab. It is also possible to produce *throw()* when there is no exception depending on the [generation settings](#).

`#{staticnl}` produce a line break when the operation is static, else an empty string. In case you do not like this notation, change the [generation settings](#) to remove this macro.

`#{body}` is replaced by the body of the operation, this macro may also be replaced by the body itself (BOUML use this way for the *get* and *set* operations associated to a relation/attribute). The usage of `#{body}` has a great advantage : when you hit **Default definition**

the body is not cleared ! At least BOUML share the definition forms of the operations (and other objects) to minimize the need memory size, `#{body}` help for this ! When *contextual body indent* is toggled the indentation of the keyword `#{body}` is added to the indent of the operation at the beginning of each line. Since release 3.4 *contextual body indent* is not set when an operation is produced by a reverse to not alter the original indent.

`#{@}` produces the annotations, when it is present in the definition the button *Edit annotation* is active, a specific dialog allows you to enter the annotations proposing the default annotations (*Deprecated ...*) and the ones defined through the classes stereotyped `@interface` :



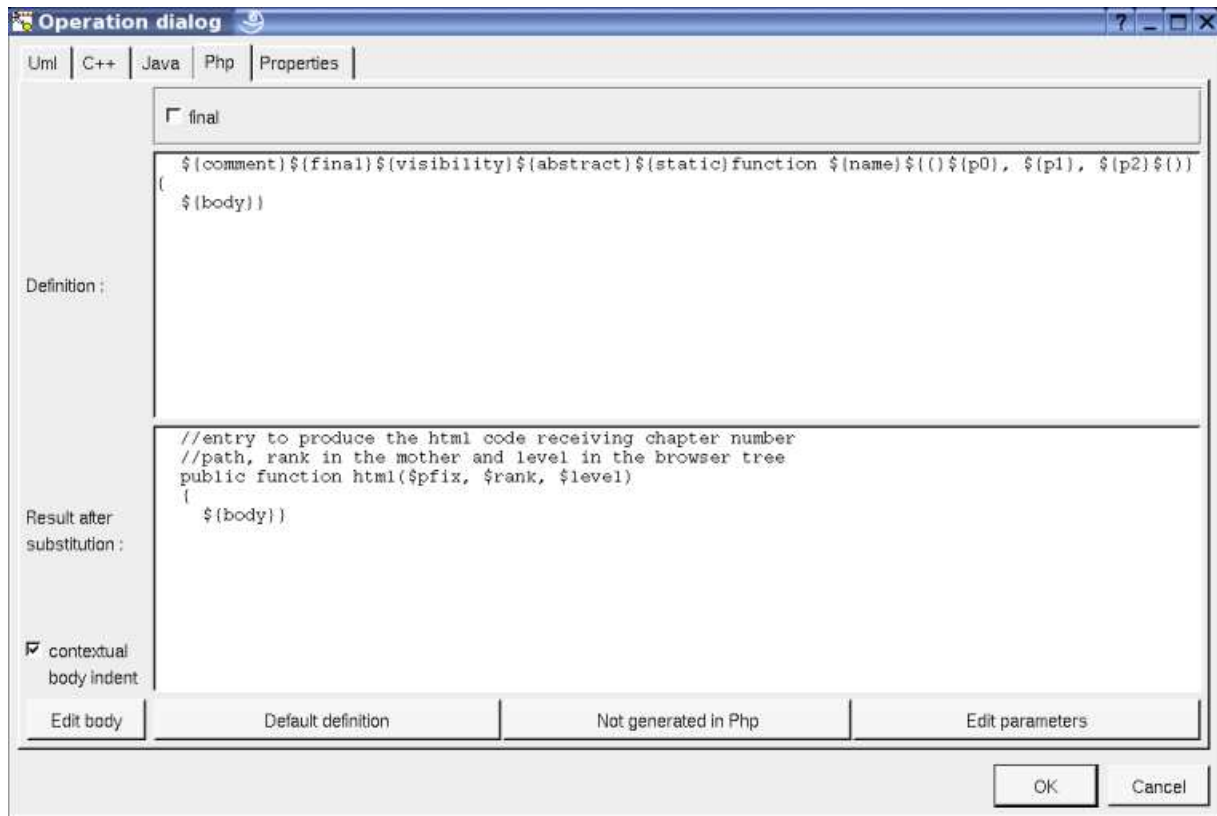
The forms `@{property}`  
are replaced by the value of the corresponding user property if it is defined for the class or at an upper level, else it is not substituted.

The **edit body** button is active when the definition contains `#{body}`, to edit it through the editor you want define the [environment variable](#) BOUML\_EDITOR, furthermore, when you set *add operation profile on body edition* through the *miscellaneous* menu the form defining the operation in the editor is added at the beginning of the edited file.



## Operation dialog, tab Php

This tab allows to give the Php definition of the operation, it is visible only if Php is set through the *Language* menu



In BOUML the generated code is obtained by the substitution of macros in a text, only the **Definition** part is editable, the other one help you to see what will be generated for Php (supposing you do not modify the Php code generator !).

When you push the button **Default definition**, the form specified through the [generation settings](#) is proposed adding all the operation parameters given in the UML tab. These forms may be modified as you want, even to produce illegal source code !

When you do not want to have this operation defined in Php, empties the definition manually or using the button **not generated in Php**.

`${visibility}` produce the visibility (see the UML tab), except if the visibility is *package*, use this case to generate Php 4 code

`${final}` produce an empty string when the check box **final** is not checked, else produce *final*

`${static}` produce an empty string when the operation is not a class operation (see the UML tab), else produce *static*

`${abstract}` produce *abstract* when the operation is abstract (see the UML tab), else an empty string.

`${name}` is replaced by the name of the operation.

`$(()` and `$(())` produce ( and ), but there are also a mark for BOUML to find the parameters list

`$(t<n>)` and `$(p<n>)`  
produce the type and the name of each parameter (count from 0), this allows you to remove a parameter, add modifiers like array or & etc ...

`${static!}` produce a line break when the operation is static, else an empty string. Not used by default, to add it change the [generation settings](#) to use this macro.

`${body}` is replaced by the body of the operation, this macro may also be replaced by the body itself (BOUML use this way for the *get* and *set* operations associated to a relation/attribute). The usage of `${body}` has a great advantage : when you hit **Default definition**

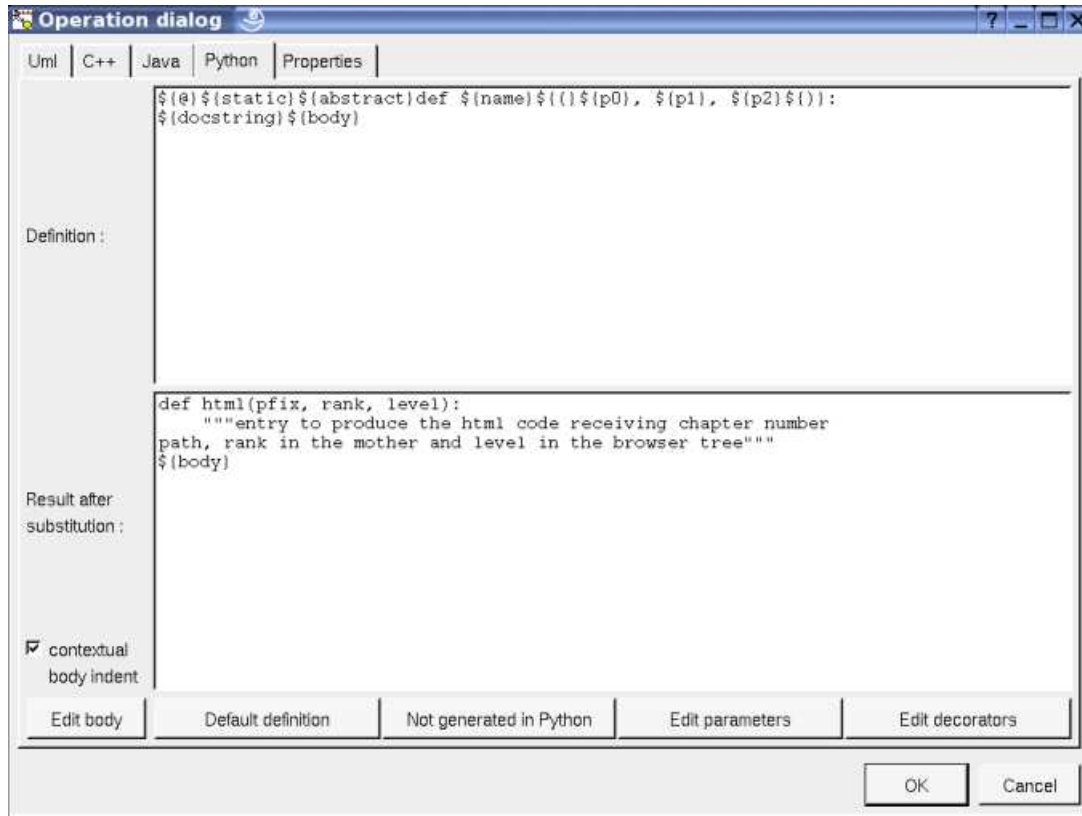
the body is not cleared ! At least BOUML share the definition forms of the operations (and other objects) to minimize the need memory size, `${body}` help for this ! When *contextual body indent* is toggled the indentation of the keyword `${body}` is added to the indent of the operation at the beginning of each line. Since release 3.4 *contextual body indent* is not set when an operation is produced by a reverse to not alter the original indent.

The `@{property}` forms  
are replaced by the value of the corresponding user property if it is defined for the class or at an upper level, else it is not substituted.

The **edit body** button is active when the definition contains `${body}`, to edit it through the editor you want define the [environment variable](#) `BOUML_EDITOR`, furthermore, when you set *add operation profile on body edition* through the *miscellaneous* menu the form defining the operation in the editor is added at the beginning of the edited file.

## Operation dialog, tab Python

This tab allows to give the Python definition of the operation, it is visible only if Python is set through the *Language* menu



In BOUML the generated code is obtained by the substitution of macros in a text, only the **Definition** part is editable, the other one help you to see what will be generated for Python (supposing you do not modify the Python code generator !).

When you push the button **Default definition**, the form specified through the [generation settings](#) is proposed adding all the operation parameters given in the UML tab. These forms may be modified as you want, even to produce illegal source code !

When you do not want to have this operation defined in Python, empties the definition manually or using the button **not generated in Python**.

`${@}` produce the decorators, don't use decorator to produce `@staticmethod` and `@abstractmethod`, see `${static}` and `${abstract}` bellow

`${static}` produce `@staticmethod` followed by a newline if the method is declared *static* in the UML tab, else an empty string

`${abstract}` produce `@abstractmethod` followed by a newline if the method is declared *abstract* in the UML tab, else an empty string

`${name}` is replaced by the name of the operation.

`${(}` and `${)}` produce ( and ), but there are also a mark for BOUML to find the parameters list

`${t<n>}` and `${p<n>}`  
produce the type and the name of each parameter (count from 0), this allows you to remove a parameter, add modifiers like array or & etc ...

`${docstring}`  
produce produce the description of the operation placed between `"""` and followed by a new line if the description is not empty, nothing is produced. If you prefer you can use `${comment}` or `${description}`

`${body}` is replaced by the body of the operation, this macro may also be replaced by the body itself (BOUML use this way for the *get* and *set* operations associated to a relation/attribute). The usage of `${body}` has a great advantage : when you hit **Default definition**

the body is not cleared ! At least BOUML share the definition forms of the operations (and other objects) to minimize the need memory size, `${body}` help for this ! When *contextual body indent* is toggled the needed indentation is added to the body (this mean at least the first line is not indented. Since release 3.4 *contextual body indent* is not set when an operation is produced by a reverse to not alter the original indent. In the special case of the `__init__` operation the initialization of the instance attributes and relations is added before the body, they are not produced if `${body}` is not part of the operation definition.

The forms `@{property}`

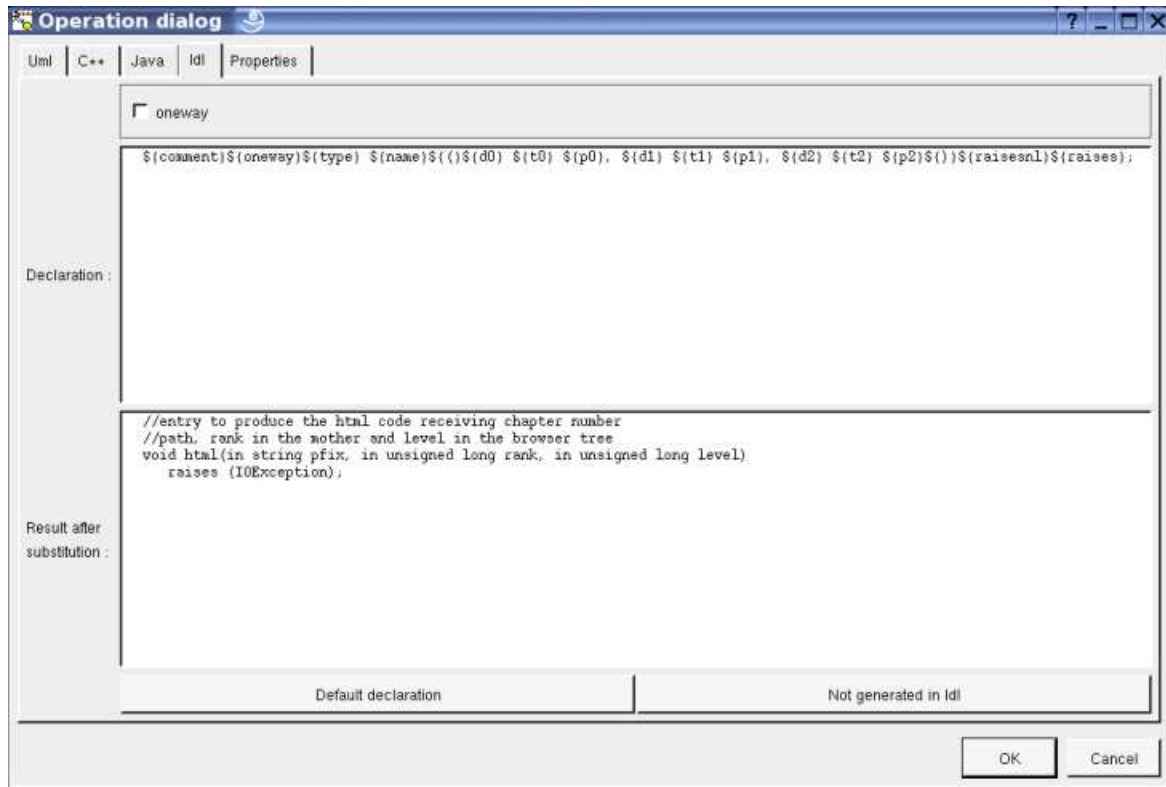


are replaced by the value of the corresponding user property if it is defined for the class or at an upper level, else it is not substituted.

The **edit body** button is active when the definition contains `#{body}`, to edit it through the editor you want define the [environment variable](#) BOUML\_EDITOR, furthermore, when you set *add operation profile on body edition* through the *miscellaneous* menu the form defining the operation in the editor is added at the beginning of the edited file.

### Operation dialog, tab Idl

This tab allows to give the Idl declaration of the operation, it is visible only if Idl is set through the *Language* menu



In BOUML the generated code is obtained by the substitution of macros in a text, only the **Definition** part is editable, the other one help you to see what will be generated for Java (supposing you do not modify the Java code generator !).

When you push the button **Default definition**, the form specified through the [generation settings](#) is proposed adding all the operation parameters given in the UML tab. These forms may be modified as you want, even to produce illegal source code !

When you do not want to have this operation defined in Java+, empties the definition manually or using the button **not generated in Idl**.

`#{oneway}` produce an empty string when the check box **oneway** is not checked, else produce *oneway*

`#{type}` is replaced by the value type (see the UML tab)

`#{name}` is replaced by the name of the operation.

`#{(}` and `#{)}` produce ( and ), but there are also a mark for BOUML to find the parameters list

`#{t<n>}` and `#{p<n>}`  
produce the type and the name of each parameter (count from 0), this allows you to remove a parameter, add type modifiers etc ...

`#{raise}` is replaced by the form *raise (...)* when at least an exception is defined in the UML tab.

`#{raisesnl}`  
produce a line break when the operation have exception, else an empty string. In case you do not like this notation, change the [generation settings](#) to remove this macro.

The `#{property}` forms `@{property}`  
are replaced by the value of the corresponding user property if it is defined for the class or at an upper level, else it is not substituted.

### [Menu](#) : duplicate

The menu entry *duplicate* clone the operation.

**Menu : delete**

The menu entry *delete* is only present when the operation is not *read-only*.

Delete the operation, and all its representation in the diagrams. After that it is possible to undelete it (from the browser) until you close the project : obviously the deleted items are not saved !

**Menu : mark**

See [mark](#)

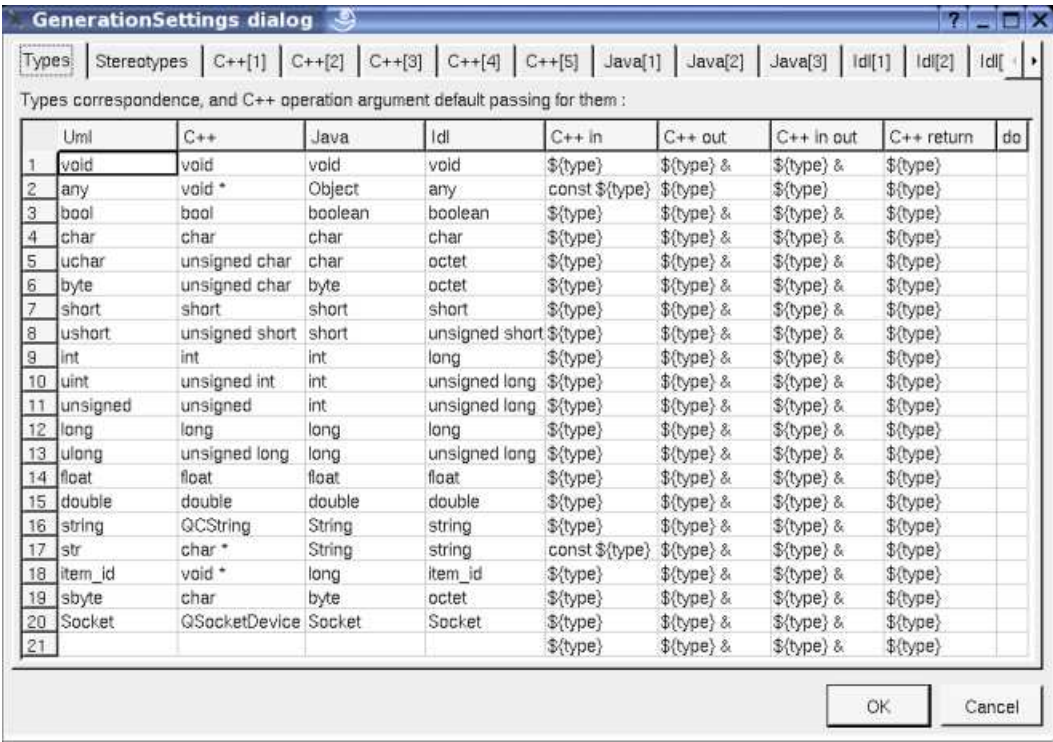
**Menu : tools**

Appears only when at least one [plug-out](#) is associated to the operations. To apply a *plug-out* on the operation.

**Generation settings**

This very important dialog allows to specify many default definitions concerning the operations, more details will be given in [C++ generator](#), [Java generator](#), [Php generator](#) and [Idl generator](#).

The first tab allows to specify how the types not supported by an UML class are translated in each language except Php, and the default C++ argument passing for these types depending on the direction more the default type form to return value :



The fourth C++ tab *Stereotypes* allows to specify the default declaration and definition of an operation (without the parameters), the default argument pass enumeration's item and an argument whose type is not one of the types defined in the first tab, and if a *throw()* must be generated when the operation does not have exception. They are visible only if C++ is set through the menu *Languages*





**GenerationSettings dialog**

Types | Stereotypes | C++[1] | C++[2] | C++[3] | **C++[4]** | C++[5] | Java[1] | Java[2] | Java[3] | Idl[1] | Idl[2] | Idl[3] | Idl[4] | ...

Get operation default definition: Visibility: ☒ public ☐ protected ☐ private Modifiers: ☒ inline ☐ const value ☐ const name:  ☐ also in uml

Set operation default definition: Visibility: ☒ public ☐ protected ☐ private Modifiers: ☐ inline ☐ const param ☐ by ref name:  ☐ also in uml

Enumeration item default definition:

Default type forms for the enums: input parameter:  output parameter:  input output parameter:  operation return: 

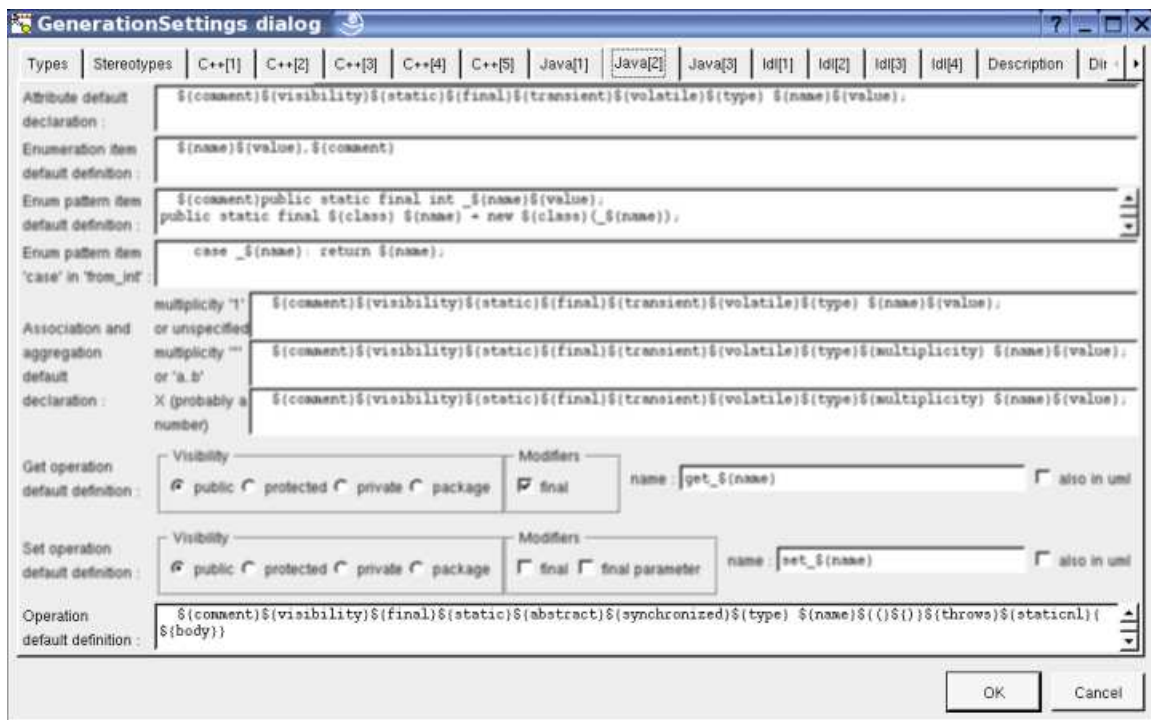
Default type forms for the types not given in the first tab: input parameter:  output parameter:  input output parameter:  operation return: 

Operation default declaration:

Operation default definition:  ☐ throw()

OK Cancel

The second Java tab allows to set the default operation definition (without the parameters) in Java, it is visible only if Java is set through the menu *Languages*



**GenerationSettings dialog**

Types | Stereotypes | C++[1] | C++[2] | C++[3] | C++[4] | C++[5] | **Java[1]** | **Java[2]** | Java[3] | Idl[1] | Idl[2] | Idl[3] | Idl[4] | Description | Dir | ...

Attribute default declaration:

Enumeration item default definition:

Enum pattern item default definition:

Enum pattern item 'case' in 'from\_int':

Association and aggregation default declaration: multiplicity '1' or unspecified:  multiplicity 'n':  multiplicity 'a..b':

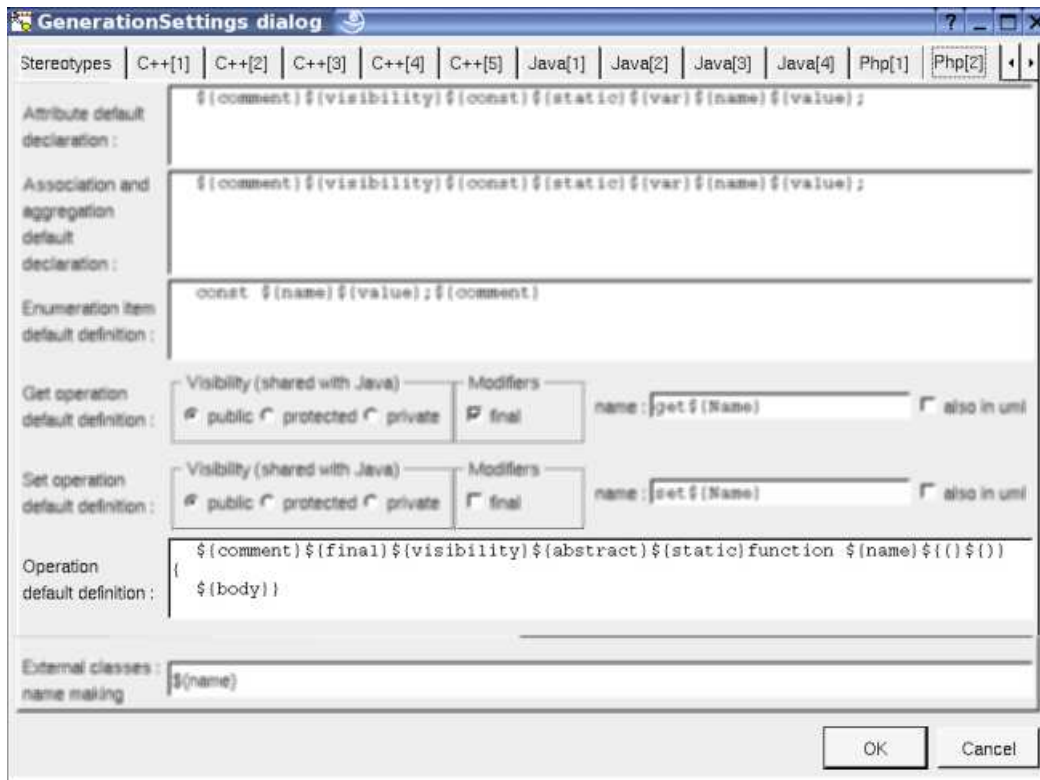
Get operation default definition: Visibility: ☒ public ☐ protected ☐ private ☐ package Modifiers: ☒ final name:  ☐ also in uml

Set operation default definition: Visibility: ☒ public ☐ protected ☐ private ☐ package Modifiers: ☐ final ☐ final parameter name:  ☐ also in uml

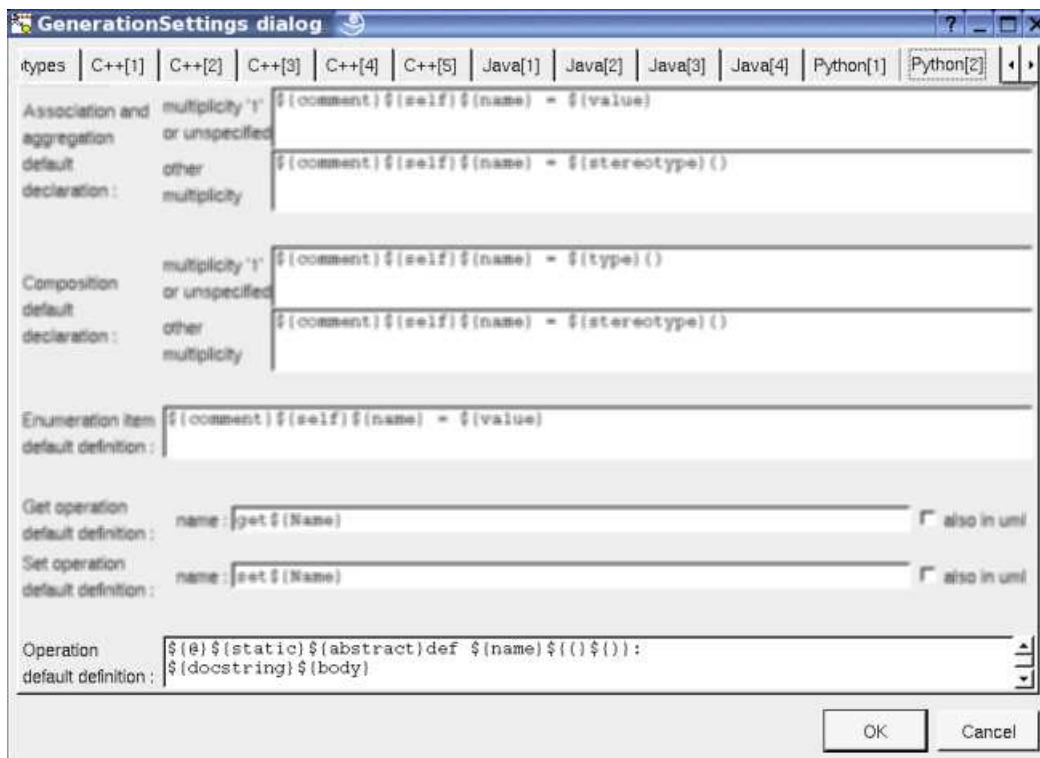
Operation default definition:

OK Cancel

The second Php tab allows to set the default operation definition (without the parameters) in Php, it is visible only if Php is set through the menu *Languages*



The second Python tab allows to set the default operation definition (without the parameters) in Python, it is visible only if Python is set through the menu *Languages*



The third Idl tab allows to set the default operation declaration in Idl, it is visible only if Idl is set through the menu *Languages*

Types	Stereotypes	C++[1]	C++[2]	C++[3]	C++[4]	C++[5]	Java[1]	Java[2]	Java[3]	Idl[1]	Idl[2]	Idl[3]	Idl[4]	Description	Dir
Attribute default declaration :	<code>\${comment}\${readonly}\${attribut} \${type} \${name};</code>														
Attribute default declaration in valuetype :	<code>\${comment}\${visibility}\${type} \${name};</code>														
Constant default declaration :	<code>\${comment}const \${type} \${name}\${value};</code>														
Union item default declaration :	<code>\${comment}case \${case} : \${readonly}\${type} \${name};</code>														
Enumeration item default declaration :	<code>\${name}, \${comment}</code>														
Get operation default definition :	name : <input type="text" value="get_\${name}"/>														<input type="checkbox"/> also in uml
Set operation default definition :	<input type="checkbox"/> oneway name : <input type="text" value="set_\${name}"/>														<input type="checkbox"/> also in uml
Operation default declaration :	<code>\${comment}\${oneway}\${type} \${name}\${()}\${}\${raisesn}\${raises};</code>														

OK Cancel

The tab *Description* allows to set a default description :

**GenerationSettings dialog**

[1]	C++[2]	C++[3]	C++[4]	C++[5]	Java[1]	Java[2]	Java[3]	Idl[1]	Idl[2]	Idl[3]	Idl[4]	Description
Artifact default description :												
Class default description :												
Operation default description :												
Attribute default description :												
Relation default description :												

OK Cancel

## Drawing

The operations are drawn with their [class](#) in a [class diagram](#), and alone in a [sequence diagram](#) or a [collaboration diagram](#).

The way an operation is written or not in its class picture in a [class diagram](#) depend on the [class drawing settings](#) options [drawing language](#), [hide operation](#), [show full members definition](#), [member max width](#) and [show member visibility](#), and also on the [class menu](#) entry [individual operation visibility](#).

For instance in a class diagram, the [drawing language](#) is UML, the full member definitions and visibility are shown, and the attributes are hidden :

UmlOperation
+ UmlOperation(in id : Item_id, in n : string) + sKind() : string + memo_ref() : void + html(in pfix : string, in rank : uint) : void + <u>ref_index() : void</u> + <u>generate_index() : void</u> - gen_cpp_decl(in s : string, in descr : bool) : void - gen_java_decl(in s : string) : void + compute_name(in s : string) : string

changing the [drawing language](#) from UML (the default) to C++ :

UmlOperation
+ UmlOperation(void * Id, const QCString & n) + QCString sKind() + void memo_ref() + void html(QCString pfix, unsigned int rank) + <u>void ref_index()</u> + <u>void generate_index()</u> - void gen_cpp_decl(QCString s, bool descr) - void gen_java_decl(QCString s) + QCString compute_name(QCString s)

changing the [drawing language](#) to Java :

UmlOperation
+ UmlOperation(long Id, String n) + String sKind() + void memo_ref() + void html(String pfix, int rank) + <u>void ref_index()</u> + <u>void generate_index()</u> - void gen_cpp_decl(String s, boolean descr) - void gen_java_decl(String s) + String compute_name(String s)

---

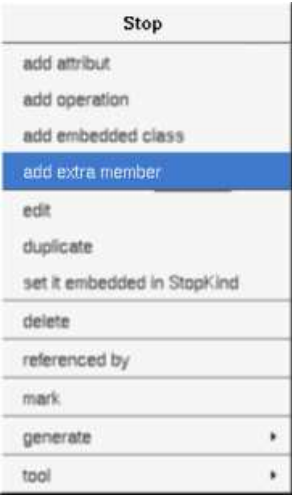
Previous : [attribute](#)

Next : [extra member](#)

# Extra member

An extra member is an alien allowing you to specify directly the generated code for each language. This allows to manage non U items, for instance C++ pre-processor directives, Java attribute initializations etc ...

An extra member is created through the class menu called from the [browser](#) :



## Menus

The extra member menu appearing with a right mouse click on its representation in the browser is something like these, supposing it is not *read-only* nor deleted :

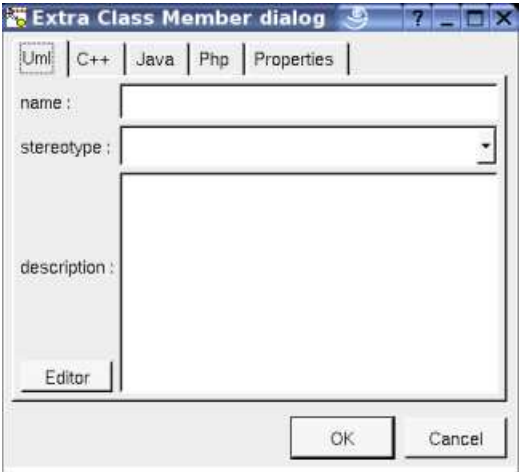


### [Menu](#) : edit

*edit*  
allows to show/modify the extra member properties. In case the extra member is read-only, the fields of the dialog are also read-only.

### Extra member dialog, tab Uml

The tab *Uml* is a global tab, independent of the language :



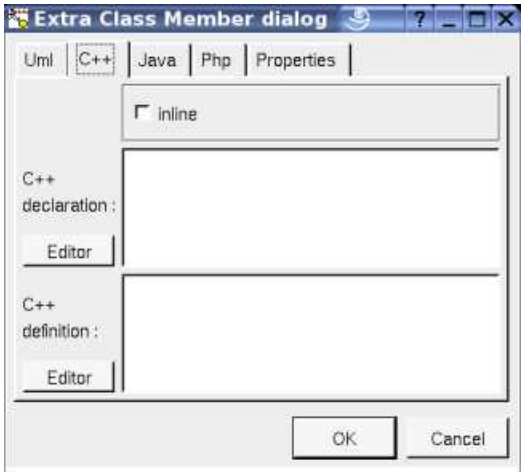
The *name* and the *stereotype* do not have meaning for BOUML.

The *description* is not used by the code generators.

The *editor* button visible above and associated here to the description, allows to edit the description in an other window, or to call an external editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note that this external editor have to create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel with BOUML which looks each second at the file contents to get the new definition until the dialog is closed (I do not like very much polling but this works even QT isn't compiled with the thread support).

**Extra member dialog, tab C++**

The tab C++ allows to specify the code generated in the header and/or the source file(s), it is visible only if C++ is set through the menu *Languages*



The *declaration* part is generated in the definition of the class containing the extra member, except if the *inline* toggle is set, in this case the generation is made after the class definition as for an inline operation.

The *definition* part is produced in the source file rather than in the header one.

For instance with :

EM	#ifdef	#endif

and :

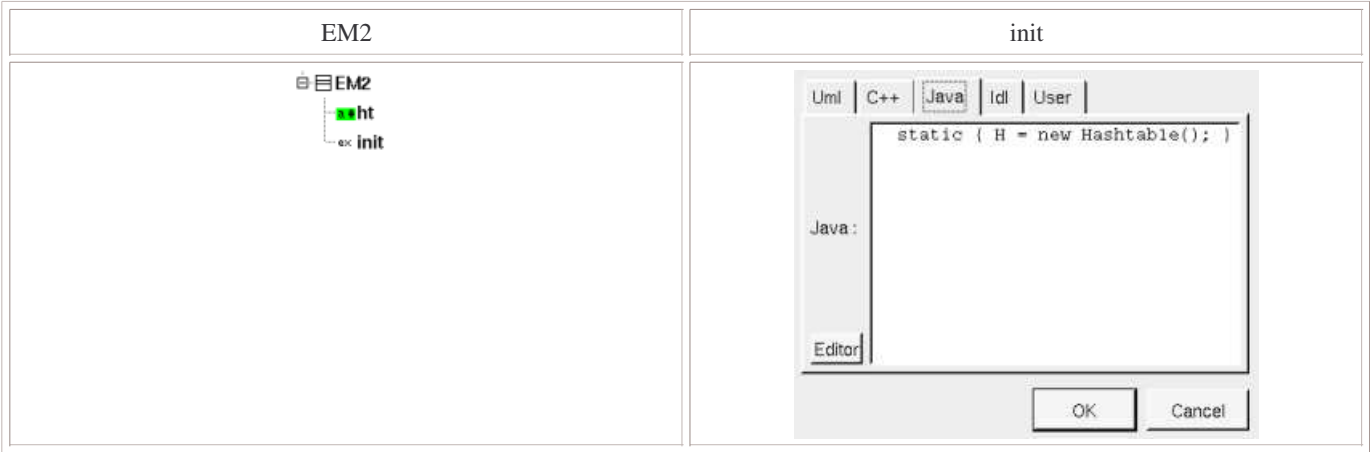
C++ header	C++ source
<pre>class EM { #ifdef NEEDED // decl      public:         void op();  #endif };</pre>	<pre>#ifdef NEEDED // def void EM::op() { }  #endif</pre>

C++ header
<pre>class EM { #ifdef NEEDED // decl      public:         void op();  #endif };  #ifdef NEEDED // def inline void EM::op() { }  #endif</pre>

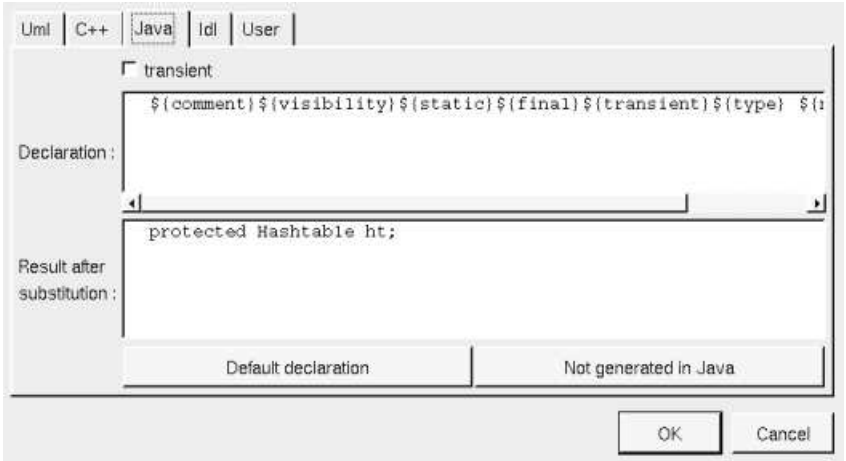
The tab Java allows to specify the code generated in Java, it is visible only if Java is set through the menu *Languages*, for instance



with :



and :



The generated code for *EM2* is :

```
class EM2 {  
    protected Hashtable ht;  
  
    static { H = new Hashtable(); }  
}
```

**Extra member dialog, tab Php**

The tab Php allows to specify the code generated in Php, it is visible only if Php is set through the menu *Languages* :



**Extra member dialog, tab Python**

The tab Python allows to specify the code generated in Python, it is visible only if Python is set through the menu *Languages* :



#### Extra member dialog, tab Idl

The tab Idl allows to specify the code generated in Idl, it is visible only if Idl is set through the menu *Languages* :



#### Menu : duplicate

The menu entry *duplicate* clone the extra member.

#### Menu : delete

The menu entry *delete* is only present when the extra member is not *read-only*.

Delete the extra member. After that it is possible to undelete it (from the browser) until you close the project : obviously the delet items are not saved !

#### Menu : mark

See [mark](#)

#### Menu : tools

Appears only when at least one [plug-out](#) is associated to the extra member. To apply a *plug-out* on the extra member.

---

Previous : [operation](#)

Next : [class instance](#)

# Class instance

A class instance may be placed under a [class view](#), a [use case view](#) or a [use case](#).

## Menus

The class menu appearing with a right mouse click on its representation is something like this, supposing the instance not *read-only* nor deleted (see also [menu in a diagram](#)) :



### Menu : edit

*edit* allows to show/modify the instance properties. In case the class is read-only, the fields of the dialog are also read-only.

#### Class dialog, tab Uml



The button *class* shows a menu proposing :

- to select in the browser the class of the instance
- if the instance is not read-only and if a class is selected in the browser : to set the class of the instance to be this class
- if the view containing the class containing the instance is not read-only : to create a new class and to set the attribute type to it

The *editor* button visible above and associated here to the description, allows to edit the description in an other window, or to call an external editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note that this external editor have to create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel with BOUML which looks each second at the file contents to get the new definition until the dialog is closed (I do not like very much polling but this works even QT isn't compiled with the thread support).

#### Class dialog, tab Attributes

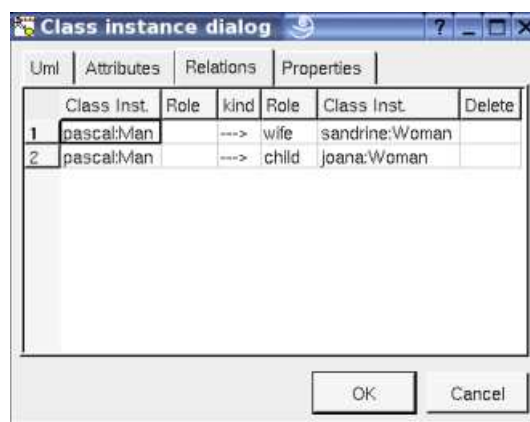
This tab allows to specify the value of the attributes of the instance, including the inherited ones :



Unfortunately an attribute can't be set several times, this is a problem when an attribute has a multiplicity greater than 1.

### Class dialog, tab Relations

This tab allows to specify the value of the relations of the instance, including the inherited ones:



Unfortunately a relation can't be set several times, this is a problem when a relation has a multiplicity greater than 1.

### Menu : duplicate

The menu entry *duplicate* clone the class instance, except the value of the relations.

### Menu : referenced by

To know who reference the current class instance

### Menu : mark

See [mark](#)

### Menu : delete

The menu entry *delete* is only present when the class instance is not *read-only*.

Delete the instance and all the representation of it in the opened diagrams. After that it is possible to undelete it (from the browser) until you close the project : obviously the deleted items are not saved !

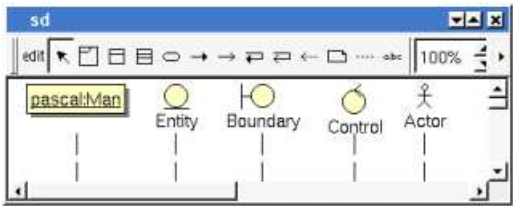
### Menu : tool

The menu entry *tool* is only present in case at least a [plug-out](#) may be applied on a class instance. The selected tool is called and applied on the current class instance.

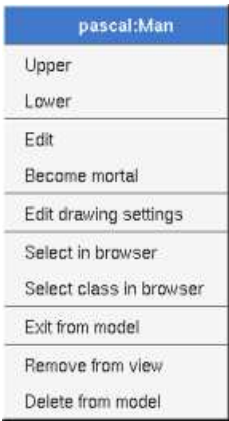
## Drawing

### Sequence diagrams

A class instance is drawn in a sequence diagram as a rectangle or an icon depending on the class's stereotype and the [drawing settings](#), the value of the attributes and relations don't appears :



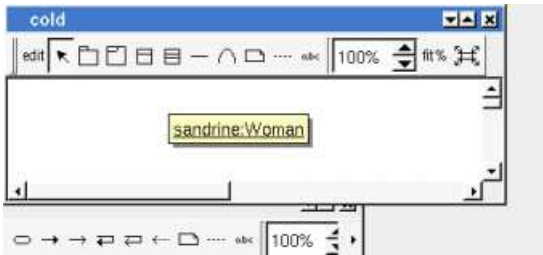
A right mouse click on a class instance in a sequence diagram calls the following menu (supposing the instance editable) :



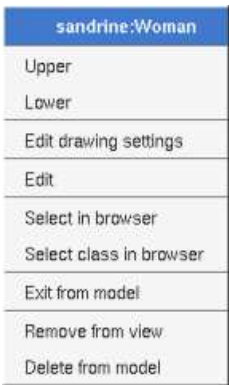
Please refer to the chapter [sequence diagram](#) for more details.

**Collaboration diagrams**

A class instance is drawn in a collaboration diagram as a rectangle, the value of the attributes and relations don't appears :



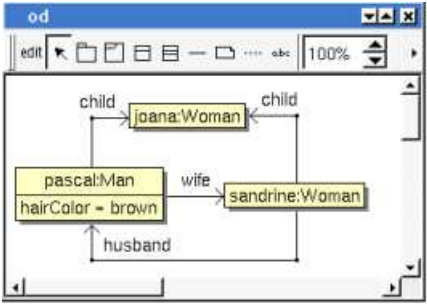
A right mouse click on a class instance in a collaboration diagram calls the following menu (supposing the instance editable) :



Please refer to the chapter [collaboration diagram](#) for more details.

**Object diagrams**

A class instance is drawn in a collaboration diagram as a rectangle, the value of the attributes and relations appears :



A right mouse click on a class instance in an object diagram calls the following menu (supposing the instance editable) :

joana:Woman
Upper
Lower
Edit drawing settings
Edit
Select in browser
Select class in browser
Select linked items
Remove from view
Delete from model

Please refer to the chapter [object diagram](#) for more details.

Previous : [extra member](#)

Next : [state](#)





The tab *Uml* is a global tab :



The proposed **stereotypes** are the default one specified through the [Default stereotypes dialog](#) more the current one (empty here). It is possible to choose into the list or to give a new one, or to empty it.

The **editor** button visible above and associated here to the description, allows to edit the description in an other window, or to call an external editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note that this external editor have to create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel with BOUML which looks each second at the file contents to get the new definition until the dialog is closed (I do not like very much polling but this works even QT isn't compiled with the thread support).

#### State dialog, tab Ocl

The *Ocl* tab allows to specify the *state* behavior using the *Ocl* language, but in fact BOUML doesn't check the syntax, this is just a convention.



#### State dialog, tab C++

This tab allows to specify the *state* behavior in C++, but in fact BOUML doesn't check the syntax, this is just a convention.



#### State dialog, tab Java

This tab allows to specify the *state* behavior in *Java*, but in fact BOUML doesn't check the syntax, this is just a convention.



### **Menu : duplicate**

The menu entry *duplicate* clone the *state* without its children

### **Menu : extract it from X**

The menu entry *extract it from X* is present when the *stste* is nested in *X* (perhaps *X* is itself nested in another one, but it is not visible here). Choosing this entry the current *stste* became a children of the parent of *X*.

### **Menu : referenced by**

To know who reference the current *stste* through a transition.

### **Menu : mark**

See [mark](#)

### **Menu : delete**

The menu entry *delete* is only present when the *state* is not *read-only*.

Delete the *state* and all its children, and all the representation of them in the opened diagrams. After that it is possible to undelete them (from the browser) until you close the project : obviously the deleted items are not saved !

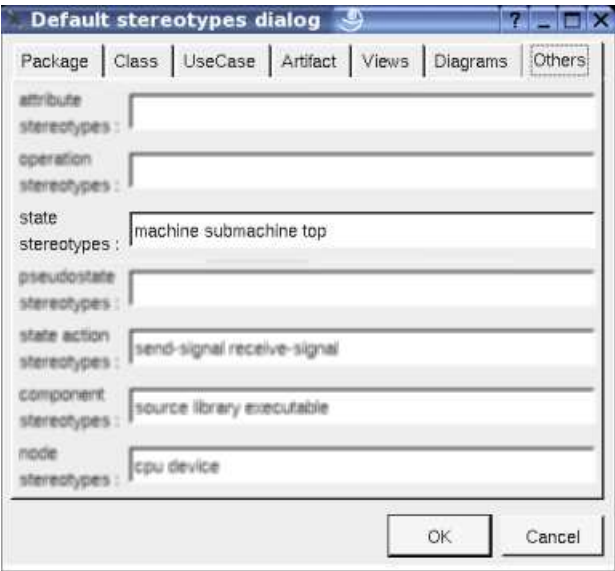
### **Menu : tool**

The menu entry *tool* is only present in case at least a [plug-out](#) may be applied on a *state*. The selected tool is called and applied on the current *state*.

---

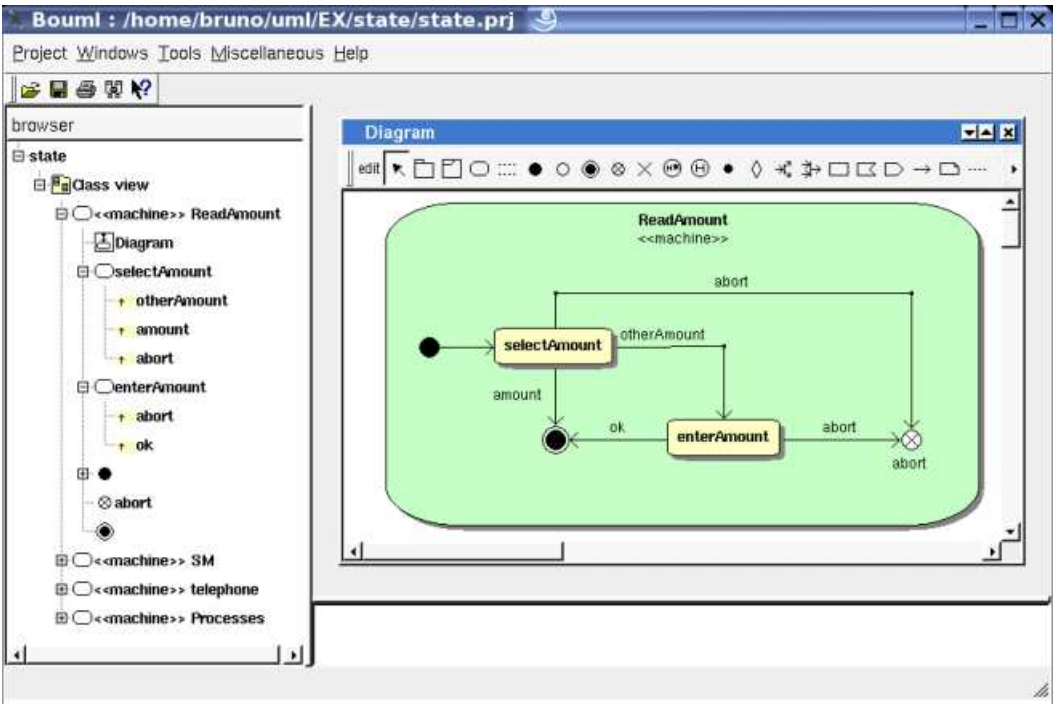
## **Default stereotypes**

The [dialog](#) allowing to set the default stereotypes :



## Drawing

A *state* is drawn in a diagram as a rectangle with rounded corners :



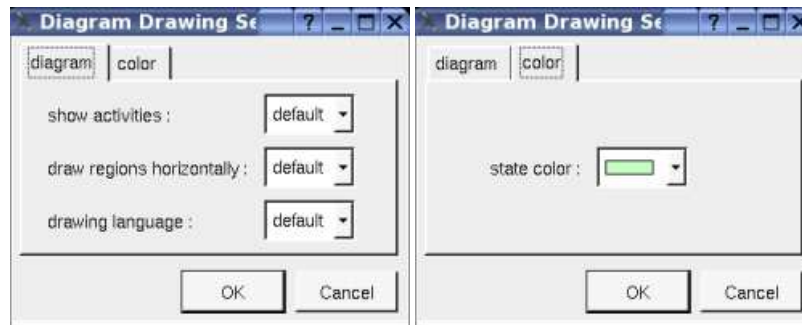
A right mouse click on a *state* in a diagram calls the following menu (supposing the *state* editable) :

enterAmount
upper
lower
edit drawing settings
edit state
select in browser
select linked items
set associated diagram
remove from view
delete from model
tool

## Menu : edit drawing settings

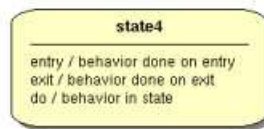
These *drawing settings* concerns only the picture for which the menu is called.

A *settings* *valuing* *default* indicates that the setting specified in the upper level (here the diagram) must be followed, obviously this one may also be *default* ... up to the project level. When you add a *state* in a diagram, all the settings are set to *default*.



### show activities :

Allows to show or not the *state*'s activities in a compartment depending on the drawing language.

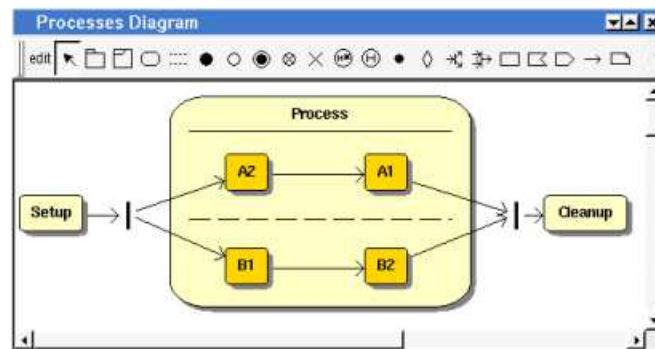


### drawing language :

Allows to specify the used language in case the *state*'s activities are shown.

### drawing regions horizontally :

Allows to indicate if the *regions* are drawn horizontally or vertically, for instance :



Note : the regions are not visible while the state picture is too small.

### State color :

To specify the fill color.

## Menu : set associated diagram

*set associated diagram* allows to automatically open the current diagram when a double mouse click is made on the *state* representation in a diagram or the browser. After that the only way to edit the *state* is to choose the *edit* entry in the menu.

## Menu tools :

Appears only when at least one *plug-out* is associated to the *states*. To apply a *plug-out* on the *state*.

Next : [state action](#)

# State Actions

In the *browser* a *state action* may contain transitions

The *state action's* properties will be exposed below.

## Menus

The *state action* menu appearing with a right mouse click on its representation in the browser is something like these, supposing the *action* not *read-only* nor deleted (see also [menu in a diagram](#)) :

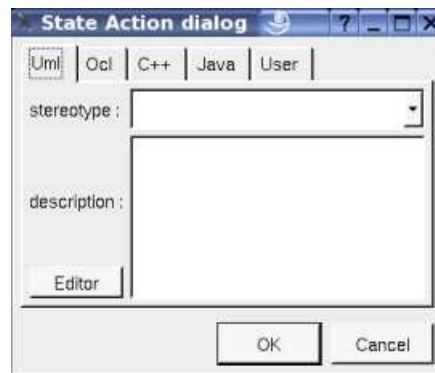


### Menu : edit

*edit* allows to show/modify the *action* properties. In case the *action* is *read-only*, the fields of the dialog are also *read-only*.

### State action dialog, tab Uml

The tab *Uml* is a global tab :

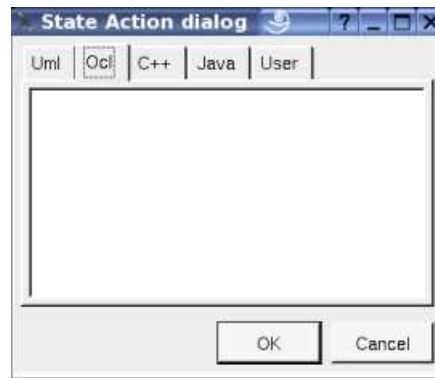


The proposed *stereotypes* are the default one specified through the [Default stereotypes dialog](#) more the current one (empty here). It is possible to choose into the list or to give a new one, or to empty it.

The *editor* button visible above and associated here to the description, allows to edit the description in an other window, or to call an external editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note that this external editor have to create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel with BOUML which looks each second at the file contents to get the new definition until the dialog is closed (I do not like very much polling but this works even QT isn't compiled with the thread support).

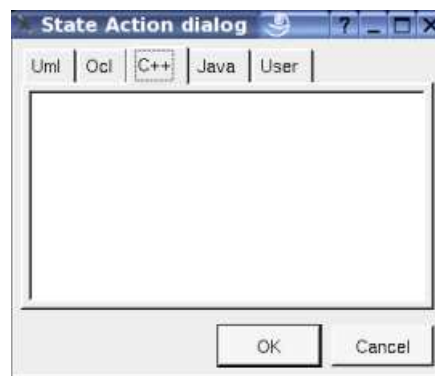
### State action dialog, tab Ocl

The *Ocl* tab allows to specify the *state action* expression using the *Ocl* language, but in fact BOUML doesn't check the syntax, this is just a convention.



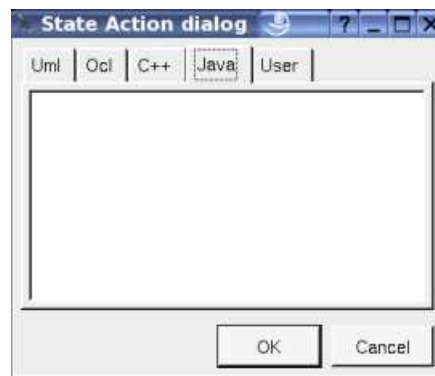
#### State action dialog, tab C++

This tab allows to specify the *state action* expression in *C++*, but in fact BOUML doesn't check the syntax, this is just a convention.



#### State action dialog, tab Java

This tab allows to specify the *state action* expression in *Java*, but in fact BOUML doesn't check the syntax, this is just a convention.



#### Menu : duplicate

The menu entry *duplicate* clone the *state action* without its children

#### Menu : referenced by

To know who reference the current *state action* through a transition.

#### Menu : mark

See [mark](#)

#### Menu : delete

The menu entry *delete* is only present when the *state action* is not *read-only*.

Delete the *state action* and all its children, and all the representation of them in the opened diagrams. After that it is possible to undelete them (from the browser) until you close the project : obviously the deleted items are not saved !

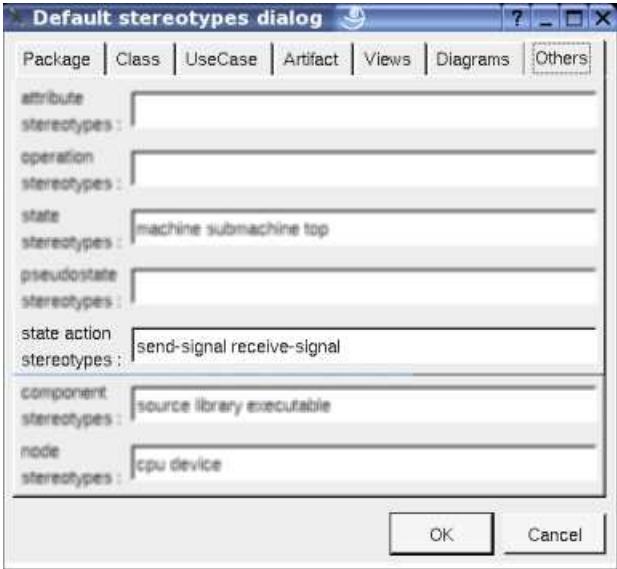


**Menu : tool**

The menu entry *tool* is only present in case at least a *plug-out* may be applied on a *state action*. The selected tool is called and applied on the current *state action*.

**Default stereotypes**

The [dialog](#) allowing to set the default stereotypes :



**Drawing**

Depending on its stereotype a *state action* is drawn in a diagram as a rectangle, a convex pentagon or a concave pentagon :



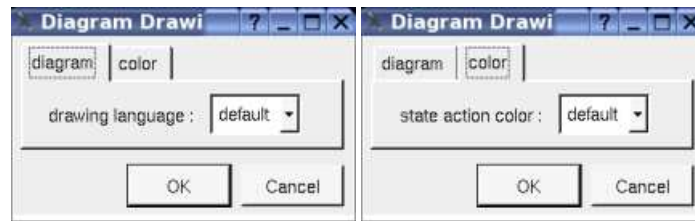
A right mouse click on a *state action* in a diagram calls the following menu (supposing the *state action* editable) :

action
upper
lower
edit drawing settings
edit state action
select in browser
remove from view
delete from model
tool ▶

**Menu : edit drawing settings**

These *drawing settings* concerns only the picture for which the menu is called.

A *settings* valuing *default* indicates that the setting specified in the upper level (here the diagram) must be followed, obviously this one may also be *default* ... up to the project level. When you add a *state action* in a diagram, all the settings are set to *default*.

**drawing language :**

Allows to specify the used language showing the expression

**State action color :**

To specify the fill color.

**Menu tools :**

Appears only when at least one *plug-out* is associated to the *state actions*. To apply a *plug-out* on the *state action*.

---

Previous : [state](#)

Next : [pseudo state](#)

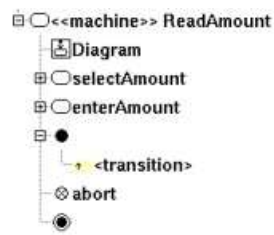
# Pseudo State

Currently BOUML manages only the *behavioral states*.

The *pseudo states* are :

- final state (for reasons of simplification)
- initial
- deep history
- shallow history
- join
- fork
- junction
- choice
- entry point
- exit point
- terminate

In the *browser* a *pseudo state* may contain transitions (when this is legal) :



The *pseudo state's* properties will be exposed below.

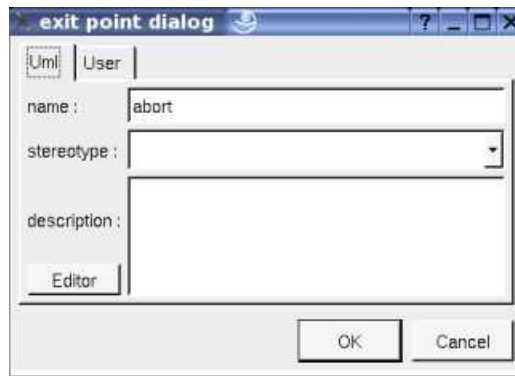
## Menus

The *pseudo state* menu appearing with a right mouse click on its representation in the browser is something like these, supposing the *pseudo state* not *read-only* nor deleted (see also [menu in a diagram](#)) :

exit point
edit
duplicate
delete
referenced by
mark
tool ▶

### Menu : edit

*edit* allows to show/modify the *pseudo state* properties. In case the *pseudo state* is *read-only*, the fields of the dialog are also *read-only*.



The proposed *stereotypes* are the default one specified through the [Default stereotypes dialog](#) more the current one (empty here). It is possible to choose into the list or to give a new one, or to empty it.

The *editor* button visible above and associated here to the description, allows to edit the description in an other window, or to call an external editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note that this external editor have to create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel with BOUML which looks each second at the file contents to get the new definition until the dialog is closed (I do not like very much polling but this works even QT isn't compiled with the thread support).

### **Menu : duplicate**

The menu entry *duplicate* clone the *pseudo state* without its children

### **Menu : referenced by**

To know who reference the current *pseudo state* through a transition.

### **Menu : mark**

See [mark](#)

### **Menu : delete**

The menu entry *delete* is only present when the *pseudo state* is not *read-only*.

Delete the *pseudo state* and all its children, and all the representation of them in the opened diagrams. After that it is possible to undelete them (from the browser) until you close the project : obviously the deleted items are not saved !

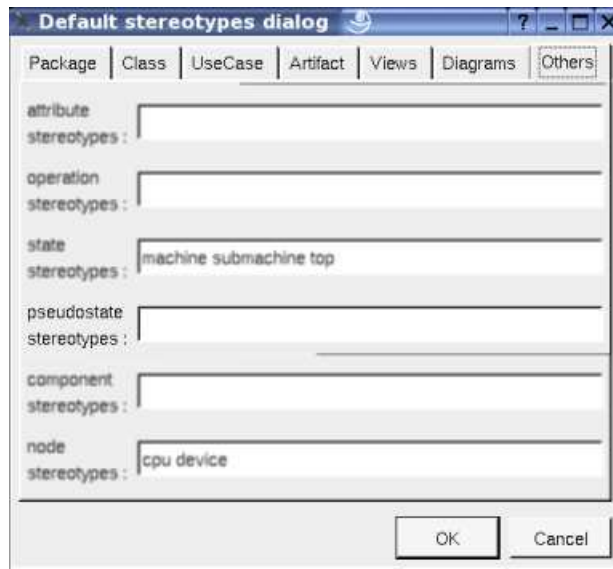
### **Menu : tool**

The menu entry *tool* is only present in case at least a [plug-out](#) may be applied on a *pseudo state* of this kind. The selected tool is called and applied on the current *pseudo state*.

---

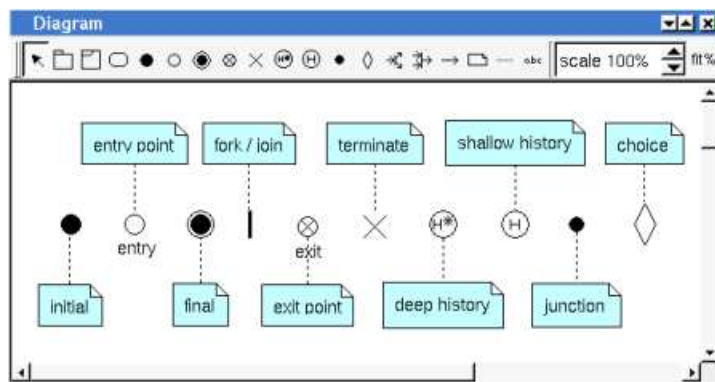
## **Default stereotypes**

The [dialog](#) allowing to set the default stereotypes :

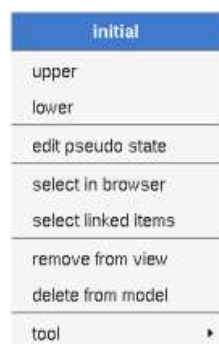


## Drawing

A *pseudo state* is drawn in a diagram depending on its kind :



A right mouse click on a *pseudo state* in a diagram calls the following menu (supposing the *pseudo state* editable) :



### Menu tools :

Appears only when at least one *plug-out* is associated to the *pseudo state* kind. To apply a *plug-out* on the *pseudo state*.

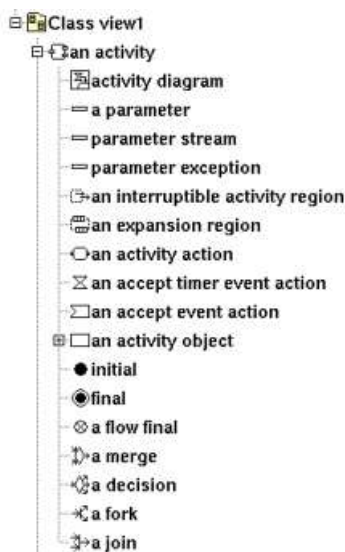
Previous : [state action](#)

Next : [activity](#)

# Activity

## Activity

In the *browser* an *activity* is placed in a [class view](#), a [use case view](#) or a [use case](#), and may contain parameters, expansion and interruptible [activity regions](#), activity nodes ([activity action](#), [object node](#) or [control node](#)) and [activity diagrams](#) :



The *activity's* properties are exposed below.

## Activity parameter

To simplify in Bouml a parameter is both a UML parameter and a UML parameter node, this means that the properties of a parameter also contains the properties of a parameter node.

In the browser a *parameter* is placed under its *activity* and may contains [flows](#). A flow can't directly connect two parameters, a parameter is connected through a [flow](#) to an *expansion node* or an *activity node* ([activity action](#), [object node](#) or [control node](#)) whose must be in the *activity* owing the parameter.

Note : *parameter sets* are defined on actions, see [actions](#).

## Activity Menus

The *activity* menu appearing with a right mouse click on its representation in the browser is something like these, supposing the *activity* not *read-only* nor deleted (see also [menu in a diagram](#)) :

an activity
new activity diagram
add parameter
add interruptible activity region
new expansion region
add activity action
add object node
edit
duplicate
delete
referenced by
mark

### Menu : add ...

These entries of the menu allow to add as [activity diagram](#), a *parameter*, a [region](#) or an activity node ([activity action](#), [object node](#) or [control node](#)). The [activity diagrams](#) nested in a *activity* are the ones for the *activity*, this means that in case you use this [activity](#)

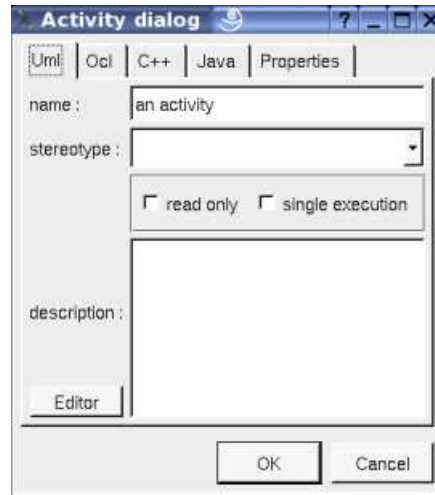
[diagram](#) to create a new [region](#) or an activity node these ones will be a child of the *activity* in the *browser*.

## **Menu : edit**

*edit* allows to show/modify the *activity* properties. In case the *activity* is read-only, the fields of the dialog are also read-only.

### **activity dialog, tab Uml**

The tab *Uml* is a global tab :



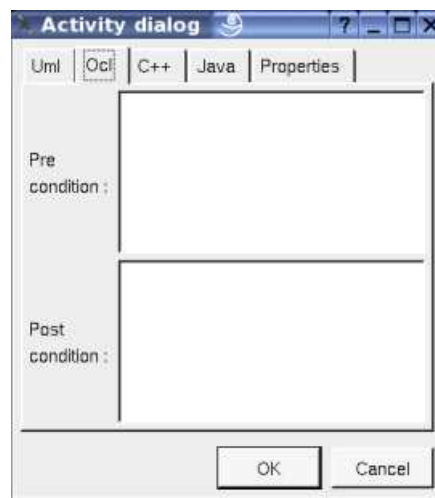
The proposed *stereotypes* are the default one specified through the [Default stereotypes dialog](#) more the current one (empty here). It is possible to choose into the list or to give a new one, or to empty it.

**Read-only** and **single execution** are the standard UML flags.

The *editor* button visible above and associated here to the description, allows to edit the description in an other window, or to call an external editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note that this external editor have to create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel with BOUML which looks each second at the file contents to get the new definition until the dialog is closed (I do not like very much polling but this works even QT isn't compiled with the thread support).

### **activity dialog, tab Ocl**

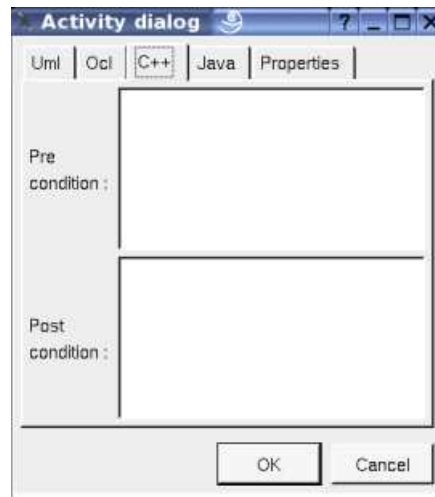
The *Ocl* tab allows to specify the *pre* and *post conditions* using the *Ocl* language, but in fact BOUML doesn't check the syntax, this is just a convention.



### **activity dialog, tab C++**

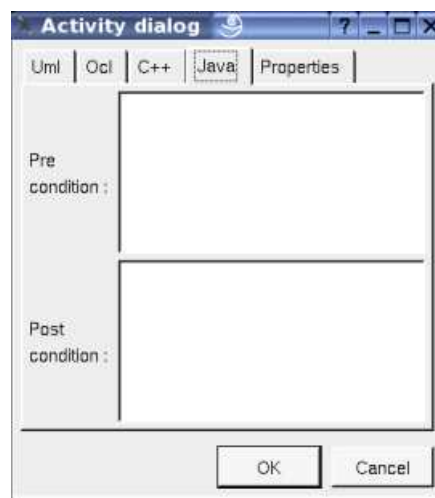
This tab allows to specify the *pre* and *post conditions* in *C++*, but in fact BOUML doesn't check the syntax, this is just a convention.





### activity dialog, tab Java

This tab allows to specify the *pre* and *post* conditions in *Java*, but in fact BOUML doesn't check the syntax, this is just a convention.



### Menu : duplicate

The menu entry *duplicate* clone the *activity* without its children

### Menu : referenced by

To know who reference the current *activity* through a *dependency* or a *call behavior action*.

### Menu : mark

See [mark](#)

### Menu : delete

The menu entry *delete* is only present when the *activity* is not *read-only* (I speak about the file permission, not about the *read-only* UML property)

Delete the *activity* and all its children, and all the representation of them in the opened diagrams. After that it is possible to undelete them (from the browser) until you close the project : obviously the deleted items are not saved !

### Menu : tool

The menu entry *tool* is only present in case at least a [plug-out](#) may be applied on an *activity*. The selected tool is called and applied on the current *activity*.

## Parameter Menus

The *parameter* menu appearing with a right mouse click on its representation in the browser is something like these, supposing the *parameter* not *read-only* nor deleted (see also [menu in a diagram](#)) :

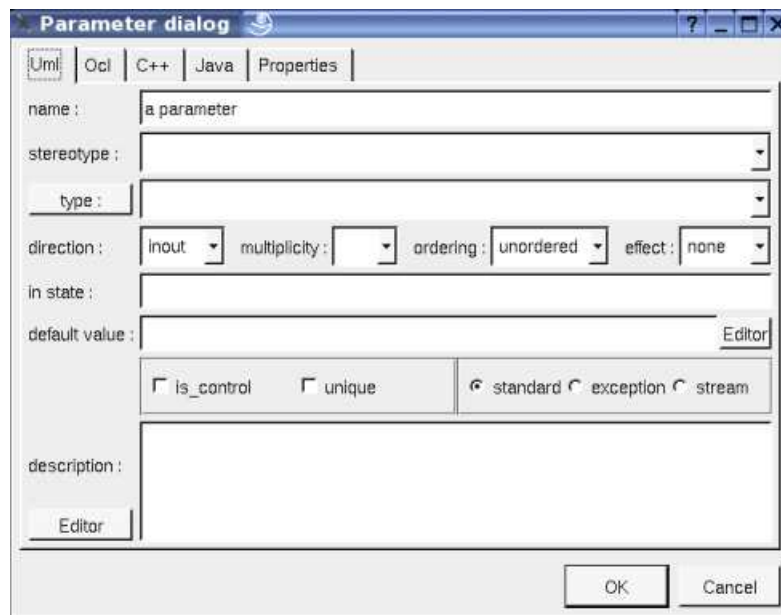


## Menu : edit

*edit* allows to show/modify the *parameter* properties. In case the *parameter* is *read-only*, the fields of the dialog are also *read-only*.

### parameter dialog, tab Uml

The tab *Uml* is a global tab :



The proposed **stereotypes** are the default one specified through the [Default stereotypes dialog](#) more the current one (empty here). It is possible to choose into the list or to give a new one, or to empty it.

The proposed **types** are the *non class* types defined in the first tab of the [generation settings](#), more all the classes defined in the project (with their localization to distinguish synonymous classes). You are not limited to the proposed list, and any form (even invalid for the target language(s)) may be given. The button *type*: shows a menu proposing :

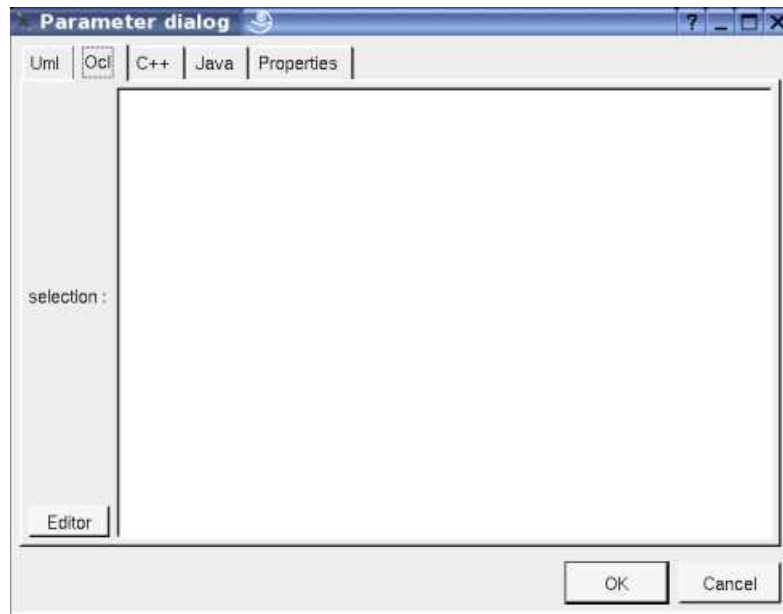
- if the current type of the parameter is a class : to select this class in the browser
- if the parameter is not *read-only* and if a class is selected in the browser : to set the type to be this class

The **direction**, **multiplicity**, **ordering**, **effect**, **in state**, **default value**, **is control**, **unique**, **exception** and **stream** are the standard UML properties. The flags **exception** and **stream** are exclusive, choose **standard** to be not *exception* nor *stream*.

The **editor** button visible above and associated here to the default value and description, allows to edit the default value or the description in other window, or to call an external editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note that this external editor have to create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel with BOUML which looks each second at the file contents to get the new definition until the dialog closed (I do not like very much this polling but this works even QT isn't compiled with the thread support).

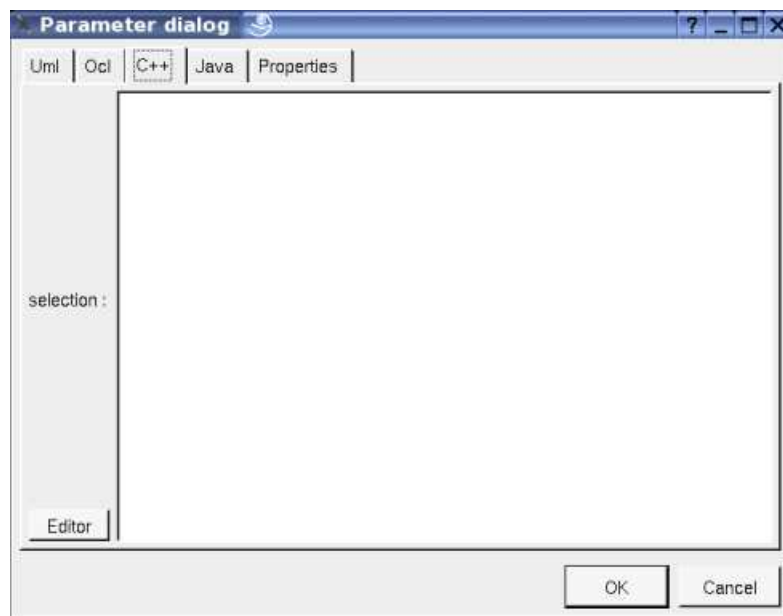
### parameter dialog, tab Ocl

The *Ocl* tab allows to specify the *selection* using the *Ocl* language, but in fact BOUML doesn't check the syntax, this is just a convention.



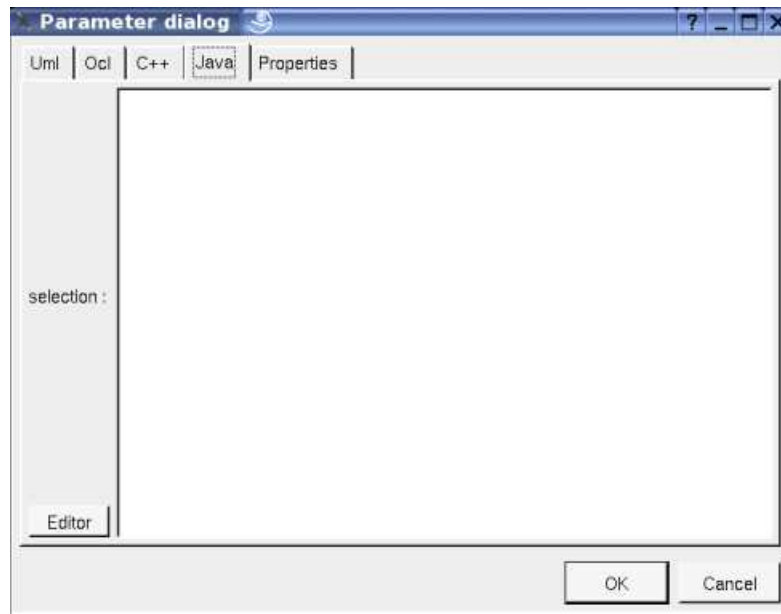
#### parameter dialog, tab C++

This tab allows to specify the *selection* in C++, but in fact BOUML doesn't check the syntax, this is just a convention.



#### parameter dialog, tab Java

This tab allows to specify the *selection* in Java, but in fact BOUML doesn't check the syntax, this is just a convention.



### **Menu : duplicate**

The menu entry *duplicate* clone the *parameter* without its children

### **Menu : referenced by**

To know who reference the current *parameter* through *flow*

### **Menu : mark**

See [mark](#)

### **Menu : delete**

The menu entry *delete* is only present when the *parameter* is not *read-only*

Delete the *parameter* and all its children, and all the representation of them in the opened diagrams. After that it is possible to undelete them (from the browser) until you close the project : obviously the deleted items are not saved !

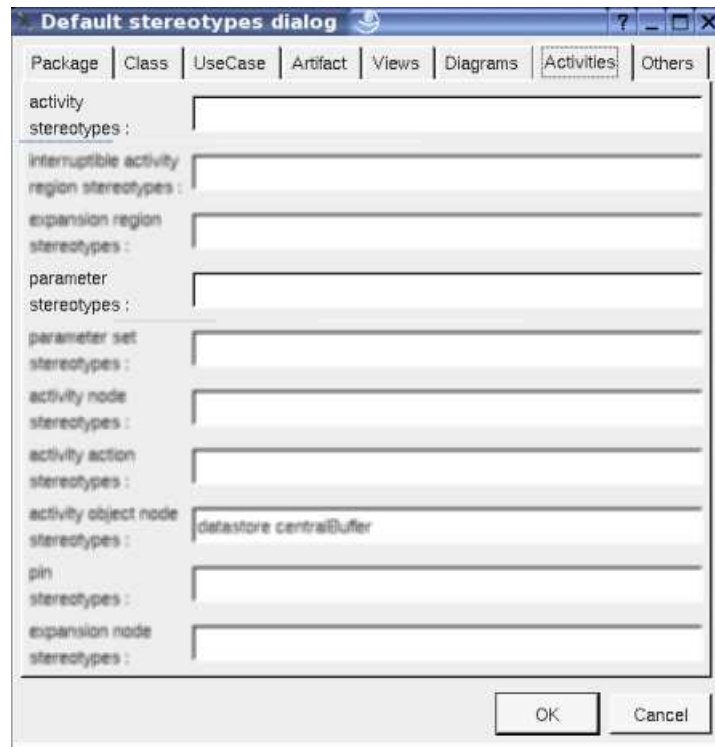
### **Menu : tool**

The menu entry *tool* is only present in case at least a [plug-out](#) may be applied on an *parameter*. The selected tool is called and applied on the current *parameter*.

---

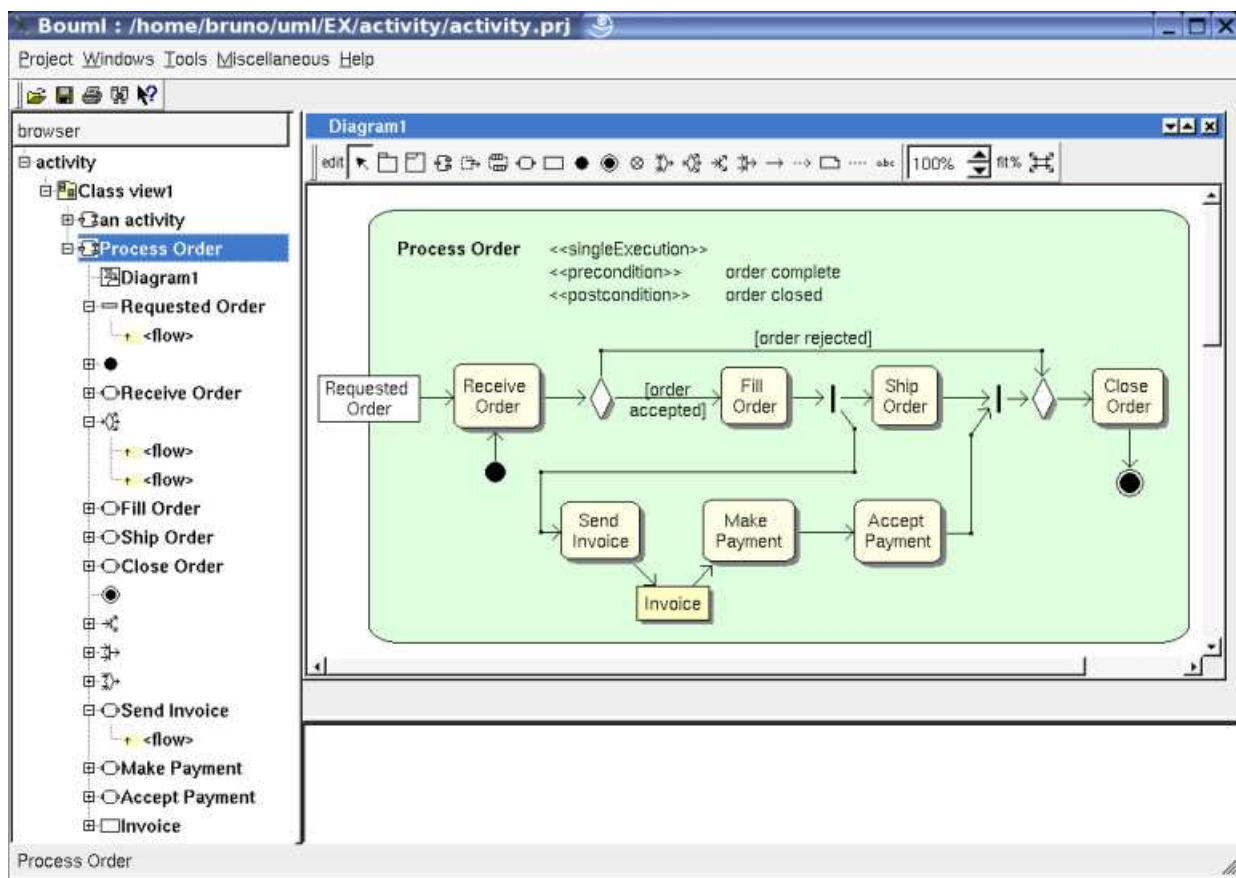
## **Default stereotypes**

The [dialog](#) allowing to set the default activity and parameter stereotypes :



## Activity Drawing

A *activity* is drawn in a diagram as a rectangle with rounded corners :



Note : the activity sub elements are forced to be inside their activity rectangle,. So, when you want to draw in a diagram both an activity and some of its elements I **strongly** recommend to add first the activity then the elements, else when you add the activity the elements may be moved to be inside the rectangle broking their pretty respective positions.

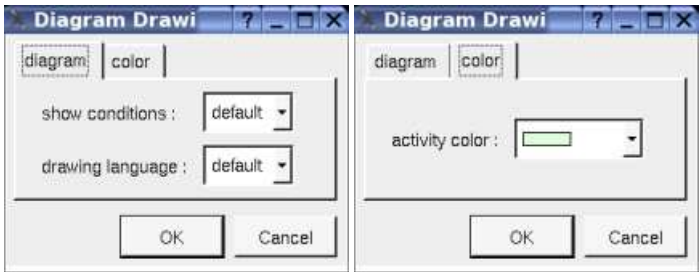
A right mouse click on a *activity* in a diagram calls the following menu (supposing the *activity* editable) :

Process Order
add parameter
upper
lower
edit drawing settings
edit activity
select in browser
set associated diagram
remove from view
delete from model

**Menu : edit drawing settings**

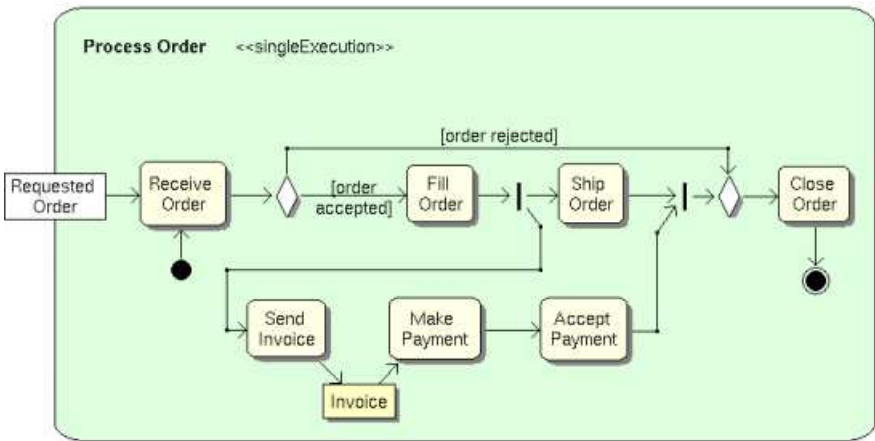
These *drawing settings* concerns only the picture for which the menu is called.

A settings valuing default indicates that the setting specified in the upper level (here the diagram) must be followed, obviously this one may also be default ... up to the project level. When you add a *activity* in a diagram, all the settings are set to *default*.



**show conditions :**

Allows to show or not the *activity's pre* and *post conditions* depending on the drawing language. Note : on an upper level, for instance at the diagram level, this drawing follow the value of drawing *show info notes*. In the diagram above *show conditions* is true, if false the drawing become :



**drawing language :**

Allows to specify the used language in case the *activity's pre* and *post conditions* are shown.

**activity color :**

To specify the fill color.

**Menu : set associated diagram**

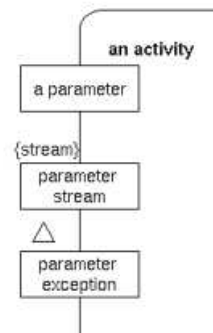
*set associated diagram* allows to automatically open the current diagram when a double mouse click is made on the *activity* representation in a diagram or the browser. After that the only way to edit the *activity* is to choose the *edit* entry in the menu.

**Menu tools :**

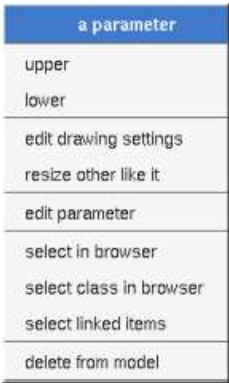
Appears only when at least one *plug-out* is associated to the *activity*s. To apply a *plug-out* on the *activity*.

**Parameter Drawing**

A *parameter node* is drawn in a diagram as a rectangle, with the indication indicating if the parameter is a *stream* or an *exception* :



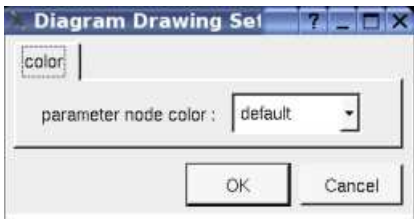
A right mouse click on a *parameter* in a diagram calls the following menu (supposing the *parameter* editable) :



**Menu : edit drawing settings**

These *drawing settings* concerns only the picture for which the menu is called.

A *settings* *valuing* *default* indicates that the setting specified in the upper level (here the diagram) must be followed, obviously this one may also be *default* ... up to the project level. When you add a *parameter* in a diagram, all the settings are set to *default*.



**Menu : select class in browser**

Appears when the type of the parameter is a *class*, to select it in the browser.

Previous : [pseudo state](#)

Next : [activity region](#)

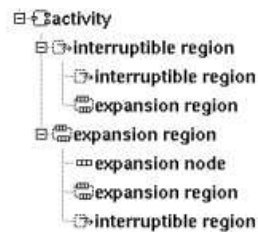


# Activity region

There are two kind of regions : *expansion region* and *interruptible region*, both may be placed under an [activity](#) or a region.

In the *browser* an *expansion region* may contain sub *activity regions*, *expansion nodes* and activity nodes ([activity action](#), [object node](#) or [control node](#))

In the *browser* an *interruptible region* may contain sub *activity regions* and activity nodes ([activity action](#), [object node](#) or [control node](#))



The *region*'s properties are exposed below.

## Expansion region menus

The *expansion region* menu appearing with a right mouse click on its representation in the browser is something like these, supposing the *region* not *read-only* nor deleted (see also [menu in a diagram](#)) :

expansion region
add expansion node
add nested expansion region
add interruptible activity region
add activity action
add object node
edit
duplicate
set it nested in region above
extract it from current parent region
delete
mark

### Menu : add ...

These entries of the menu allow to add an expansion node, a nested region or an activity node ([activity action](#), [object node](#) or [control node](#))

### Menu : edit

*edit* allows to show/modify the *region* properties. In case the *region* is read-only, the fields of the dialog are also read-only.

### Expansion region dialog, tab Uml



The proposed *stereotypes* are the default one specified through the [Default stereotypes dialog](#) more the current one (empty here). It is possible to choose into the list or to give a new one, or to empty it.

The **mode** and the flag **must isolate** are the standard UML information.

The *editor* button visible above and associated here to the description, allows to edit the description in an other window, or to call an external editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note that this external editor have to create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel with BOUML which looks each second at the file contents to get the new definition until the dialog is closed (I do not like very much polling but this works even QT isn't compiled with the thread support).

### Menu : duplicate

The menu entry *duplicate* clone the *region* without its children

### Menu : set it nested in the region above

The menu entry *set it nested in the region above* is present when the previous element in the browser is a *region*, allows to move the *region* inside

### Menu : extract it from current parent region

The menu entry *extract it from current parent region* is present when the *region* is nested in an other one, allows to extract the region and place it in its grand parent..

### Menu : mark

See [mark](#)

### Menu : delete

The menu entry *delete* is only present when the *region* is not *read-only*.

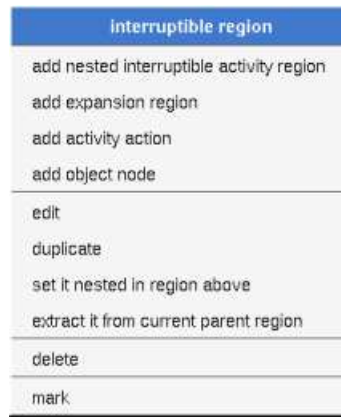
Delete the *region* and all its children, and all the representation of them in the opened diagrams. After that it is possible to undelete them (from the browser) until you close the project : obviously the deleted items are not saved !

### Menu : tool

The menu entry *tool* is only present in case at least a [plug-out](#) may be applied on an expansion region. The selected tool is called and applied on the current *region*.

## Interruptible region menus

The interruptible *region* menu appearing with a right mouse click on its representation in the browser is something like these, supposing the *region* not *read-only* nor deleted (see also [menu in a diagram](#)) :



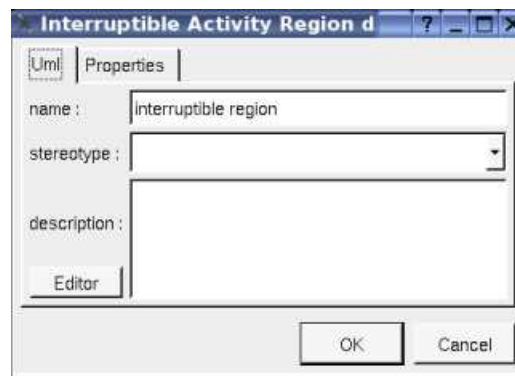
### **Menu : add ...**

These entries of the menu allow to add a nested region or an activity node ([activity action](#), [object node](#) or [control node](#))

### **Menu : edit**

*edit* allows to show/modify the *region* properties. In case the *region* is read-only, the fields of the dialog are also read-only.

### **Interruptible region dialog, tab Uml**



The proposed *stereotypes* are the default one specified through the [Default stereotypes dialog](#) more the current one (empty here). It is possible to choose into the list or to give a new one, or to empty it.

The *editor* button visible above and associated here to the description, allows to edit the description in an other window, or to call an external editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note that this external editor have to create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel with BOUML which looks each second at the file contents to get the new definition until the dialog is closed (I do not like very much polling but this works even QT isn't compiled with the thread support).

### **Menu : duplicate**

The menu entry *duplicate* clone the *region* without its children

### **Menu : set it nested in the region above**

The menu entry *set it nested in the region above* is present when the previous element in the browser is a *region*, allows to move the *region* inside

### **Menu : extract it from current parent region**

The menu entry *extract it from current parent region* is present when the *region* is nested in an other one, allows to extract the region and place it in its grand parent..

### **Menu : mark**

See [mark](#)

### **Menu : delete**

The menu entry *delete* is only present when the *region* is not *read-only*.

Delete the *region* and all its children, and all the representation of them in the opened diagrams. After that it is possible to undelete them (from the browser) until you close the project : obviously the deleted items are not saved !

## Menu : tool

The menu entry *tool* is only present in case at least a [plug-out](#) may be applied on an interruptible region. The selected tool is called and applied on the current *region*.

## Expansion node menus

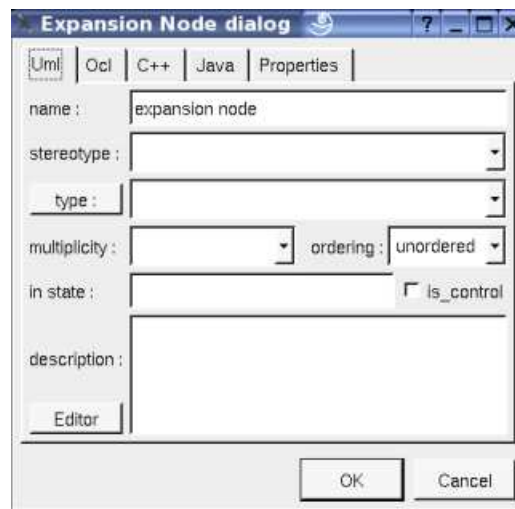
The menu appearing with a right mouse click on its *expansion node* representation in the browser is something like these, supposing the *node* not *read-only* nor deleted (see also [menu in a diagram](#)) :



## Menu : edit

*edit* allows to show/modify the *expansion node* properties. In case the *node* is *read-only*, the fields of the dialog are also *read-only*.

### Expansion node dialog, tab Uml



The proposed *stereotypes* are the default one specified through the [Default stereotypes dialog](#) more the current one (empty here). It is possible to choose into the list or to give a new one, or to empty it.

The proposed *types* are the *non class* types defined in the first tab of the [generation settings](#), more all the classes defined in the project (with their localization to distinguish synonymous classes). You are not limited to the proposed list, and any form (even invalid for the target language(s)) may be given. The button *type*: shows a menu proposing :

- if the current type of the *expansion node* is a class : to select this class in the browser
- if the *expansion node* is not *read-only* and if a class is selected in the browser : to set the type to be this class

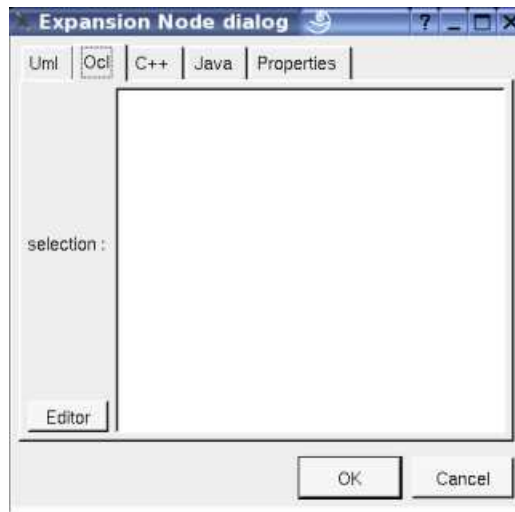
The **multiplicity**, **ordering**, **in state**, and **is control** are the standard UML properties.

The *editor* button visible above and associated here to the description, allows to edit the description in an other window, or to call an external editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note that this external editor have to create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel wi

BOUML which looks each second at the file contents to get the new definition until the dialog is closed (I do not like very much polling but this works even QT isn't compiled with the thread support).

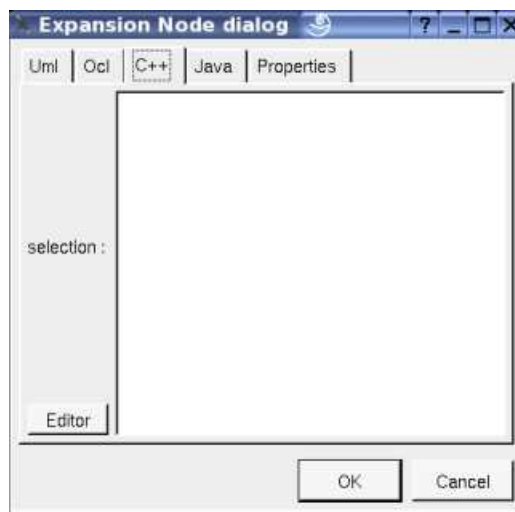
### Expansion node dialog, tab Ocl

The *Ocl* tab allows to specify the *selection* using the *Ocl* language, but in fact BOUML doesn't check the syntax, this is just a convention.



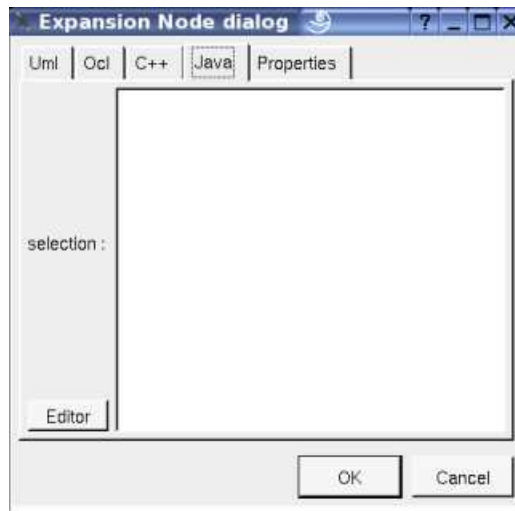
### Expansion node dialog, tab C++

This tab allows to specify the *selection* in *C++*, but in fact BOUML doesn't check the syntax, this is just a convention.



### Expansion node dialog, tab Java

This tab allows to specify the *selection* in *Java*, but in fact BOUML doesn't check the syntax, this is just a convention.



### Menu : duplicate

The menu entry *duplicate* clone the *expansion node* without its children

### Menu : delete

The menu entry *delete* is only present when the *region* is not *read-only*.

Delete the *region* and all its children, and all the representation of them in the opened diagrams. After that it is possible to undelete them (from the browser) until you close the project : obviously the deleted items are not saved !

### Menu : referenced by

To know who reference the current *parameter* through *flow*

### Menu : mark

See [mark](#)

### Menu : tool

The menu entry *tool* is only present in case at least a [plug-out](#) may be applied on an interruptible region. The selected tool is called and applied on the current *region*.

## Default stereotypes

The [dialog](#) allowing to set the default stereotypes :



## Expansion region drawing

An *expansion region* is drawn in a diagram as a dashed rectangle with rounded corners, with the indication of the *mode* :



A right mouse click on a the region in a diagram calls the following menu (supposing the *region* editable) :

expansion region
add expansion node
upper
lower
edit drawing settings
edit expansion region
select in browser
set associated diagram
remove from view
delete from model

### Menu : add expansion node

These entries of the menu allow to add an expansion node.

### Menu : edit drawing settings

These *drawing settings* concerns only the picture for which the menu is called.

A settings valuing default



indicates that the setting specified in the upper level (here the diagram) must be followed, obviously this one may also be *default* ... up to the project level. When you add a *region* in a diagram, all the settings are set to *default*.



To specify the fill color.

**Menu : set associated diagram**

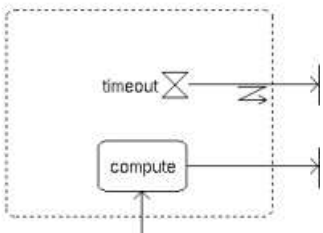
*set associated diagram* allows to automatically open the current diagram when a double mouse click is made on the *region* representation in a diagram or the browser. After that the only way to edit the *region* is to choose the *edit* entry in the menu.

**Menu tools :**

Appears only when at least one *plug-out* is associated to the *regions*. To apply a *plug-out* on the *region*.

**Interruptible region drawing**

An *interruptible activity region* is drawn in a diagram as a dashed rectangle with rounded corners :



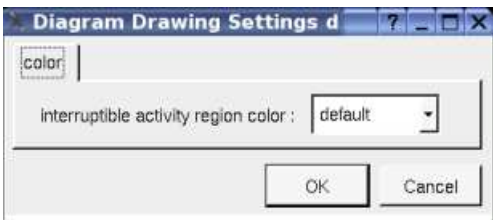
A right mouse click on a *region* in a diagram calls the following menu (supposing the *region* editable) :

interruptible region
upper
lower
edit drawing settings
edit interruptible activity region
select in browser
set associated diagram
remove from view
delete from model

**Menu : edit drawing settings**

These *drawing settings* concerns only the picture for which the menu is called.

A settings valuing default indicates that the setting specified in the upper level (here the diagram) must be followed, obviously this one may also be *default* ... up to the project level. When you add a *region* in a diagram, all the settings are set to *default*.



To specify the fill color.

**Menu : set associated diagram**

*set associated diagram* allows to automatically open the current diagram when a double mouse click is made on the *region* representation in a diagram or the browser. After that the only way to edit the *region* is to choose the *edit* entry in the menu.

**Menu tools :**

Appears only when at least one *plug-out* is associated to the *regions*. To apply a *plug-out* on the *region*.

---

**Expansion node drawing**

A *expansion node* is drawn in a diagram as pins :



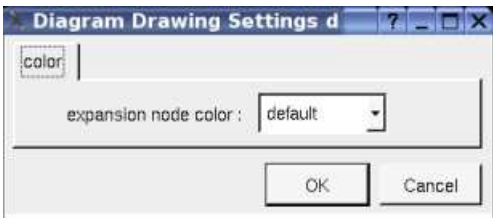
A right mouse click on as *expansion node* in a diagram calls the following menu (supposing the *node* editable) :



**Menu : edit drawing settings**

These *drawing settings* concerns only the picture for which the menu is called.

A *settings* *valuing* *default* indicates that the setting specified in the upper level (here the diagram) must be followed, obviously this one may also be *default ...* up to the project level. When you add a *node* in a diagram, all the settings are set to *default*.



To specify the fill color.

**Menu tools :**

Appears only when at least one *plug-out* is associated to the *expansion nodes*. To apply a *plug-out* on the *node*.

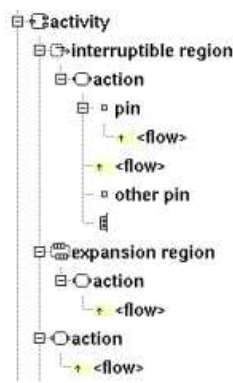
---

Previous : [activity](#)

Next : [activity action](#)

# Activity Actions

An *activity action* is an *activity node*. In the *browser* an *activity action* is placed in an [activity](#) or an [activity region](#), and may contain pins, parameter sets, and [flows](#) :



There are several kinds of *action*, Bouml doesn't propose all the kinds defined in UML but the main ones : *opaque*, *accept event*, *read variable value*, *clear variable value*, *write variable value*, *add variable value*, *remove variable value*, *call behavior*, *call operation*, *send object*, *send signal*, *broadcast signal*, *unmarshall* and *value specification*.

The *pins* are added automatically by Bouml depending on the kind of the action, for the *call behavior* and *call operation* and actions on a variable the pins are set when the *behavior/operation/variable* is set or changed. When something is changed all the current *pins* are removed then new ones may be added. Note that you may change or delete the default *pins* or add new ones, and the *pins* are not changed when the corresponding *behavior/operation/variable* is modified.

The *parameter sets* are defined on the actions and have sense only for *call behavior action* when the *behavior* is an *activity* (but this is not checked by Bouml), so a *parameter set* contains *pins* corresponding to *parameters*. To define *parameter sets* on *action* rather than *activity* allows you to use them as you want.

The *activity action's* and *pin's* properties are exposed below.

## Activity action menus

The *activity* menu appearing with a right mouse click on its representation in the browser is something like these, supposing the *action* not *read-only* nor deleted (see also [menu in a diagram](#)) :

action
add pin
add parameter set
edit
duplicate
delete
referenced by
mark

### [Menu](#) : add ...

To add a *pin* or a *parameter set*.

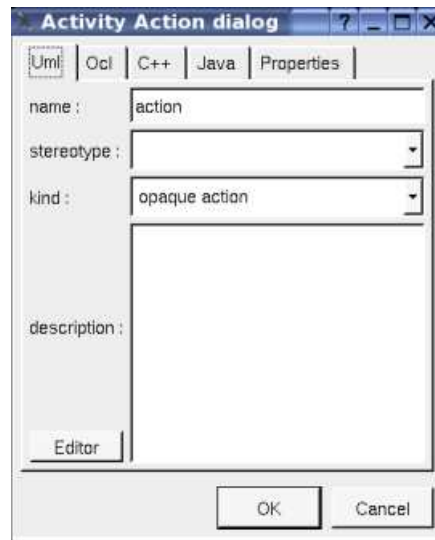
### [Menu](#) : edit

*edit* allows to show/modify the *action* properties. In case the *action* is *read-only*, the fields of the dialog are also *read-only*.

The aspect of the dialog depend on the kind of action :

### Activity opaque action dialog, tab Uml

The tab *Uml* is a global tab :



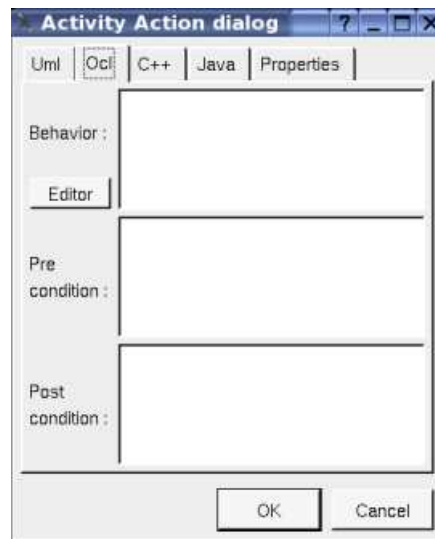
The proposed *stereotypes* are the default one specified through the [Default stereotypes dialog](#) more the current one (empty here). It is possible to choose into the list or to give a new one, or to empty it.

It is possible to change the *kind* of the action, this modify the other editor *tabs*.

The *editor* button visible above and associated here to the description, allows to edit the description in an other window, or to call an external editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note that this external editor have to create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel wi BOUML which looks each second at the file contents to get the new definition until the dialog is closed (I do not like very much polling but this works even QT isn't compiled with the thread support).

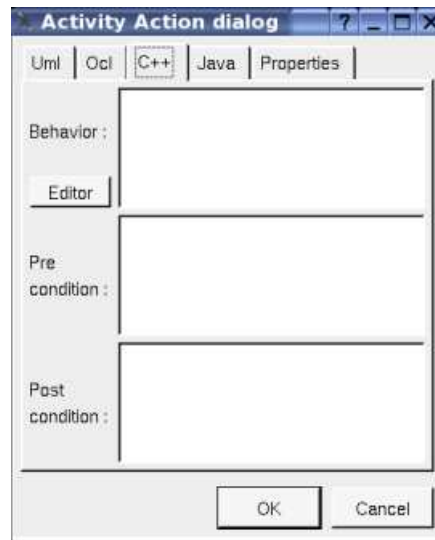
#### Activity opaque action dialog, tab Ocl

The *Ocl* tab allows to specify the *activity action behavior* and the *pre* and *post condition* using the *Ocl* language, but in fact BOUML doesn't check the syntax, this is just a convention.



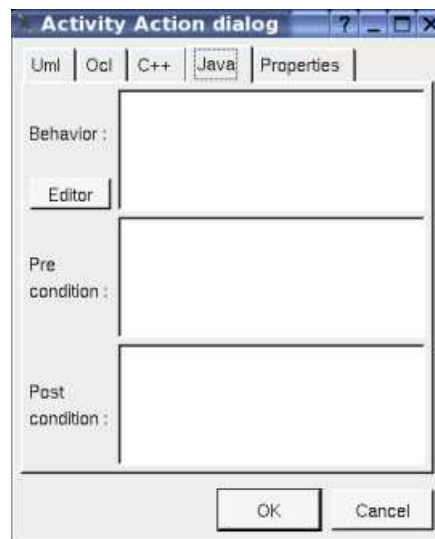
#### Activity opaque action dialog, tab C++

This tab allows to specify the *activity action behavior* and the *pre* and *post condition* in C++, but in fact BOUML doesn't check the syntax, this is just a convention.

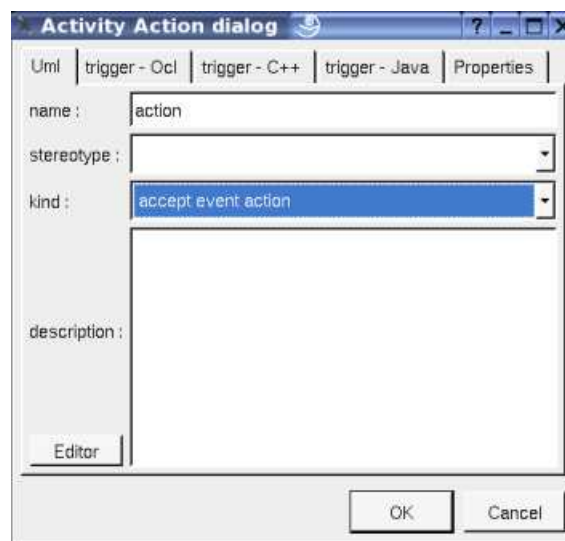


### Activity opaque action dialog, tab Java

This tab allows to specify the *activity action behavior* and the *pre* and *post condition* in *Java*, but in fact BOUML doesn't check the syntax, this is just a convention.

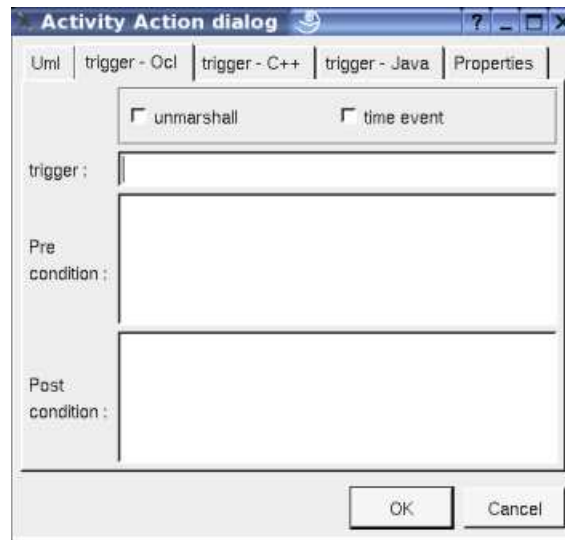


### Activity accept event action dialog, tab Uml



### Activity accept event action dialog, tab Ocl

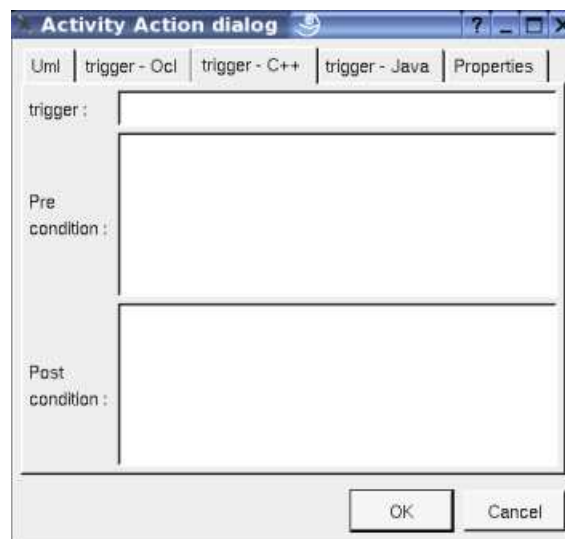
The *Ocl* tab allows to specify the *trigger* and the *pre* and *post condition* using the *Ocl* language, but in fact BOUML doesn't check the syntax, this is just a convention.



**Unmarshall** is the standard UML property, **time event** allows to indicate is the event is a timer to choose between the two representations.

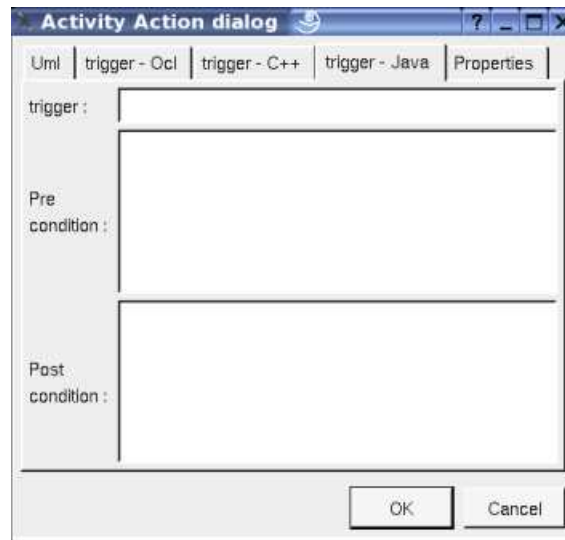
#### Activity accept event action dialog, tab C++

This tab allows to specify the *trigger* and the *pre* and *post condition* in C++, but in fact BOUML doesn't check the syntax, this is just a convention.

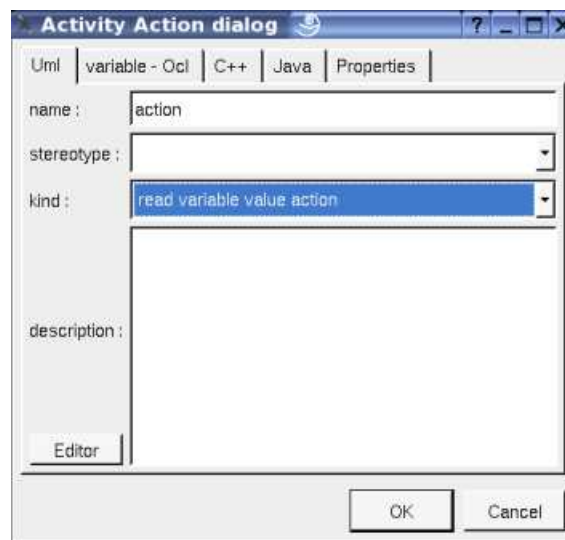


#### Activity accept event dialog, tab Java

This tab allows to specify the *trigger* and the *pre* and *post condition* in Java, but in fact BOUML doesn't check the syntax, this is just a convention.

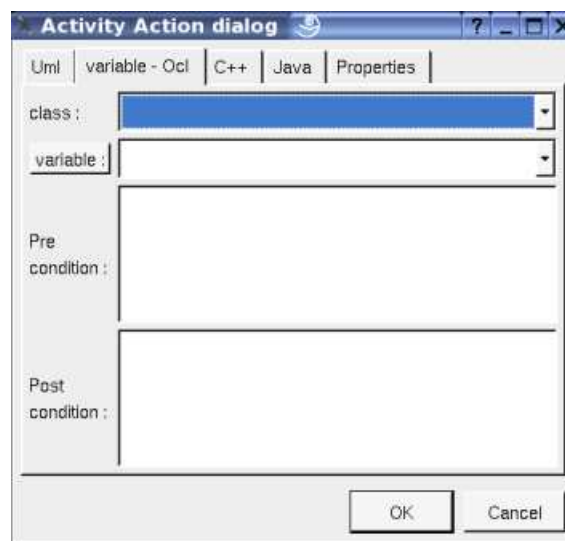


Activity read variable value action dialog, tab Uml



Activity read variable value action dialog, tab Ocl

The *Ocl* tab allows to specify the *variable* (class attribute or relation) and the *pre* and *post condition* using the *Ocl* language, but in fact BOUML doesn't check the syntax, this is just a convention.



The button *variable* shows a menu proposing :

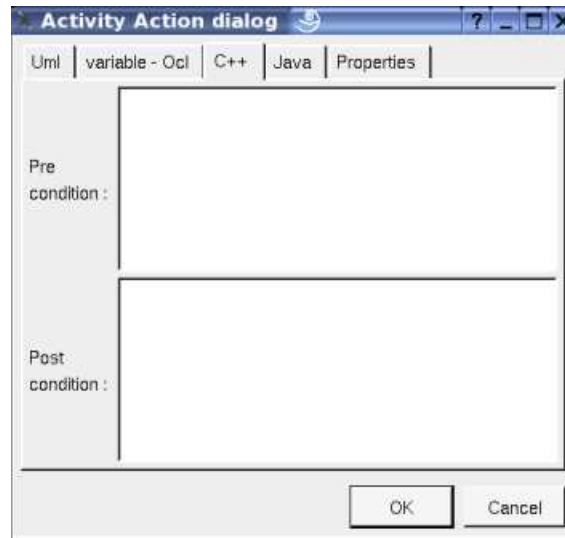
- if the current variable is set : to select this attribute or relation in the browser



- if the action is not read-only and if a class attribute or relation is selected in the browser : to set the variable to be the selected element
- if the class is not read-only : to create a new attribute and to choose it

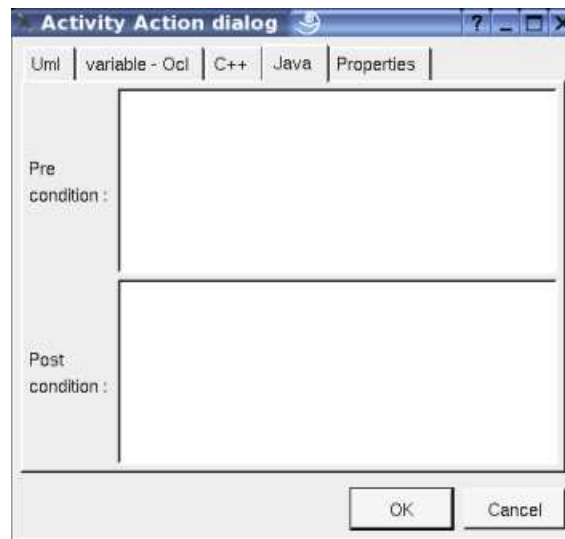
#### Activity read variable value action dialog, tab C++

This tab allows to specify the *pre* and *post condition* in C++, but in fact BOUML doesn't check the syntax, this is just a convention.



#### Activity read variable value action dialog, tab Java

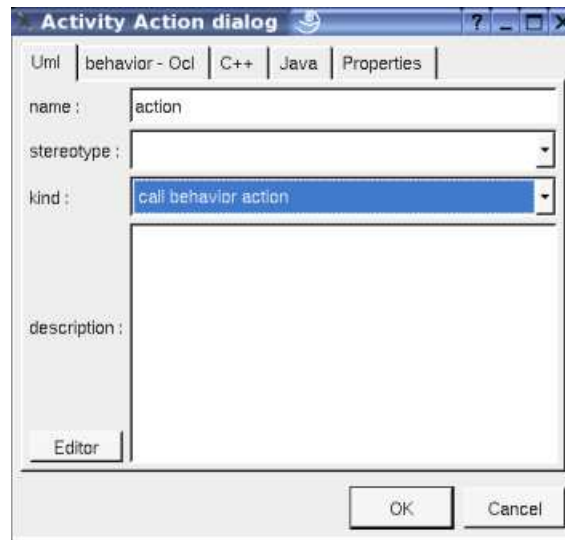
This tab allows to specify the *pre* and *post condition* in Java, but in fact BOUML doesn't check the syntax, this is just a convention.



Activity clear variable value action  
 Activity write variable value action  
 Activity add variable value action  
 Activity remove variable value action

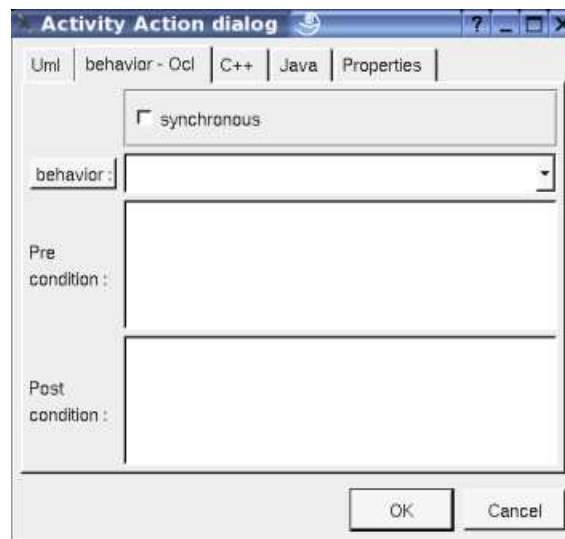
The dialog is equals to the *read variable value action* case

#### Activity call behavior action dialog, tab Uml



#### Activity call behavior action dialog, tab Ocl

The *Ocl* tab allows to specify the *behavior* (a *state* or an *activity*) and the *pre* and *post condition* using the *Ocl* language, but in fact BOUML doesn't check the syntax, this is just a convention.

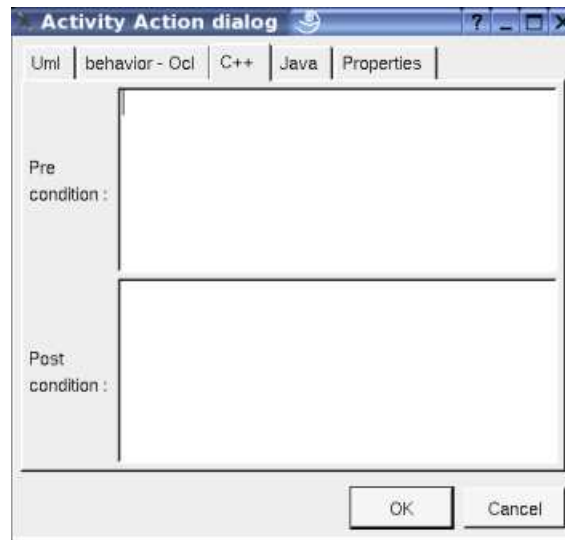


The button *behavior* shows a menu proposing :

- if the current behavior is set : to select this class in the browser
- if the action is not read-only and if a *state* or *activity* is selected in the browser : to set the behavior to be the selected element
- if the view containing the activity is not read-only : to create an activity or a state machine and to choose it

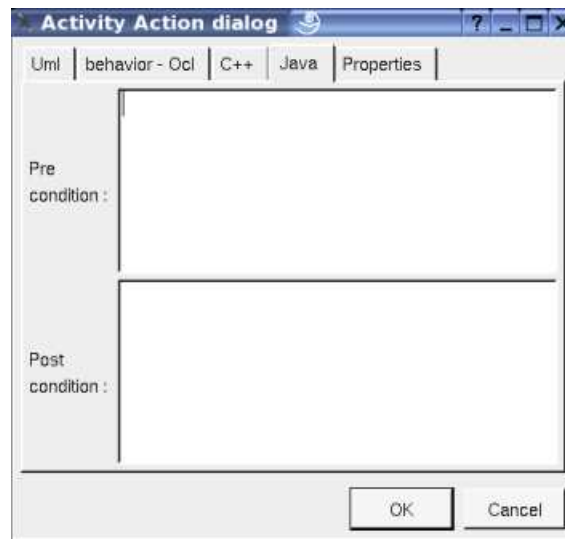
#### Activity call behavior action dialog, tab C++

This tab allows to specify the *pre* and *post condition* in *C++*, but in fact BOUML doesn't check the syntax, this is just a convention.

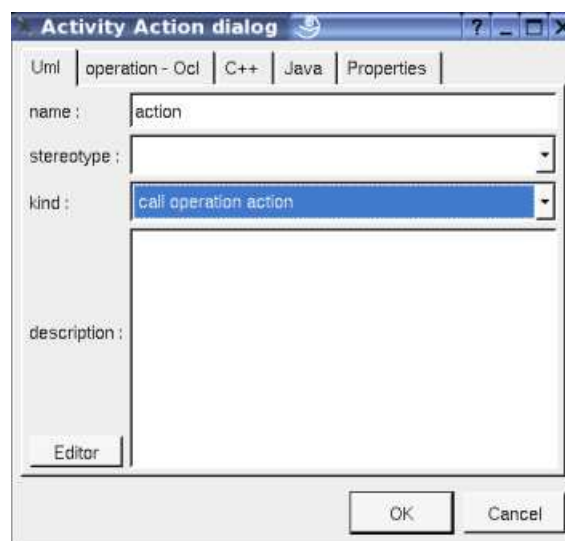


#### Activity call behavior action dialog, tab Java

This tab allows to specify the *pre* and *post condition* in *Java*, but in fact BOUML doesn't check the syntax, this is just a convention.

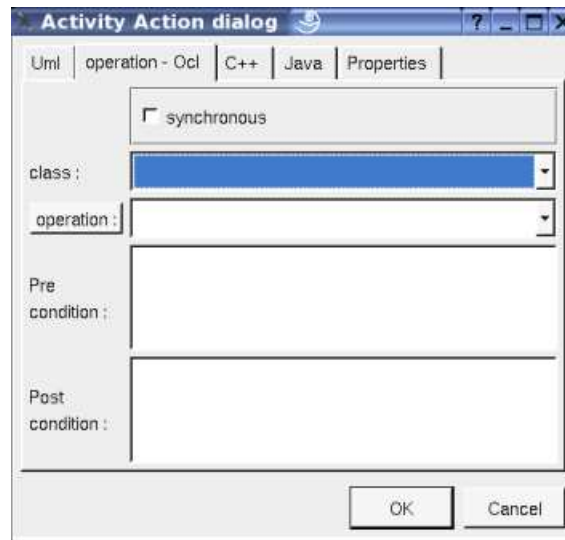


#### Activity call operation action dialog, tab Uml



#### Activity call operation action dialog, tab Ocl

The *Ocl* tab allows to specify the *operation* and the *pre* and *post condition* using the *Ocl* language, but in fact BOUML doesn't check the syntax, this is just a convention.

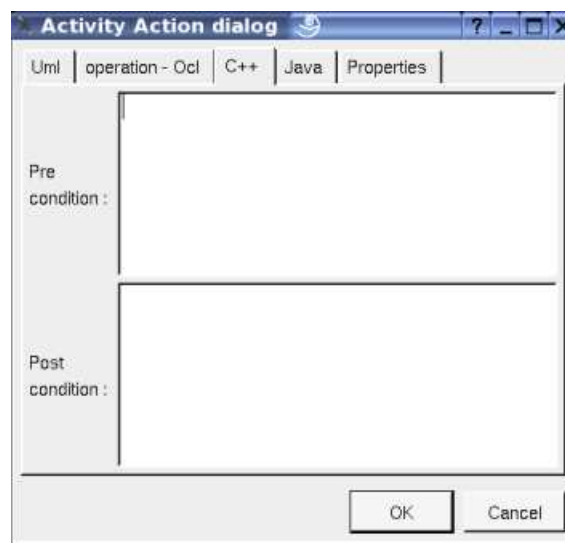


The button *operation* shows a menu proposing :

- if the current operation is set : to select this class in the browser
- if the action is not read-only and if an operation selected in the browser : to set the operation to be the selected one
- if the class is not read-only : to create an operation and to choose it

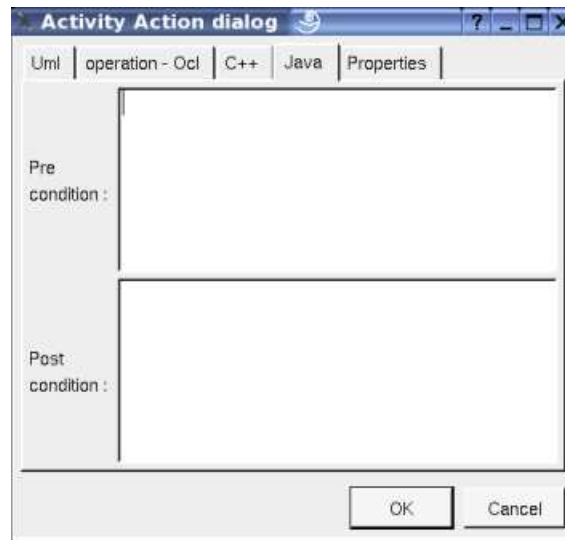
#### Activity call operation action dialog, tab C++

This tab allows to specify the *pre* and *post condition* in C++, but in fact BOUML doesn't check the syntax, this is just a convention.

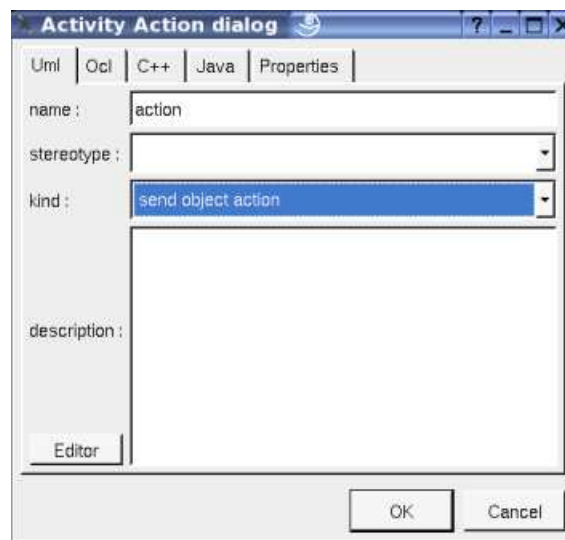


#### Activity call operation action dialog, tab Java

This tab allows to specify the *pre* and *post condition* in Java, but in fact BOUML doesn't check the syntax, this is just a convention.

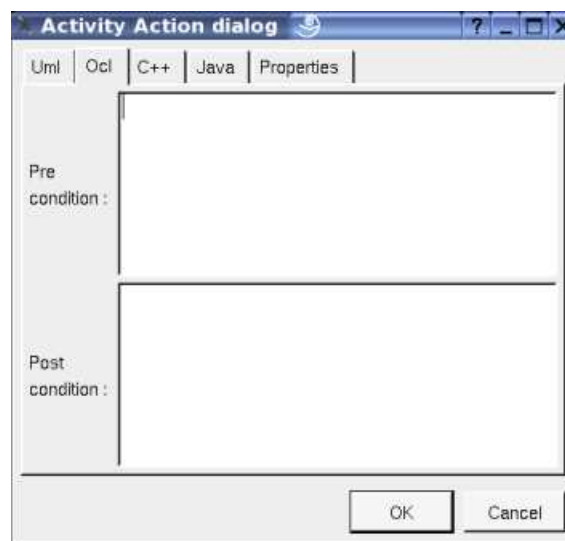


Activity send object action dialog, tab Uml



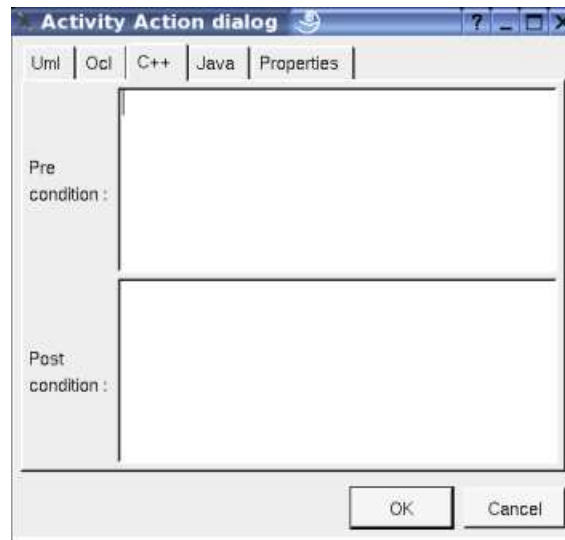
Activity send object action dialog, tab Ocl

The *Ocl* tab allows to specify the *pre* and *post condition* using the *Ocl* language, but in fact BOUML doesn't check the syntax, this is just a convention.



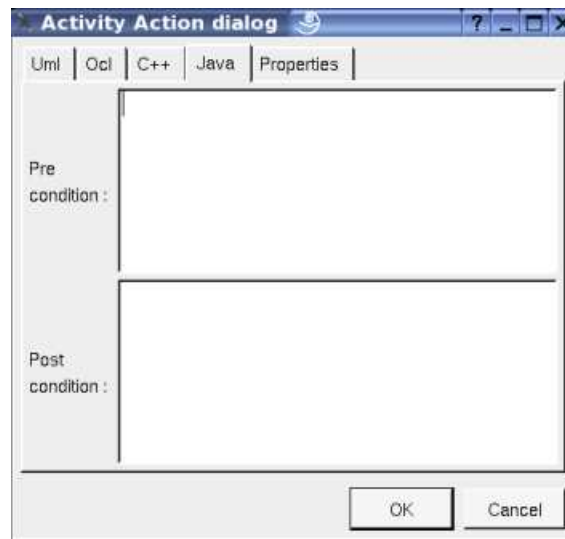
Activity send object action dialog, tab C++

This tab allows to specify the *pre* and *post condition* in *C++*, but in fact BOUML doesn't check the syntax, this is just a convention.



### Activity send object action dialog, tab Java

This tab allows to specify the *pre* and *post* condition in *Java*, but in fact BOUML doesn't check the syntax, this is just a convention.



### Menu : duplicate

The menu entry *duplicate* clone the *activity action* without its children

### Menu : referenced by

To know who reference the current *activity action* through a [flow](#).

### Menu : mark

See [mark](#)

### Menu : delete

The menu entry *delete* is only present when the *activity action* is not *read-only*.

Delete the *activity action* and all its children, and all the representation of them in the opened diagrams. After that it is possible to undelete them (from the browser) until you close the project : obviously the deleted items are not saved !

### Menu : tool

The menu entry *tool* is only present in case at least a [plug-out](#) may be applied on a *activity action*. The selected tool is called and applied on the current *activity action*.

## Pin menus

The *pin* menu appearing with a right mouse click on its representation in the browser is something like these, supposing the *pin* not *read-only* nor *deleted* (see also [menu in a diagram](#)) :

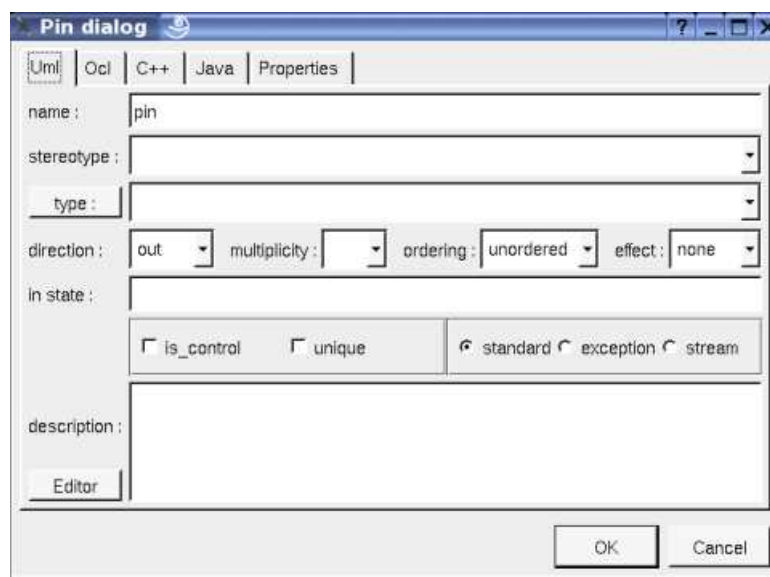


### Menu : edit

*edit* allows to show/modify the *pin* properties. In case the *pin* is *read-only*, the fields of the dialog are also *read-only*.

### Pin dialog, tab Uml

The tab *Uml* is a global tab :



The proposed **stereotypes** are the default one specified through the [Default stereotypes dialog](#) more the current one (empty here). It is possible to choose into the list or to give a new one, or to empty it.

The proposed **types** are the *non class* types defined in the first tab of the [generation settings](#), more all the classes defined in the project (with their localization to distinguish synonymous classes). You are not limited to the proposed list, and any form (even invalid for the target language(s)) may be given. The button *type*: shows a menu proposing :

- if the current type of the pin is a class : to select this class in the browser
- if the pin is not *read-only* and if a class is selected in the browser : to set the type to be this class
- if the view containing the activity is not *read-only* : to create a class and to choose it

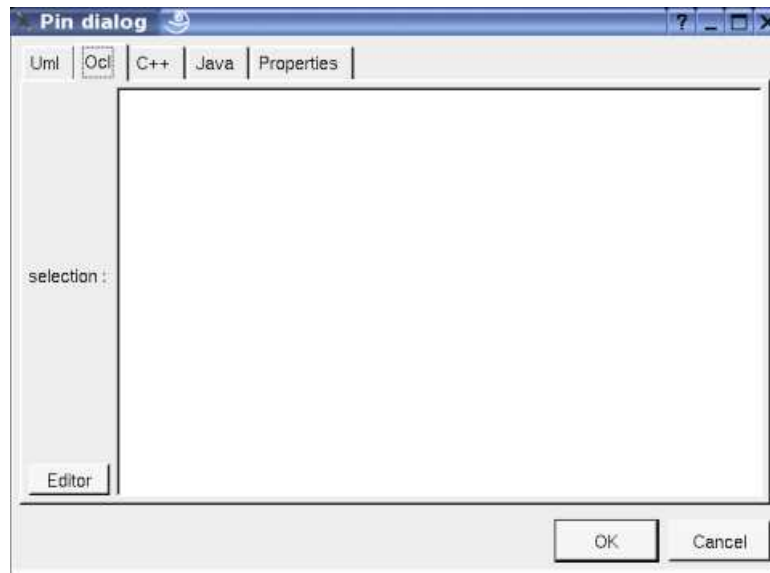
The **direction**, **multiplicity**, **ordering**, **effect**, **in state**, **default value**, **is control**, **unique**, **exception** and **stream** are the standard UML properties. The flags **exception** and **stream** are exclusive, choose **standard** to be not *exception* nor *stream*.

The **editor** button visible above and associated here to the default value and description, allows to edit the default value or the description in other window, or to call an external editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note that this external editor have to create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel with BOUML which looks each second at the file contents to get the new definition until the dialog closed (I do not like very much this polling but this works even QT isn't compiled with the thread support).

### Pin dialog, tab Ocl

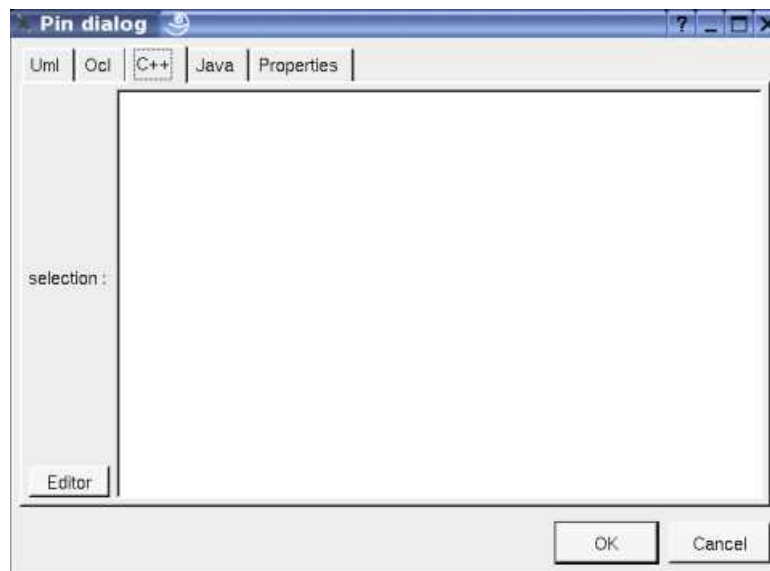
The *Ocl* tab allows to specify the *selection* using the *Ocl* language, but in fact BOUML doesn't check the syntax, this is just a convention.





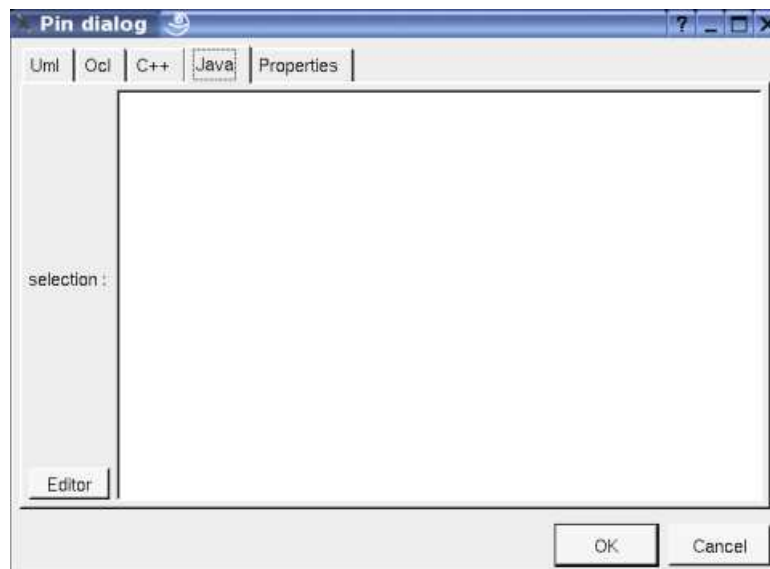
#### Activity opaque action dialog, tab C++

This tab allows to specify the *selection* in *C++*, but in fact BOUML doesn't check the syntax, this is just a convention.



#### Activity opaque action dialog, tab Java

This tab allows to specify the *selection* in *Java*, but in fact BOUML doesn't check the syntax, this is just a convention.



## Menu : duplicate

The menu entry *duplicate* clone the *pin* without its children

## Menu : referenced by

To know who reference the current *pin* through a [flow](#).

## Menu : mark

See [mark](#)

## Menu : delete

The menu entry *delete* is only present when the *pin* is not *read-only*.

Delete the *pin* and all its children, and all the representation of them in the opened diagrams. After that it is possible to undelete them (from the browser) until you close the project : obviously the deleted items are not saved !

## Menu : tool

The menu entry *tool* is only present in case at least a [plug-out](#) may be applied on a *pin*. The selected tool is called and applied on the current *pin*.

## Parameter set menus

The *parameter set* menu appearing with a right mouse click on its representation in the browser is something like these, supposing the *set* not *read-only* nor deleted (see also [menu in a diagram](#)) :

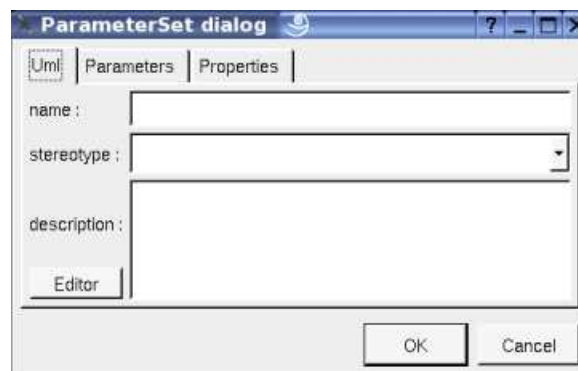


## Menu : edit

*edit* allows to show/modify the *parameter set* properties. In case the *set* is read-only, the fields of the dialog are also read-only.

### Parameter set dialog, tab Uml

The tab *Uml* is a global tab :



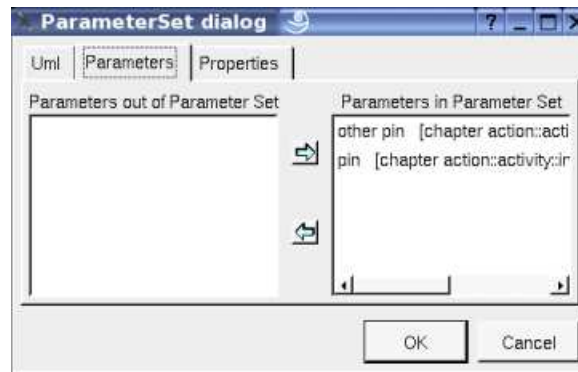
The proposed *stereotypes* are the default one specified through the [Default stereotypes dialog](#) more the current one (empty here). It is possible to choose into the list or to give a new one, or to empty it.

The *editor* button visible above and associated here to the default value and description, allows to edit the default value or the description in other window, or to call an external editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note

that this external editor have to create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel with BOUML which looks each second at the file contents to get the new definition until the dialog closed (I do not like very much this polling but this works even QT isn't compiled with the thread support).

### Pin dialog, tab parameters

Allows to specify the *pins* part of the *parameter set*



### Menu : duplicate

The menu entry *duplicate* clone the *pin* without its children

### Menu : mark

See [mark](#)

### Menu : delete

The menu entry *delete* is only present when the *pin* is not *read-only*.

Delete the *pin* and all its children, and all the representation of them in the opened diagrams. After that it is possible to undelete them (from the browser) until you close the project : obviously the deleted items are not saved !

### Menu : tool

The menu entry *tool* is only present in case at least a [plug-out](#) may be applied on a *parameter set*. The selected tool is called and applied on the current *set*.

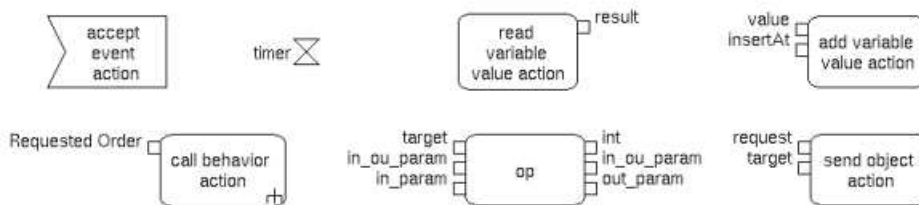
## Default stereotypes

The [dialog](#) allowing to set the default stereotypes :



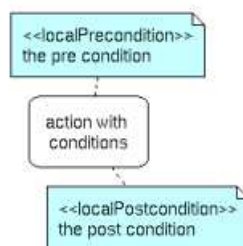
## Activity action drawing

Depending on its stereotype a *activity action* is drawn in a diagram as a rectangle, a convex pentagon or a concave pentagon :



By default the name of an action is its kind, this one is written inside the action, except if the action is an *opaque* action and you ask to see the code.

If the action has *pre* and/or *post condition* these ones are show (except if you ask to not show the *conditions*) :



A right mouse click on a *activity action* in a diagram calls the following menu (supposing the *activity action* editable) :

action	read variable value action	call behavior action	call operation action
add pin	add pin	add pin	add pin
upper	upper	upper	upper
lower	lower	lower	lower
edit drawing settings	edit drawing settings	edit drawing settings	edit drawing settings
edit activity action	edit activity action	edit activity action	edit activity action
select in browser	select in browser	select in browser	select in browser
select linked items	select variable in browser	select behavior in browser	select operation in browser
set associated diagram	set associated diagram	set associated diagram	set associated diagram
remove from view	remove from view	remove from view	remove from view
delete from model	delete from model	delete from model	delete from model

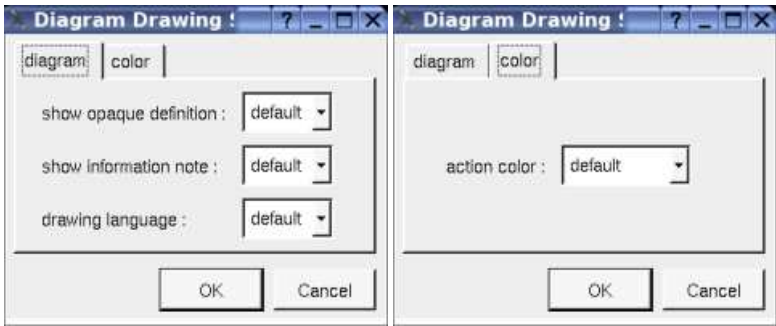
**Menu :add pin**

To add a *pin*

**Menu : edit drawing settings**

These *drawing settings* concerns only the picture for which the menu is called.

A *settings* *valuing* *default* indicates that the setting specified in the upper level (here the diagram) must be followed, obviously this one may also be *default* ... up to the project level. When you add a *activity action* in a diagram, all the settings are set to *default*.



**show opaque definition :**

To write an opaque action's name or its behavior

**show information note :**

to show or not the conditions

**drawing language :**

Allows to specify the used language used to indicate opaque action behavior or the conditions

**action color :**

To specify the fill color.

**Menu select X in browser :**

Appears only when the action is associated to a *variable*, *behavior* or an *operation*, to select this one the the browser

**Menu select associated diagram from behavior :**

Appears only when the action is a *call behavior* and the behavior (an *activity* or a *state*) has an associated diagram, set this last to be the diagram associated to the action, this allows to goes to the diagram associated to the behavior on a double click on the action.

**Menu tools :**

Appears only when at least one *plug-out* is associated to the *activity actions*. To apply a *plug-out* on the *activity action*.

## Pin drawing

A Pin is drawn as a small square, with the stream or exception indication :



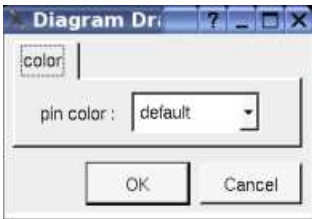
A right mouse click on a *pin* in a diagram calls the following menu (supposing the *pin* editable) :



### Menu : edit drawing settings

These *drawing settings* concerns only the picture for which the menu is called.

A settings valuing default indicates that the setting specified in the upper level (here the diagram) must be followed, obviously this one may also be default ... up to the project level. When you add a *pin* in a diagram, all the settings are set to *default*.



To specify the fill color.

### Menu : select class in browser

Appears when the type of the pin is a class, to select it in the browser.

### Menu tools :

Appears only when at least one *plug-out* is associated to the *pins*. To apply a *plug-out* on the *pin*.

## Parameter set drawing

A *parameter set* is a rectangle drawn around the corresponding *pins*, it is highly recommended to place the pins on the same are to have a pretty result:



A right mouse click on a *pin* in a diagram calls the following menu (supposing the *pin* editable) :



**Menu : edit drawing settings**

These *drawing settings* concerns only the picture for which the menu is called.

A settings valuing *default* indicates that the setting specified in the upper level (here the diagram) must be followed, obviously this one may also be *default* ... up to the project level. When you add a *set* in a diagram, all the settings are set to *default*.



To specify the fill color.

**Menu tools :**

Appears only when at least one *plug-out* is associated to the *parameter sets*. To apply a *plug-out* on the *set*.

---

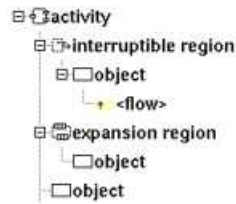
Previous : [activity region](#)

Next : [activity object](#)

# Activity object

Note : *pins* are *activity object*, they are described with the [activity actions](#).

In the *browser* an *activity object* is placed in an [activity](#) or an [activity region](#), and may contain [flows](#) :



The *activity object*'s properties are exposed below.

## Menus

The *activity object* menu appearing with a right mouse click on its representation in the browser is something like these, supposing the *object* not *read-only* nor deleted (see also [menu in a diagram](#)) :

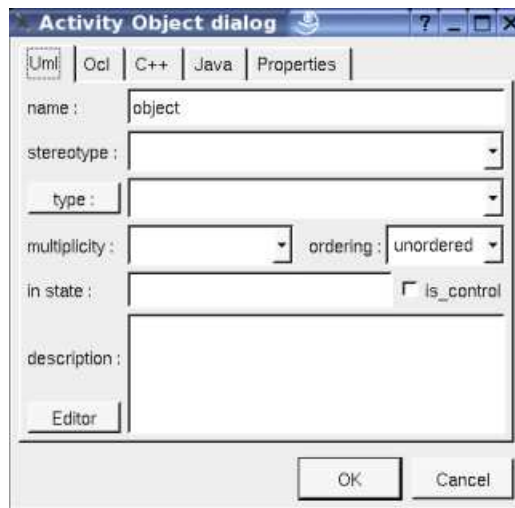


### [Menu](#) : edit

*edit* allows to show/modify the *object* properties. In case the *object* is *read-only*, the fields of the dialog are also *read-only*.

### Activity object dialog, tab Uml

The tab *Uml* is a global tab :



The proposed *stereotypes* are the default one specified through the [Default stereotypes dialog](#) more the current one (empty here). It is possible to choose into the list or to give a new one, or to empty it.

The proposed *types* are the *non class* types defined in the first tab of the [generation settings](#), more all the classes defined in the project (with their localization to distinguish synonymous classes). You are not limited to the proposed list, and any form (even invalid for the target language(s)) may be given. The button *type*: shows a menu proposing :

- if the current type of the object is a class : to select this class in the browser
- if the object is not *read-only* and if a class is selected in the browser : to set the type to be this class

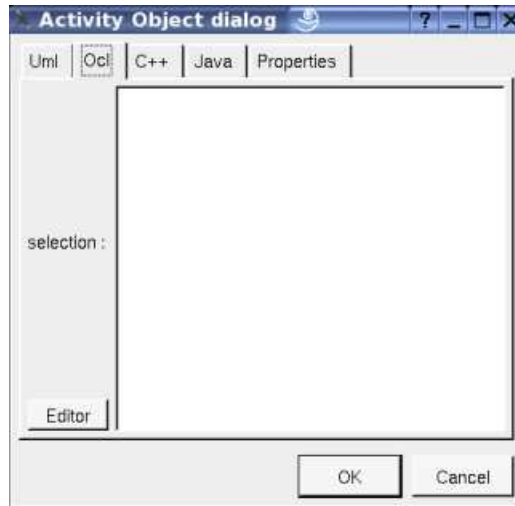


The **multiplicity**, **ordering**, **in state** and **is control** are the standard UML properties.

The *editor* button visible above and associated here to the description, allows to edit the description in an other window, or to call an external editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note that this external editor have to create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel with BOUML which looks each second at the file contents to get the new definition until the dialog is closed (I do not like very much polling but this works even QT isn't compiled with the thread support).

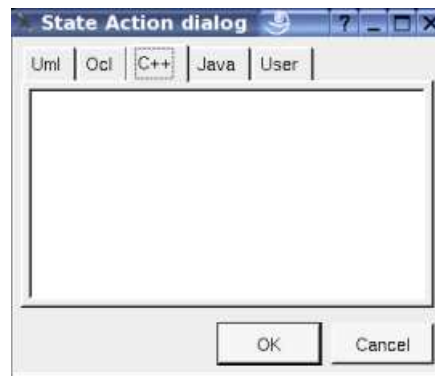
### Activity object dialog, tab Ocl

The *Ocl* tab allows to specify the *selection* using the *Ocl* language, but in fact BOUML doesn't check the syntax, this is just a convention.



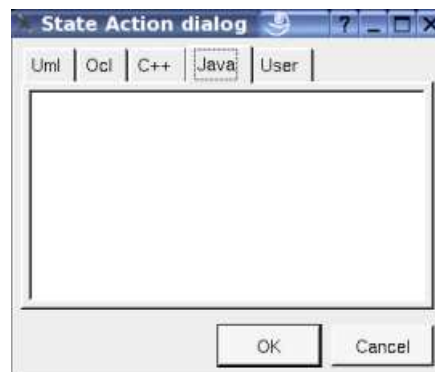
### Activity object dialog, tab C++

This tab allows to specify the *selection* in C++, but in fact BOUML doesn't check the syntax, this is just a convention.



### Activity object dialog, tab Java

This tab allows to specify the *selection* in Java, but in fact BOUML doesn't check the syntax, this is just a convention.



[Menu](#) : duplicate

The menu entry *duplicate* clone the *activity object* without its children

**Menu : referenced by**

To know who reference the current *activity object* through a transition.

**Menu : mark**

See [mark](#)

**Menu : delete**

The menu entry *delete* is only present when the *activity object* is not *read-only*.

Delete the *activity object* and all its children, and all the representation of them in the opened diagrams. After that it is possible to undelete them (from the browser) until you close the project : obviously the deleted items are not saved !

**Menu : tool**

The menu entry *tool* is only present in case at least a *plug-out* may be applied on a *activity object*. The selected tool is called and applied on the current *activity object*.

---

## Default stereotypes

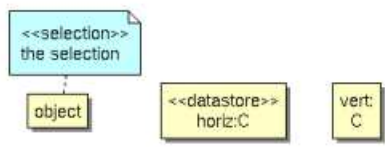
The [dialog](#) allowing to set the default stereotypes :



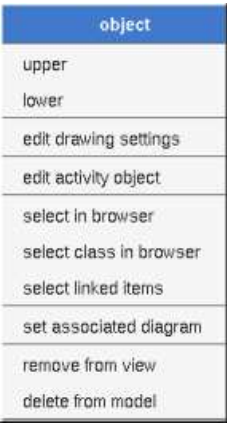
---

## Drawing

An *activity object* is drawn in a diagram as a rectangle, with its stereotype, its type (if set) and its selection (except if you ask to not show it) :



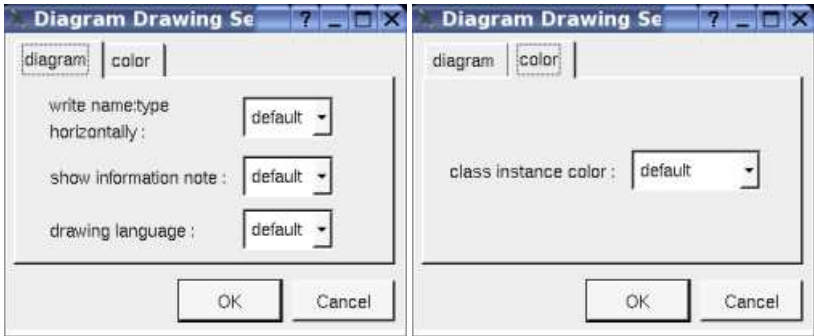
A right mouse click on a *activity object* in a diagram calls the following menu (supposing the *activity object* editable) :



**Menu : edit drawing settings**

These *drawing settings* concerns only the picture for which the menu is called.

A settings valuing default indicates that the setting specified in the upper level (here the diagram) must be followed, obviously this one may also be default ... up to the project level. When you add a *activity object* in a diagram, all the settings are set to *default*.



**write name:type horizontally:**

usefull when the object has a name and a type, to write them on the same line of on two lines

**show information note :**

to show or not the selection

**drawing language :**

Allows to specify the used language showing the expression

**Class instance color :**

To specify the fill color.

**Menu tools :**

Appears only when at least one *plug-out* is associated to the *activity objects*. To apply a *plug-out* on the *activity object*.

Next : [activity control node](#)

# Activity control node

The *activity control nodes* are :

- ● initial
- ● final
- ⊗ flow final
- ↗ merge
- ✕ decision
- ↘ fork
- ↙ join

In the *browser* a *activity control node* is placed under its [activity](#) or [region](#) and may contain *flows* (when this is legal).

The *activity control node's* properties will be exposed below.

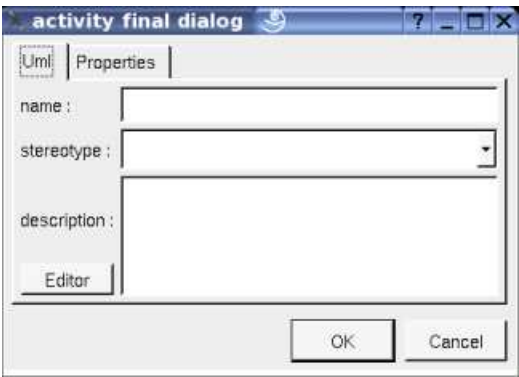
## Menus

The *activity control node* menu appearing with a right mouse click on its representation in the browser is something like these, supposing the *activity control node* not *read-only* nor deleted (see also [menu in a diagram](#)) :



### Menu : edit

*edit* allows to show/modify the *activity control node* properties. In case the *activity control node* is *read-only*, the fields of the dialog are also *read-only*.



The proposed *stereotypes* are the default one specified through the [Default stereotypes dialog](#) more the current one (empty here). It is possible to choose into the list or to give a new one, or to empty it.

The *editor* button visible above and associated here to the description, allows to edit the description in an other window, or to call an external editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note that this external editor have to create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel wi BOUML which looks each second at the file contents to get the new definition until the dialog is closed (I do not like very much polling but this works even QT isn't compiled with the thread support).

### Menu : duplicate

The menu entry *duplicate* clone the *activity control node* without its children

### **Menu : referenced by**

To know who reference the current *activity control node* through a transition.

### **Menu : mark**

See [mark](#)

### **Menu : delete**

The menu entry *delete* is only present when the *activity control node* is not *read-only*.

Delete the *activity control node* and all its children, and all the representation of them in the opened diagrams. After that it is possible to undelete them (from the browser) until you close the project : obviously the deleted items are not saved !

### **Menu : tool**

The menu entry *tool* is only present in case at least a [plug-out](#) may be applied on a *activity control node* of this kind. The selected tool is called and applied on the current *activity control node*.

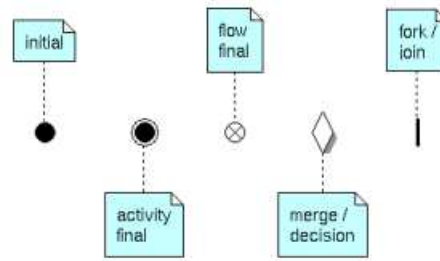
## **Default stereotypes**

The [dialog](#) allowing to set the default stereotypes :



## **Drawing**

A *activity control node* is drawn in a diagram depending on its kind :



A right mouse click on a *activity control node* in a diagram calls the following menu (supposing the *activity control node* editable) :

merge
upper
lower
edit activity node
select in browser
select linked items
remove from view
delete from model

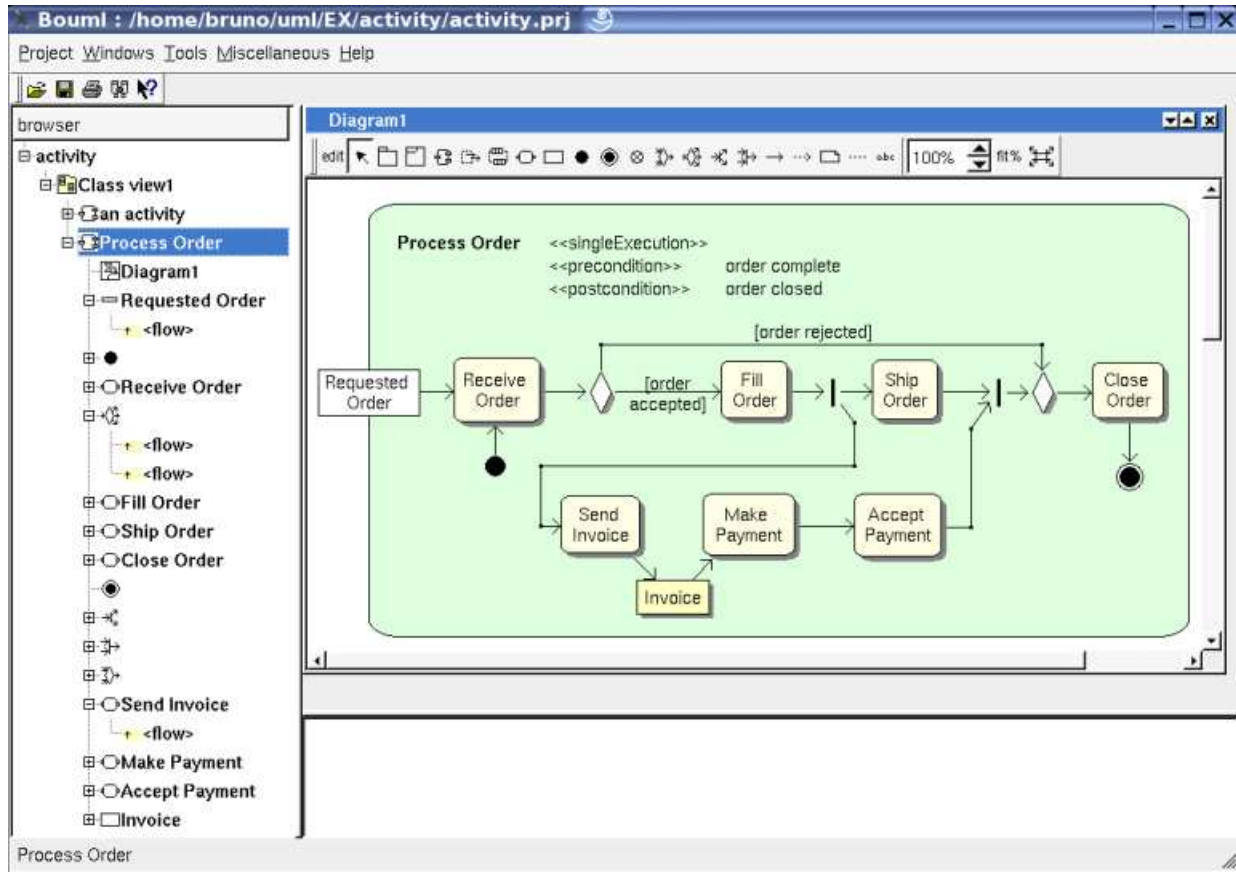
Previous : [activity object](#)

Next : [flow](#)

# Flow

We consider in this chapter both the *control flows* and the *object flows*, there are not distinguished by a given flag but by their usage. When you try to create a *flow* through a diagram Bouml checks the validity of the flow, for instance you can't have a *flow* starting from an *output pin*, but you may after edit the extremities of the *flow*, producing an invalid case, you are responsible of that, perhaps the non consistent *flow* is temporary. To add a flow you have to use an [activity diagram](#) (obviously the diagram or the flow drawing may be deleted just after !), it is not possible to add a flow through the browser. It is also not possible to move a flow out of its start or end container in the browser.

In the browser are written the name of the flows (by default <<flow>>), the stereotype may be may be showed/hidden through the [Miscellaneous menu](#).



## Menus

The flow menu appearing with a right mouse click on its representation in the browser is something like these, supposing it is not *read-only* nor deleted (see also [menu in a diagram](#)) :

<<flow>>
edit
delete
select decision
mark

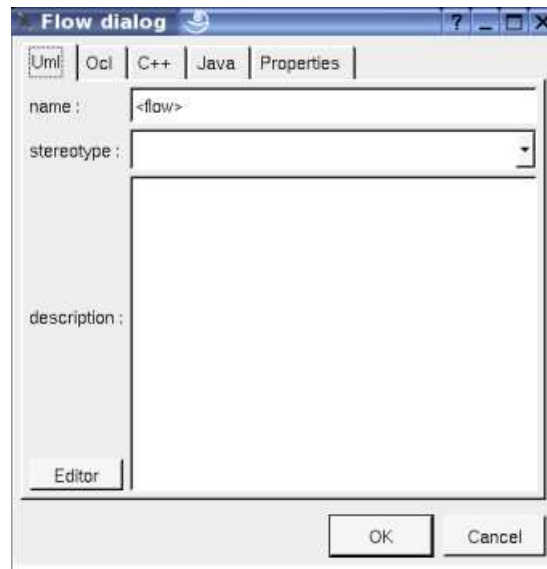
### [Menu](#) : edit

*edit* allows to show/modify the flows properties. In case the flow is *read-only*, the fields of the dialog are also *read-only*.

### Flow dialog, tab Uml

The tab *Uml* is a global tab, independent of the language :



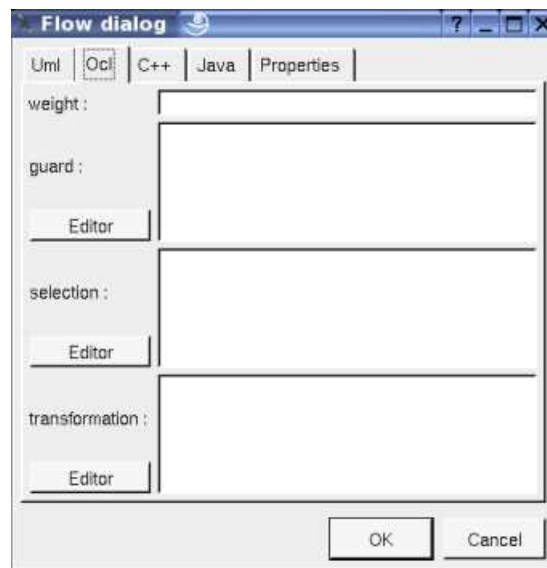


The proposed *stereotypes* are the default ones specified through the [Default stereotypes dialog](#) more the current one (empty here). It is possible to choose into the list or to give a new one, or to empty it. The stereotype and the multiplicity are used both (may be with the stereotype of the class(es) in Idl) to compute the default declaration.

The *editor* button visible above and associated here to the description and initial value, allows to edit the description or initial value in an window, or to call an external editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note that this external editor have to create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel with BOUML which looks each second at the file contents to get the new definition until the dialog is closed (I do not like very much this polling but this works even QT isn't compiled with the thread support).

#### Flow dialog, tab Ocl

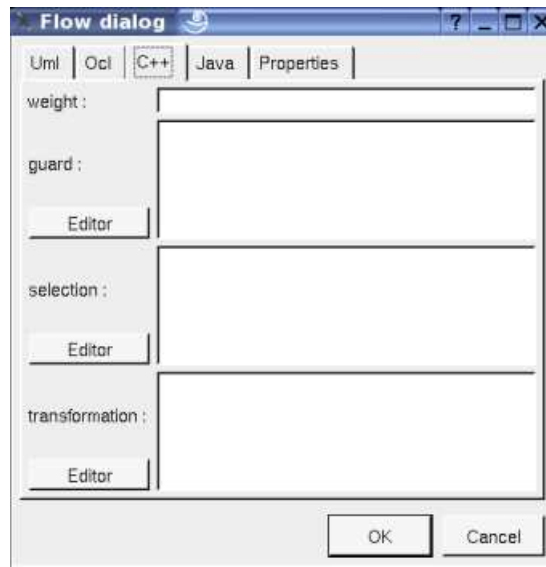
This tab allows to give the Ocl definition of the flow, but in fact BOUML doesn't check the syntax, this is just a convention.



The **weight**, **guard**, **selection** and **transformation** are the standard UML properties of a flow.

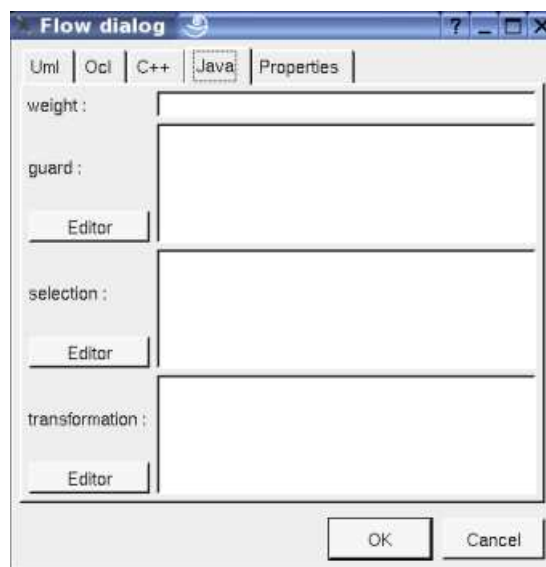
#### Flow dialog, tab C++

This tab allows to give the C++ definition of the flow, but in fact BOUML doesn't check the syntax, this is just a convention.



### Flow dialog, tab Java

This tab allows to give the Java definition of the flow, but in fact BOUML doesn't check the syntax, this is just a convention.



### Menu : delete

The menu entry *delete* is only present when the flow is not *read-only*.

Delete the flow, associated get and set operations when they exist, and all the representation of them in the opened diagrams. After that it is possible to undelete them (from the browser) until you close the project : obviously the deleted items are not saved !

### Menu : select X

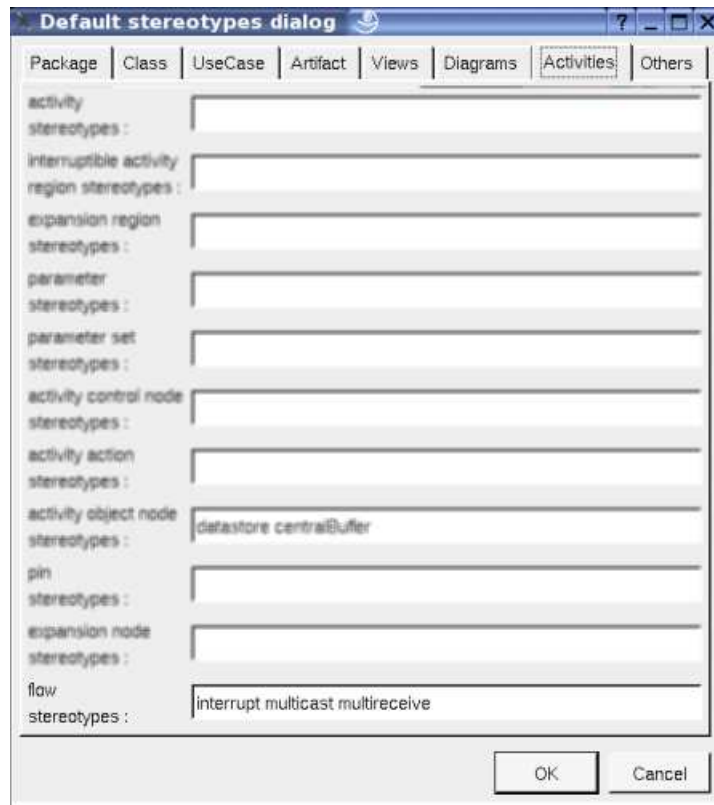
To select the target of the *flow*.

### Menu : mark

See [mark](#)

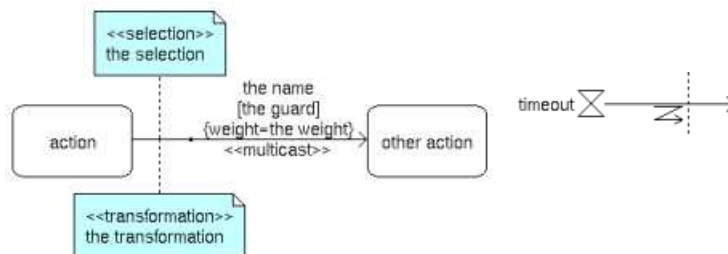
## Default stereotypes

The [dialog](#) allowing to set the default stereotypes has a tab reserved for the flows :



## Drawing

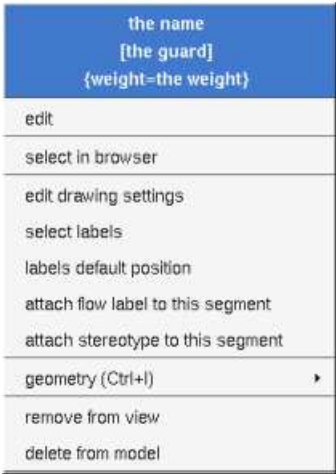
A flow is drawn in a diagram with a line may be with the associated labels : flow's name, stereotypes (the stereotype *interrupt* is specially managed and allows to show a *zigzag*), the weight, the *guard*, the *selection* and *transformation*, depending on the *drawing settings* :



To add a new flow between elements, select the flow through the buttons on the top of the diagram sub window, click on the start element with the left mouse button, move the mouse, each time the mouse button is raised a new line break is introduced, at least click on the end element. To abort a flow construction press the right mouse button or do a double click with the left mouse button.

A line may be broken, during its construction or after clicking on the line with the left mouse button and moving the mouse with mouse button still pushed. To remove a line break, a double click on the point with the left mouse button is enough, or use the line break menu of the point using the right mouse button.

A right mouse click on a flow in a diagram or a double click with left mouse button calls the following menu (supposing the flow is editable) :



**Menu : edit**

To edit the flow

**Menu : select labels**

To select the labels (flow's name, stereotypes) associated to the flow. Useful when you are lost after many label movings.

**Menu : labels default position**

To place the labels (flow's name, stereotypes, role(s) and multiplicity(ies)) associated to the flow in the default position. Useful wh you are lost after many label movings.

By default when you move a class or a flow point break or edit the flow, the associated labels are moved to their default position, this may be irritating. To ask BOUML to not move the associated labels on the flows in a diagram, use the drawing settings of the diagrams.

**Menu : geometry**

Allows to choose one of the line geometry :



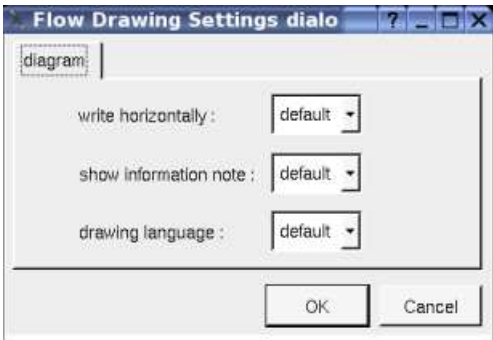
Note : if you manually move the central line of the last two geometries this one stop to be automatically updated when you move one of the two extremities of the flow.

**Menu : tools**

Appears only when at least one plug-out is associated to the flow. To apply a *plug-out* on the flow.

**Drawing settings**

The drawing settings associated to the flows in a diagram may be set through the the diagram itself or one of its parent, for instance :



A **settings** **valuing** *default*  
indicates that the setting specified in the upper level (here the diagram) must be followed, obviously this one may also be *default* ... up to the project level.

**write horizontally :**

To made the label indicating the name, *weight* and *guard* on one line or several.

**show information note :**

To show or not the *selection* and *transformation*.

**Drawing language :**

To choose the language used to write the *weight*, *guard*, *selection* and *transformation*.

---

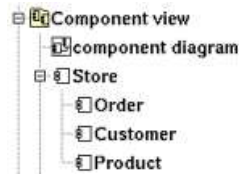
Previous : [activity control node](#)

Next : [component](#)

# Component

A component is an autonomous unit with well defined interfaces that is replaceable within its environment. In the old release BOUML (up to the 1.5.1), the behavior of the [artifact](#) was defined at the component level.

A component may be placed in a [component view](#) or is nested in another component



A component may be created through the *new component* entry of the [component view menu](#) or through the component button of a [component diagram](#).

## Browser Menu

The component menu appearing with a right mouse click on its representation in the browser is something like these, suppos component not *read-only* nor deleted (see also [menu in a diagram](#)) :

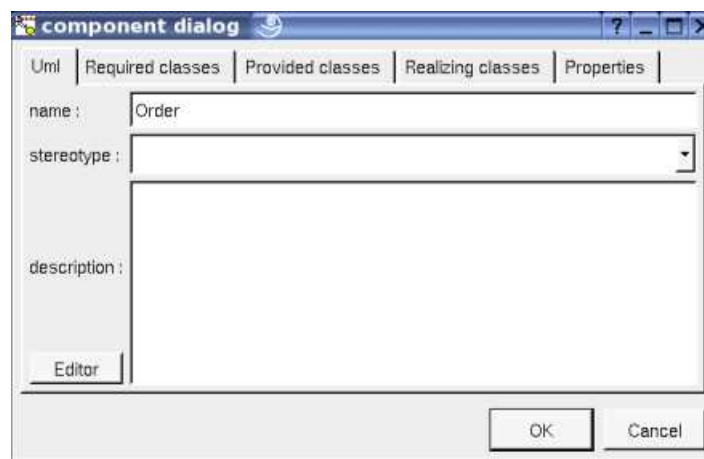


### [Menu](#) : edit

*edit* allows to show/modify the component properties. In case the component is read-only, the fields of the dialog are also read-only.

### Component dialog, tab Uml

The tab *Uml* is a global tab, independent of the language :



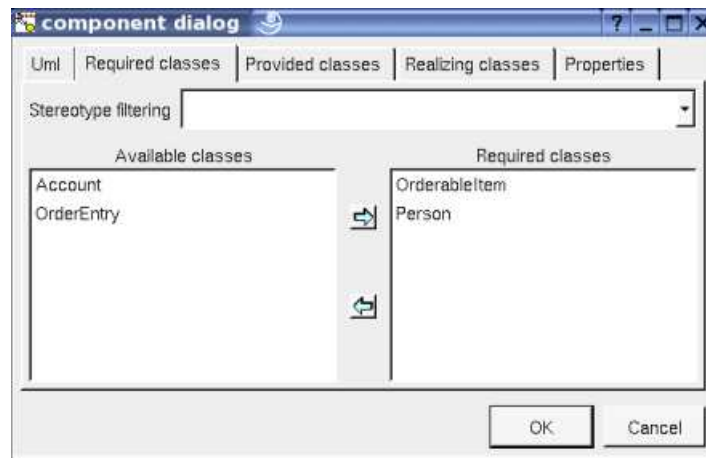
The proposed *stereotypes* are the default one specified through the [Default stereotypes dialog](#) more the current one (empty here). It is possible to choose into the list or to give a new one, or to empty it.

The *editor* button visible above and associated here to the description, allows to edit the description in an other window, or to call an extern:

editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note that this external editor have to create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel with BOUML which looks each second at the file contents to get the new definition until the dialog is closed (I do not like very much polling but this works even QT isn't compiled with the thread support).

### Component dialog, tab Required classes

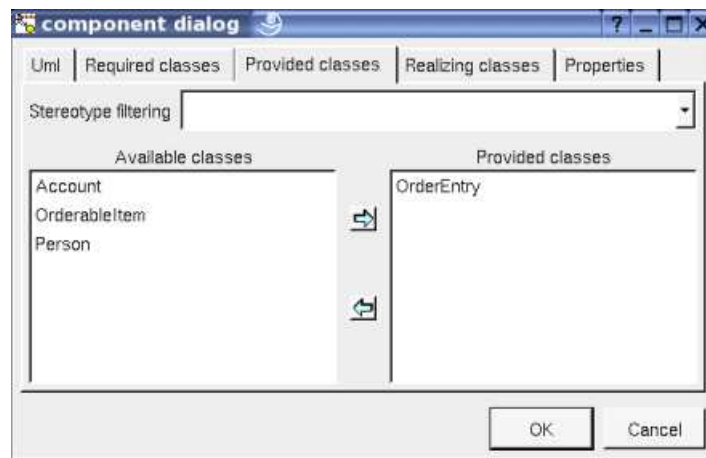
This tab allows to specify the list of interface the component requires.



When a component C contains nested components, it is useless to repeat the list of required interfaces at the level of C already defined by the sub components, the right list will be provided when you will add  $\neg C$  or  $\neg O$ .

### Component dialog, tab Provided classes

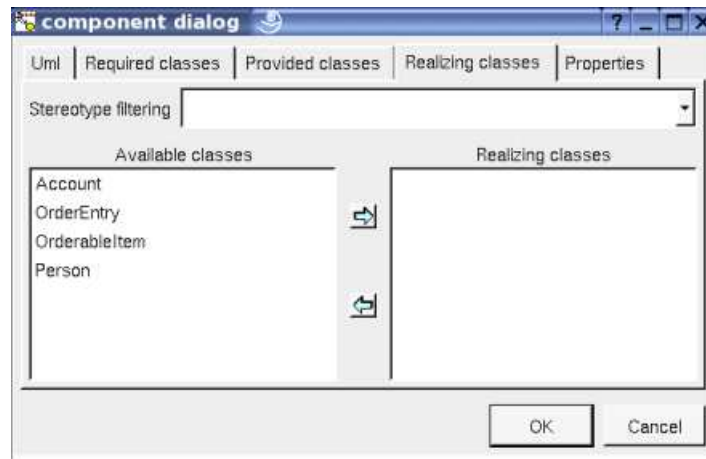
This tab allows to specify the list of interface the component provides.



When a component C contains nested components, it is impossible to repeat the list of provided/realized interfaces at the level of C already defined by the sub components, the right list will be provided when you will add  $\neg C$  or  $\neg O$ .

### Component dialog, tab Realizing classes

This tab allows to specify the list of classes realizing the behavior of the component.



### **Menu : set it nested in X**

The menu entry *set it nested in X* is present when the component follow the component X in the browser. Choosing this entry the current component became a sub component of X.

### **Menu : extract it from X**

The menu entry *extract it from X* is present when the component is nested in X (perhaps X is itself nested in another one, but it is not visible here). Choosing this entry the current component became a children of the parent of X.

### **Menu : delete**

The menu entry *delete* is only present when the component/artifact is not *read-only*.

Delete the component and all its representation in the opened diagrams. After that it is possible to undelete it (from the browser) un you close the project : obviously the deleted items are not saved !

### **Menu : referenced by**

To know who are the classes associated to the component, and the components referencing the current one through a relation.

### **Menu : mark**

See [mark](#)

### **Menu : tool**

Appears only when at least one [plug-out](#) is associated to the components. To apply a *plug-out* on the component.

### **Menu : select the required class**

### **Menu : select a required class**

### **Menu : select the realized class**

### **Menu : select a realized class**

### **Menu : select the provided class**

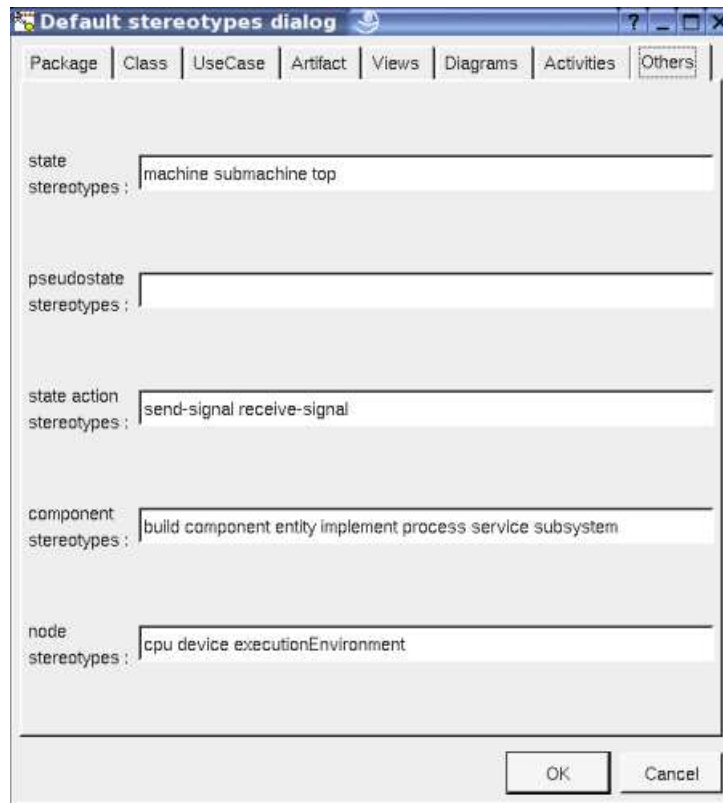
### **Menu : select a provided class**

Appears only when at least one class is required/provided/realized by the components. To quickly select the appropriate class, the current component may also be quickly [selected](#) from its associated classes.

## **Default stereotypes**

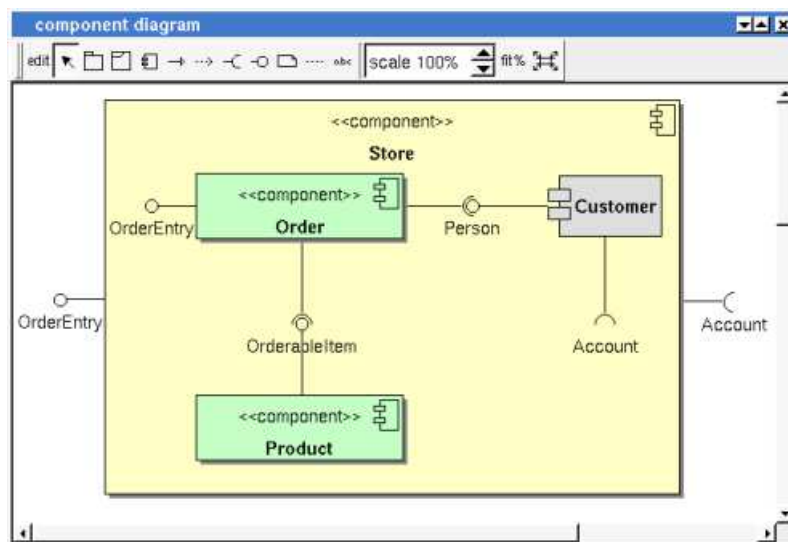
The [dialog](#) allowing to set the default stereotypes has a tab reserved for the components :





## Drawing

A component is drawn in a [component diagram](#) or a [deployment diagram](#), following the [drawing settings](#) they may be drawn as a rectangle containing the component icon (UML 2.0 notation) or as a component icon being a rectangle with two lobes (UML 1.5 notation) :



The default color of a component may be set through the [drawing settings](#).

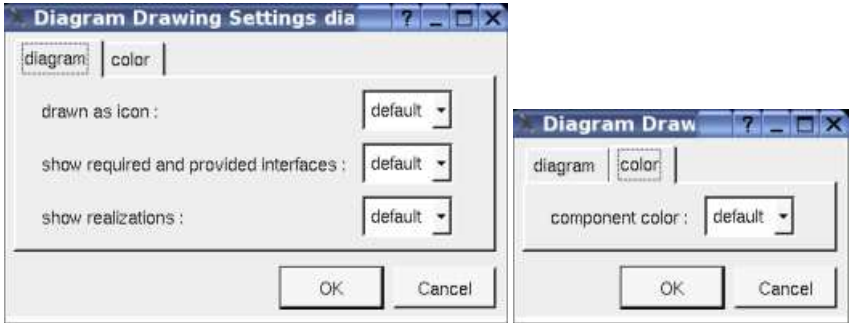
A right mouse click on a component in a diagram calls the following menu (supposing the component editable) :

Order
upper
lower
edit drawing settings
edit component
select in browser
select linked items
set associated diagram
remove from view
delete from model
tool

**Menu : edit drawing settings**

These *drawing settings* concerns only the picture for which the menu is called.

A *settings* valuing *default* indicates that the setting specified in the upper level (here the diagram) must be followed, obviously this one may also be *default ...* up to the project level. When you add a component in a diagram, all the settings are set to *default*.



*show ...* : allows to write or not the component's interfaces and realization into its drawing as a compartment.

**Menu : set associated diagram**

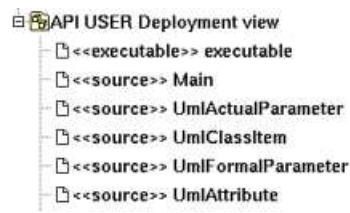
*set* *associated* *diagram* allows to automatically open the current diagram when a double mouse click is made on the component representation in a diagram or the browser. After that the only way to edit the component is to choose the *edit* entry in the menu.

Previous : [flow](#)

Next : [artifact](#)

# Artifact

Artifacts are used to produce source file(s) when the stereotype is *source*, or to specified how libraries or an executable are composed.



An artifact stereotyped *source* (potentially) associated to classes to produce their code, but you may directly give its source(s) contain for instance to define the C++ *main* function, as it is made in the [plug-outs](#).

An artifact stereotyped *text* is not associated to classes and allows to produce its C++, Java, Php, python or Idl definition as it is set, without any changes. generated file is named like the stereotype, including the extension.

The non source nor text artifacts may be associated to other artifacts, this allows to define libraries and executable, to genera Makefiles etc ... see the *plug-out* [genpro](#) producing a *.pro* file from the artifact *executable* of the *plug-outs*.

An artifact may be created through the *new artifact* entry of the [deployment view menu](#), through the artifact button of a [deployment diagram](#), or through the *create source artifact* of the [class menu](#) (in case the class view containing the class has an associated deployment view).

## Browser Menu

The artifact menu appearing with a right mouse click on its representation in the browser is something like these, supposing the artifact not *read-only* nor deleted (see also [menu in a diagram](#)) :

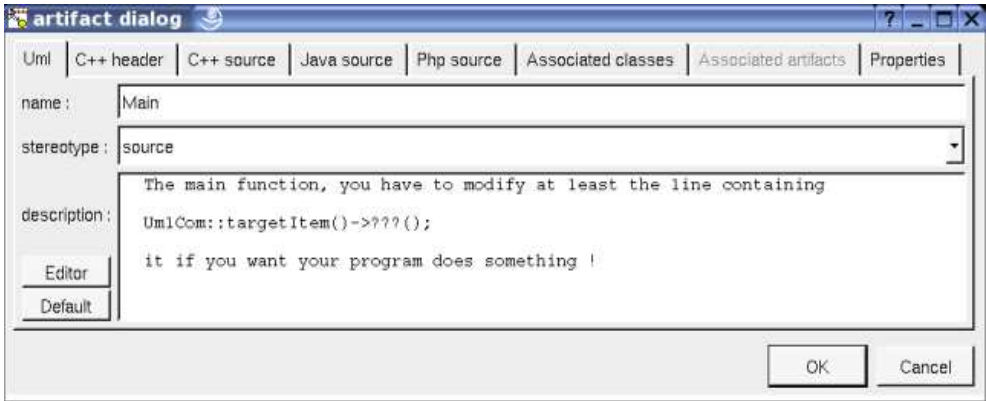
executable	executable
edit	edit
delete	delete
referenced by	referenced by
mark	mark
tool	tool

### [Menu](#) : edit

*edit* allows to show/modify the artifact properties. In case the artifact is read-only, the fields of the dialog are also read-only.

### Artifact dialog, tab Uml

The tab *Uml* is a global tab, independent of the language :



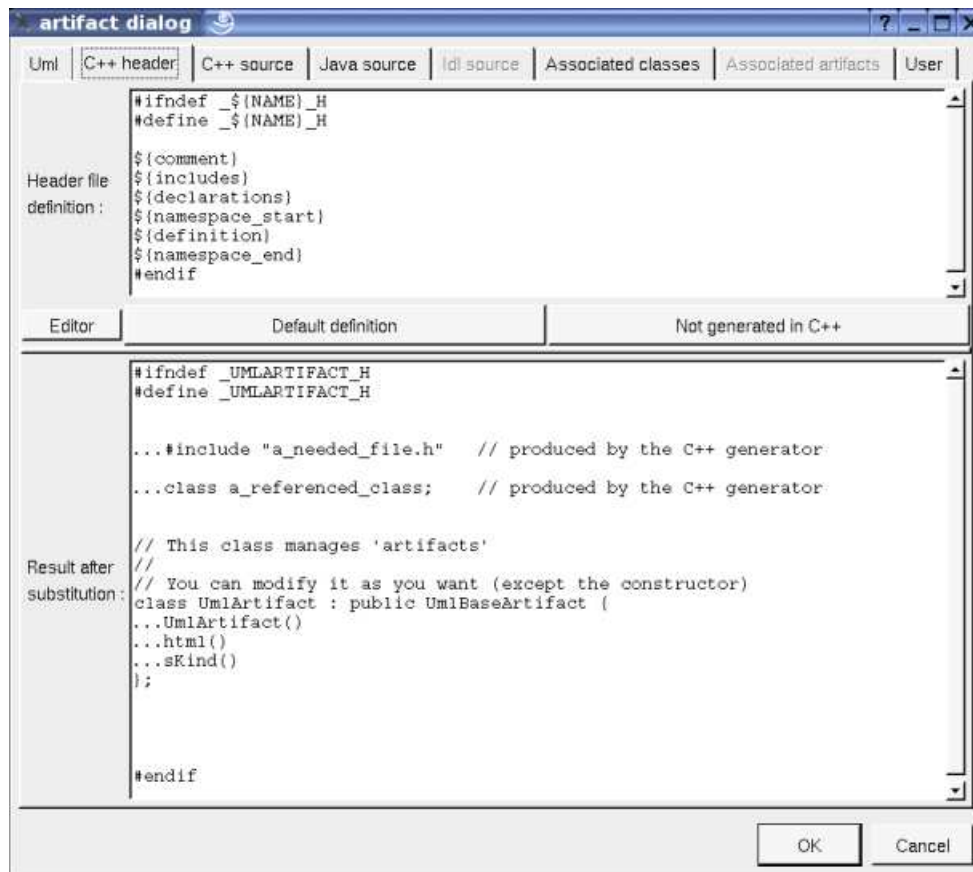
The proposed *stereotypes* are the default one specified through the [Default stereotypes dialog](#) more the current one (empty here). It is possible to choose into the list or to give a new one, or to empty it.

The **editor** button visible above and associated here to the description, allows to edit the description in an other window, or to call an external editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note that this external editor have to create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel with BOUML which looks each second at the file contents to get the new definition until the dialog is closed (I do not like very much polling but this works even QT isn't compiled with the thread support).

The **default** button visible above associated to the description allows to set the description with a default content specified through the [generation settings](#). Useful to generate comments compatible Java Doc or Doxygen for instance.

### Artifact dialog, tab C++ header

This tab is only available for the source artifacts and allows to give the definition of the C++ header file generated for the artifact, it is visible only if C++ is set through the menu *Languages*



In BOUML the generated code is obtained by the substitution of macros in a text, only the **Definition** part is editable, the other one help you to see what will be generated for C++ (supposing you do not modify the C++ code generator !).

When you push the button **default declaration**, the form specified through the [generation settings](#) is proposed, this last may be modified as you want, even to produce illegal source code.

When you do not want to generate the C++ header file (for instance for the *main*), empties the declaration manually or using the button **not generated in C++**.

`${NAME}` produce the artifact name capitalized, `$(Name)` produce the artifact name with the first letter capitalized, at least `$(name)` produce the artifact name without modification.

`$(comment)` is replaced by the artifact description adding //

`$(description)` is replaced by the artifact description without adding //

`$(includes)` is replaced by the automatically generated `#include` and `using` forms (done by the C++ code generator, not made by the dialog which produce a fixed form). It is also a good place to add your `#include` and `using` forms, when the ones produced by the code generator are not sufficient. The C++ code generator does not look at in the operations body, only the operation's profiles: relations and attributes, classes inheritances etc ... are used to compute the needed `#include` list. You can also use dependencies to ask for to add `#includes` specifying if these ones must be placed in header or source file. `$(includes)` and `$(all_includes)` are exclusive.

`$(all_includes)` is replaced by all the automatically generated `#include` and `using` forms, it is also a good place to add your `#include`

and using  
forms, when the ones produced by the code generator are not sufficient. In this case the code generator doesn't produce declarations in the header file nor `#include` in the source file, except the ones added by you. `$(all_includes)` and `$(includes)` are exclusive.

`$(declarations)`

is replaced by the class declarations generated automatically (done by the C++ code generator, not made by the dialog which produce a fixed form). It is also a good place to add your declarations, when the ones produced by the code generator are not sufficient.

`$(namespace_start)` is replaced by the `namespace xx {` forms, dependent on the [namespace](#) specifications associated to the package containing the [deployment view](#) where the artifact is defined.

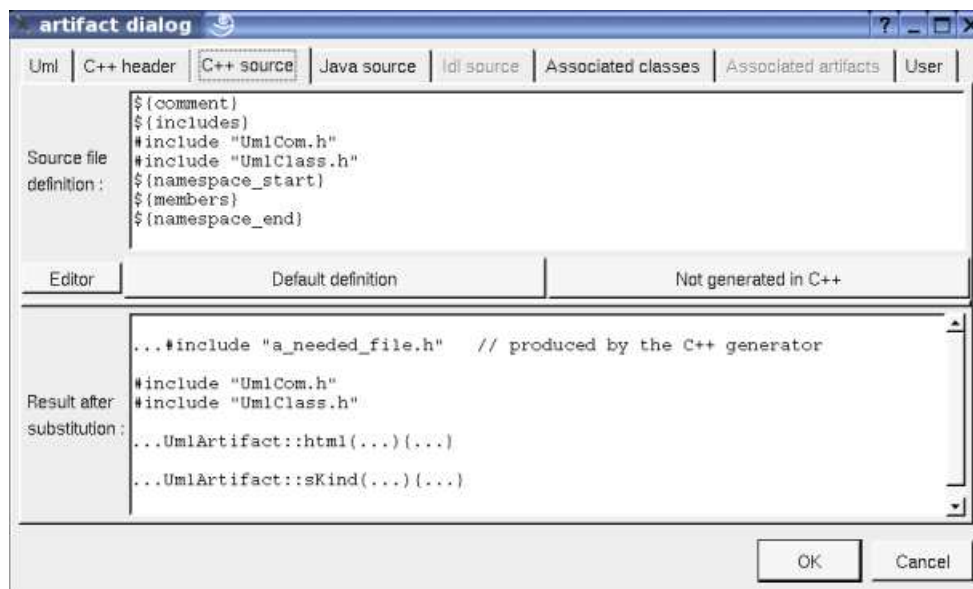
`$(definition)` is replaced by the definition of the classes associated to the artifact.

`$(namespace_end)` is replaced by the `}` forms, dependent on the [namespace](#) specifications associated to the package containing the [deployment view](#) where the artifact is defined.

The @{property}  
are replaced by the value of the corresponding user property if it is defined for the class or at an upper level, else it is not substituted.

### Artifact dialog, tab C++ source

This tab is only available for the source artifacts and allows to give the definition of the C++ source file generated for the artifact, it is visible only if C++ is set through the menu *Languages*



When you push the button **default declaration**, the form specified through the [generation settings](#) is proposed, this last may be modified as you want, even to produce illegal source code.

When you do not want to generate the C++ source file (for instance the artifact is only associated to an *enum*), empties the declaration manually or using the button **not generated in C++**.

`$(comment)` is replaced by the artifact description

`$(description)` is replaced by the artifact description without adding `//`

`$(includes)` if `$(all_includes)` is not used in the header definition, it is replaced by the automatically generated `#include` and `using` forms (done by the C++ code generator, not made by the dialog which produce a fixed form). If `$(all_includes)` is used in the header definition, it is replaced by the `#include` of the header. It is also a good place to add your `#include` and `using` forms, the code generator does not look inside the bodies to compute the needed ones.

`$(namespace_start)` is replaced by the `namespace xx {` forms, dependent on the [namespace](#) specifications associated to the package containing the [deployment view](#) where the artifact is defined.

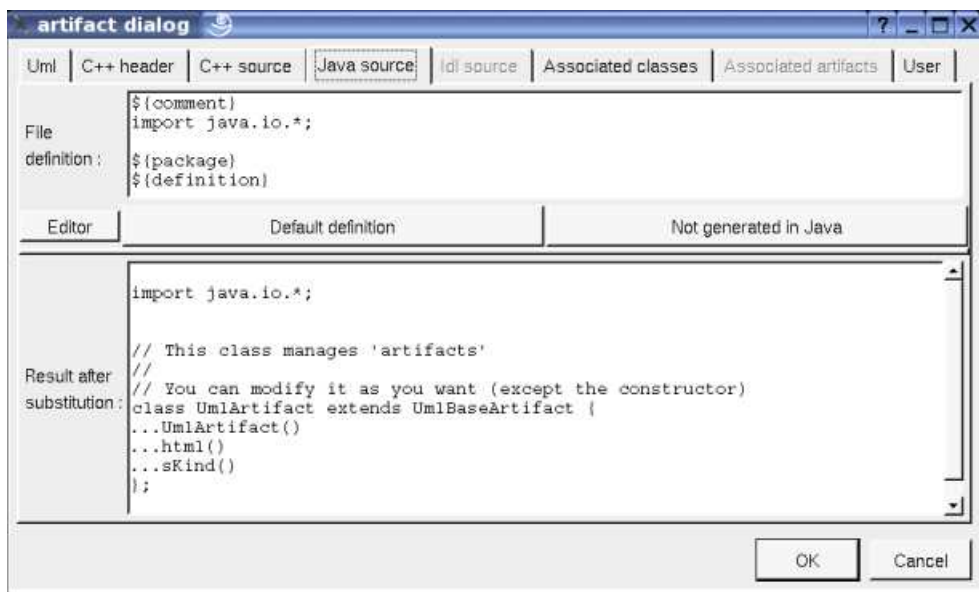
`$(members)` is replaced by the non *inline* members of the classes associated to the artifact.

`$(namespace_end)` is replaced by the `}` forms, dependent on the [namespace](#) specifications associated to the package containing the [deployment view](#) where the artifact is defined.

The @{property}  
are replaced by the value of the corresponding user property if it is defined for the class or at an upper level, else it is not substituted.

## Artifact dialog, tab Java source

This tab is only available for the source artifacts and allows to give the definition of the Java source file generated for the artifact, it is visible only if Java is set through the menu *Languages*



When you push the button **default declaration**, the form specified through the [generation settings](#) is proposed, this last may be modified as you want, even to produce illegal source code.

When you do not want to generate the Java source file, empties the declaration manually or using the button **not generated in Java**.

`${comment}` is replaced by the artifact description adding `/* */`

`${description}` is replaced by the artifact description without adding `/* */`

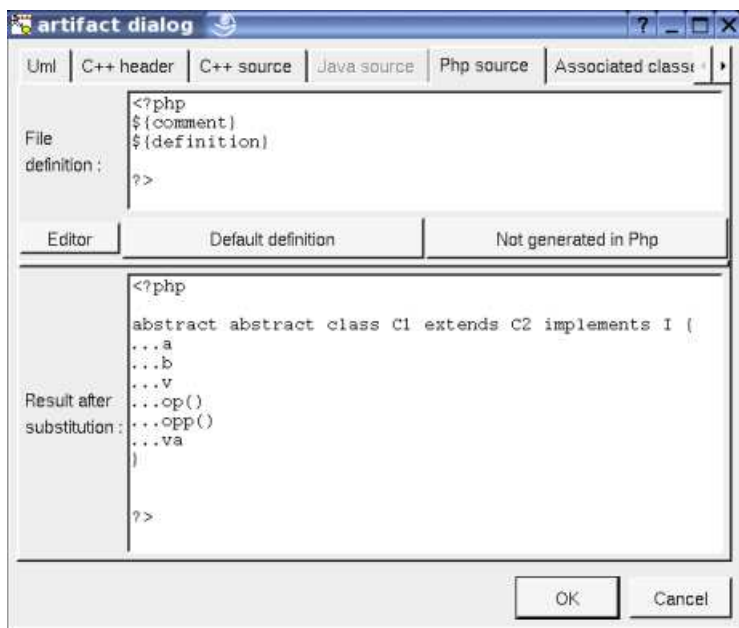
`${package}` is replaced by the `package xx` forms, dependent on the [package](#) specifications associated to the BOUML package containing the [deployment view](#) where the artifact is defined.

`${definition}` is replaced by the definition of the classes associated to the artifact.

The `@{property}` forms are replaced by the value of the corresponding user property if it is defined for the class or at an upper level, else it is not substituted.

## Artifact dialog, tab Php source

This tab is only available for the source artifacts and allows to give the definition of the Php source file generated for the artifact, it is visible only if Php is set through the menu *Languages*





When you push the button **default declaration**, the form specified through the [generation settings](#) is proposed, this last may be modified as you want, even to produce illegal source code.

When you do not want to generate the Php source file, empties the declaration manually or using the button **not generated in Php**.

`${comment}` is replaced by the artifact description adding `/* */`

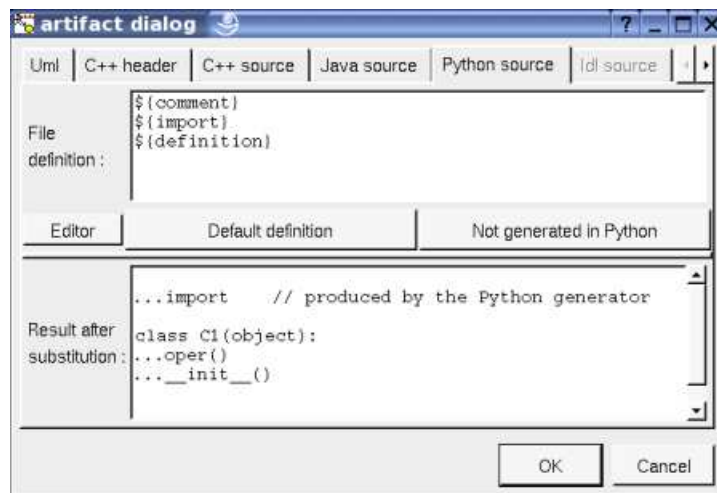
`${description}` is replaced by the artifact description without adding `/* */`

`${definition}` is replaced by the definition of the classes associated to the artifact.

The forms `@{property}` are replaced by the value of the corresponding user property if it is defined for the class or at an upper level, else it is not substituted.

### Artifact dialog, tab Python source

This tab is only available for the source artifacts and allows to give the definition of the Python source file generated for the artifact, it is visible only if Python is set through the menu *Languages*



When you push the button **default declaration**, the form specified through the [generation settings](#) is proposed, this last may be modified as you want, even to produce illegal source code.

When you do not want to generate the Python source file, empties the declaration manually or using the button **not generated in Python**.

`${comment}` is replaced by the artifact description adding `#` at the beginning of each line

`${description}` is replaced by the artifact description without adding `#`

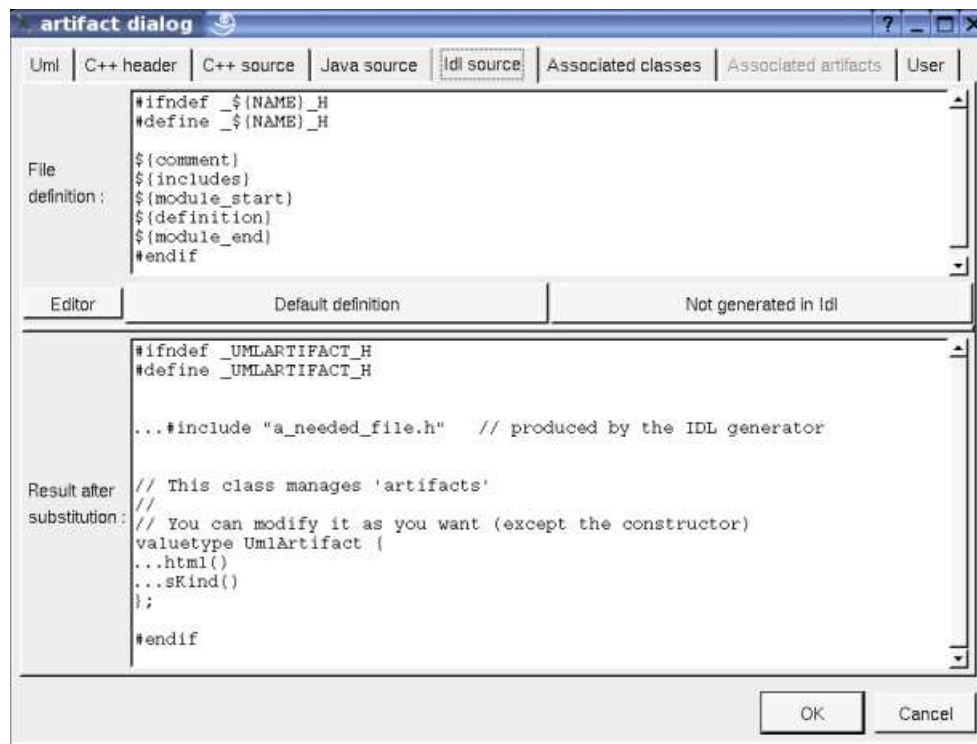
`${imports}` is replaced by the automatically generated *import* and *from .. import* forms (done by the Python code generator, not made by the dialog which produce a fixed form) from the dependencies between classes and between *artifacts* stereotyped *import* or *from*. A dependency stereotyped *import* between artifact produces an *import*, stereotyped *from* this produces a *from ... import \**. A dependency stereotyped *import* between classes produces an *import*, stereotyped *from* this produces a form *from ... import*.

`${definition}` is replaced by the definition of the classes associated to the artifact.

The forms `@{property}` are replaced by the value of the corresponding user property if it is defined for the class or at an upper level, else it is not substituted.

### Artifact dialog, tab Idl source

This tab is only available for the source artifacts and allows to give the definition of the Idl source file generated for the artifact, it is visible only if Idl is set through the menu *Languages*. In the picture given for the [UML tab](#), the Idl tab is not enabled because the artifact is not associated to at least one class defined in Idl, else :



When you push the button **default declaration**, the form specified through the [generation settings](#) is proposed, this last may be modified as you want, even to produce illegal source code.

When you do not want to generate the Idl source file, empties the declaration manually or using the button **not generated in Idl**.

`$(NAME)` produce the artifact name capitalized, `$(Name)` produce the artifact name with the first letter capitalized, at least `$(name)` produce the artifact name without modification.

`$(comment)` is replaced by the artifact description adding `//`

`$(description)` is replaced by the artifact description without adding `//`

`$(includes)` will be replaced by the automatically generated `#include` forms (not made by the dialog which produce a fixed form) ... in a future version of the Idl generator, it is a good place to add your `#include` forms.

`$(module_start)` is replaced by the `module xx {` forms, dependent on the [module](#) specifications associated to the BOUML package containing the [deployment view](#) where the artifact is defined.

`$(definition)` is replaced by the definition of the classes associated to the artifact.

`$(module_end)` is replaced by the `}` forms, dependent on the [module](#) specifications associated to the BOUML package containing the [deployment view](#) where the artifact is defined.

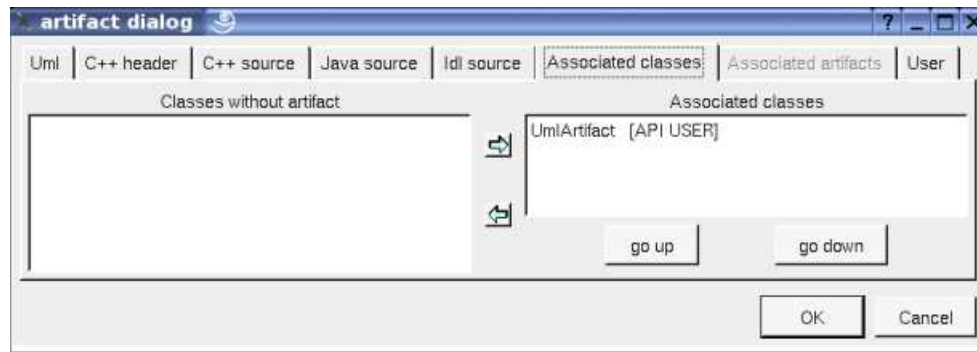
Refer to the [generation settings](#) for more details.

The `forms` `@{property}` are replaced by the value of the corresponding user property if it is defined for the class or at an upper level, else it is not substituted.

### Artifact dialog, tab associated classes

This tab is only available for the source artifacts and allows to set the list of classes associated with the artifact. The order of associated classes is important because it is followed by the code generators to produce the class definitions.



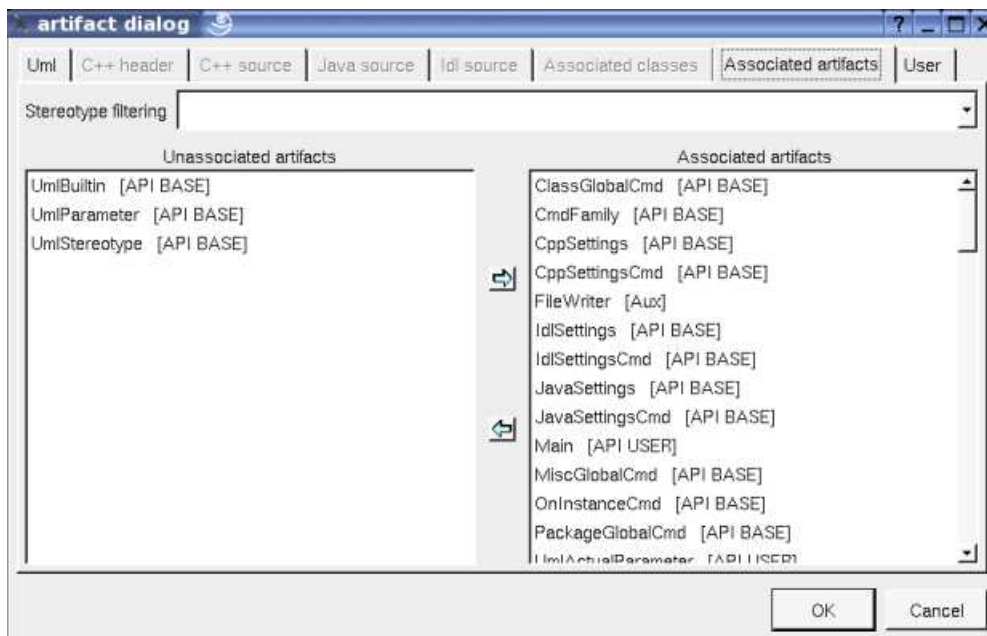


All the classes without associated artifact are given in the left table (above all the classes are associated with an artifact), the classes associated with the current artifact and given in the table on the right. To move classes from a table to the other one, select them a click on the appropriate arrow.

*go up* and *go down* allows to change the order of the associated classes. Select the wrong ordered classes and use the appropriate button.

### Artifact dialog, tab associated artifacts

This tab is not enabled for a source artifact, to set the unordered list of artifacts associated to the current one. For instance for the artifact *executable* of the *plug-out html* :



### Menu : delete

The menu entry *delete* is only present when the artifact is not *read-only*.

Delete the artifact and all its representation in the opened diagrams. After that it is possible to undelete it (from the browser) until you close the project : obviously the deleted items are not saved !

### Menu : generate

The menu entry *generate* is only present when the artifact is a *source artifact*.

Allows to call the [C++ generator](#), [Java generator](#), [Php Generator](#) and [Idl generator](#).

### Menu : edit

The menu entry *edit* is only present when the artifact is not *read only* and is a *source artifact*.

Allows to edit one of the generated files, for instance the header file :

```

UmlArtifact.h
#ifndef _UMLARTIFACT_H
#define _UMLARTIFACT_H

#include "UmlBaseArtifact.h"
#include <QString>

// This class manages 'artifacts'
//
// You can modify it as you want (except the constructor)
class UmlArtifact : public UmlBaseArtifact {
public:
    UmlArtifact(void * id, const QString & n) : UmlBaseArtifact(id, n) {};

    //entry to produce the html code receiving chapter number
    //path, rank in the mother and level in the browser tree
    virtual void html(QString pfir, unsigned int rank, unsigned int level);

    //returns a string indicating the kind of the element
    virtual QString sKind();
};

#endif
line 1      column 1

```

### **Menu : referenced by**

To know who are the classes associated to the artifact, and the artifact referencing the current one through a relation.

### **Menu : mark**

See [mark](#)

### **Menu : tool**

Appears only when at least one [plug-out](#) is associated to the artifacts. To apply a *plug-out* on the artifact.

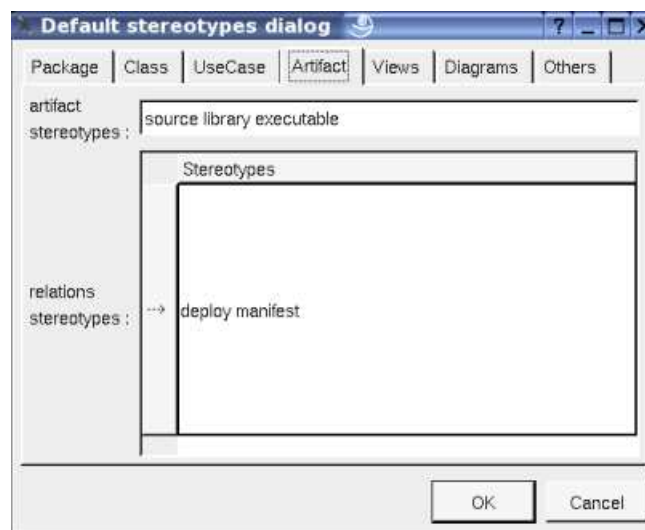
### **Menu : select associated class**

### **Menu : select an associated class**

Only for the artifact source, appears only when at least one class is associated to the artifacts. To quickly select the appropriate class, the current artifact may also be quickly [selected](#) from its associated classes.

## **Default stereotypes**

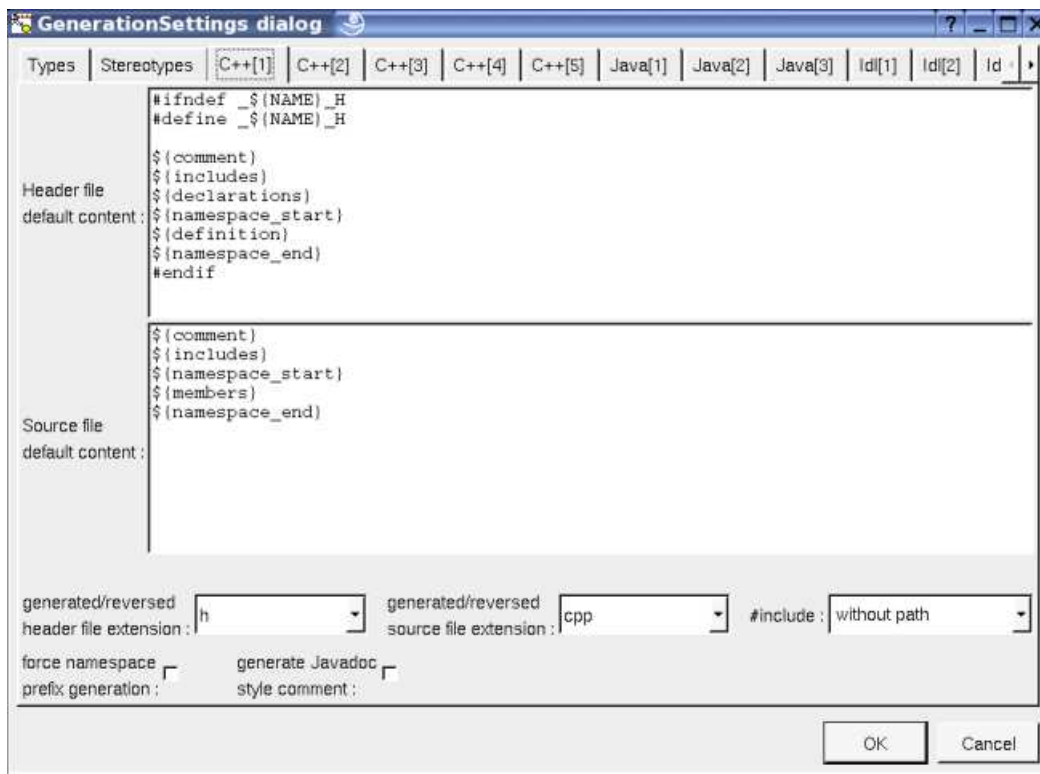
The [dialog](#) allowing to set the default stereotypes has a tab reserved for the artifacts :



## **Generation settings**

This very important dialog allows to specify many default definitions concerning the classes, more details will be given in [C++ generator](#), [Java generator](#) and [Idl generator](#).

The first C++ tab allows to specify the default header and source file definitions, it is visible only if C++ is set through the menu *Languages* :

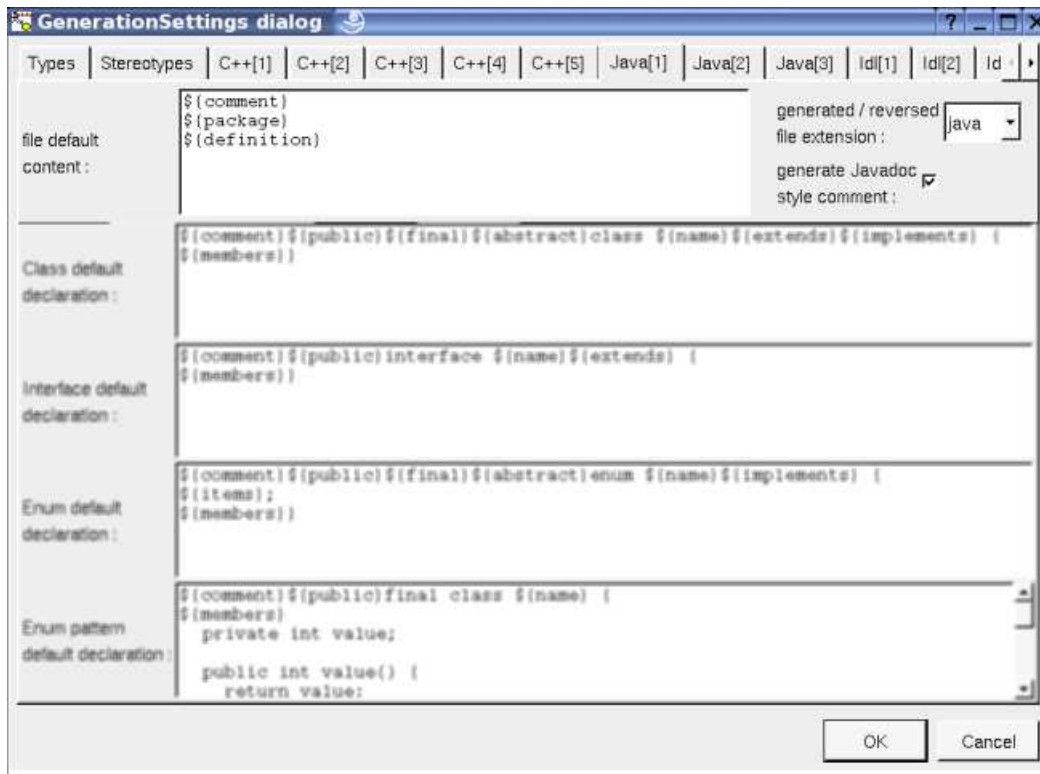


As you can see it is possible to choose the extension of the header and source files, to force the production of the namespace header and to ask for the Javadoc style comment.

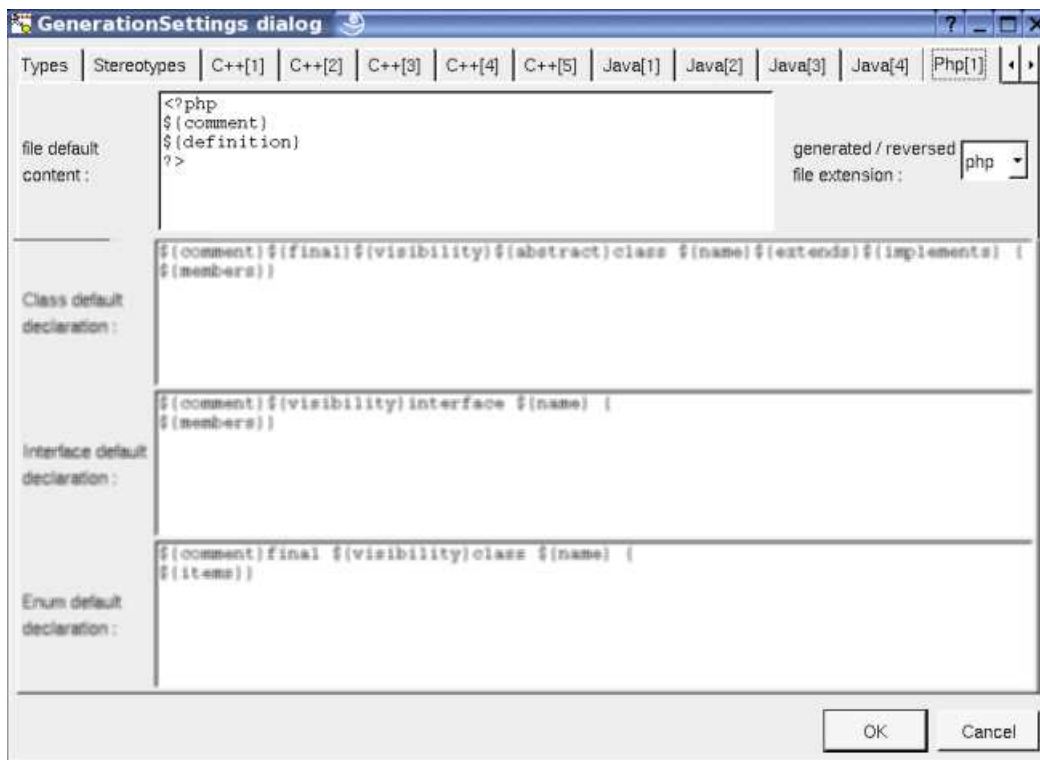
There are four ways to produce #include :

- *without path* : ask for the C++ code generator to generate the #include without relative or absolute path
- *with absolute path* : ask for the C++ code generator to generate the absolute path of the automatically included files.
- *with relative path* : ask for the C++ code generator to generate the relative path of the automatically included files, relative to the file containing the #include
- *with root relative path* : ask for the C++ code generator to generate the relative path of the automatically included files, relative to the directory specified by the *generation settings*

The first Java tab allows to set the default Java file content associated to an artifact, it is visible only if Java is set through the menu *Languages* :



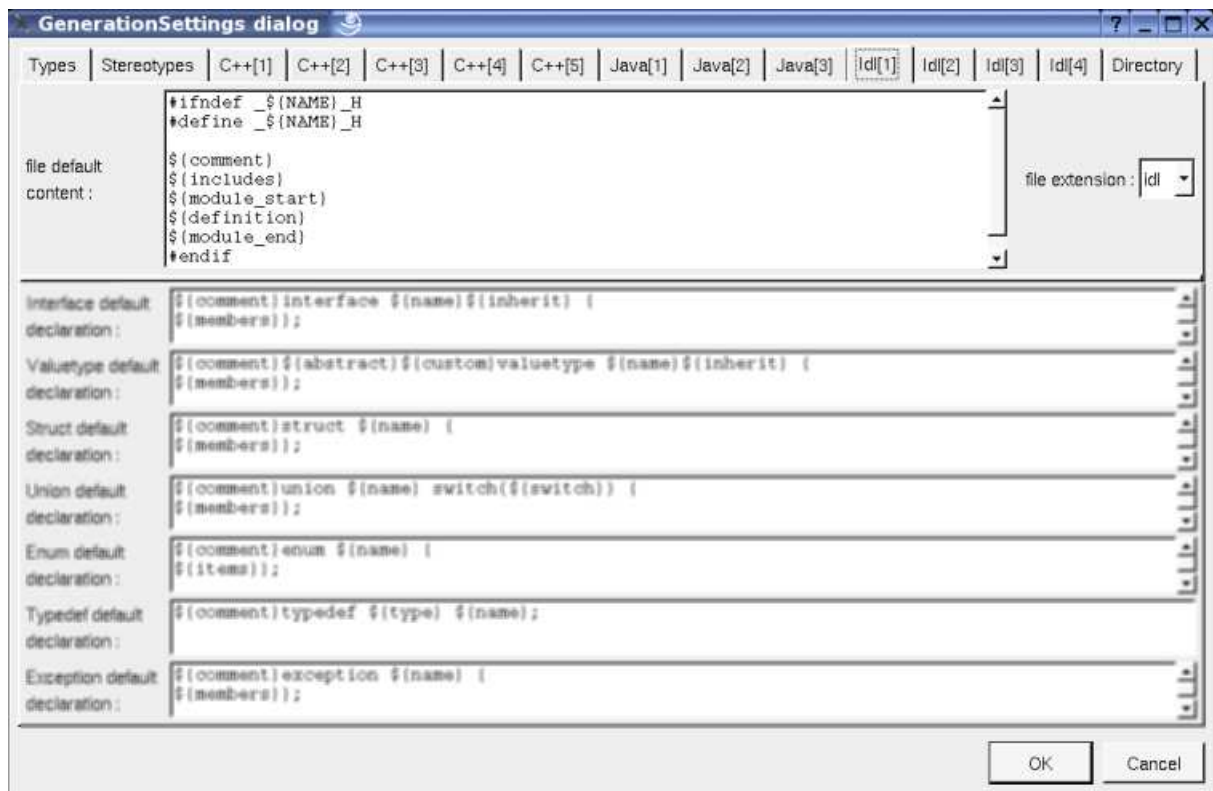
The first Php tab allows to set the default Php file content associated to an artifact, it is visible only if Php is set through the menu *Languages* :



The first Python tab allows to set the default Python file content associated to an artifact, it is visible only if Python is set through the menu *Languages* :

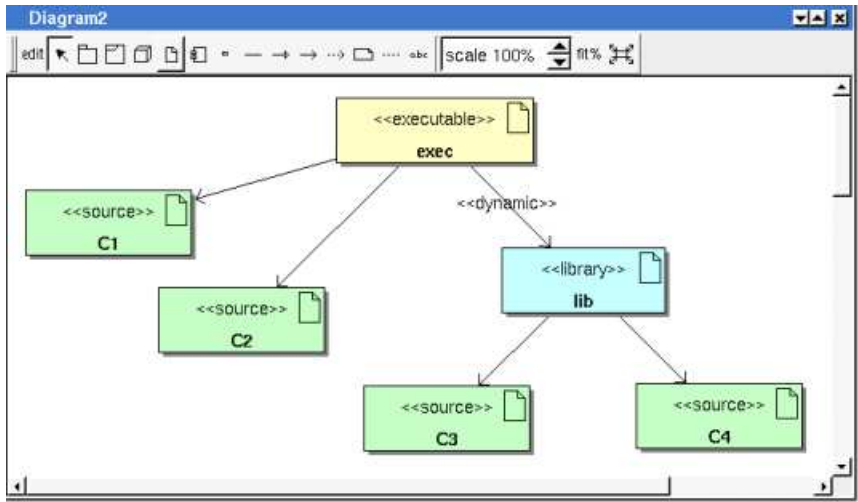


The first Idl tab allows to set the default Idl file content associated to an artifact, it is visible only if Idl is set through the menu *Languages* :



## Drawing

An artifact is drawn in a [deployment diagram](#) as a rectangle containing the artifact icon :



The default color of an artifact may be set through the [drawing settings](#).

A right mouse click on an artifact in a diagram calls the following menu (supposing the artifact editable) :

lib
Upper
Lower
Edit drawing settings
Edit artifact
Select in browser
Select linked items
Set associated diagram
Remove diagram association
Remove from view
Delete from model
Generate
Tool

**Menu : edit drawing settings**

These *drawing settings* concerns only the picture for which the menu is called.

A *settings* valuing *default* indicates that the setting specified in the upper level (here the diagram) must be followed, obviously this one may also be *default* ... up to the project level. When you add an artifact in a diagram, all the settings are set to *default*.

**Menu : set associated diagram**

*set associated diagram* allows to automatically open the current diagram when a double mouse click is made on the artifact representation in a diagram or the browser. After that the only way to edit the artifact is to choose the *edit* entry in the menu.

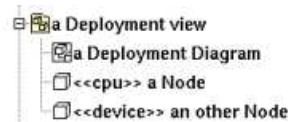
**Menu : remove diagram association**

To stop to associate a diagram to the artifact.

Next : [deployment node](#)

# Node

Nodes are used to specify how the deployment is made, a node may represent a CPU, a device etc ...



A node may be created through the *new node* entry of a [deployment view](#) or through the component button of a [deployment diagram](#).

## Menus

The node menu appearing with a right mouse click on its representation in the browser is something like these, supposing the node not *read-only* nor deleted (see also [menu in a diagram](#)) :

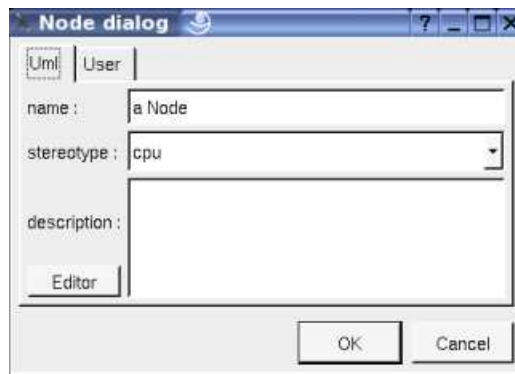


### [Menu](#) : edit

*edit* allows to show/modify the node properties. In case the node is read-only, the fields of the dialog are also read-only.

#### Node dialog, tab Uml

The tab *Uml* is a global tab, independent of the language :



The proposed *stereotypes* are the default one specified through the [Default stereotypes dialog](#) more the current one (empty here). It is possible to choose into the list or to give a new one, or to empty it.

The *editor* button visible above and associated here to the description, allows to edit the description in an other window, or to call an external editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note that this external editor have to create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel with BOUML which looks each second at the file contents to get the new definition until the dialog is closed (I do not like very much polling but this works even QT isn't compiled with the thread support).

### [Menu](#) : delete

The menu entry *delete* is only present when the node is not *read-only*.

Delete the node and all its representation in the opened diagrams. After that it is possible to undelete it (from the browser) until you close the project : obviously the deleted items are not saved !

### [Menu](#) : mark



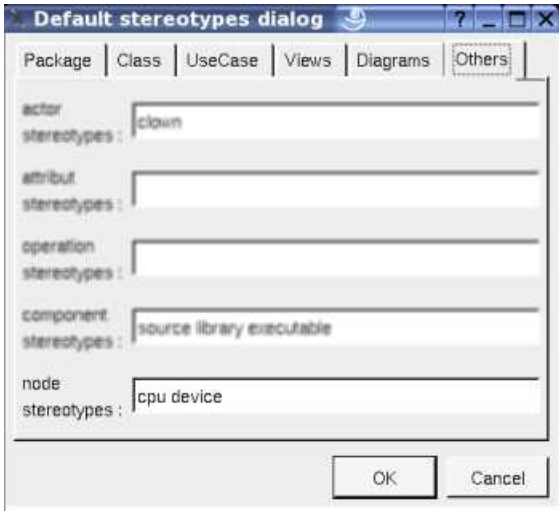
See [mark](#)

**Menu : tool**

Appears only when at least one [plug-out](#) is associated to the nodes. To apply a *plug-out* on the node.

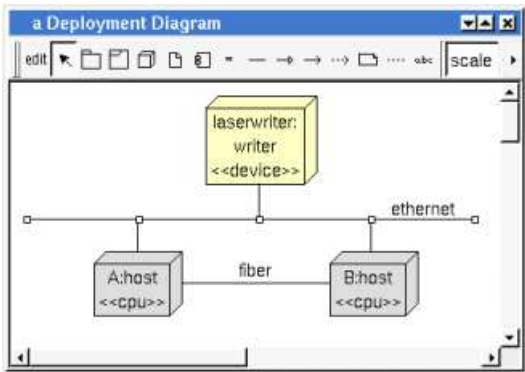
**Default stereotypes**

The [dialog](#) allowing to set the default stereotypes has a tab reserved for the components :



**Drawing**

A component is drawn in a [deployment diagram](#) as a parallelepiped depending on the [drawing settings](#).



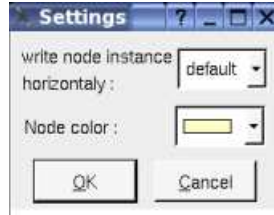
A right mouse click on a node in a diagram calls the following menu (supposing the node editable) :

writer
upper
lower
edit
edit drawing settings
select node in browser
select linked items
set node associated diagram
remove from view
tool

**Menu : edit drawing settings**

These *drawing settings* concerns only the picture for which the menu is called.

A *settings* *valuing* *default*  
 indicates that the setting specified in the upper level (here the diagram) must be followed, obviously this one may also be *default* ... up to the project level. When you add a node in a diagram, all the settings are set to *default*.



### **write node instance horizontally**

Allows to write the the node type and the instance name on the same line (see host [below](#)) or on two lines (see laserwriter [below](#)).

### **Menu : select linked items**

To select all the items connected by one or several lines (whatever the line represent : relation, anchor ...), help to quickly select them for a moving etc ...

### **Menu : set associated diagram**

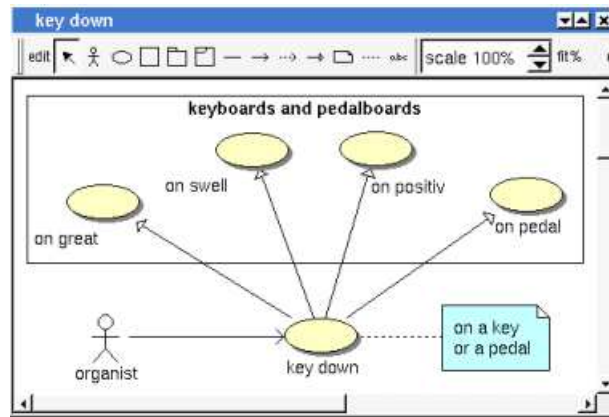
*set* *associated* *diagram*  
 allows to automatically open the current diagram when a double mouse click is made on the node representation in a diagram ( browser). After that the only way to edit the node is to choose the *edit* entry in the menu.

---

Previous : [artifact](#)

Next : [use case diagram](#)

# Use case diagram



The use case diagrams may be placed in a [use case view](#) or a [use case](#)



A use case diagram is created through the *new use case diagram* entry of the [use case view](#) and [use case](#) browser menus.

## Browser Menu

The node menu appearing with a right mouse click on its representation in the browser is something like these, supposing the diagram not *read-only* nor deleted :



### [Browser Menu](#) : show

*show* allows to show the diagram [picture](#), and edit it in case it is not read only.

### [Browser Menu](#) : edit

*edit* allows to show/modify the diagram properties. In case the diagram is read-only, the fields of the dialog are also read-only.

### Use case diagram dialog, tab Uml

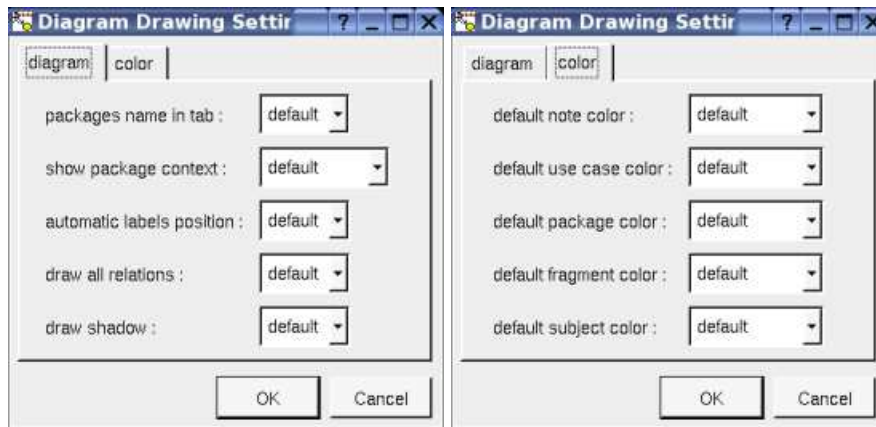


The proposed *stereotypes* are the default one specified through the [Default stereotypes dialog](#) more the current one (empty here). It is possible to choose into the list or to give a new one, or to empty it. The *stereotype* of a diagram doesn't have a special meaning for BOUML.

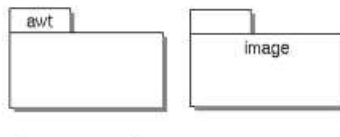
The *editor*  
button visible above and associated here to the description, allows to edit the description in an other window, or to call an external editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note that this external editor have to create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel with BOUML which looks each second at the file contents to get the new definition until the dialog is closed (I do not like very much polling but this works even QT isn't compiled with the thread support).

### Browser Menu : edit drawing settings

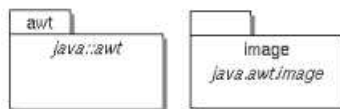
This dialog allows to specify how the items of the diagram must be drawn by default, you may choose *default* for each value, in this case the effective value if specified in the upper level (which itself may specify *default* etc ... up to the *project* level).



- *package* *name* *in* *tab*  
: to indicate if the package's name must be written in the tab which is the small rectangle on the top (note : a transparent *package* doesn't have shadow) :



- *show* *packages* *context*  
: to indicate if the context where the package is defined must be written, it is not the case just above. The context may be the "UML context" which is the path of the package in the browser (choice followed for *awt* below), or the C++ namespace specified by the package, or the Java package specified by the package (choice followed for *picture* below) or at least the Idl module :



- *automatic* *labels* *position*  
: To ask BOUML to move or to not move the associated labels on the relations in a diagram when the relation is edited connected items are moved etc ...

### Browser Menu : duplicate

Clone the diagram in a new one

### Browser Menu : delete

The *delete* entry is only present when the diagram is not *read-only*.

Delete the diagram and all its representation in the opened diagrams. After that it is possible to undelete it (from the browser) until you close the project : obviously the deleted items are not saved !

**Menu : mark**

See [mark](#)

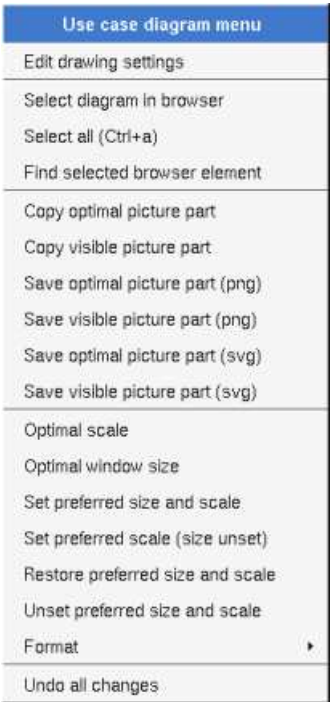
**Browser Menu : tools**

The menu entry *tool* is only present in case at least a [plug-out](#) may be applied on the diagram. The selected tool is called and applied on the current diagram.

---

**Picture menu**

The diagram menu appearing with a right mouse click out off any item in the diagram picture is something like these, supposin diagram not *read-only* :



**Picture Menu : edit drawing settings**

see [edit drawing settings](#).

**Picture Menu : select all**

Select all the items of the current diagram, also done through a *control a*.

**Picture Menu : find selected browser element**

To find in the diagram a representation of the elements selected in the browser

**Picture Menu : optimal scale**

similar to the *fit scale* button, change scale to see all the diagram's elements at great as possible

**Picture Menu : optimal window size**

Similar to the button , change the size of the windows to see all the diagram's elements with the current scale

**Picture Menu : copy visible part**

Put the visible part of the current diagram in the clipboard (*copy*), it is a screen copy, the current scale have impact on the picture quality. To easily get all the diagram's elements use *optimal window size*

It is also possible to [print](#) all (not only the visible part) the current diagram on printer or a file through the button

### **Picture Menu : save visible picture part**

To save the visible part of the current diagram in a file, it is a screen copy, the current scale have impact on the picture quality.

### **Picture Menu : set preferred size and scale**

To memorize the current diagram sub window size and the current scale. These ones will be used the next time the diagram will be opened.

### **Picture Menu : set preferred scale**

To memorize the current scale. These one will be used the next time the diagram will be opened.

### **Picture Menu : restore preferred size and scale**

In case a preferred sub window size and preferred scale was set, follow them.

### **Picture Menu : unset preferred and scale**

To stop to define a preferred size and scale was set

### **Picture Menu : format**

By default the diagram's canvas size is 840 x 1188 points corresponding to an iso A4 format (210\*4 x 297\*4), all the available formats are :

- iso A5 : 592 x 840
- iso A4 : 840 x 1188
- iso A3 : 1188 x 1680
- iso A2 : 1680 x 2376
- iso A1 : 2376 x 3364
- iso A0 : 3364 x 4756
- USA A : 864 x 1116
- USA B : 1116 x 1728
- USA C : 1728 x 2236
- USA D : 2236 x 3456
- USA E : 3456 x 4472

Each format also exists in *landscape*

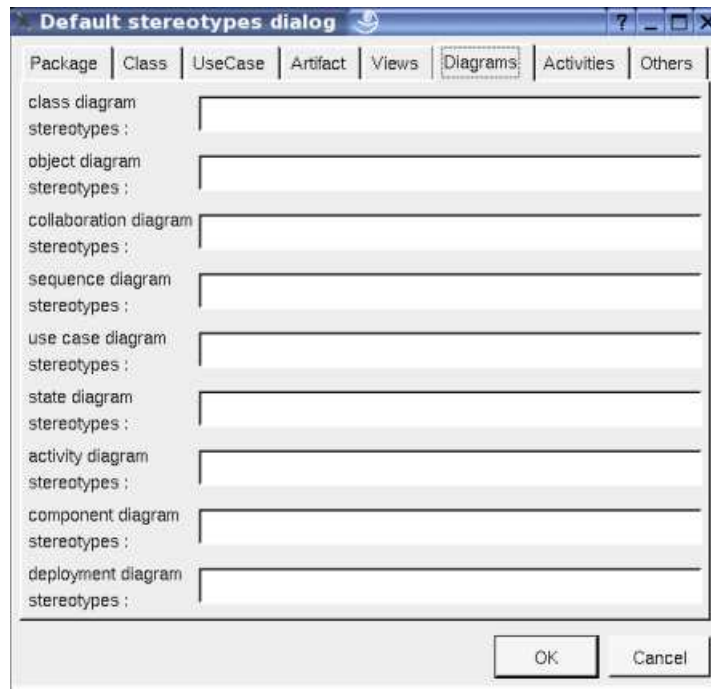
### **Picture Menu : undo changes**

Undo all the changes made since the diagram was opened the last time. Note that the diagram is restored independently of the project saving. Obviously the deleted browser items was not restored !

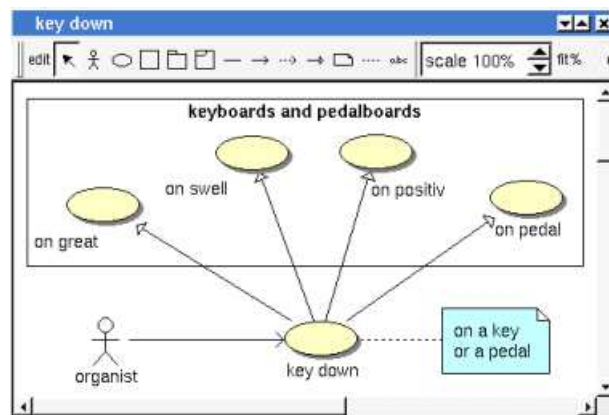
---

## **Default stereotypes**

The [dialog](#) allowing to set the default stereotypes has a tab reserved for the diagrams :



## Drawing



A use case diagram may contain [actors](#), [classes](#), [use cases](#), [packages](#), [fragment](#), [notes](#), [texts](#), [diagram icons](#), [subjects](#) and the relations.

To place these elements in the diagram, use the buttons on the top (to create a new element or choose from the already defined one) then click at the appropriate position in the diagram, or *grab* the element from the browser into the diagram.

The *scale* may be changed from 30 up to 200 %. To allows to see the relation kinds even with a small scale, the size of the arrows and unmodified.

*fit%* allows to set the largest scale allowing to show all the picture, as it is possible. Because of the scrollbar management, hitting several times on *fit%* may give several scales.

Classically, the left mouse click may be used to select an element, or several when the *control* key is down, a multiple selection may also be done defining a region using the mouse left click out of any element and moving the mouse with the click and maintaining the click down. The diagram's elements may be moved after a left mouse click on an element and maintaining the click down during mouse move. The right mouse click is used to show context dependent menus.

*control s* is a shortcut to save the project

*control a* is a shortcut to select all the diagram's elements

*control d* is a shortcut to delete the selected element(s), the model is affected

*suppr* is a shortcut to remove the selected element(s) from the diagram, these elements are not deleted

*control c* is a shortcut to copy the selected diagram's elements

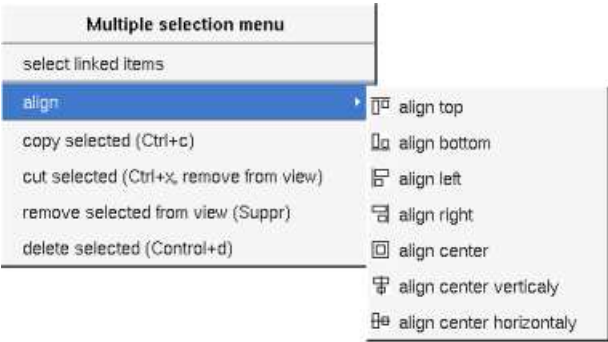
*control x* is a shortcut to cut (remove from view) the selected diagram's elements

*control v* is a shortcut to paste the selected diagram's elements

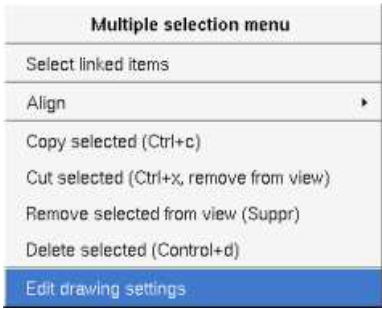
the keyboard arrows allow to move the selected items

You can define your own shortcut using the [shortcut editor](#) through the *Miscellaneous* menu.

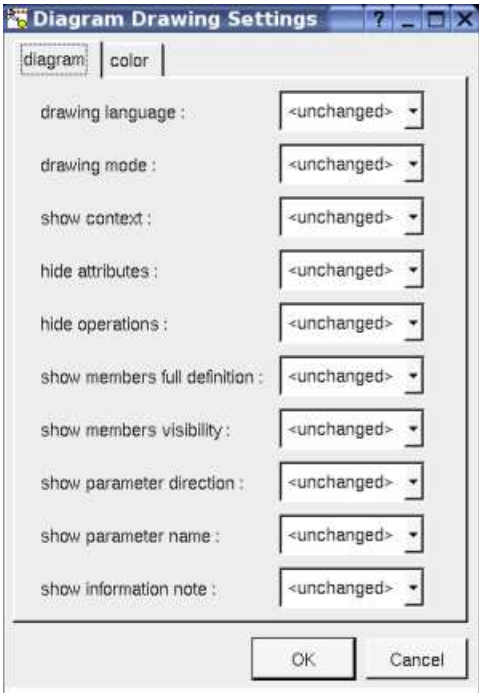
When several diagram's elements are selected, the menu appearing on a left mouse click allows to align them :



If all the selected elements has the same kind (for instance all are a class), you edit their *drawing settings* globally :



and for instance when the elements are classes in a class diagram :





The settings valuing are not changed for the selected elements, in the other case the same value is set for all the elements. <unchanged>

Fragment

A *fragment* is resizeable. To change the fragment's name choose the *edit* entry of its menu or double click on it

A right mouse click on a note show the following menu :



*Edit drawing settings* allows to change the fill color

Note

A *note* may contain any text, a note is resizeable. To change the text of a note choose the *edit* entry of its menu or double click on it, this open a window containing the text :



A right click on the edited note calls a dialog allowing some text manipulations thanks Qt.

A right mouse click on a note show the following menu :



The *font* may be changed, by default the font is the one used to write the other texts.

The color of the text may be changed, a text is black by default.

*Edit drawing settings* allows to change the fill color

*Select linked items* select all the items connected this the current one through any line, in the previous picture this select all except the package, the text “*upper diagram* :” and the use case diagram icon.

Text

*Text* is another way to write a text, contrary to the note a *text* is written alone (see “*upper diagram* :” on the picture below), by default a text contains *blabla*: a text cannot be empty. The edition of a text is done like for a note. A text is resizeable, a resize is not automatically done when the contains is modified.

A right mouse click on a *text* show the following menu :

Text
Upper
Lower
Edit
Color
Font
Select linked items
Remove from view

Diagram icon

A diagram icon is a shortcut to a diagram, this allows to quickly navigate through the diagram without limitation (a actor or some other item kinds may be [associated](#) to only one diagram). The used icon indicates the diagram kind, but its name is not written.

A double mouse left click on a diagram icon opens the diagram, the *open* entry of the diagram icon menu does the same thing.

A right mouse click on a diagram icon show the following menu :

Overall short cut
upper
lower
open
select diagram in browser
select linked items
remove from view

The other diagram's elements have an own chapter, refer to them.

Subject

A *subject* is drawn as a rectangle, to set/modify the title do a double left mouse click on the representation or call the menu through a right mouse click and choose *edit* :

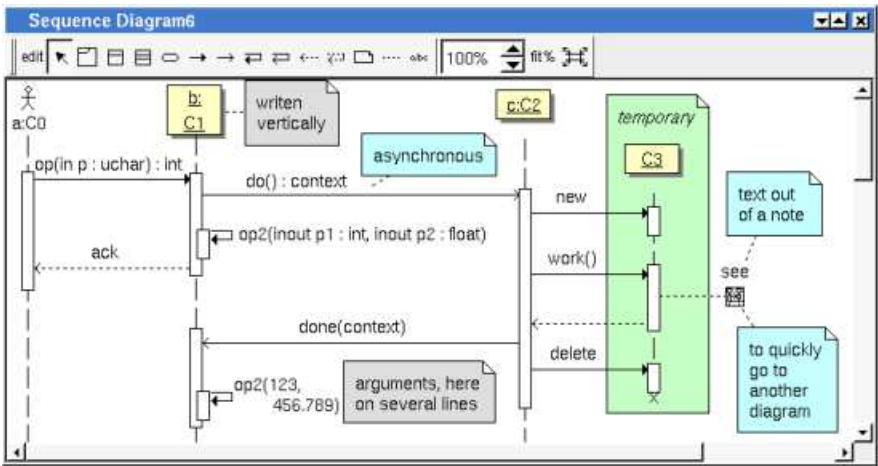
Subject
upper
lower
edit
edit drawing settings
remove from view

---

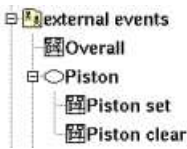
Previous : [deployment node](#)

Next : [sequence diagram](#)

# Sequence diagram



The sequence diagrams may be placed in a [use case view](#) or a [use case](#)



A sequence diagram is created through the *new sequence diagram* entry of the [use case view](#) and [use case](#) browser menus.

## Browser menu

The node menu appearing with a right mouse click on its representation in the browser is something like these, supposing the diagram not *read-only* nor deleted :

Overall
show
edit
edit drawing settings
duplicate
delete
mark
tool ▶

### [Browser Menu](#) : show

*show* allows to show the diagram [picture](#), and edit it in case it is not read only.

### [Browser Menu](#) : edit

*edit* allows to show/modify the diagram properties. In case the diagram is read-only, the fields of the dialog are also read-only.

**sequence diagram dialog, tab Uml**

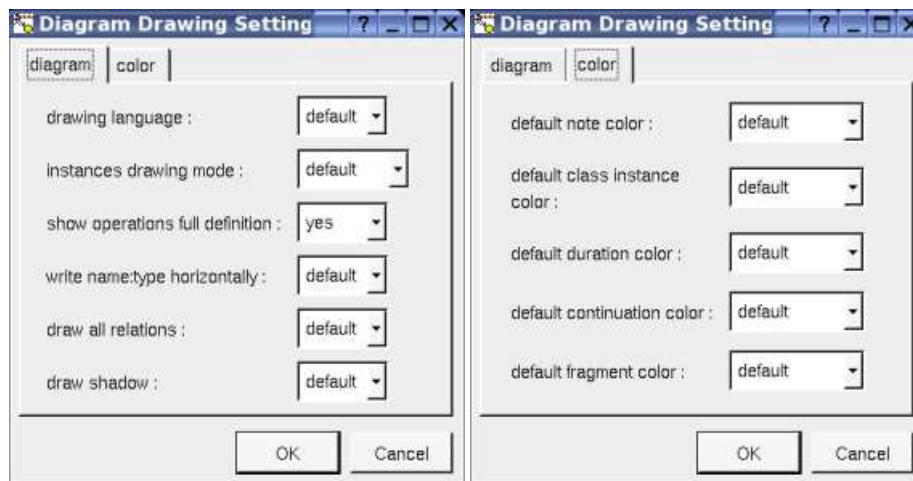


The proposed *stereotypes* are the default one specified through the [Default stereotypes dialog](#) more the current one (empty here). It is possible to choose into the list or to give a new one, or to empty it. The *stereotype* of a diagram doesn't have a special meaning for BOUML.

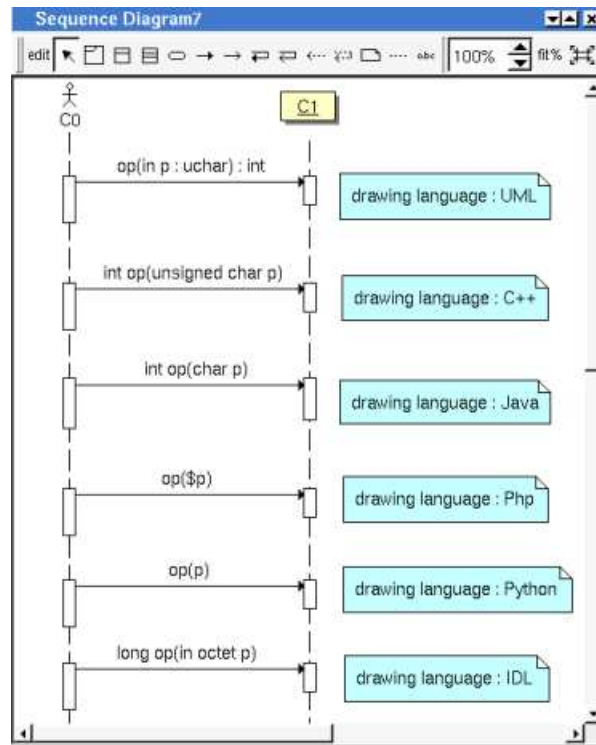
The *editor* button visible above and associated here to the description, allows to edit the description in an other window, or to call an external editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note that this external editor have to create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel with BOUML which looks each second at the file contents to get the new definition until the dialog is closed (I do not like very much polling but this works even QT isn't compiled with the thread support).

### [Browser Menu](#) : edit drawing settings

This dialog allows to specify how the items of the diagram must be drawn by default, you may choose *default* for each value, in this case the effective value if specified in the upper level (which itself may specify *default* etc ... up to the *project* level).



- *drawing*
*language*
 : to indicate in which language the operations must be written when the full definition is showed (see below), like for the [class drawing](#) in a [class diagram](#). For instance, the class *C* has the operation *op* returning an *uchar* (supposed translated *unsigned char* in C++, *char* in Java and *octet* in Idl) and having the *input* parameter *p* having the type *uchar* and only in Idl a second *out* parameter *p2* of the same type. Depending on the *drawing language* :



- *instances drawing mode* : to draw an instance in the standard way or to use one of the icons actor boundary, control or entity.
- *show operations full definition* : to show all the operation profile or just its name (may be with the arguments specified at the operation level in the diagram, see [edit](#))
- *write class instance horizontally* : To write the instance type and name on the same line, or on two lines

### **Browser Menu : duplicate**

Clone the diagram in a new one

### **Browser Menu : delete**

The *delete* entry is only present when the diagram is not *read-only*.

Delete the diagram and all its representation in the opened diagrams. After that it is possible to undelete it (from the browser) until you close the project : obviously the deleted items are not saved !

### **Menu : mark**

See [mark](#)

### **Browser Menu : tools**

The menu entry *tool* is only present in case at least a *plug-out* may be applied on the diagram. The selected tool is called and applied on the current diagram.

## **Picture menu**

The diagram menu appearing with a right mouse click out off any item in the diagram picture is something like these, supposing diagram not *read-only* :

Sequence diagram menu
Transform to flat activity bars
Edit drawing settings
Select diagram in browser
Select all (Ctrl+a)
Find selected browser element
Copy optimal picture part
Copy visible picture part
Save optimal picture part (png)
Save visible picture part (png)
Save optimal picture part (svg)
Save visible picture part (svg)
Optimal scale
Optimal window size
Set preferred size and scale
Set preferred scale (size unset)
Restore preferred size and scale
Unset preferred size and scale
Format
Undo all changes

**Picture Menu : transform to flat activity bars / transform to overlapping activity bars**

Allows to automatically modify the diagram to use flat or overlapping activity bars.

The overlapping level introduced by a reflexive message is propagated to the next messages, you may have to add explicit returns before transforming flat activity bars to overlapping ones. Known problem : the fragments 'alt' are not taken into account transformation, cutting first the activity bars crossing over the separators may help for. The transformation is not undoable, to save the project before may be a good idea !

In a diagram using overlapping activity bars, the reception of a synchronous message create a new overlapping activity bars, this is not the case for the reception of an asynchronous message (but you can ask for to create an overlapping activity bars through the menu of the message).

in a diagram using overlapping activity bars, a reflexive synchronous messages is always at the beginning of an activity bars a reflexive returns at the end, you can't move them.

**Picture Menu : edit drawing settings**

see [edit drawing settings](#).

**Picture Menu : select all**

Select all the items of the current diagram, also done through a *control a*


**Picture Menu : find selected browser element**

To find in the diagram a representation of the elements selected in the browser

**Picture Menu : optimal scale**

similar to the *fit scale* button, change scale to see all the diagram's elements at great as possible

**Picture Menu : optimal window size**

Similar to the button , change the size of the windows to see all the diagram's elements with the current scale

**Picture Menu : copy optimal picture part**

Copy all the diagram elements, equivalent to a *optimal windows size* followed by a *copy visible picture part* then a restore of the original windows size

### **Picture Menu : copy visible part**

Put the visible part of the current diagram in the clipboard (*copy*), it is a screen copy, the current scale have impact on the picture quality.

It is also possible to [print](#) all (not only the visible part) the current diagram on printer or a file through the button

### **Picture Menu : save optimal picture part**

To save all the diagram elements in a file, equivalent to a *optimal windows size* followed by a *save visible picture part* then a restore of the original windows size

### **Picture Menu : save visible picture part**

To save the visible part of the current diagram in a file, it is a screen copy, the current scale have impact on the picture quality.

### **Picture Menu : set preferred size and scale**

To memorize the current diagram sub window size and the current scale. These ones will be used the next time the diagram wil opened.

### **Picture Menu : set preferred scale**

To memorize the current scale. These one will be used the next time the diagram will be opened.

### **Picture Menu : restore preferred and scale**

In case a preferred sub window size and preferred scale was set, follow them.

### **Picture Menu : unset preferred and scale**

To stop to define a preferred size and scale was set

### **Picture Menu : format**

By default the diagram's canvas size is 840 x 1188 points corresponding to an iso A4 format (210\*4 x 297\*4), all the available formats are :

- iso A5 : 592 x 840
- iso A4 : 840 x 1188
- iso A3 : 1188 x 1680
- iso A2 : 1680 x 2376
- iso A1 : 2376 x 3364
- iso A0 : 3364 x 4756
- USA A : 864 x 1116
- USA B : 1116 x 1728
- USA C : 1728 x 2236
- USA D : 2236 x 3456
- USA E : 3456 x 4472

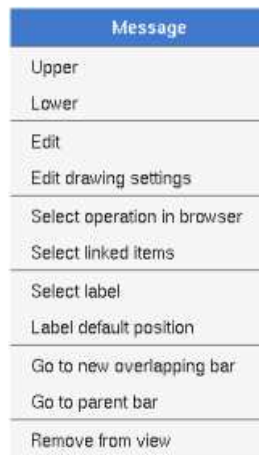
Each format also exists in *landscape*

### **Picture Menu : undo changes**

Undo all the changes made since the diagram was opened the last time. Note that the diagram is restored independently of the th project saving. Obviously the deleted browser items was not restored !

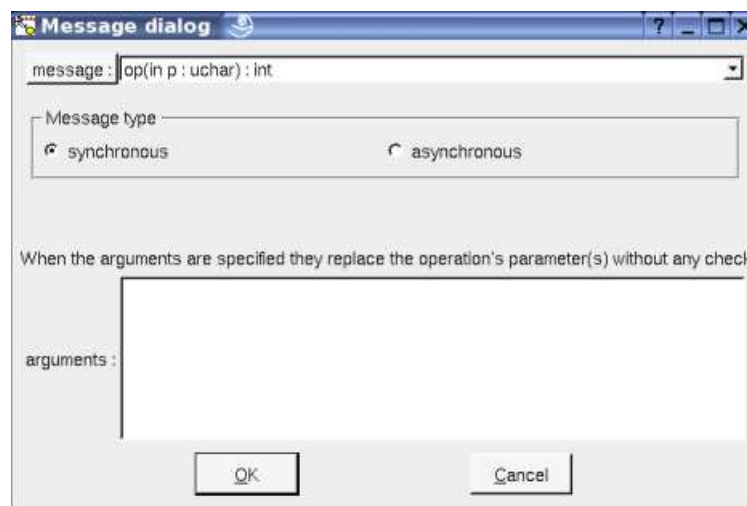
## Operation/message Menu

The menu appearing with a right mouse click on an operation/message arrow is something like these, supposing the diagram not *read-only* :



### Message Menu : edit

*edit* allows to edit the message. In case the diagram is read-only, the fields of the dialog are also read-only

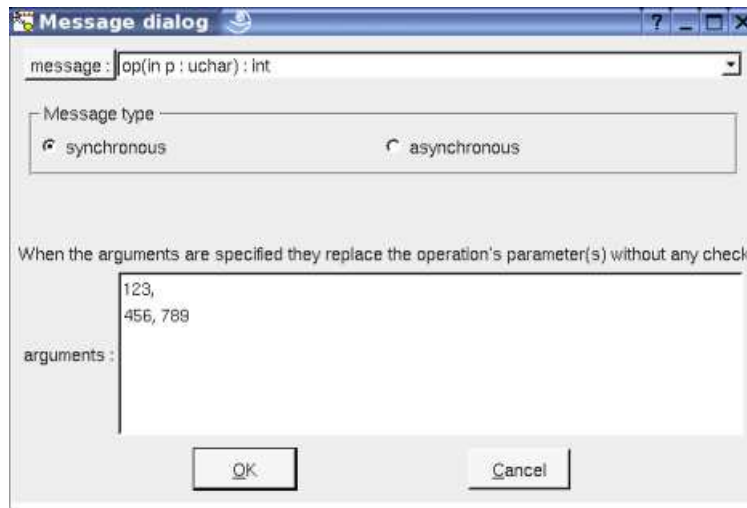


The proposed messages are the operations defined on the message receiver more the current one, but you may also give any o message.

It is possible to change the synchronous/asynchronous state.

It is also possible to specify explicitly the message/operation arguments, is possible to give then on several lines, obviously in this case BOUML produce the appropriate indent :





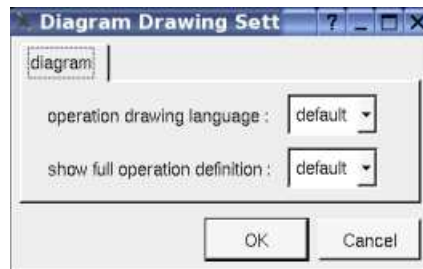
The edition of an explicit return allows to indicate the returned value :



The [Return](#) button [message](#) allows to create a new operation in the class of the instance and to select this new operation. If the current message is an operation this button also allows to select this operation in the browser.

### **Message Menu : edit drawing settings**

This dialog allows to specify how the operation/message must be drawn, you may choose *default* for each value, in this case the effective value if specified in the upper level (which itself may specify *default* etc ... up to the *project* level).



see [operation drawing language](#) and [show full operation definition](#).

### **Message Menu : select linked items**

*Select linked items* select all the items connected this the current one through any line.

### **Message Menu : select label**

Select the label given the associated message/operation

### **Message Menu : label default position**

Move the label given the associated message/operation to its default position.

### **Message Menu : go to new overlapping bar / go to parent bar**

When the diagram use overlapping activity bars, to move the message in a new overlapping bar or in the bar containing receiving/emitting the message

## **Activity bar Menu**

The top and the bottom of an activity bars may be changed using the mouse, clicking down the left mouse button on the top or bottom and moving the mouse. The size of an activity bar is updated when you try to move a message above the top or below the bottom of the bar.

An activity bar can be drawn as a standard activity bar or a coregion.

The menu appearing with a right mouse click on an activity bar rectangle is something like these, supposing the diagram not *read-only* :

Activity bar		Activity bar
Upper		Upper
Lower		Lower
Draw as a coregion		Draw as activity bar
Edit drawing settings		Edit drawing settings
Select linked items		Select linked items
Remove from view		Remove from view
Cut here		Cut here
Merge juxtaposed activity bars		Merge juxtaposed activity bars
Collapse in parent bar		

**Message Menu : edit drawing settings**

This dialog allows to specify the color of the activity duration rectangle.

**Message Menu : select linked items**

*Select linked items* select all the items connected this the current one through any line.

**Message Menu : cut here**

Cut the activity bar at the mouse place to obtain two activity bars, resize the bar when there is no message above or below the position (an empty activity bar is not created)

**Message Menu : merge juxtaposed activity bars**

When a least two activity bars (of the same life line) are juxtaposed, replace these ones by only one activity bar.

**Message Menu : collapse in parent bars**

Available when the bar in overlapped, to remove it moving the messages in the bar containing the current bar.

---

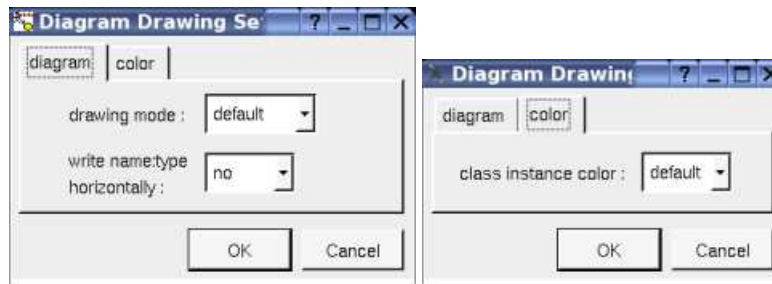
**Class instance Menu**

The menu appearing with a right mouse click on a class [instance](#) is something like these, supposing the diagram not *read-only* :

b:C1	C3
Upper	Upper
Lower	Lower
Edit	Edit
Become mortal	Become immortal
Edit drawing settings	Edit drawing settings
Select class in browser	Select in browser
Insert in model	Select class in browser
Replace it	Exit from model
Remove from view	Remove from view
	Delete from model

**Message Menu : edit drawing settings**

This dialog allows to specify how a the class instance must be drawn, you may choose *default* for each value, in this case the effective value if specified in the upper level (which itself may specify *default* etc ... up to the *project* level).





*write class instance horizontally* : To write the instance type and name on the same line, or on two lines.

### **Message Menu : become mortal / immortal**

By default an instance is immortal : its life line is infinite, else a X is automatically add under the last activity bar and the life line is finite. Note that it is possible to add a new message below the end of the instance life, the position of the end of life is updated, this is done also when you move/resize/delete the last activity bar.

Note that the vertical position of the instance is not fixed, to help you by default an instance is inserted on the top of the picture, use the mouse to move it horizontally or vertically.

### **Message Menu : Insert in model / exit from model**

There are two kinds of class instances : [instances](#) part of the model (visible in the browser), and pure graphic instances (to not pollute the model with many useless instances). The button  add or create an instance part of the model. The button  add or create a pure graphical instances. But after that you are able to transform a pure graphical instance to a modeled one, or to replace the drawing of a modeled instance by a pure graphic one (the modeled class instance itself is unchanged).

### **Message Menu : Replace it**

This entry appears on a pure graphical instance and allows to replace it by a modeled instance of the same class.

## **Continuation Menu**

The size of an activity bar is updated when it is renamed and the name is too long to be written inside the drawing, else the continuation drawing may be resized by hand.

The menu appearing with a right mouse click on a continuation is something like these, supposing the diagram not *read-only* :



### **Message Menu : edit drawing settings**

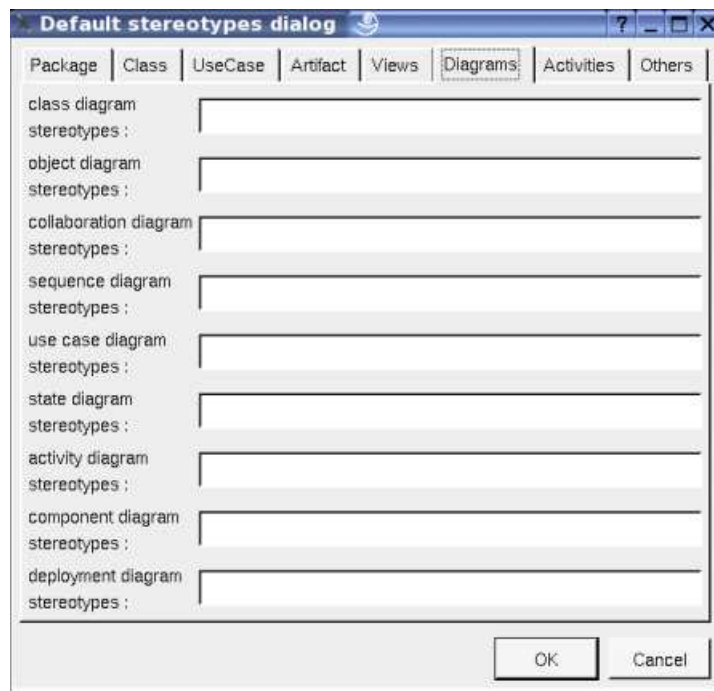
This dialog allows to specify the color of the continuation ellipsis

### **Message Menu : select linked items**

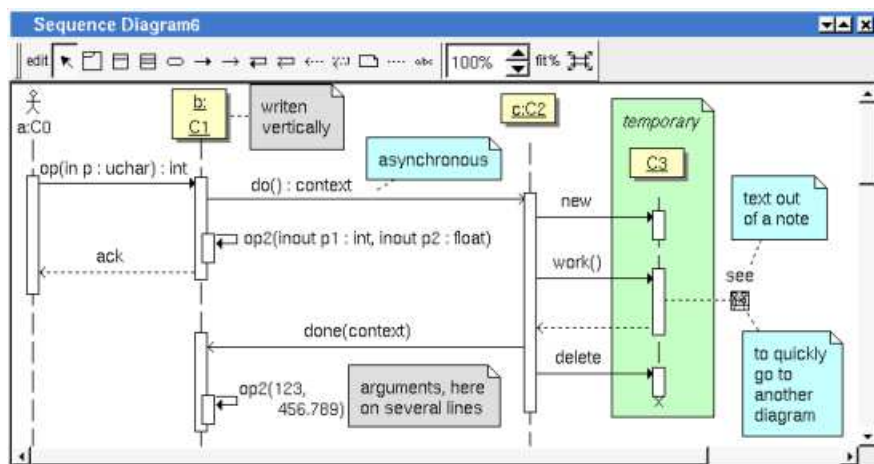
*Select linked items* select all the items connected this the current one through any line.

## **Default stereotypes**

The [dialog](#) allowing to set the default stereotypes has a tab reserved for the diagrams :



## Drawing



A sequence diagram may contain [actors](#), [classes instances](#) (part of the model or only graphical), [fragment](#), [notes](#), [texts](#), [diagram icons](#) and the operations/messages.

To place these elements in the diagram, use the buttons on the top (to create a new element or choose from the already defined one) then click at the appropriate position in the diagram, or *grab* the element from the browser into the diagram.

To add an operation/message select the appropriate button (the synchronous/asynchronous state may be changed through operation/message edit menu), click on the sender with the left mouse button, except for the self operation/message move the mouse to the receiver maintaining the mouse button down, then mouse button up.

BOUML doesn't fix the vertical size of the rectangle representing the activity duration, you are the master ! To do this click down the left mouse button on the top or bottom and move the mouse. The size of an activity bar is updated when you try to move a message above the top or below the bottom of the bar. When you move vertically a message/operation out of its current activity duration rectangle, a new one is automatically added is needed.

The *scale* may be changed from 30 up to 200 %. To allow to see the operation/message kinds even with a small scale, the size of the unmodified.

*fit%* allows to set the largest scale allowing to show all the picture, as it is possible. Because of the scrollbar management, hitting several times on *fit%* may give several scales.

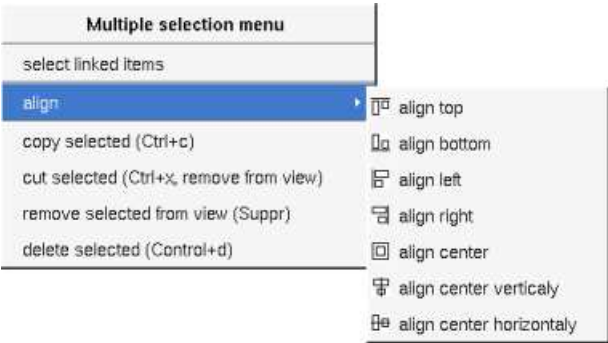
Classically, the left mouse click may be used to select an element, or several when the *control* key is down, a multiple selection may also be done defining a region using the mouse left click out of any element and moving the mouse with the click and maintaining the

click down. The diagram's elements may be moved after a left mouse click on an element and maintaining the click down during mouse move. The right mouse click is used to show context dependent menus.

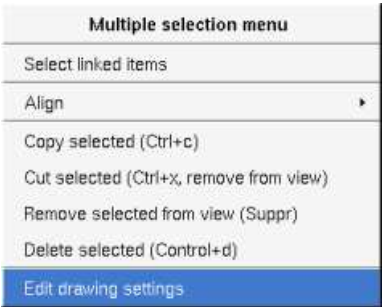
- control s* is a shortcut to save the project
- control a* is a shortcut to select all the diagram's elements
- control d* is a shortcut to delete the selected element(s), the model is affected
- suppr* is a shortcut to remove the selected element(s) from the diagram, these elements are not deleted
- control c* is a shortcut to copy the selected diagram's elements
- control x* is a shortcut to cut (remove from view) the selected diagram's elements
- control v* is a shortcut to paste the selected diagram's elements
- control z* and *control u* are a shortcut of *undo*
- control r* and *control y* are a shortcut of *redo*
- the keyboard arrows allow to move the selected items

You can define your own shortcut using the [shortcut editor](#) through the *Miscellaneous* menu.

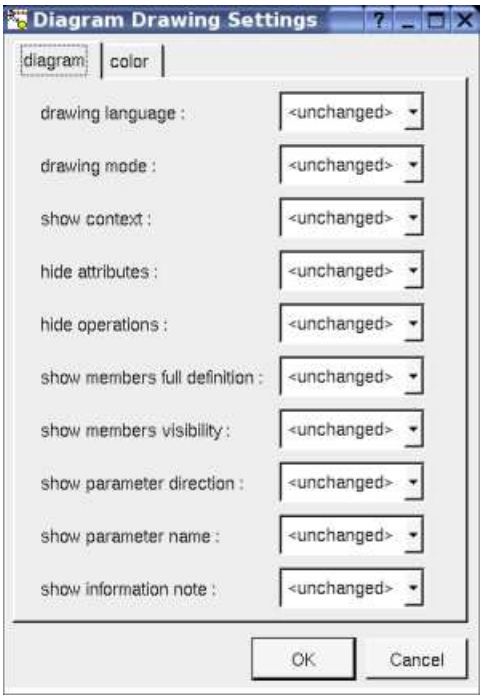
When several diagram's elements are selected, the menu appearing on a left mouse click allows to align them :



If all the selected elements has the same kind (for instance all are a class), you edit their *drawing settings* globally :



and for instance when the elements are classes in a class diagram :



The settings valuing *<unchanged>* are not changed for the selected elements, in the other case the same value is set for all the elements.

**Fragment**

A *fragment* is resizable. To change the fragment's name choose the *edit* entry of its menu or double click on it

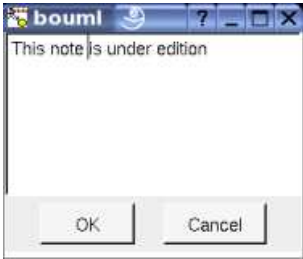
A right mouse click on a note show the following menu :



*Edit drawing settings* allows to change the fill color

**Note**

A *note* may contain any text, a note is resizable. To change the text of a note choose the *edit* entry of its menu or double click on it, this open a window containing the text, close the window to end the edition :



A right click on the edited note calls a dialog allowing some text manipulations thanks Qt.

A right mouse click on a note show the following menu :

Note
Upper
Lower
Edit
Color of text
Font
Edit drawing settings
Select linked items
Remove from view

The *font* may be changed, by default the font is the one used to write the other texts.

The color of the text may be changed, a text is black by default.

*Edit drawing settings* allows to change the fill color

*Select linked items* select all the items connected this the current one through any line, in the previous picture this select all except the package, the text “*upper diagram :*” and the use case diagram icon.

### Text

*Text* is another way to write a text, contrary to the note a *text* is written alone (see “*upper diagram :*” on the picture below), by default a text contains *blabla*: a text cannot be empty. The edition of a text is done like for a note. A text is resizeable, a resize is not automatically done when the contains is modified.

A right mouse click on a *text* show the following menu :

Text
Upper
Lower
Edit
Color
Font
Select linked items
Remove from view

### Diagram icon

A diagram icon is a shortcut to a diagram, this allows to quickly navigate through the diagram without limitation (a actor or some other item kinds may be associated to only one diagram). The used icon indicates the diagram kind, but its name is not written.

A double mouse left click on a diagram icon opens the diagram, the *open* entry of the diagram icon menu does the same thing.

A right mouse click on a diagram icon show the following menu :

Overall short cut
upper
lower
open
select diagram in browser
select linked items
remove from view

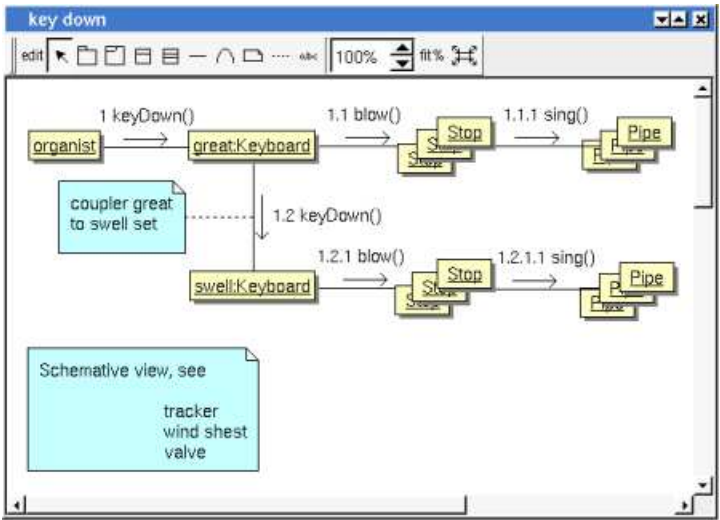
The other diagram's elements have an own chapter, refer to them.

Previous : [use case diagram](#)

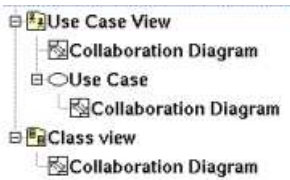
Next : [collaboration diagram](#)



# Collaboration diagram



The collaboration diagrams may be placed in a [use case view](#), a [use case](#) or a [class view](#)



A collaboration diagram is created through the *new collaboration diagram* entry of the [use case view](#), [use case](#) and [class view](#) browser menus..

## Browser menu

The node menu appearing with a right mouse click on its representation in the browser is something like these, supposing the diagram not *read-only* nor deleted :

key down
show
edit
edit drawing settings
duplicate
delete
mark
tool

### [Browser Menu](#) : show

*show* allows to show the diagram [picture](#), and edit it in case it is not read only.

### [Browser Menu](#) : edit

*edit* allows to show/modify the diagram properties. In case the diagram is read-only, the fields of the dialog are also read-only.

**collaboration diagram dialog, tab Uml**



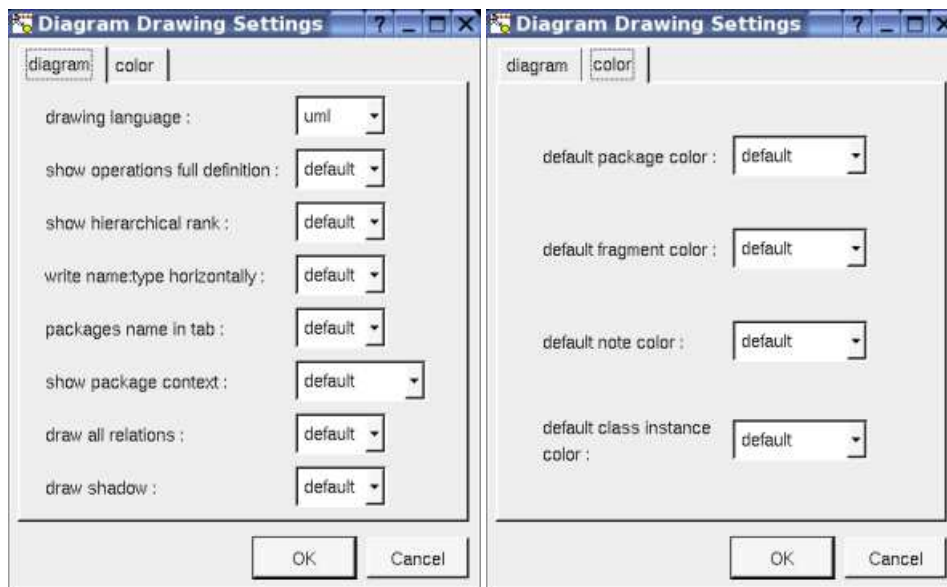


The proposed *stereotypes* are the default one specified through the [Default stereotypes dialog](#) more the current one (empty here). It is possible to choose into the list or to give a new one, or to empty it. The *stereotype* of a diagram doesn't have a special meaning for BOUML.

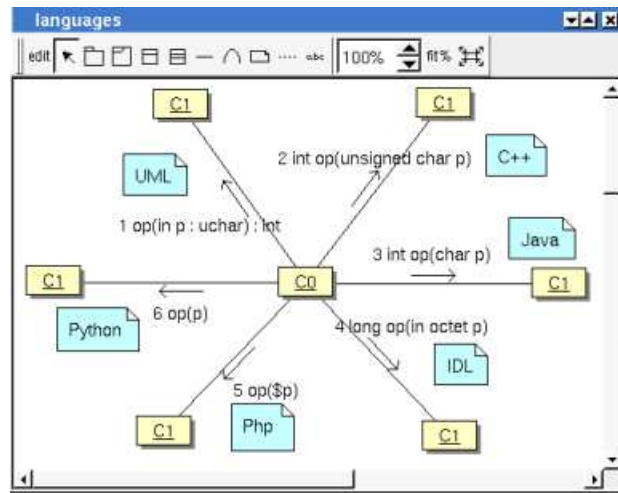
The *editor* button visible above and associated here to the description, allows to edit the description in an other window, or to call an external editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note that this external editor have to create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel with BOUML which looks each second at the file contents to get the new definition until the dialog is closed (I do not like very much polling but this works even QT isn't compiled with the thread support).

### **Browser Menu : edit drawing settings**

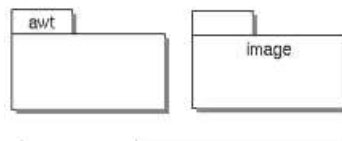
This dialog allows to specify how the items of the diagram must be drawn by default, you may choose *default* for each value, in this case the effective value if specified in the upper level (which itself may specify *default* etc ... up to the *project* level).



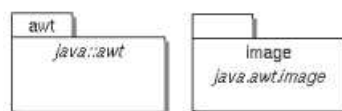
- drawing*** *language*  
 : to indicate in which language the operations must be written when the full definition is showed (see below), like for the [class drawing](#) in a [class diagram](#). For instance, the class *C* has the operation *op* returning an *int* (supposed translated *int* in C++ and Java, *octet* in Idl) and having the *input* parameter *p* having the type *uchar* (supposed translated *unsigned char* in C++, *char* in Java and *octet* in Idl). Depending on the *drawing language* :



- *show* *operations* *full* *definition*  
: to show all the operation profile or just its name (may be with the arguments specified at the operation level in the diagram, see [edit](#))
- *show* *hierarchical* *rank*  
: a message may be numbered using its rank (a simple number) or its hierarchical rank (numbers separated with dot)
- *write class instance horizontally* : To write the instance type and name on the same line, or on two lines
- *package* *name* *in* *tab*  
: to indicate if the package's name must be written in the tab which is the small rectangle on the top (note : a transparent package doesn't have shadow) :



- *show* *packages* *context*  
: to indicate if the context where the package is defined must be written, it is not the case just above. The context may be the "UML context" which is the path of the package in the browser (choice followed for *awt* below), or the C++ namespace specified by the package, or the Java package specified by the package (choice followed for *picture* below) or at least the Idl module :



## Browser Menu : duplicate

Clone the diagram in a new one

## Browser Menu : delete

The *delete* entry is only present when the diagram is not *read-only*.

Delete the diagram and all its representation in the opened diagrams. After that it is possible to undelete it (from the browser) until you close the project : obviously the deleted items are not saved !

## Menu : mark

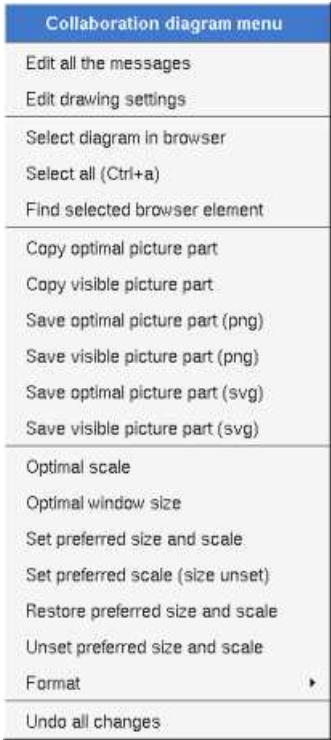
See [mark](#)

## Browser Menu : tools

The menu entry *tool* is only present in case at least a [plug-out](#) may be applied on the diagram. The selected tool is called and applied on the current diagram.

## Picture menu

The diagram menu appearing with a right mouse click out off any item in the diagram picture is something like these, supposin diagram not *read-only* :



### Picture Menu : edit all the messages

To edit all the messages appearing in the diagram, to change the messages themselves or their numbering (recursively or not), or delete some of them (recursively or not) :

Collaboration diagram message dialog					
Rank	Hierarchical rank	From	Message	To	*
1	1	:organist	keyDown()	greatkeyboard	*
2	1.1	greatkeyboard	blow()	:stop	*
3	1.1.1	:stop	sing()	:pipe	*
4	1.2	greatkeyboard	keyDown()	swell.keyboard	*
5	1.2.1	swell.keyboard	blow()	:stop	*
6	1.2.1.1	:stop	sing()	:pipe	*

The proposed messages are the operations defined on the message receiver more the current one, but you may also give any o message.

A mouse click on the appropriate line and column allows to modify the appropriate cell.

### Picture Menu : edit drawing settings

see [edit drawing settings](#).

### Picture Menu : select all

Select all the items of the current diagram, also done through a *control a*

### Picture Menu : find selected browser element

To find in the diagram a representation of the elements selected in the browser

### Picture Menu : optimal scale

similar to the *fit scale* button, change scale to see all the diagram's elements at great as possible

### **Picture Menu : optimal window size**

Similar to the button , change the size of the windows to see all the diagram's elements with the current scale

### **Picture Menu : copy optimal picture part**

Copy all the diagram elements, equivalent to a *optimal windows size* followed by a *copy visible picture part* then a restore of the original windows size

### **Picture Menu : copy visible part**

Put the visible part of the current diagram in the clipboard (*copy*), it is a screen copy, the current scale have impact on the picture quality.

It is also possible to [print](#) all (not only the visible part) the current diagram on printer or a file through the button

### **Picture Menu : save optimal picture part**

To save all the diagram elements in a file, equivalent to a *optimal windows size* followed by a *save visible picture part* then a restore of the original windows size

### **Picture Menu : save visible picture part**

To save the visible part of the current diagram in a file, it is a screen copy, the current scale have impact on the picture quality.

### **Picture Menu : set preferred size and scale**

To memorize the current diagram sub window size and the current scale. These ones will be used the next time the diagram will opened.

### **Picture Menu : set preferred scale**

To memorize the current scale. These one will be used the next time the diagram will be opened.

### **Picture Menu : restore preferred size and scale**

In case a preferred sub window size and preferred scale was set, follow them.

### **Picture Menu : unset preferred and scale**

To stop to define a preferred size and scale was set

### **Picture Menu : format**

By default the diagram's canvas size is 840 x 1188 points corresponding to an iso A4 format (210\*4 x 297\*4), all the available formats are :

- iso A5 : 592 x 840
- iso A4 : 840 x 1188
- iso A3 : 1188 x 1680
- iso A2 : 1680 x 2376
- iso A1 : 2376 x 3364
- iso A0 : 3364 x 4756
- USA A : 864 x 1116
- USA B : 1116 x 1728
- USA C : 1728 x 2236
- USA D : 2236 x 3456
- USA E : 3456 x 4472

Each format also exists in *landscape*

### Picture Menu : undo changes

Undo all the changes made since the diagram was opened the last time. Note that the diagram is restored independently of the project saving. Obviously the deleted browser items was not restored !

## Operation/message Menu

The menu appearing with a right mouse click on link is something like these, supposing the diagram not *read-only* :



The two first entries allows to add a message in each direction, you have to choose the new message rank among the proposed ones but you may change it after with [edit its message](#) or [edit all the messages](#). For instance in case a new message is added from the *keyboard* *great* to a *stop* (see the [first picture](#)), the proposed ranks are :



### Message Menu : edit its message

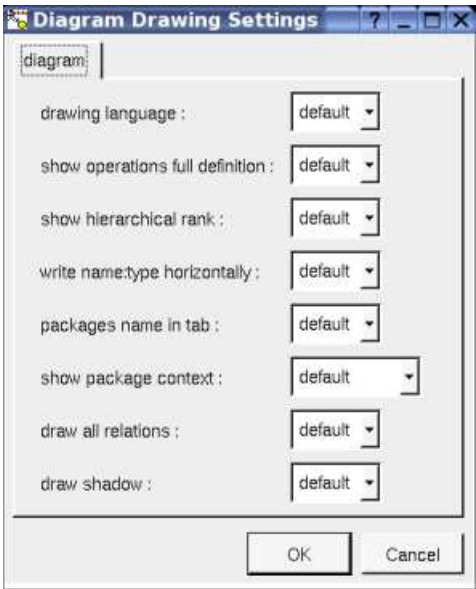
*edit* allows to edit the message of the link, or delete some of them (recursively or not), see [edit](#).

### Message Menu : edit all the messages

To edit all the messages appearing in the diagram, to change the messages themselves or their numbering (recursively or not), or delete some of them (recursively or not), see [edit](#).

### Message Menu : edit drawing settings

This dialog allows to specify how the operation/message must be drawn, you may choose *default* for each value, in this case the effective value if specified in the upper level (which itself may specify *default* etc ... up to the *project* level).



see [operation drawing language](#) and [show full operation definition](#).

**Message Menu : select linked items**

*Select linked items* select all the items connected this the current one through any line.

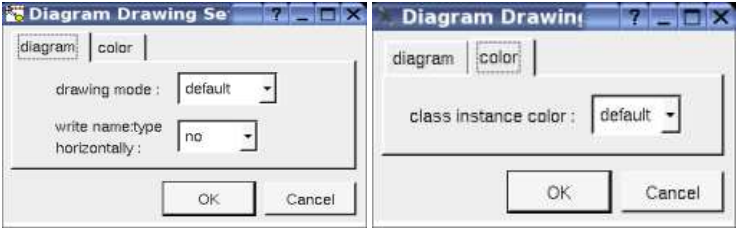
**Class instance Menu**

The menu appearing with a right mouse click on a class [instance](#) is something like these, supposing the diagram not *read-only* :

great:Keyboard	swell:Keyboard
Upper	Upper
Lower	Lower
Edit drawing settings	Edit drawing settings
Edit	Edit
Select class in browser	Select in browser
Select linked items	Select class in browser
Insert in model	Select linked items
Replace it	Exit from model
Remove from view	Remove from view
	Delete from model



**Message Menu : edit drawing settings**

This dialog allows to specify how a the class instance must be drawn, you may choose *default* for each value, in this case the effective value if specified in the upper level (which itself may specify *default* etc ... up to the *project* level).



*write class instance horizontaly* : To write the instance type and name on the same line, or on two lines.

**Message Menu : Insert in model / exit from model**

There are two kinds of class instances : [instances](#) part of the model (visible in the browser), and pure graphic instances (to not pollute the model with many useless instances). The button  add or create an instance part of the model. The button  add or create a pure graphical instances. But after that you are

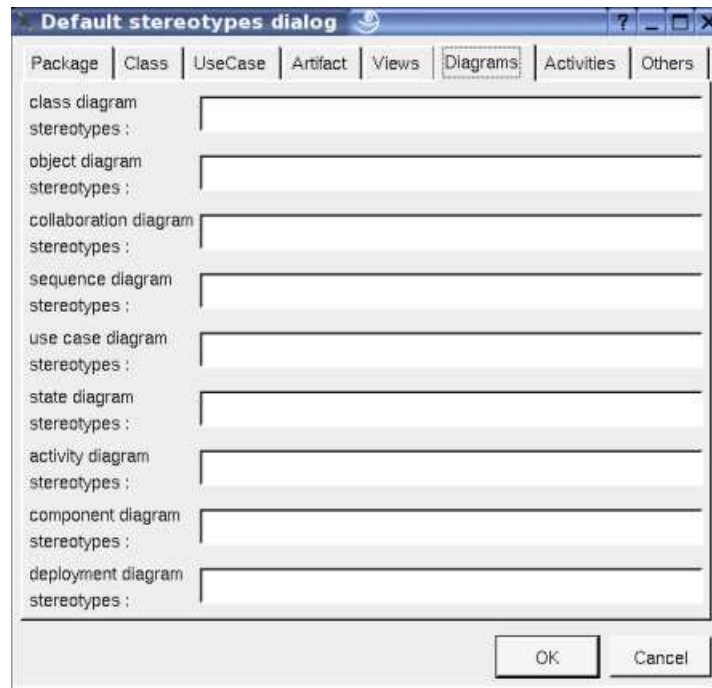
able to transform a pure graphical instance to a modeled one, or to replace the drawing of a modeled instance by a pure graphic one (the modeled class instance itself is unchanged).

### Message Menu : Replace it

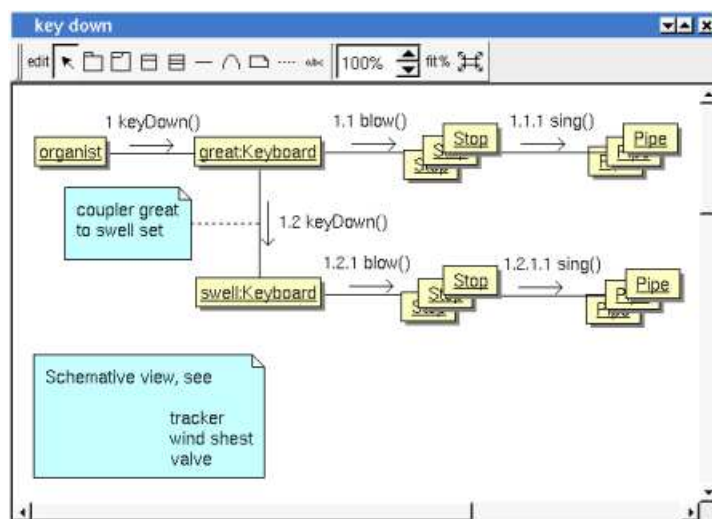
This entry appears on a pure graphical instance and allows to replace it by a modeled instance of the same class.

## Default stereotypes

The [dialog](#) allowing to set the default stereotypes has a tab reserved for the diagrams :



## Drawing



A collaboration diagram may contain [actors](#), [classes instances](#), [notes](#), [texts](#), [diagram icons](#), [packages](#) and the operations/messages.

To place these elements in the diagram, use the buttons on the top (to create a new element or choose from the already defined one) then click at the appropriate position in the diagram, or *grab* the element from the browser into the diagram.

To add an operation/message on an already existing link, double mouse click on it or use the [menu](#) appearing though a right mouse click.

The *scale*  
may be changed from 30 up to 200 %. To allows to see the operation/message kinds even with a small scale, the size of the



unmodified.

*fit%* allows to set the largest scale allowing to show all the picture, as it is possible. Because of the scrollbar management, hitting several times on *fit%* may give several scales.

Classically, the left mouse click may be used to select an element, or several when the *control* key is down, a multiple selection may also be done defining a region using the mouse left click out of any element and moving the mouse with the click and maintaining the click down. The diagram's elements may be moved after a left mouse click on an element and maintaining the click down during mouse move. The right mouse click is used to show context dependent menus.

*control s* is a shortcut to save the project

*control a* is a shortcut to select all the diagram's elements

*control d* is a shortcut to delete the selected element(s), the model is affected

*suppr* is a shortcut to remove the selected element(s) from the diagram, these elements are not deleted

*control c* is a shortcut to copy the selected diagram's elements

*control x* is a shortcut to cut (remove from view) the selected diagram's elements

*control v* is a shortcut to paste the selected diagram's elements

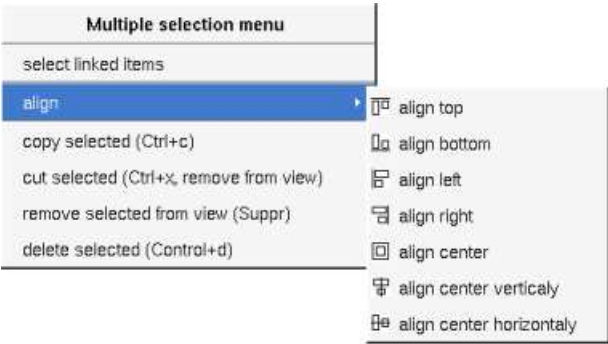
*control z* and *control u* are a shortcut of *undo*

*control r* and *control y* are a shortcut of *redo*

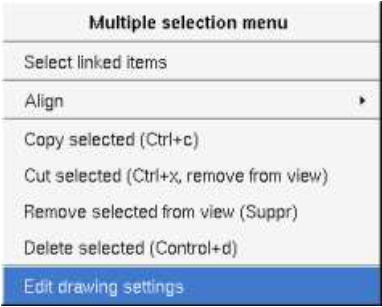
the keyboard arrows allow to move the selected items

You can define your own shortcut using the [shortcut editor](#) through the *Miscellaneous* menu.

When several diagram's elements are selected, the menu appearing on a left mouse click allows to align them :

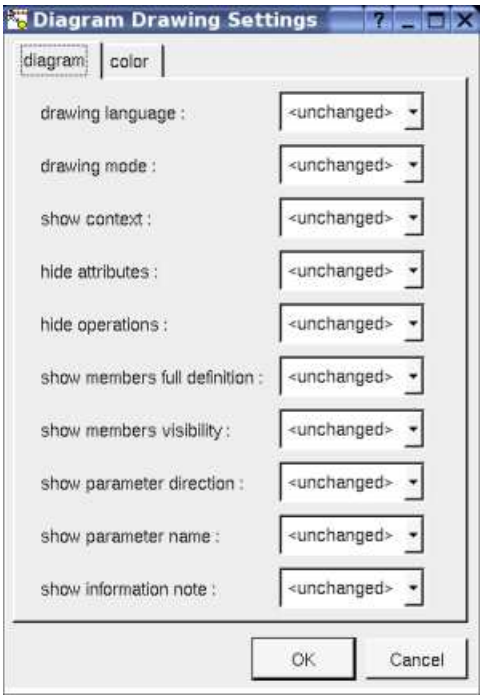


If all the selected elements has the same kind (for instance all are a class), you edit their *drawing settings* globally :



and for instance when the elements are classes in a class diagram :





The settings valuing *<unchanged>* are not changed for the selected elements, in the other case the same value is set for all the elements.

**Note**

A *note* may contain any text, a note is resizable. To change the text of a note choose the *edit* entry of its menu or double click on it, this open a window containing the text, close the window to end the edition :



A right click on the edited note calls a dialog allowing some text manipulations thanks Qt.

A right mouse click on a note show the following menu :



The *font* may be changed, by default the font is the one used to write the other texts.

The color of the text may be changed, a text is black by default.

*Edit drawing settings* allows to change the fill color

*Select linked items* select all the items connected this the current one through any line, in the previous picture this select all except the package, the text “upper diagram :” and the use case diagram icon.

**Text**

*Text* is another way to write a text, contrary to the note a *text* is written alone (see “*upper diagram* :” on the picture below), by default a text contains *blabla*: a text cannot be empty. The edition of a text is done like for a note. A text is resizeable, a resize is not automatically done when the contains is modified.

A right mouse click on a *text* show the following menu :

Text
Upper
Lower
Edit
Color
Font ▶
Select linked items
Remove from view

**Diagram icon**

A diagram icon is a shortcut to a diagram, this allows to quickly navigate through the diagram without limitation (a actor or some other item kinds may be associated to only one diagram). The used icon indicates the diagram kind, but its name is not written.

A double mouse left click on a diagram icon opens the diagram, the *open* entry of the diagram icon menu does the same thing.

A right mouse click on a diagram icon show the following menu :

Overall short cut
upper
lower
open
select diagram in browser
select linked items
remove from view

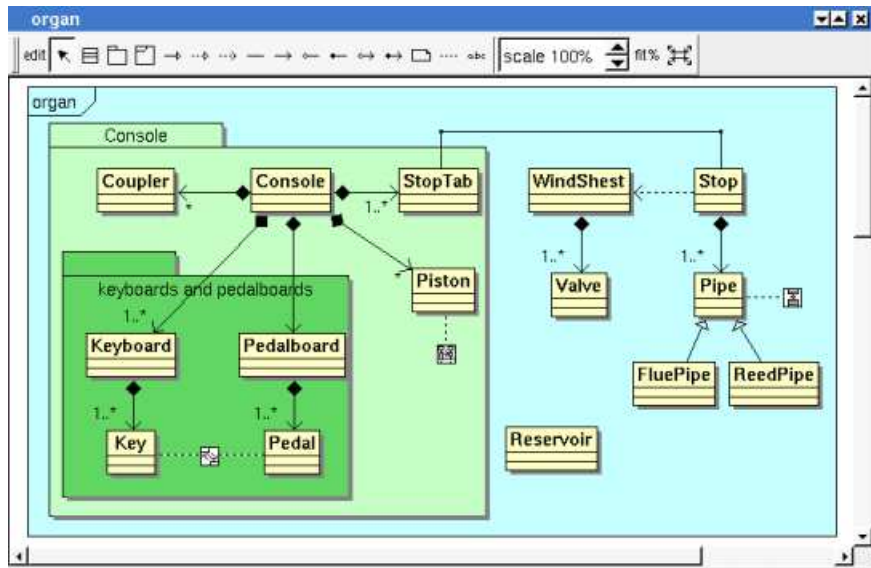
The other diagram's elements have an own chapter, refer to them.

---

Previous : [sequence diagram](#)

Next : [class diagram](#)

# Class diagram



The class diagrams may be placed in a [class view](#)



A class diagram is created through the *new static class diagram* entry of the [class view](#) browser menus.

## Browser menu

The node menu appearing with a right mouse click on its representation in the browser is something like these, supposing the diagram not *read-only* nor deleted :

organ
show
edit
edit drawing settings
duplicate
delete
mark
tool ▶

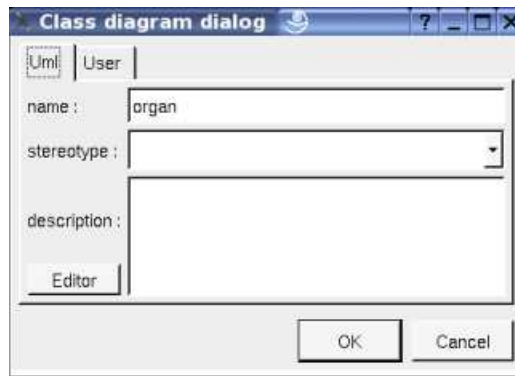
### [Browser Menu](#) : show

*show* allows to show the diagram [picture](#), and edit it in case it is not read only.

### [Browser Menu](#) : edit

*edit* allows to show/modify the diagram properties. In case the diagram is read-only, the fields of the dialog are also read-only.

class diagram dialog, tab Uml

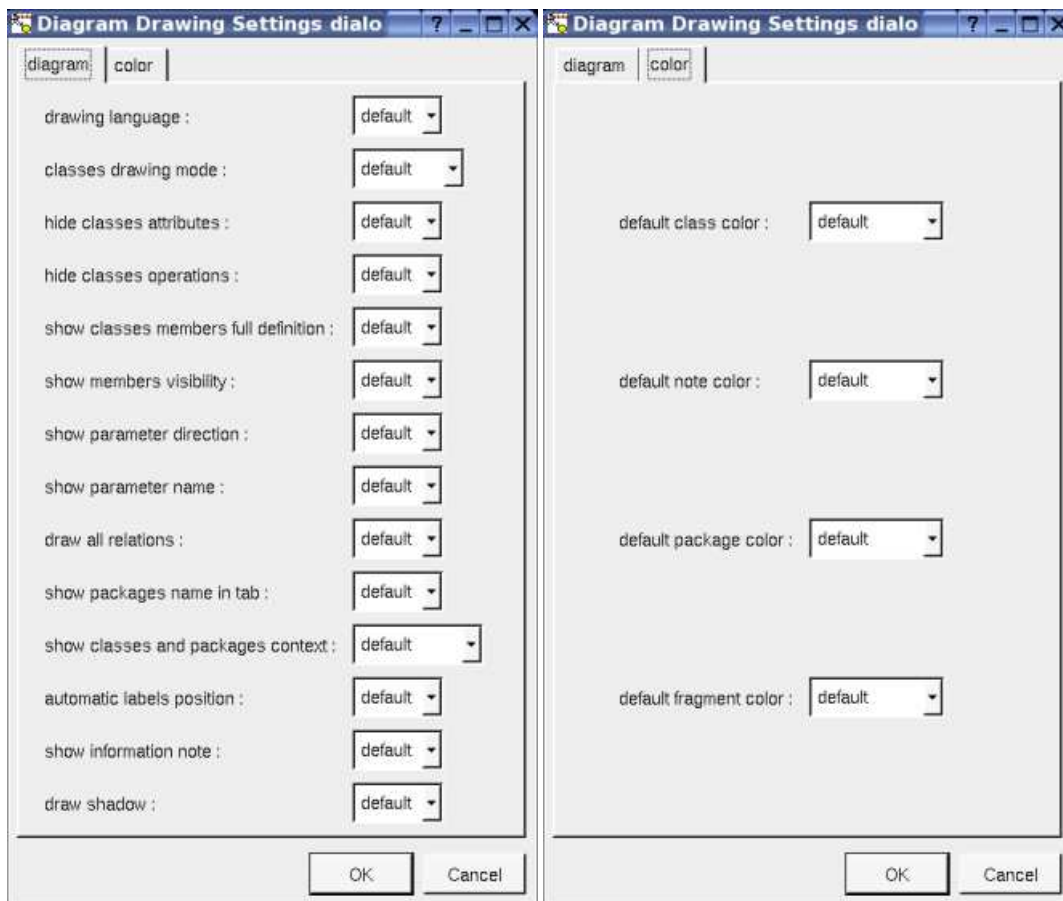


The proposed *stereotypes* are the default one specified through the [Default stereotypes dialog](#) more the current one (empty here). It is possible to choose into the list or to give a new one, or to empty it. The *stereotype* of a diagram doesn't have a special meaning for BOUML.

The *editor* button visible above and associated here to the description, allows to edit the description in an other window, or to call an external editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note that this external editor have to create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel with BOUML which looks each second at the file contents to get the new definition until the dialog is closed (I do not like very much polling but this works even QT isn't compiled with the thread support).

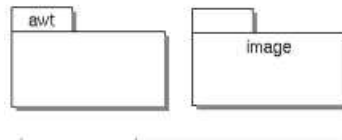
### **Browser Menu : edit drawing settings**

This dialog allows to specify how the items of the diagram must be drawn by default, you may choose *default* for each value, in this case the effective value if specified in the upper level (which itself may specify *default* etc ... up to the *project* level).



- *drawing language* : to indicate in which language the operations must be written when the full definition is showed (see below), see [class drawing](#) in a class diagram.
- *classes drawing mode* : to draw the class using a rectangle or an icon, see [drawing mode](#) in a class diagram.
- *hide classes attributes* : to hide or not the attributes, it is also possible to [specify](#) the visibility for each one.

- *hide classes operations* : to hide or not the operations, it is also possible to [specify](#) the visibility for each one.
- *show classes member full definition* : to show all the attributes and operation profile or just their name
- *draw* *all* *classes* *relations*  
: to automatically draw or not the new relations, obviously a relation will be added in a diagram only when the start and end classes (may be the same) are drawn.
- *package* *name* *in* *tab*  
: to indicate if the package's name must be written in the tab which is the small rectangle on the top (note : a transparent package doesn't have shadow) :



- *show* *classes* *and* *packages* *context*  
: To indicate if the context where the class/package is defined must be written, and if yes, how. The context may be the "context" which is the path of the class/package in the browser, or the C++ *namespace*, or the Java *package*, or the Python *package* or at least the Idl *module*. See [class context](#) and [package context](#).
- *automatic* *labels* *position*  
: To ask BOUML to move or to not move the associated labels on the relations in a diagram when the relation is edited associated class pictures moved etc ...

## **Browser Menu : duplicate**

Clone the diagram in a new one

## **Browser Menu : delete**

The *delete* entry is only present when the diagram is not *read-only*.

Delete the diagram and all its representation in the opened diagrams. After that it is possible to undelete it (from the browser) until you close the project : obviously the deleted items are not saved !

## **Menu : mark**

See [mark](#)

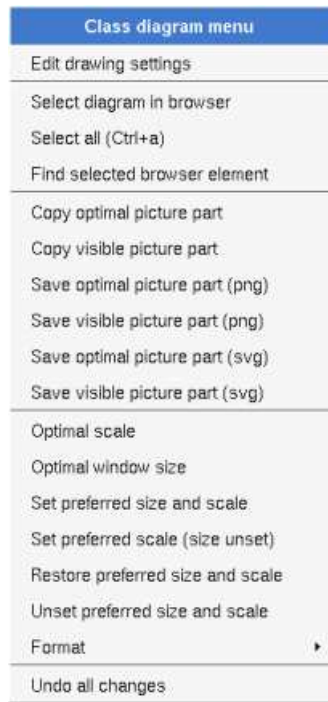
## **Browser Menu : tools**

The menu entry *tool* is only present in case at least a [plug-out](#) may be applied on the diagram. The selected tool is called and applied on the current diagram.

---

## **Picture menu**

The diagram menu appearing with a right mouse click out off any item in the diagram picture is something like these, supposing diagram not *read-only* :



### **Picture Menu : add classes of the selected class view**

Only appears when a [class view](#) is selected in the browser, all the classes of the *class view* whose are not yet present in the diagram are added in the diagram, allowing to not have to do several *drag & drop* from the browser to ass these classes, but the position of the classes is probably not the good one !

### **Picture Menu : edit drawing settings**

see [edit drawing settings](#).

### **Picture Menu : select all**

Select all the items of the current diagram, may also be done through *control a*

### **Picture Menu : find selected browser element**

To find in the diagram a representation of the elements selected in the browser

### **Picture Menu : optimal scale**

Similar to the *fit scale* button, change scale to see all the diagram's elements at great as possible

### **Picture Menu : optimal window size**

Similar to the button , change the size of the windows to see all the diagram's elements with the current scale

### **Picture Menu : copy optimal picture part**

Copy all the diagram elements, equivalent to a *optimal windows size* followed by a *copy visible picture part* then a restore of the original windows size

### **Picture Menu : copy visible picture part**

Put the visible part of the current diagram in the clipboard (*copy*), it is a screen copy, the current scale have impact on the picture quality.

It is also possible to [print](#) all (not only the visible part) the current diagram on printer or a file through the button

### **Picture Menu : save optimal picture part**

To save all the diagram elements in a file, equivalent to a *optimal windows size* followed by a *save visible picture part* then a restore of the original windows size

### **Picture Menu : save visible picture part**

To save the visible part of the current diagram in a file, it is a screen copy, the current scale have impact on the picture quality.

### **Picture Menu : set preferred size and scale**

To memorize the current diagram sub window size and the current scale. These ones will be used the next time the diagram will opened.

### **Picture Menu : set preferred scale**

To memorize the current scale. These one will be used the next time the diagram will be opened.

### **Picture Menu : restore preferred size and scale**

In case a preferred sub window size and preferred scale was set, follow them.

### **Picture Menu : unset preferred and scale**

To stop to define a preferred size and scale was set

### **Picture Menu : format**

By default the diagram's canvas size is 840 x 1188 points corresponding to an iso A4 format (210\*4 x 297\*4), all the available formats are :

- iso A5 : 592 x 840
- iso A4 : 840 x 1188
- iso A3 : 1188 x 1680
- iso A2 : 1680 x 2376
- iso A1 : 2376 x 3364
- iso A0 : 3364 x 4756
- USA A : 864 x 1116
- USA B : 1116 x 1728
- USA C : 1728 x 2236
- USA D : 2236 x 3456
- USA E : 3456 x 4472

Each format also exists in *landscape*

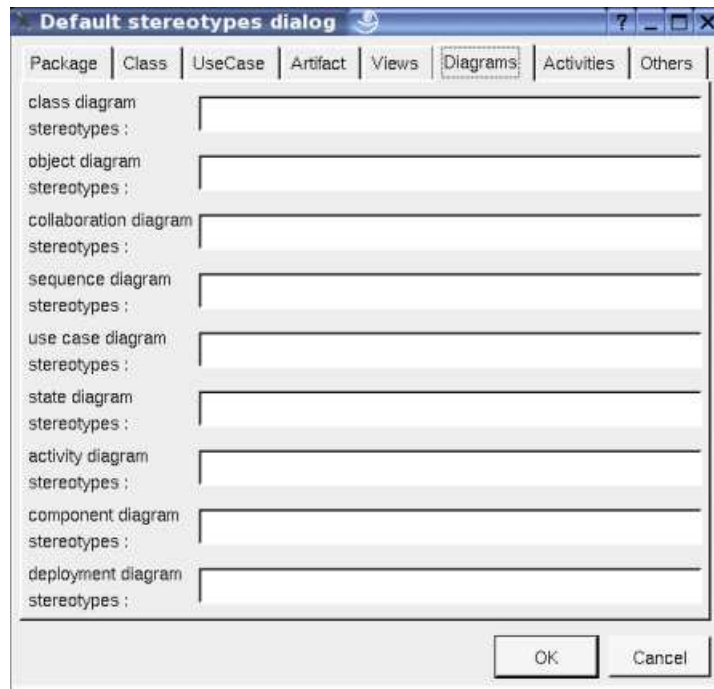
### **Picture Menu : undo changes**

Undo all the changes made since the diagram was opened the last time. Note that the diagram is restored independently of the project saving. Obviously the deleted browser items was not restored !

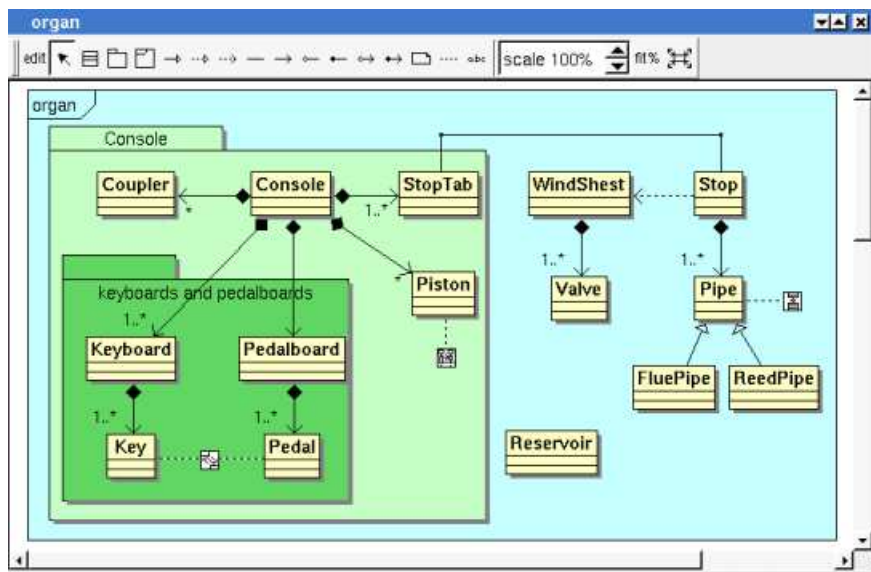
---

## **Default stereotypes**

The [dialog](#) allowing to set the default stereotypes has a tab reserved for the diagrams :



## Drawing



A class diagram may contain [classes](#), [packages](#), [fragment](#), [notes](#), [texts](#), [diagram icons](#) and the [relations](#).

To place these elements in the diagram, use the buttons on the top (to create a new element or choose from the already defined one) then click at the appropriate position in the diagram, or *grab* the element from the browser into the diagram.

To add a new relation between classes, select the appropriate relation through the buttons on the top of the diagram sub window, click on the start class with the left mouse button, move the mouse, each time the mouse button is raised a new line break is introduced, at least click on the end class. To abort a relation construction press the right mouse button or do a double click with the left mouse button.

A line may be broken, during its construction or after clicking on the line with the left mouse button and moving the mouse with mouse button still pushed. To remove a line break, a double click on the point with the left mouse button is enough, or use the line break menu of the point using the right mouse button.

The *scale* may be changed from 30 up to 200 %. To allow to see the operation/message kinds even with a small scale, the size of the unmodified.

*fit%* allows to set the largest scale allowing to show all the picture, as it is possible. Because of the scrollbar management, hitting seven



times on *fit%* may give several scales.

Classically, the left mouse click may be used to select an element, or several when the *control* key is down, a multiple selection may also be done defining a region using the mouse left click out of any element and moving the mouse with the click and maintaining the click down. The diagram's elements may be moved after a left mouse click on an element and maintaining the click down during mouse move. The right mouse click is used to show context dependent menus.

*control s* is a shortcut to save the project

*control a* is a shortcut to select all the diagram's elements

*control d* is a shortcut to delete the selected element(s), the model is affected

*suppr* is a shortcut to remove the selected element(s) from the diagram, these elements are not deleted

*control c* is a shortcut to copy the selected diagram's elements

*control x* is a shortcut to cut (remove from view) the selected diagram's elements

*control v* is a shortcut to paste the selected diagram's elements

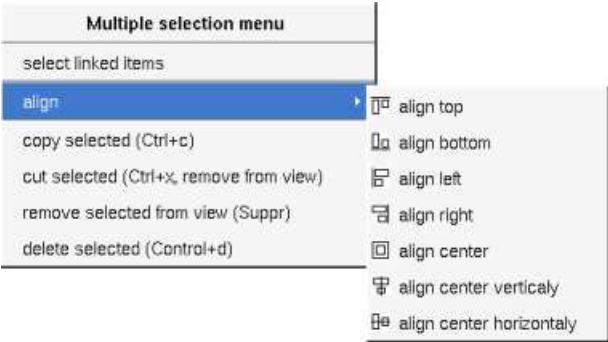
*control z* and *control u* are a shortcut of *undo*

*control r* and *control y* are a shortcut of *redo*

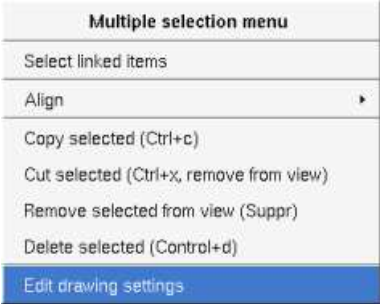
the keyboard arrows allow to move the selected items

You can define your own shortcut using the [shortcut editor](#) through the *Miscellaneous* menu.

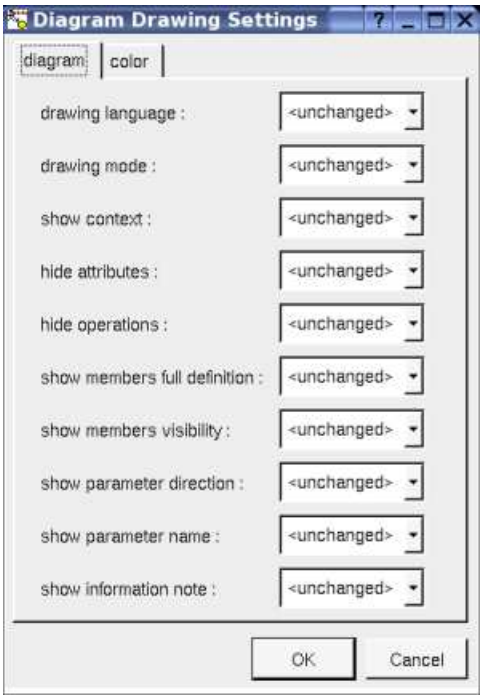
When several diagram's elements are selected, the menu appearing on a left mouse click allows to align them :



If all the selected elements has the same kind (for instance all are a class), you edit their *drawing settings* globally :



and for instance when the elements are classes in a class diagram :



The settings valuing *<unchanged>* are not changed for the selected elements, in the other case the same value is set for all the elements.

**Fragment**

A *fragment* is resizable. To change the fragment's name choose the *edit* entry of its menu or double click on it

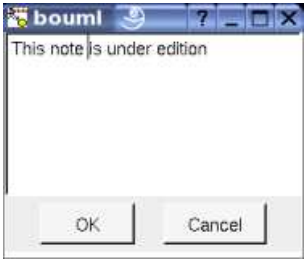
A right mouse click on a note show the following menu :



*Edit drawing settings* allows to change the fill color

**Note**

A *note* may contain any text, a note is resizable. To change the text of a note choose the *edit* entry of its menu or double click on it, this open a window containing the text, close the window to end the edition :



A right click on the edited note calls a dialog allowing some text manipulations thanks Qt.

A right mouse click on a note show the following menu :

Note
Upper
Lower
Edit
Color of text
Font
Edit drawing settings
Select linked items
Remove from view

The *font* may be changed, by default the font is the one used to write the other texts.

The color of the text may be changed, a text is black by default.

*Edit drawing settings* allows to change the fill color

*Select linked items* select all the items connected this the current one through any line, in the previous picture this select all except the package, the text “*upper diagram :*” and the use case diagram icon.

## Text

*Text* is another way to write a text, contrary to the note a *text* is written alone (see “*upper diagram :*” on the picture below), by default a text contains *blabla*: a text cannot be empty. The edition of a text is done like for a note. A text is resizeable, a resize is not automatically done when the contains is modified.

A right mouse click on a *text* show the following menu :

Text
Upper
Lower
Edit
Color
Font
Select linked items
Remove from view

## Diagram icon

A diagram icon is a shortcut to a diagram, this allows to quickly navigate through the diagram without limitation (a actor or some other item kinds may be associated to only one diagram). The used icon indicates the diagram kind, but its name is not written.

A double mouse left click on a diagram icon opens the diagram, the *open* entry of the diagram icon menu does the same thing.

A right mouse click on a diagram icon show the following menu :

Overall short cut
upper
lower
open
select diagram in browser
select linked items
remove from view

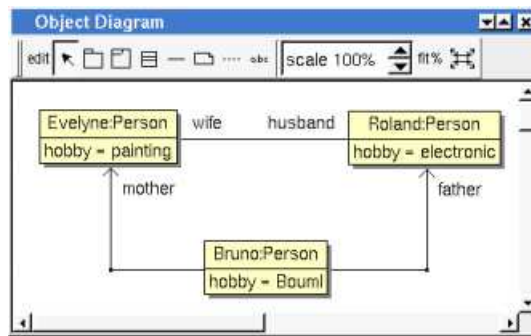
The other diagram's elements have an own chapter, refer to them.

---

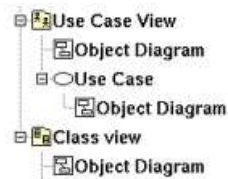
Previous : [collaboration diagram](#)

Next : [object diagram](#)

# Object diagram



The object diagrams may be placed in a [use case view](#), a [use case](#) or a [class view](#)



An object diagram is created through the *new object diagram* entry of the [use case view](#), [use case](#) and [class view](#) browser menus.

## Browser menu

The node menu appearing with a right mouse click on its representation in the browser is something like these, supposing the diagram not *read-only* nor deleted :



### [Browser Menu](#) : show

*show* allows to show the diagram [picture](#), and edit it in case it is not read only.

### [Browser Menu](#) : edit

*edit* allows to show/modify the diagram properties. In case the diagram is read-only, the fields of the dialog are also read-only.

**object diagram dialog, tab Uml**

```

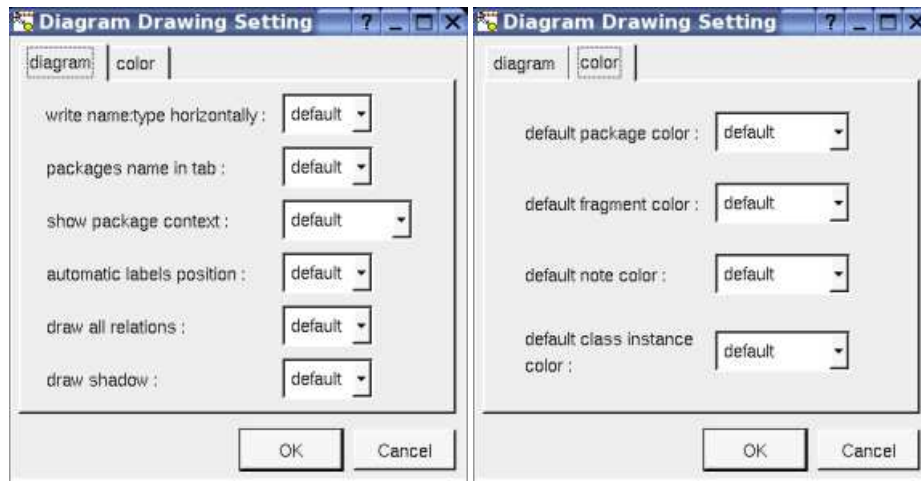
classDiagram
    class "Object Diagram" {
        name : Object Diagram
        stereotype : 
        description : 
    }
  
```

The proposed *stereotypes* are the default one specified through the [Default stereotypes dialog](#) more the current one (empty here). It is possible to choose into the list or to give a new one, or to empty it. The *stereotype* of a diagram doesn't have a special meaning for BOUML.

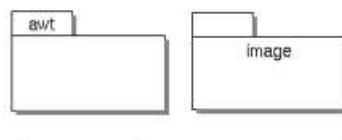
The *editor* button visible above and associated here to the description, allows to edit the description in an other window, or to call an external editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note that this external editor have to create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel with BOUML which looks each second at the file contents to get the new definition until the dialog is closed (I do not like very much polling but this works even QT isn't compiled with the thread support).

## **Browser Menu : edit drawing settings**

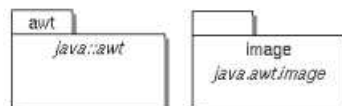
This dialog allows to specify how the items of the diagram must be drawn by default, you may choose *default* for each value, in this case the effective value if specified in the upper level (which itself may specify *default* etc ... up to the *project* level).



- *write class instance horizontally* : To write the instance type and name on the same line, or on two lines
- *package name in tab* : to indicate if the package's name must be written in the tab which is the small rectangle on the top (note : a transparent package doesn't have shadow) :



- *show packages context* : to indicate if the context where the package is defined must be written, it is not the case just above. The context may be the "UML context" which is the path of the package in the browser (choice followed for *awt* below), or the C++ namespace specified by the package, or the Java package specified by the package (choice followed for *picture* below), or the Python package specified by the package or at least the Idl module :



- *automatic labels position* : To ask BOUML to move or to not move the associated labels on the relations in a diagram when the relation is edited associated class pictures moved etc ...

## **Browser Menu : duplicate**

Clone the diagram in a new one

## **Browser Menu : delete**

The *delete* entry is only present when the diagram is not *read-only*.

Delete the diagram and all its representation in the opened diagrams. After that it is possible to undelete it (from the browser) until you close the project : obviously the deleted items are not saved !

## Menu : mark

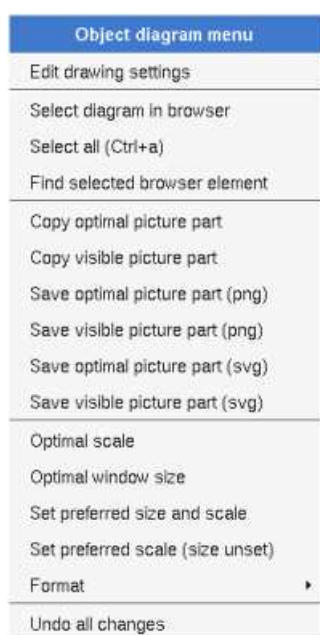
See [mark](#)

## Browser Menu : tools

The menu entry *tool* is only present in case at least a [plug-out](#) may be applied on the diagram. The selected tool is called and applied on the current diagram.

## Picture menu

The diagram menu appearing with a right mouse click out off any item in the diagram picture is something like these, supposin diagram not *read-only* :



### Picture Menu : edit drawing settings

see [edit drawing settings](#).

### Picture Menu : select all

Select all the items of the current diagram, also done through a *control a*

### Picture Menu : find selected browser element

To find in the diagram a representation of the elements selected in the browser

### Picture Menu : optimal scale

similar to the *fit scale* button, change scale to see all the diagram's elements at great as possible

### Picture Menu : optimal window size

Similar to the button , change the size of the windows to see all the diagram's elements with the current scale

### Picture Menu : copy optimal picture part

Copy all the diagram elements, equivalent to a *optimal windows size* followed by a *copy visible picture part* then a restore of the original windows size

### **Picture Menu : copy visible part**

Put the visible part of the current diagram in the clipboard (*copy*), it is a screen copy, the current scale have impact on the picture quality.

It is also possible to [print](#) all (not only the visible part) the current diagram on printer or a file through the button

### **Picture Menu : save optimal picture part**

To save all the diagram elements in a file, equivalent to a *optimal windows size* followed by a *save visible picture part* then a restore of the original windows size

### **Picture Menu : save visible picture part**

To save the visible part of the current diagram in a file, it is a screen copy, the current scale have impact on the picture quality.

### **Picture Menu : set preferred size and scale**

To memorize the current diagram sub window size and the current scale. These ones will be used the next time the diagram wil opened.

### **Picture Menu : set preferred scale**

To memorize the current scale. These one will be used the next time the diagram will be opened.

### **Picture Menu : restore preferred size and scale**

In case a preferred sub window size and preferred scale was set, follow them.

### **Picture Menu : unset preferred and scale**

To stop to define a preferred size and scale was set

### **Picture Menu : format**

By default the diagram's canvas size is 840 x 1188 points corresponding to an iso A4 format (210\*4 x 297\*4), all the available formats are :

- iso A5 : 592 x 840
- iso A4 : 840 x 1188
- iso A3 : 1188 x 1680
- iso A2 : 1680 x 2376
- iso A1 : 2376 x 3364
- iso A0 : 3364 x 4756
- USA A : 864 x 1116
- USA B : 1116 x 1728
- USA C : 1728 x 2236
- USA D : 2236 x 3456
- USA E : 3456 x 4472

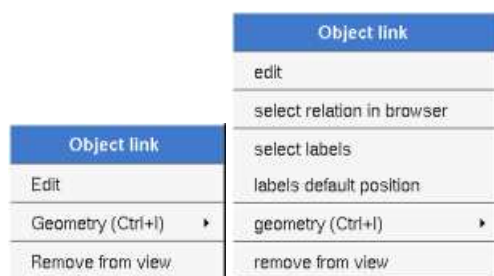
Each format also exists in *landscape*

### **Picture Menu : undo changes**

Undo all the changes made since the diagram was opened the last time. Note that the diagram is restored independently of the th project saving. Obviously the deleted browser items was not restored !

## Link Menu

The menu appearing with a right mouse click on link is something like these, supposing the diagram not *read-only* (in the first case the link is doesn't have an associated relation) :



### Link Menu : edit

*edit*  
allows to choose the relation supported by the link, the available relations depends on the type of each extremities, the current relation (if set) is shown by the selection :



### Link Menu : select relation in browser

Is of course only available when the relation is set

### Message Menu : select labels

Of course only available when the relation is set, to select the labels (relation's name, stereotypes, role(s) and multiplicity(ies) associated to the relation. Useful when you are lost after many label movings.

### Message Menu : labels default position

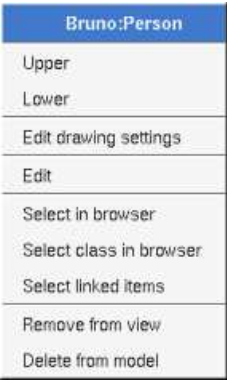
Of course only available when the relation is set, to place the roles's name associated to the relation in the default position. Useful when you are lost after many label movings.

By default when you move a class instance or a link point break or edit the link, the labels are moved to their default position, this may be irritating. To ask BOUML to not move the labels in a diagram, use the [drawing settings](#) of the diagrams.

## Class instance Menu

The menu appearing with a right mouse click on a class [instance](#) is something like this, supposing the diagram not *read-only* :

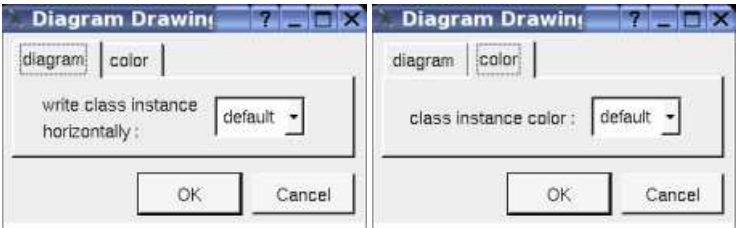




Contrarily to the [sequence](#) and [collaboration](#) diagrams, an object diagram show modeled instances : there is no pure graphical instances.

**Message Menu : edit drawing settings**

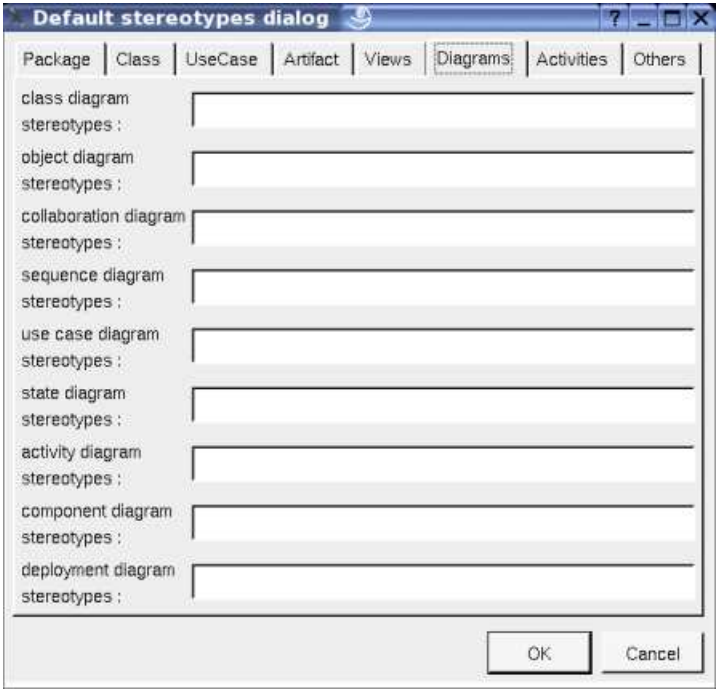
This dialog allows to specify how a the class instance must be drawn, you may choose *default* for each value, in this case the effective value if specified in the upper level (which itself may specify *default* etc ... up to the *project* level).



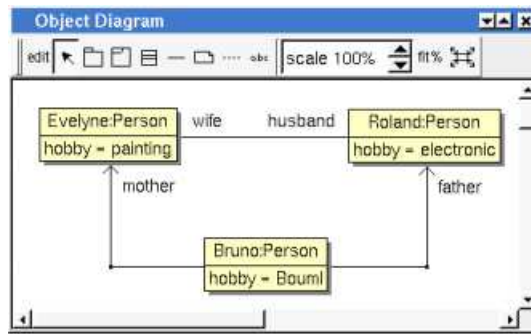
*write class instance horizontally* : To write the instance type and name on the same line, or on two lines.

**Default stereotypes**

The [dialog](#) allowing to set the default stereotypes has a tab reserved for the diagrams :



**Drawing**



An object diagram may contain [classes](#) [instances](#), [notes](#), [texts](#), [diagram icons](#), [packages](#), fragment and links.

To place these elements in the diagram, use the buttons on the top (to create a new element or choose from the already defined one then click at the appropriate position in the diagram, or *grab* the element from the browser into the diagram.

To show a relation between two class instances, add a link between the instance then edit the link through a double click or through right mouse click.

The *scale* may be changed from 30 up to 200 %. To allow to see the relation kinds even with a small scale, the size of the arrows is unmodified.

*fit%* allows to set the largest scale allowing to show all the picture, as it is possible. Because of the scrollbar management, hitting several times on *fit%* may give several scales.

Classically, the left mouse click may be used to select an element, or several when the *control* key is down, a multiple selection may also be done defining a region using the mouse left click out of any element and moving the mouse with the click and maintaining the click down. The diagram's elements may be moved after a left mouse click on an element and maintaining the click down during mouse move. The right mouse click is used to show context dependent menus.

*control s* is a shortcut to save the project

*control a* is a shortcut to select all the diagram's elements

*control d* is a shortcut to delete the selected element(s), the model is affected

*suppr* is a shortcut to remove the selected element(s) from the diagram, these elements are not deleted

*control c* is a shortcut to copy the selected diagram's elements

*control x* is a shortcut to cut (remove from view) the selected diagram's elements

*control v* is a shortcut to paste the selected diagram's elements

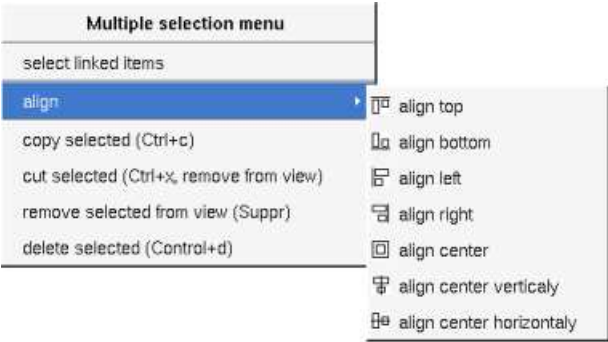
*control z* and *control u* are a shortcut of *undo*

*control r* and *control y* are a shortcut of *redo*

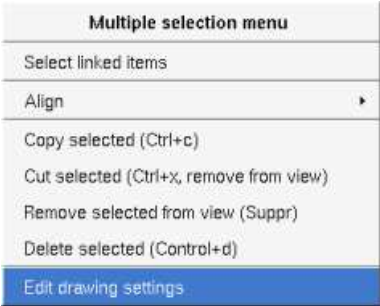
the keyboard arrows allow to move the selected items

You can define your own shortcut using the [shortcut editor](#) through the *Miscellaneous* menu.

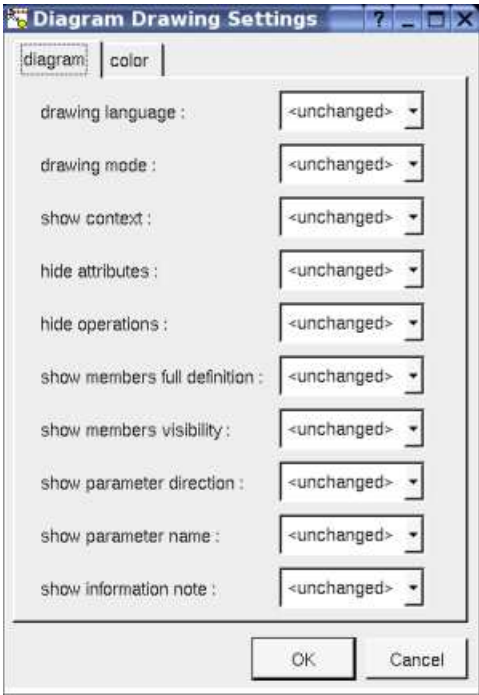
When several diagram's elements are selected, the menu appearing on a left mouse click allows to align them :



If all the selected elements has the same kind (for instance all are a class), you edit their *drawing settings* globally :



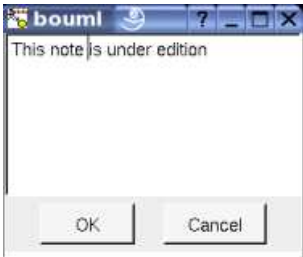
and for instance when the elements are classes in a class diagram :



The settings valuing *<unchanged>* are not changed for the selected elements, in the other case the same value is set for all the elements.

**Note**

A *note* may contain any text, a note is resizeable. To change the text of a note choose the *edit* entry of its menu or double click on it, this open a window containing the text, close the window to end the edition :



A right click on the edited note calls a dialog allowing some text manipulations thanks Qt.

A right mouse click on a note show the following menu :

Note
Upper
Lower
Edit
Color of text
Font
Edit drawing settings
Select linked items
Remove from view

The *font* may be changed, by default the font is the one used to write the other texts.

The color of the text may be changed, a text is black by default.

*Edit drawing settings* allows to change the fill color

*Select linked items* select all the items connected this the current one through any line, in the previous picture this select all except the package, the text “upper diagram :” and the use case diagram icon.

### Text

*Text* is another way to write a text, contrary to the note a *text* is written alone (see “upper diagram :” on the picture below), by default a text contains *blabla*: a text cannot be empty. The edition of a text is done like for a note. A text is resizeable, a resize is not automatically done when the contains is modified.

A right mouse click on a *text* show the following menu :

Text
Upper
Lower
Edit
Color
Font
Select linked items
Remove from view

### Diagram icon

A diagram icon is a shortcut to a diagram, this allows to quickly navigate through the diagram without limitation (a actor or some other item kinds may be [associated](#) to only one diagram). The used icon indicates the diagram kind, but its name is not written.

A double mouse left click on a diagram icon opens the diagram, the *open* entry of the diagram icon menu does the same thing.

A right mouse click on a diagram icon show the following menu :

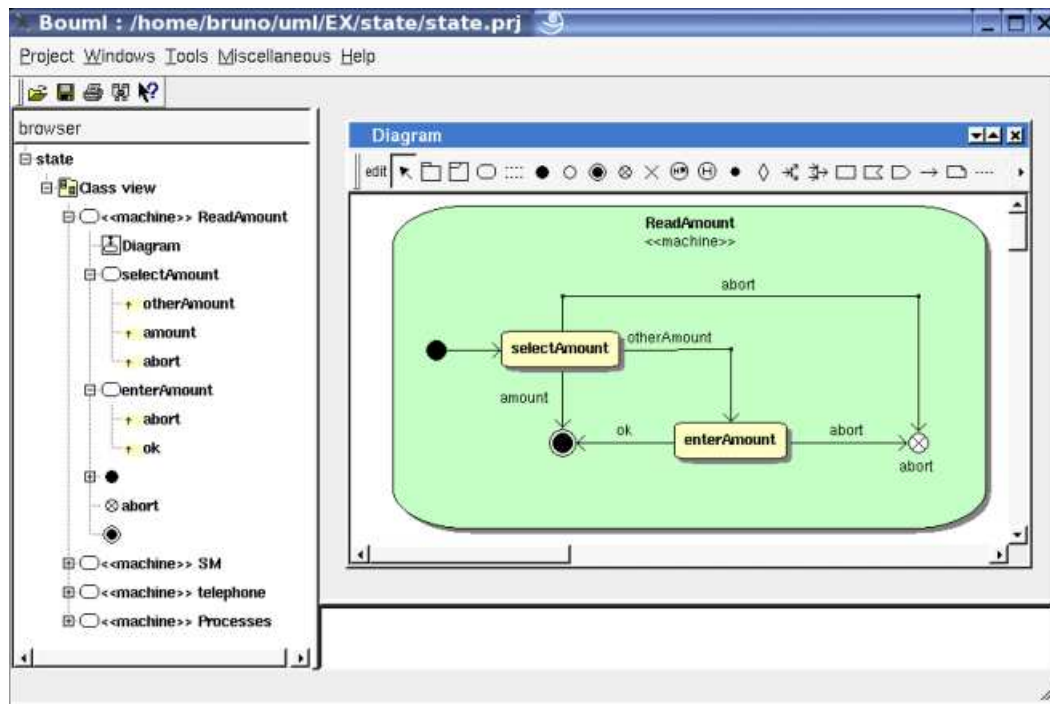
Overall short cut
upper
lower
open
select diagram in browser
select linked items
remove from view

The other diagram's elements have an own chapter, refer to them.

Previous : [class diagram](#)

Next : [state diagram](#)

# State diagram



The *state diagrams* may be placed in a [state](#)

A *state diagram* is created through the *new state diagram* entry of the [state](#) browser menus.

## Browser menu

The node menu appearing with a right mouse click on its representation in the browser is something like these, supposing the diagram not *read-only* nor deleted :



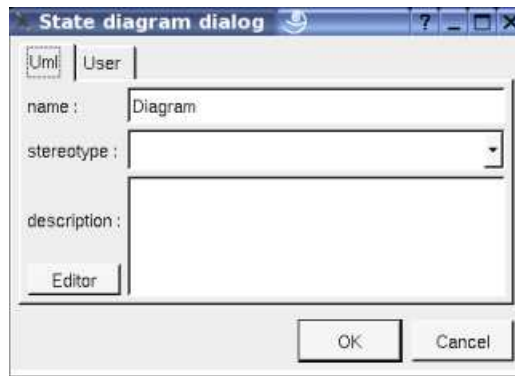
### [Browser Menu](#) : show

*show* allows to show the diagram [picture](#), and edit it in case it is not read only.

### [Browser Menu](#) : edit

*edit* allows to show/modify the diagram properties. In case the diagram is read-only, the fields of the dialog are also read-only.

### State diagram dialog, tab Uml

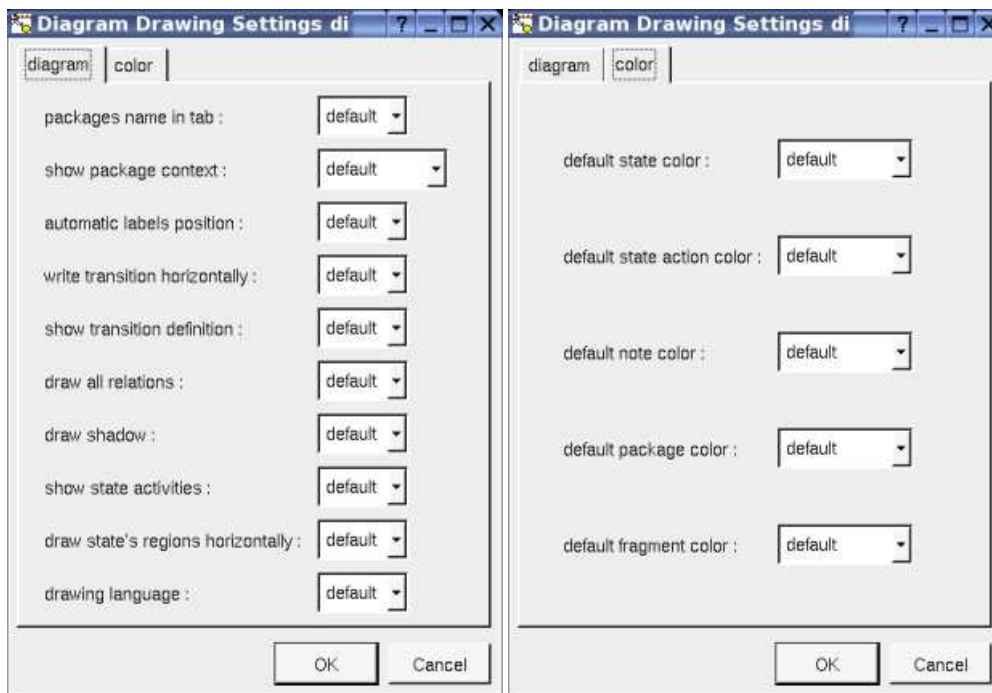


The proposed *stereotypes* are the default one specified through the [Default stereotypes dialog](#) more the current one (empty here). It is possible to choose into the list or to give a new one, or to empty it. The *stereotype* of a diagram doesn't have a special meaning for BOUML.

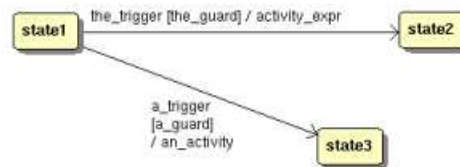
The *editor* button visible above and associated here to the description, allows to edit the description in an other window, or to call an external editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note that this external editor have to create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel with BOUML which looks each second at the file contents to get the new definition until the dialog is closed (I do not like very much polling but this works even QT isn't compiled with the thread support).

### **Browser Menu : edit drawing settings**

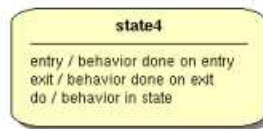
This dialog allows to specify how the items of the diagram must be drawn by default, you may choose *default* for each value, in this case the effective value if specified in the upper level (which itself may specify *default* etc ... up to the *project* level).



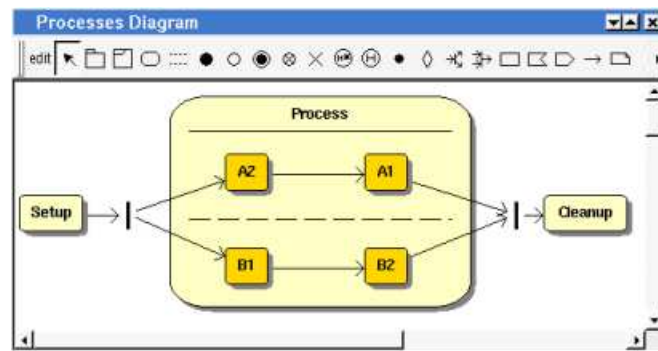
- *show* *packages* *context*  
: To indicate if the context where the package is defined must be written, and if yes, how. The context may be the “UML context” which is the path of the package in the browser, or the C++ *namespace*, or the Java *package* or at least the Idl *module*. See [package context](#).
- *automatic* *labels* *position*  
: To ask BOUML to move or to not move the associated labels on the relations in a diagram when the relation is edited associated State pictures moved etc ...
- *write transition horizontally* : to write the transition's *trigger*, *guard constraint* and *activity* on only one line or each one on a different line.



- *show transition definition* : to write the name of the transition or its *trigger*, *guard constraint* and *activity*
- *show state activities* : to show or not the *state's* activities in a compartment depending on the drawing language.



- *draw state's regions horizontally* : to indicate if the *regions* are drawn horizontally or vertically, for instance :



Note : the regions are not visible while the state picture is too small.

- *drawing language* : to indicate in which language the transition and state's compartment must be written

### **Browser Menu : duplicate**

Clone the diagram in a new one

### **Browser Menu : delete**

The *delete* entry is only present when the diagram is not *read-only*.

Delete the diagram and all its representation in the opened diagrams. After that it is possible to undelete it (from the browser) until you close the project : obviously the deleted items are not saved !

### **Menu : mark**

See [mark](#)

### **Browser Menu : tools**

The menu entry *tool* is only present in case at least a *plug-out* may be applied on the diagram. The selected tool is called and applied on the current diagram.

---

## **Picture menu**

The diagram menu appearing with a right mouse click out off any item in the diagram picture is something like these, supposin diagram not *read-only* :



State diagram menu
Edit drawing settings
Select diagram in browser
Select all (Ctrl+a)
Optimal scale
Optimal window size
Copy optimal picture part
Copy visible picture part
Save optimal picture part (png)
Save visible picture part (png)
Save optimal picture part (svg)
Save visible picture part (svg)
Set preferred size and scale
Set preferred scale (size unset)
Restore preferred size and scale
Unset preferred size and scale
Format
Undo all changes

**Picture Menu : edit drawing settings**

see [edit drawing settings](#).

**Picture Menu : select all**

Select all the items of the current diagram, may also be done through *control a*

**Picture Menu : optimal scale**

similar to the *fit scale* button, change scale to see all the diagram's elements at great as possible

**Picture Menu : optimal window size**

change the size of the windows to see all the diagram's elements with the current scale

**Picture Menu : copy optimal picture part**

Copy all the diagram elements, equivalent to a *optimal windows size* followed by a *copy visible picture part* then a restore of the original windows size

**Picture Menu : copy visible picture part**

Put the visible part of the current diagram in the clipboard (*copy*), it is a screen copy, the current scale have impact on the picture quality.

It is also possible to [print](#) all (not only the visible part) the current diagram on printer or a file through the button

**Picture Menu : save optimal picture part**

To save all the diagram elements in a file, equivalent to a *optimal windows size* followed by a *save visible picture part* then a restore of the original windows size

**Picture Menu : save visible picture part**

To save the visible part of the current diagram in a file, it is a screen copy, the current scale have impact on the picture quality.

**Picture Menu : set preferred size and scale**

To memorize the current diagram sub window size and the current scale. These ones will be used the next time the diagram will opened.

**Picture Menu : set preferred scale**

To memorize the current scale. These one will be used the next time the diagram will be opened.

### **Picture Menu : restore preferred size and scale**

In case a preferred sub window size and preferred scale was set, follow them.

### **Picture Menu : unset preferred and scale**

To stop to define a preferred size and scale was set

### **Picture Menu : format**

By default the diagram's canvas size is 840 x 1188 points corresponding to an iso A4 format (210\*4 x 297\*4), all the available formats are :

- iso A5 : 592 x 840
- iso A4 : 840 x 1188
- iso A3 : 1188 x 1680
- iso A2 : 1680 x 2376
- iso A1 : 2376 x 3364
- iso A0 : 3364 x 4756
- USA A : 864 x 1116
- USA B : 1116 x 1728
- USA C : 1728 x 2236
- USA D : 2236 x 3456
- USA E : 3456 x 4472

Each format also exists in *landscape*

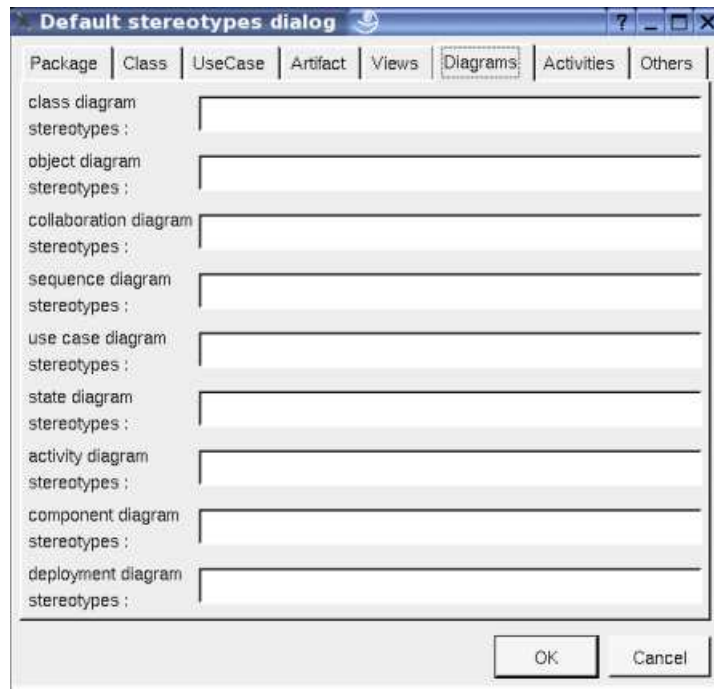
### **Picture Menu : undo changes**

Undo all the changes made since the diagram was opened the last time. Note that the diagram is restored independently of the th project saving. Obviously the deleted browser items was not restored !

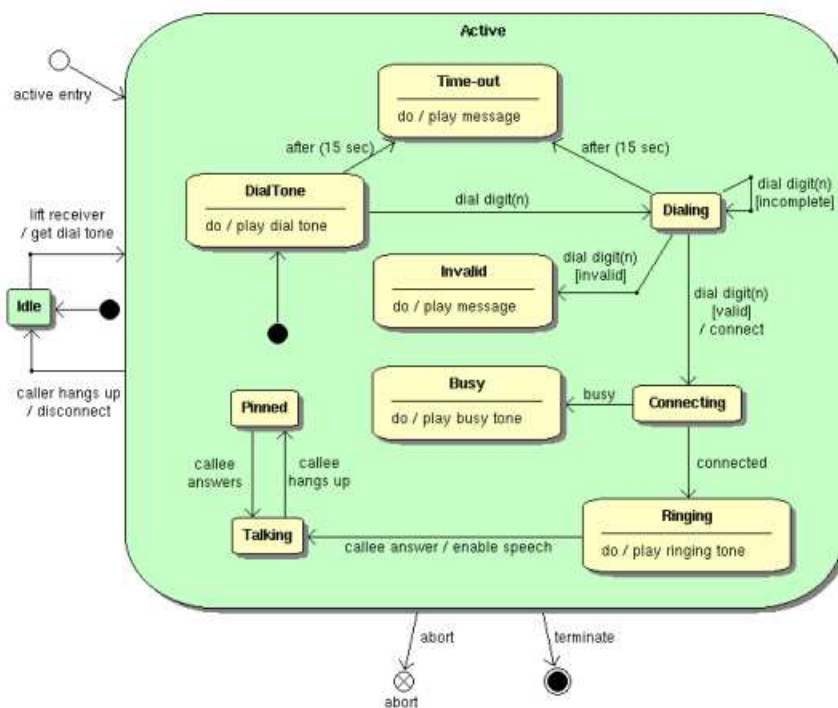
---

## **Default stereotypes**

The [dialog](#) allowing to set the default stereotypes has a tab reserved for the diagrams :



## Drawing



A *state diagram* may contain [states](#), [packages](#), [fragment](#), [notes](#), [texts](#), [diagram icons](#), [pseudo states](#) and the transitions.

To place these elements in the diagram, use the buttons on the top (to create a new element or choose from the already defined one) then click at the appropriate position in the diagram, or *grab* the element from the browser into the diagram.

**Warning :** when you use the buttons to add a *state* or *pseudo state* and you click on a *state* drawing, the new state will be nested in the other state. In case you place the new *state* or *pseudo state* out of any *state* drawing, the new one will be nested in *state* owning the diagram.

To add a new relation between *states* and *pseudo states*, select the appropriate relation through the buttons on the top of the diagram sub window, click on the start *state* or *pseudo state* with the left mouse button, move the mouse, each time the mouse button is raised a new line break is introduced, at least click on the end *state* or *pseudo state*. To abort a relation construction press the right mouse button or do a double click with the left mouse button.

A line may be broken, during its construction or after clicking on the line with the left mouse button and moving the mouse with mouse button still pushed. To remove a line break, a double click on the point with the left mouse button is enough, or use the line break menu of the point using the right mouse button.

The *scale* may be changed from 30 up to 200 %. To allow to see the operation/message kinds even with a small scale, the size of the unmodified.

*fit%* allows to set the largest scale allowing to show all the picture, as it is possible. Because of the scrollbar management, hitting several times on *fit%* may give several scales.

Classically, the left mouse click may be used to select an element, or several when the *control* key is down, a multiple selection may also be done defining a region using the mouse left click out of any element and moving the mouse with the click and maintaining the click down. The diagram's elements may be moved after a left mouse click on an element and maintaining the click down during mouse move. The right mouse click is used to show context dependent menus.

*control s* is a shortcut to save the project

*control a* is a shortcut to select all the diagram's elements

*control d* is a shortcut to delete the selected element(s), the model is affected

*suppr* is a shortcut to remove the selected element(s) from the diagram, these elements are not deleted

*control c* is a shortcut to copy the selected diagram's elements

*control x* is a shortcut to cut (remove from view) the selected diagram's elements

*control v* is a shortcut to paste the selected diagram's elements

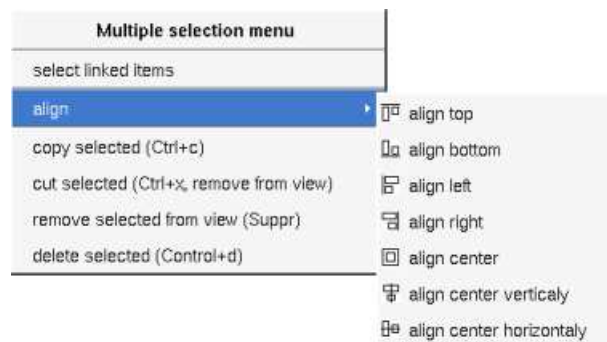
*control z* and *control u* are a shortcut of *undo*

*control r* and *control y* are a shortcut of *redo*

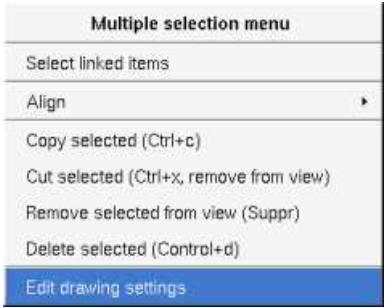
the keyboard arrows allow to move the selected items

You can define your own shortcut using the [shortcut editor](#) through the *Miscellaneous* menu.

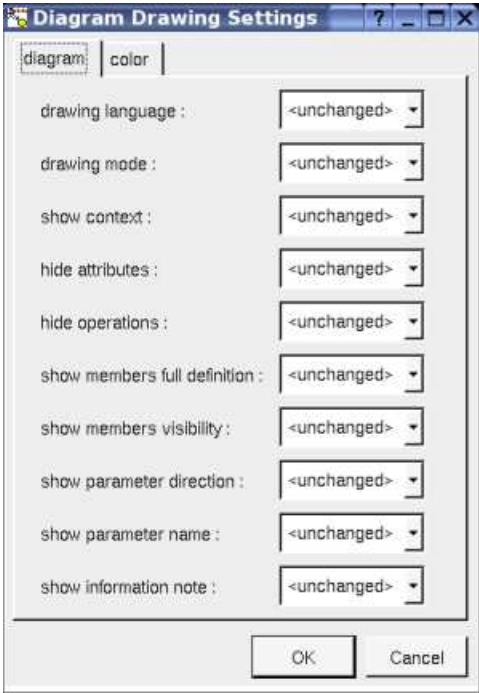
When several diagram's elements are selected, the menu appearing on a left mouse click allows to align them :



If all the selected elements has the same kind (for instance all are a class), you edit their *drawing settings* globally :



and for instance when the elements are classes in a class diagram :



The settings valuing *<unchanged>* are not changed for the selected elements, in the other case the same value is set for all the elements.

**Fragment**

A *fragment* is resizable. To change the fragment's name choose the *edit* entry of its menu or double click on it

A right mouse click on a note show the following menu :



*Edit drawing settings* allows to change the fill color

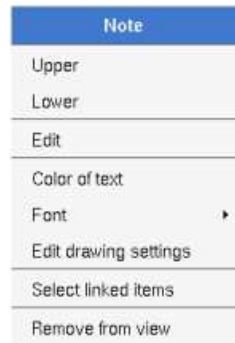
**Note**

A *note* may contain any text, a note is resizable. To change the text of a note choose the *edit* entry of its menu or double click on it, this open a window containing the text, close the window to end the edition :



A right click on the edited note calls a dialog allowing some text manipulations thanks Qt.

A right mouse click on a note show the following menu :



The *font* may be changed, by default the font is the one used to write the other texts.

The color of the text may be changed, a text is black by default.

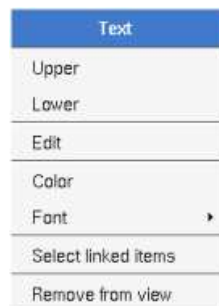
*Edit drawing settings* allows to change the fill color

*Select linked items* select all the items connected this the current one through any line, in the previous picture this select all except the package, the text “*upper diagram :*” and the use case diagram icon.

## Text

*Text* is another way to write a text, contrary to the note a *text* is written alone (see “*upper diagram :*” on the picture below), by default a text contains *blabla*: a text cannot be empty. The edition of a text is done like for a note. A text is resizeable, a resize is not automatically done when the contains is modified.

A right mouse click on a *text* show the following menu :



## Diagram icon

A diagram icon is a shortcut to a diagram, this allows to quickly navigate through the diagram without limitation (a actor or some other item kinds may be [associated](#) to only one diagram). The used icon indicates the diagram kind, but its name is not written.

A double mouse left click on a diagram icon opens the diagram, the *open* entry of the diagram icon menu does the same thing.

A right mouse click on a diagram icon show the following menu :

Overall short cut
upper
lower
open
select diagram in browser
select linked items
remove from view

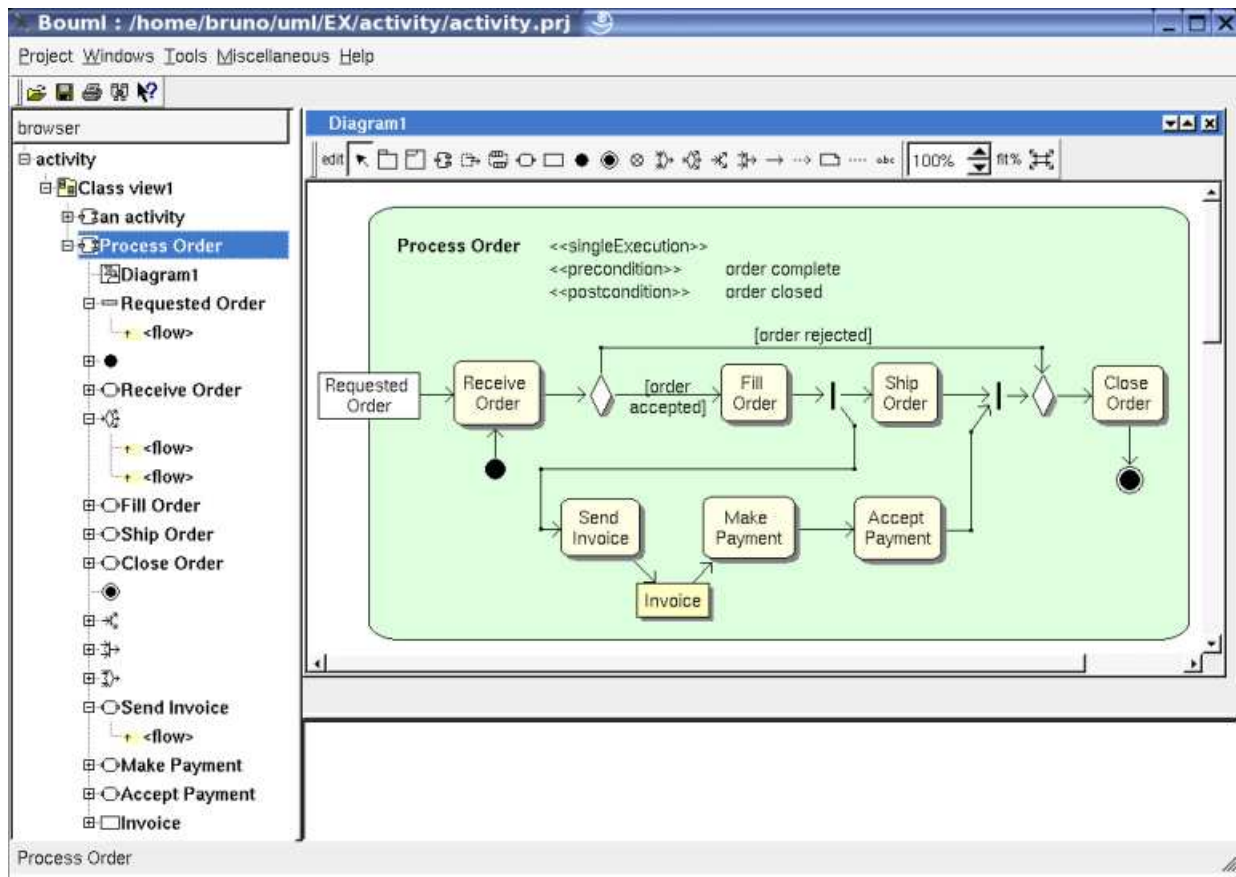
The other diagram's elements have an own chapter, refer to them.

---

Previous : [class diagram](#)

Next : [activity diagram](#)

# Activity diagram



The *activity diagrams* may be placed in an [activity](#)

A *activity diagram* is created through the *new activity diagram* entry of the [activity](#) browser menus.

## Browser menu

The node menu appearing with a right mouse click on its representation in the browser is something like these, supposing the diagram not *read-only* nor deleted :

Diagram1
show
edit
edit drawing settings
duplicate
delete
mark

### [Browser Menu](#) : show

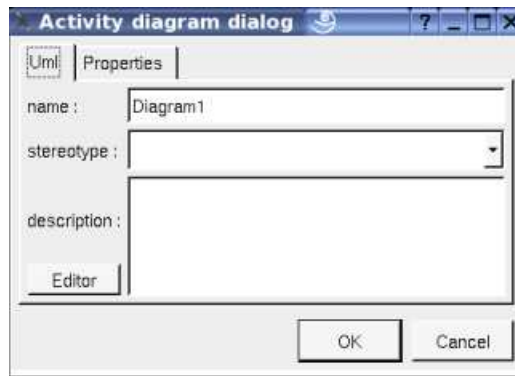
*show* allows to show the diagram [picture](#), and edit it in case it is not read only.

### [Browser Menu](#) : edit

*edit* allows to show/modify the diagram properties. In case the diagram is read-only, the fields of the dialog are also read-only.

### Activity diagram dialog, tab Uml



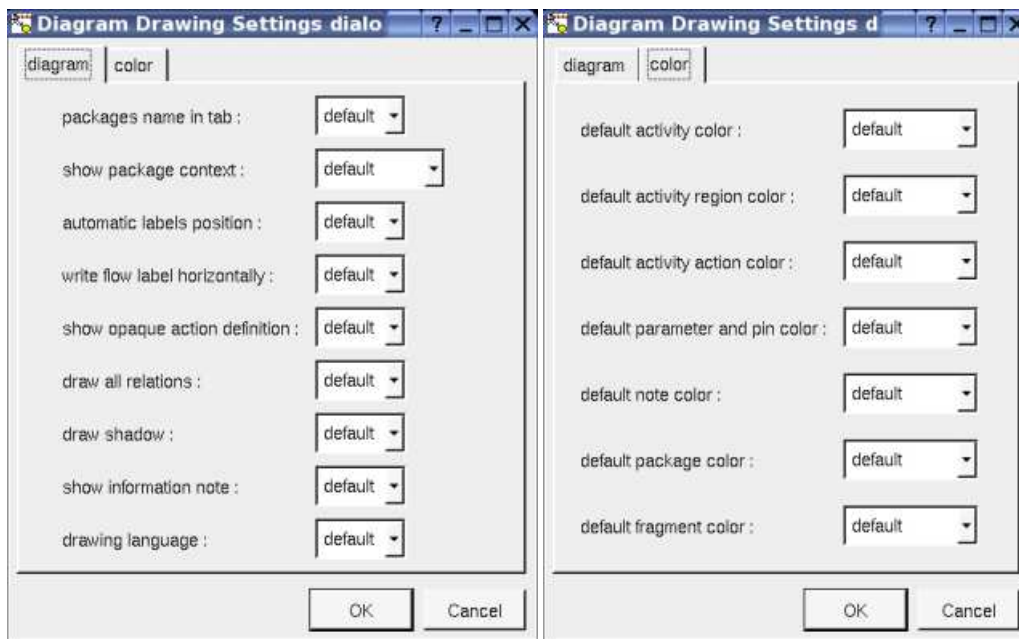


The proposed *stereotypes* are the default one specified through the [Default stereotypes dialog](#) more the current one (empty here). It is possible to choose into the list or to give a new one, or to empty it. The *stereotype* of a diagram doesn't have a special meaning for BOUML.

The *editor* button visible above and associated here to the description, allows to edit the description in an other window, or to call an external editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note that this external editor have to create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel with BOUML which looks each second at the file contents to get the new definition until the dialog is closed (I do not like very much polling but this works even QT isn't compiled with the thread support).

### **Browser Menu : edit drawing settings**

This dialog allows to specify how the items of the diagram must be drawn by default, you may choose *default* for each value, in this case the effective value if specified in the upper level (which itself may specify *default* etc ... up to the *project* level).



- *show packages context* : To indicate if the context where the package is defined must be written, and if yes, how. The context may be the “UML context” which is the path of the package in the browser, or the C++ *namespace*, or the Java *package* or at least the Idl *module*. See [package context](#).
- *automatic labels position* : To ask BOUML to move or to not move the associated labels on the relations in a diagram when the relation is edited associated Activity pictures moved etc ...
- *write flow label horizontally* : to write the flow description on only one line or each information on a different line.
- *show opaque action definition* : to write the name of an *opaque action* or its *behavior* definition
- *show information note* : to show or not the notes associated to the *condition*, *selection and transformation*, and also to write or not the *activity's* conditions.
- *drawing language* : to indicate which language must be used to write the definitions

**Browser Menu : duplicate**

Clone the diagram in a new one

**Browser Menu : delete**

The *delete* entry is only present when the diagram is not *read-only*.

Delete the diagram and all its representation in the opened diagrams. After that it is possible to undelete it (from the browser) until you close the project : obviously the deleted items are not saved !

**Menu : mark**

See [mark](#)

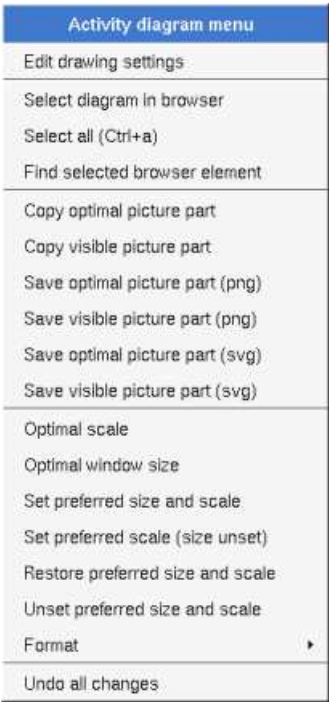
**Browser Menu : tools**

The menu entry *tool* is only present in case at least a [plug-out](#) may be applied on the diagram. The selected tool is called and applied on the current diagram.

---

**Picture menu**

The diagram menu appearing with a right mouse click out off any item in the diagram picture is something like these, supposin diagram not *read-only* :



**Picture Menu : edit drawing settings**

see [edit drawing settings](#).

**Picture Menu : select all**

Select all the items of the current diagram, may also be done through *control a*

**Picture Menu : find selected browser element**

To find in the diagram a representation of the elements selected in the browser

**Picture Menu : optimal scale**

similar to the *fit scale* button, change scale to see all the diagram's elements at great as possible

### **Picture Menu : optimal window size**

change the size of the windows to see all the diagram's elements with the current scale

### **Picture Menu : copy optimal picture part**

Copy all the diagram elements, equivalent to a *optimal windows size* followed by a *copy visible picture part* then a restore of the original windows size

### **Picture Menu : copy visible picture part**

Put the visible part of the current diagram in the clipboard (*copy*), it is a screen copy, the current scale have impact on the picture quality.

It is also possible to [print](#) all (not only the visible part) the current diagram on printer or a file through the button

### **Picture Menu : save optimal picture part**

To save all the diagram elements in a file, equivalent to a *optimal windows size* followed by a *save visible picture part* then a restore of the original windows size

### **Picture Menu : save visible picture part**

To save the visible part of the current diagram in a file, it is a screen copy, the current scale have impact on the picture quality.

### **Picture Menu : set preferred size and scale**

To memorize the current diagram sub window size and the current scale. These ones will be used the next time the diagram will opened.

### **Picture Menu : set preferred scale**

To memorize the current scale. These one will be used the next time the diagram will be opened.

### **Picture Menu : restore preferred size and scale**

In case a preferred sub window size and preferred scale was set, follow them.

### **Picture Menu : unset preferred and scale**

To stop to define a preferred size and scale was set

### **Picture Menu : format**

By default the diagram's canvas size is 840 x 1188 points corresponding to an iso A4 format (210\*4 x 297\*4), all the available formats are :

- iso A5 : 592 x 840
- iso A4 : 840 x 1188
- iso A3 : 1188 x 1680
- iso A2 : 1680 x 2376
- iso A1 : 2376 x 3364
- iso A0 : 3364 x 4756
- USA A : 864 x 1116
- USA B : 1116 x 1728
- USA C : 1728 x 2236
- USA D : 2236 x 3456

- USA E : 3456 x 4472

Each format also exists in *landscape*

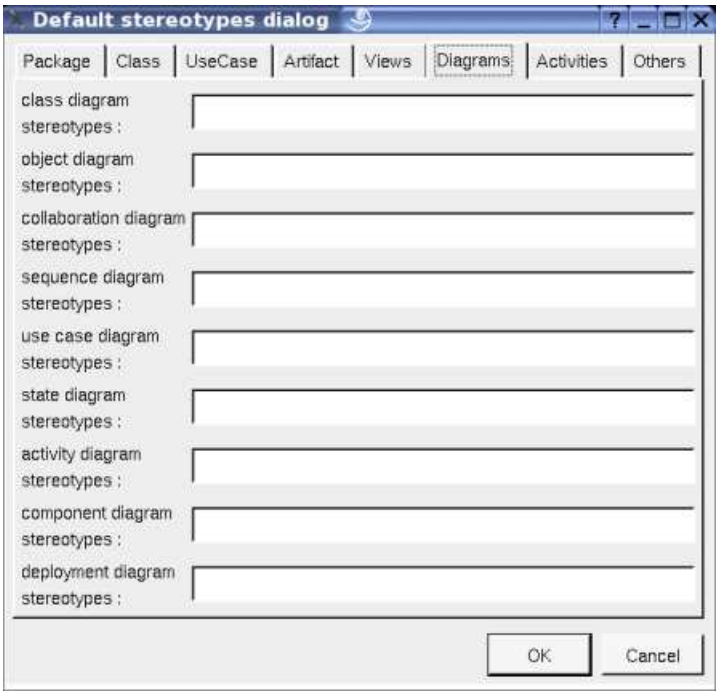
**Picture Menu : undo changes**

Undo all the changes made since the diagram was opened the last time. Note that the diagram is restored independently of the th project saving. Obviously the deleted browser items was not restored !

---

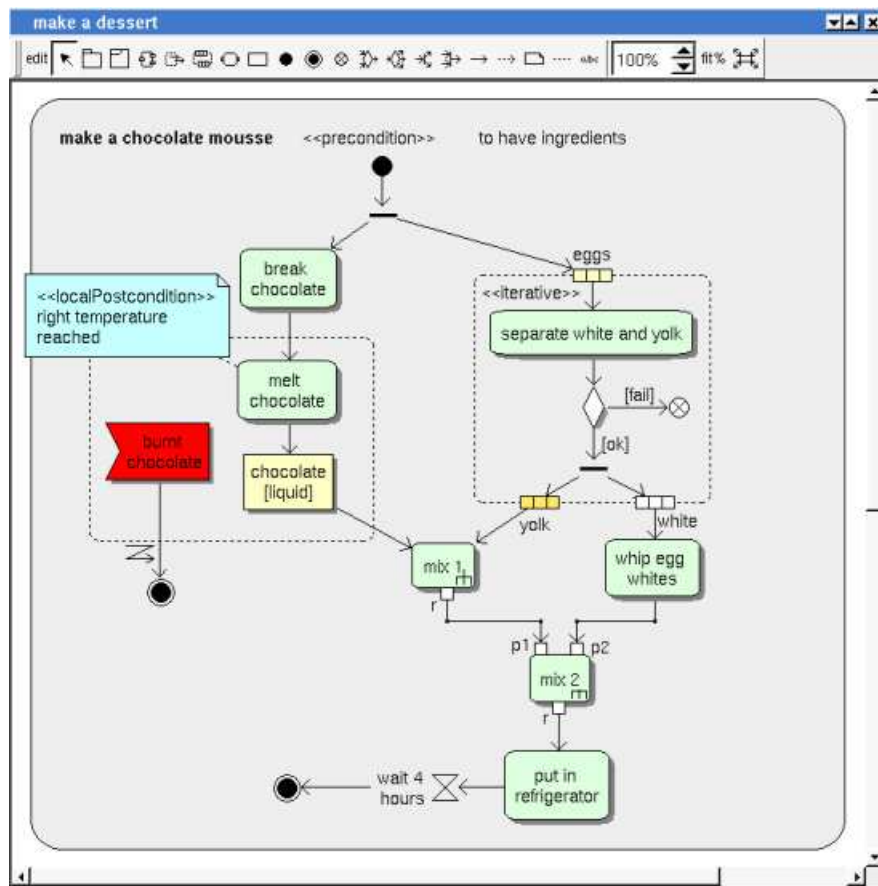
## Default stereotypes

The [dialog](#) allowing to set the default stereotypes has a tab reserved for the diagrams :



---

## Drawing



A *activity diagram* may contain [activity](#), [activity regions](#), [activity actions](#), [activity objects](#), [activity control nodes](#), [packages](#), [fragment](#), [notes](#), [texts](#), [diagram icons](#), and the [flows](#).

To place these elements in the diagram, use the buttons on the top (to create a new element or choose from the already defined one) then click at the appropriate position in the diagram, or *grab* the element from the browser into the diagram.

Warning : when you use the buttons to add an *activity node* and you click on a *activity* or *region* drawing, the new activity will be nested in the *activity* or *region*. In case you place the new *element* out of any *activity* or *region* drawing, the new one will be nested in *activity* owning the diagram.

To add a new flow or dependency between elements, select the appropriate arrow through the buttons on the top of the diagram sub window, click on the start element with the left mouse button, move the mouse, each time the mouse button is raised a new line break is introduced, at least click on the end element. To abort a line construction press the right mouse button or do a double click with the left mouse button.

A line may be broken, during its construction or after clicking on the line with the left mouse button and moving the mouse with mouse button still pushed. To remove a line break, a double click on the point with the left mouse button is enough, or use the line break menu of the point using the right mouse button.

The *scale* may be changed from 30 up to 200 %. To allow to see the operation/message kinds even with a small scale, the size of the unmodified.

*fit%* allows to set the largest scale allowing to show all the picture, as it is possible. Because of the scrollbar management, hitting several times on *fit%* may give several scales.

Classically, the left mouse click may be used to select an element, or several when the *control* key is down, a multiple selection may also be done defining a region using the mouse left click out of any element and moving the mouse with the click and maintaining the click down. The diagram's elements may be moved after a left mouse click on an element and maintaining the click down during mouse move. The right mouse click is used to show context dependent menus.

*control s* is a shortcut to save the project

*control a* is a shortcut to select all the diagram's elements

*control d* is a shortcut to delete the selected element(s), the model is affected

*suppr* is a shortcut to remove the selected element(s) from the diagram, these elements are not deleted

*control c* is a shortcut to copy the selected diagram's elements

*control x* is a shortcut to cut (remove from view) the selected diagram's elements

*control v* is a shortcut to paste the selected diagram's elements

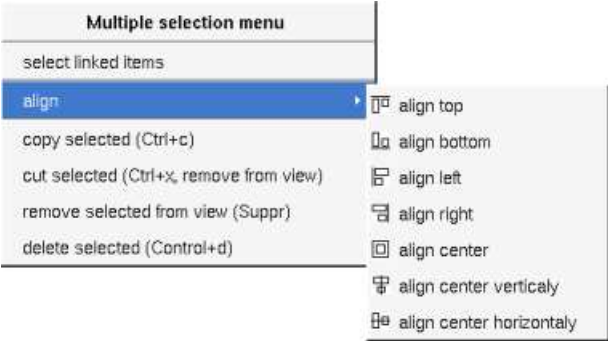
*control z* and *control u* are a shortcut of *undo*

*control r* and *control y* are a shortcut of *redo*

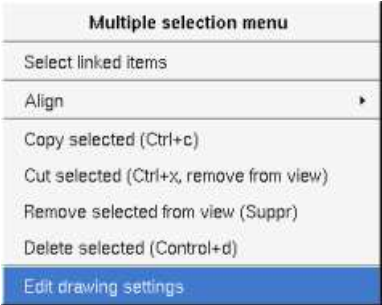
the keyboard arrows allow to move the selected items

You can define your own shortcut using the [shortcut editor](#) through the *Miscellaneous* menu.

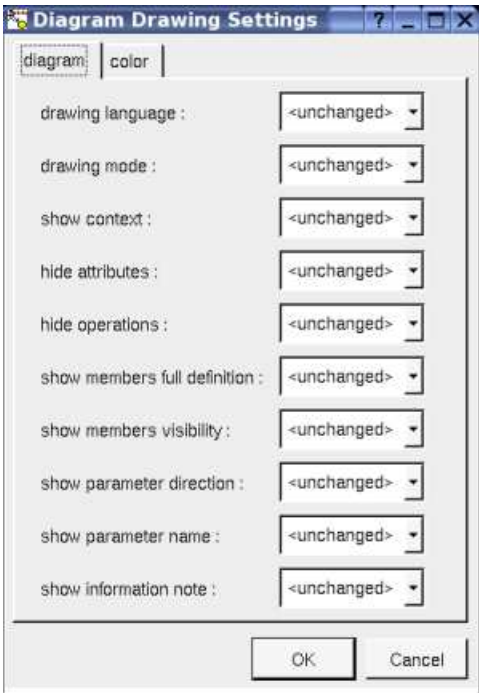
When several diagram's elements are selected, the menu appearing on a left mouse click allows to align them :



If all the selected elements has the same kind (for instance all are a class), you edit their *drawing settings* globally :



and for instance when the elements are classes in a class diagram :



The settings valuing *<unchanged>* are not changed for the selected elements, in the other case the same value is set for all the elements.

**Fragment**

A *fragment* is resizable. To change the fragment's name choose the *edit* entry of its menu or double click on it

A right mouse click on a note show the following menu :



*Edit drawing settings* allows to change the fill color

**Note**

A *note* may contain any text, a note is resizable. To change the text of a note choose the *edit* entry of its menu or double click on it, this open a window containing the text, close the window to end the edition :



A right click on the edited note calls a dialog allowing some text manipulations thanks Qt.

A right mouse click on a note show the following menu :

Note
Upper
Lower
Edit
Color of text
Font ▶
Edit drawing settings
Select linked items
Remove from view

The *font* may be changed, by default the font is the one used to write the other texts.

The color of the text may be changed, a text is black by default.

*Edit drawing settings* allows to change the fill color

*Select linked items* select all the items connected this the current one through any line, in the previous picture this select all except the package, the text “*upper diagram :*” and the use case diagram icon.

**Text**

*Text* is another way to write a text, contrary to the note a *text* is written alone (see “*upper diagram :*” on the picture below), by default a text contains *blabla*: a text cannot be empty. The edition of a text is done like for a note. A text is resizeable, a resize is not automatically done when the contains is modified.

A right mouse click on a *text* show the following menu :

Text
Upper
Lower
Edit
Color
Font ▶
Select linked items
Remove from view

**Diagram icon**

A diagram icon is a shortcut to a diagram, this allows to quickly navigate through the diagram without limitation (a actor or some other item kinds may be associated to only one diagram). The used icon indicates the diagram kind, but its name is not written.

A double mouse left click on a diagram icon opens the diagram, the *open* entry of the diagram icon menu does the same thing.

A right mouse click on a diagram icon show the following menu :

Overall short cut
upper
lower
open
select diagram in browser
select linked items
remove from view

The other diagram's elements have an own chapter, refer to them.

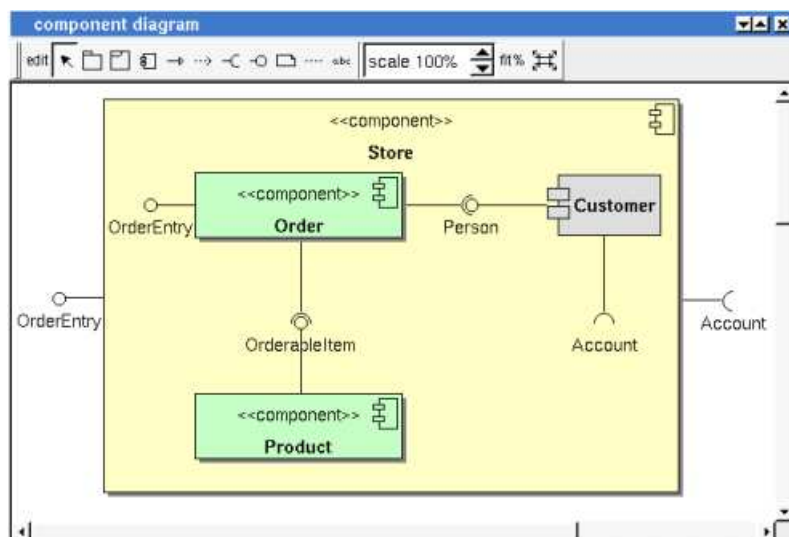


Previous : [state diagram](#)

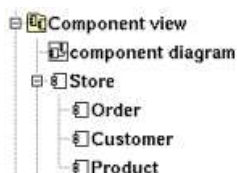
Next : [component diagram](#)



# Component diagram



The component diagrams may be placed in a [component view](#)



A component diagram is created through the *new component diagram* entry of the [component view](#) browser menus.

## Browser menu

The node menu appearing with a right mouse click on its representation in the browser is something like these, supposing the diagram not *read-only* nor deleted :

organ
show
edit
edit drawing settings
duplicate
delete
mark
tool ▶

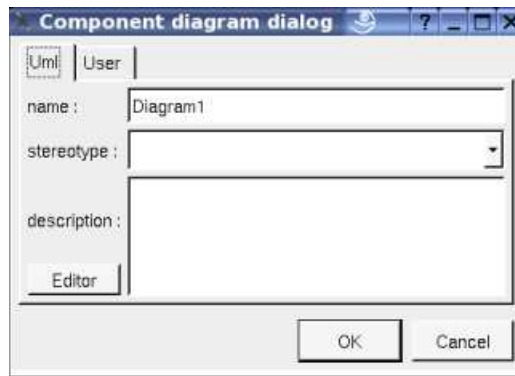
### [Browser Menu](#) : show

*show* allows to show the diagram [picture](#), and edit it in case it is not read only.

### [Browser Menu](#) : edit

*edit* allows to show/modify the diagram properties. In case the diagram is read-only, the fields of the dialog are also read-only.

**component diagram dialog, tab Uml**

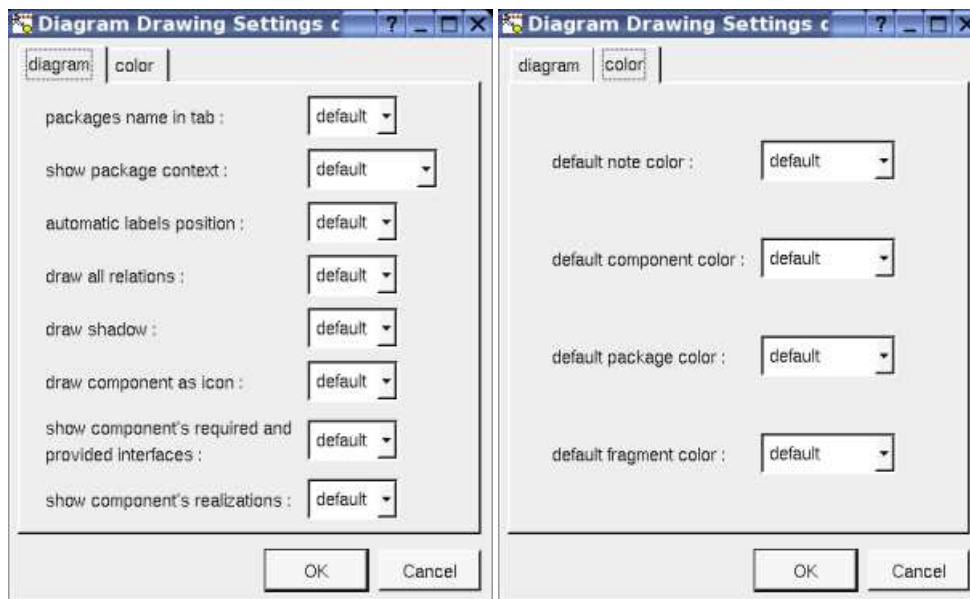


The proposed *stereotypes* are the default one specified through the [Default stereotypes dialog](#) more the current one (empty here). It is possible to choose into the list or to give a new one, or to empty it. The *stereotype* of a diagram doesn't have a special meaning for BOUML

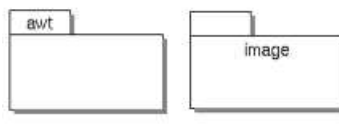
The *editor* button visible above and associated here to the description, allows to edit the description in an other window, or to call an external editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note that this external editor have to create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel with BOUML which looks each second at the file contents to get the new definition until the dialog is closed (I do not like very much polling but this works even QT isn't compiled with the thread support).

### **Browser Menu : edit drawing settings**

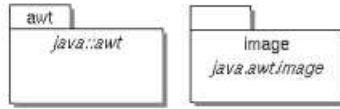
This dialog allows to specify how the items of the diagram must be drawn by default, you may choose *default* for each value, in this case the effective value if specified in the upper level (which itself may specify *default* etc ... up to the *project* level).



- *package name in tab* : to indicate if the packages's name must be written in the tab which is the small rectangle on the top :



- *show package context* : to indicate if the context where the packages are defined must be written, it is not the case just above. The context may be "UML context" which is the path of the package in the browser (choice followed for *awt* below), or the C++ namespace specified by the package, or the Java package specified by the package (choice followed for *picture* below) or at least the Idl module :



- *automatic* *labels* *position*  
: To ask BOUML to move or to not move the associated labels on the relations in the diagram when the relation is edited or the linked items moved etc ...
- *draw* *component* *as* *icon*  
: by default the component drawing follows the UML 2.0 notation, this setting allows to use the old representation (see [customer](#))
- *show component's required and provided interfaces*: to indicate if the required and provided interfaces of the component must be written in the drawing of the component as a compartment.
- *show* *component's* *realization*  
: to indicate if the realizations of the component must be written in the drawing of the component as a compartment.

### **Browser Menu : duplicate**

Clone the diagram in a new one in the same diagram view

### **Browser Menu : delete**

The *delete* entry is only present when the diagram is not *read-only*.

Delete the diagram and all its representation in the opened diagrams. After that it is possible to undelete it (from the browser) until you close the project : obviously the deleted items are not saved !

### **Menu : mark**

See [mark](#)

### **Browser Menu : tools**

The menu entry *tool* is only present in case at least a [plug-out](#) may be applied on the diagram. The selected tool is called and applied on the current diagram.

## **Picture menu**

The diagram menu appearing with a right mouse click out off any item in the diagram picture is something like these, supposin diagram not *read-only* :

Component diagram menu
Edit drawing settings
Select diagram in browser
Select all (Ctrl+a)
Find selected browser element
Copy optimal picture part
Copy visible picture part
Save optimal picture part (png)
Save visible picture part (png)
Save optimal picture part (svg)
Save visible picture part (svg)
Optimal scale
Optimal window size
Set preferred size and scale
Set preferred scale (size unset)
Restore preferred size and scale
Unset preferred size and scale
Format
Undo all changes

**Picture Menu : edit drawing settings**

see [edit drawing settings](#).

**Picture Menu : select all**

Select all the items of the current diagram, may also be done through *control a*

**Picture Menu : find selected browser element**

To find in the diagram a representation of the elements selected in the browser

**Picture Menu : optimal scale**

similar to the *fit scale* button, change scale to see all the diagram's elements at great as possible

**Picture Menu : optimal window size**

Similar to the button , change the size of the windows to see all the diagram's elements with the current scale

**Picture Menu : copy optimal picture part**

Copy all the diagram elements, equivalent to a *optimal windows size* followed by a *copy visible picture part* then a restore of the original windows size

**Picture Menu : copy visible part**

Put the visible part of the current diagram in the clipboard (*copy*), it is a screen copy, the current scale have impact on the picture quality.

It is also possible to [print](#) all (not only the visible part) the current diagram on printer or a file through the button

**Picture Menu : save optimal picture part**

To save all the diagram elements in a file, equivalent to a *optimal windows size* followed by a *save visible picture part* then a restore of the original windows size

**Picture Menu : save visible picture part**

To save the visible part of the current diagram in a file, it is a screen copy, the current scale have impact on the picture quality.

**Picture Menu : set preferred size and scale**

To memorize the current diagram sub window size and the current scale. These ones will be used the next time the diagram will be opened.

### **Picture Menu : set preferred scale**

To memorize the current scale. These one will be used the next time the diagram will be opened.

### **Picture Menu : restore preferred size and scale**

In case a preferred sub window size and preferred scale was set, follow them.

### **Picture Menu : unset preferred and scale**

To stop to define a preferred size and scale was set

### **Picture Menu : format**

By default the diagram's canvas size is 840 x 1188 points corresponding to an iso A4 format (210\*4 x 297\*4), all the available formats are :

- iso A5 : 592 x 840
- iso A4 : 840 x 1188
- iso A3 : 1188 x 1680
- iso A2 : 1680 x 2376
- iso A1 : 2376 x 3364
- iso A0 : 3364 x 4756
- USA A : 864 x 1116
- USA B : 1116 x 1728
- USA C : 1728 x 2236
- USA D : 2236 x 3456
- USA E : 3456 x 4472

Each format also exists in *landscape*

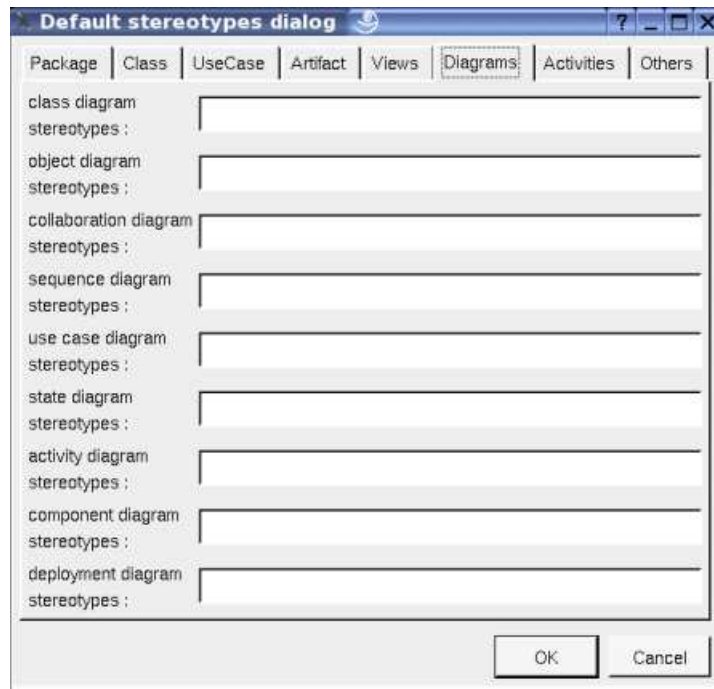
### **Picture Menu : undo changes**

Undo all the changes made since the diagram was opened the last time. Note that the diagram is restored independently of the th project saving. Obviously the deleted browser items was not restored !

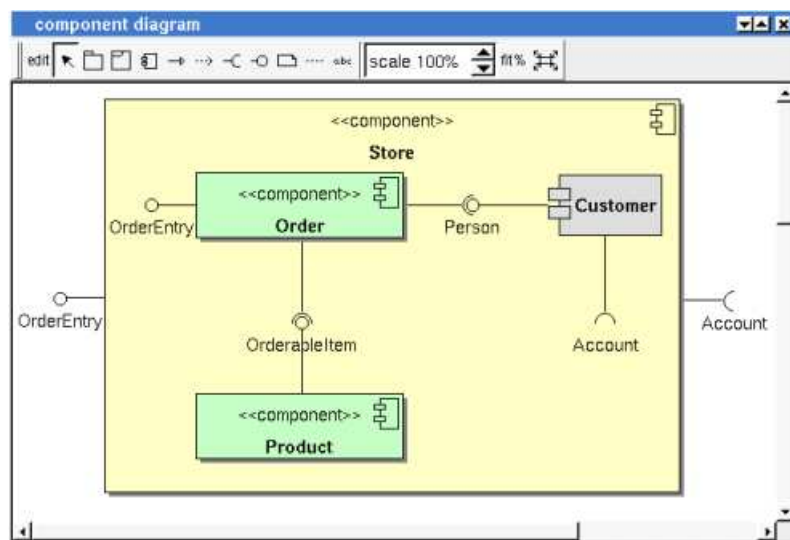
---

## **Default stereotypes**

The [dialog](#) allowing to set the default stereotypes has a tab reserved for the diagrams :





## Drawing



A component diagram may contain [component](#), [package](#), [fragment](#), [notes](#), [texts](#), [diagram icons](#) and the relations.

Note that the association between components does not appears in the browser, there are establish/deleted throw the compone diagrams or when you [edit](#) the component from where the relation start.

To place these elements in the diagram, use the buttons on the top (to create a new element or choose from the already defined one then click at the appropriate position in the diagram, or *grab* the element from the browser into the diagram.

To add *required* and *provided* interface indications, use the appropriate button, click on the component, move the mouse click down and release the click on a already placed  or  or nowhere, in this least case BOUML will ask you to select a class from the list set by editing the component (*OrderEntry* and *Account* are available for the component *Store* because *Customer* and *Order* are nested in *Store*).

The *scale* may be changed from 30 up to 200 %. To allows to see the relation kinds even with a small scale, the size of the arrows is unmodified.

*fit%* allows to set the largest scale allowing to show all the picture, as it is possible. Because of the scrollbar management, hitting seve times on *fit%* may give several scales.

Classically, the left mouse click may be used to select an element, or several when the *control* key is down, a multiple selection may

also be done defining a region using the mouse left click out of any element and moving the mouse with the click and maintaining the click down. The diagram's elements may be moved after a left mouse click on an element and maintaining the click down during mouse move. The right mouse click is used to show context dependent menus.

*control s* is a shortcut to save the project

*control a* is a shortcut to select all the diagram's elements

*control d* is a shortcut to delete the selected element(s), the model is affected

*suppr* is a shortcut to remove the selected element(s) from the diagram, these elements are not deleted

*control c* is a shortcut to copy the selected diagram's elements

*control x* is a shortcut to cut (remove from view) the selected diagram's elements

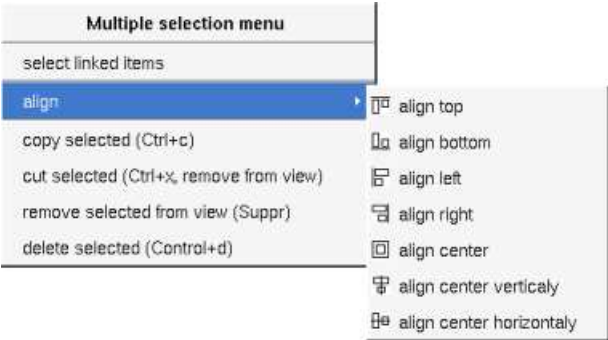
*control v* is a shortcut to paste the selected diagram's elements

*control z* and *control u* are a shortcut of *undo*

*control r* and *control y* are a shortcut of *redo*

the keyboard arrows allow to move the selected items

When several diagram's elements are selected, the menu appearing on a left mouse click allows to align them :



Fragment

A *fragment* is resizeable. To change the fragment's name choose the *edit* entry of its menu or double click on it

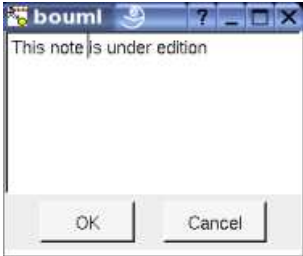
A right mouse click on a note show the following menu :



*Edit drawing settings* allows to change the fill color

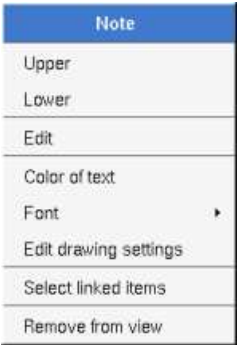
Note

A *note* may contain any text, a note is resizeable. To change the text of a note choose the *edit* entry of its menu or double click on it, this open a window containing the text, close the window to end the edition :



A right click on the edited note calls a dialog allowing some text manipulations thanks Qt.

A right mouse click on a note show the following menu :



The *font* may be changed, by default the font is the one used to write the other texts.

The color of the text may be changed, a text is black by default.

*Edit drawing settings* allows to change the fill color

*Select linked items* select all the items connected this the current one through any line, in the previous picture this select all except the package, the text “*upper diagram :*” and the use case diagram icon.

**Text**

*Text* is another way to write a text, contrary to the note a *text* is written alone (see “*upper diagram :*” on the picture below), by default a text contains *blabla*: a text cannot be empty. The edition of a text is done like for a note. A text is resizeable, a resize is not automatically done when the contains is modified.

A right mouse click on a *text* show the following menu :



**Diagram icon**

A diagram icon is a shortcut to a diagram, this allows to quickly navigate through the diagram without limitation (a actor or some other item kinds may be [associated](#) to only one diagram). The used icon indicates the diagram kind, but its name is not written.

A double mouse left click on a diagram icon opens the diagram, the *open* entry of the diagram icon menu does the same thing.

A right mouse click on a diagram icon show the following menu :



Overall short cut
upper
lower
open
select diagram in browser
select linked items
remove from view

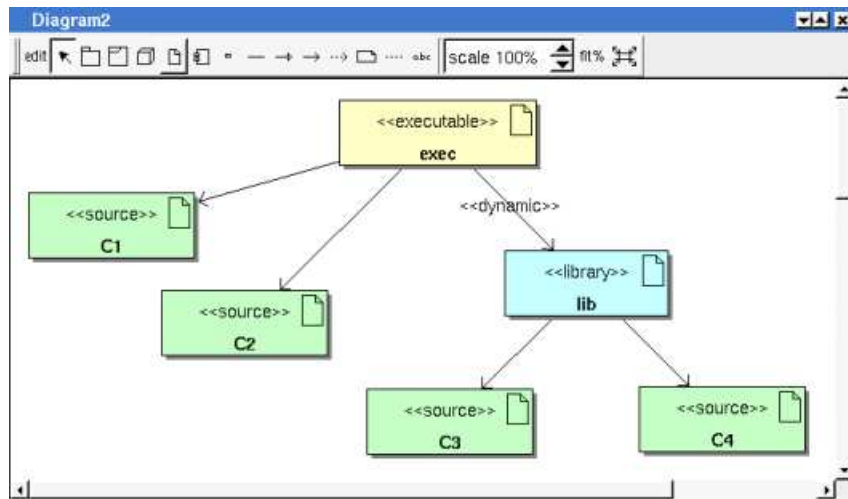
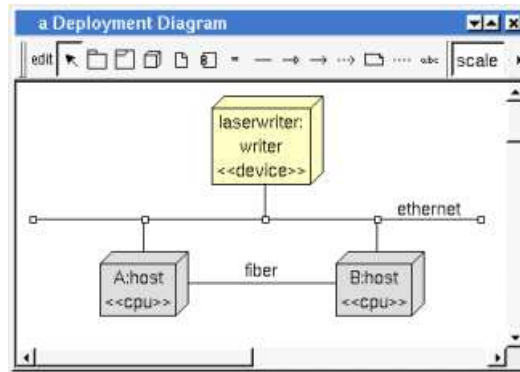
The other diagram's elements have an own chapter, refer to them.

---

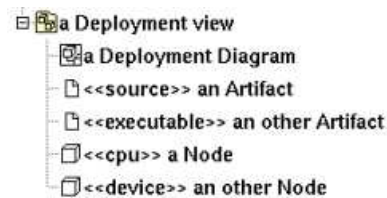
Previous : [class diagram](#)

Next : [deployment diagram](#)

# Deployment diagram



The deployment diagrams may be placed in a [deployment view](#)



A deployment diagram is created through the *new deployment diagram* entry of the [deployment view](#) browser menus.

## Browser menu

The node menu appearing with a right mouse click on its representation in the browser is something like these, supposing the diagram not *read-only* nor deleted :



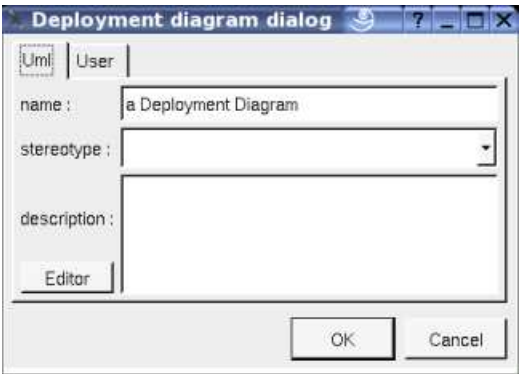
### [Browser Menu](#) : show

*show* allows to show the diagram [picture](#), and edit it in case it is not read only.

### Browser Menu : edit

*edit* allows to show/modify the diagram properties. In case the diagram is read-only, the fields of the dialog are also read-only.

#### deployment diagram dialog, tab Uml

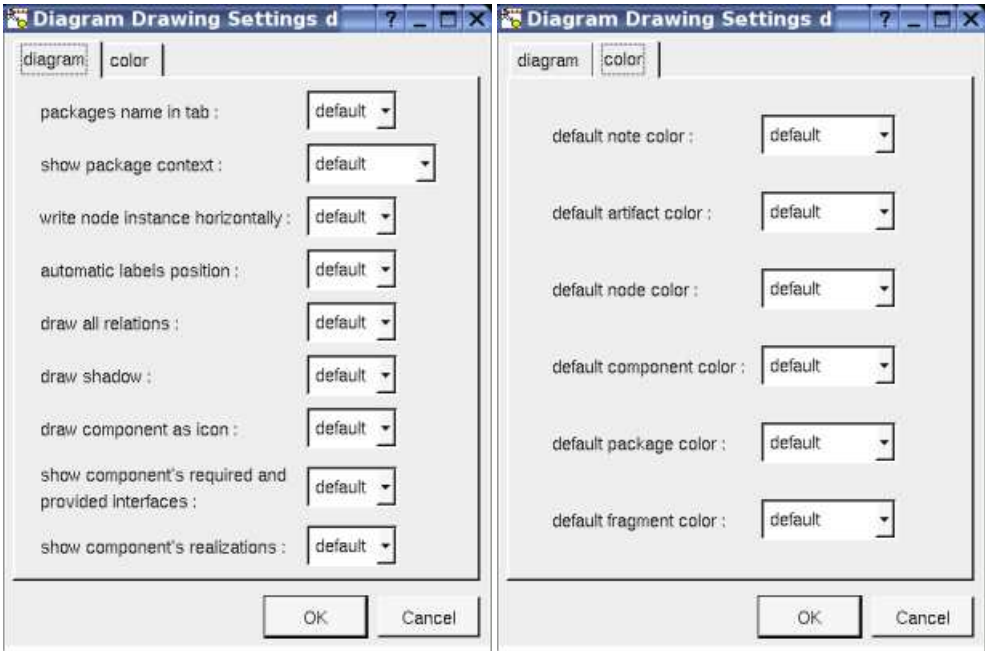


The proposed *stereotypes* are the default one specified through the [Default stereotypes dialog](#) more the current one (empty here). It is possible to choose into the list or to give a new one, or to empty it. The *stereotype* of a diagram doesn't have a special meaning for BOUML

The *editor* button visible above and associated here to the description, allows to edit the description in an other window, or to call an external editor (for instance *Xcoral*) specified through the [environment variable](#) BOUML\_EDITOR. Note that this external editor have to create an own window, its parameter is the pathname of the file containing the description, its execution is done in parallel wi BOUML which looks each second at the file contents to get the new definition until the dialog is closed (I do not like very much polling but this works even QT isn't compiled with the thread support).

### Browser Menu : edit drawing settings

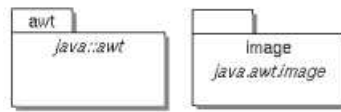
This dialog allows to specify how the items of the diagram must be drawn by default, you may choose *default* for each value, in this case the effective value if specified in the upper level (which itself may specify *default* etc ... up to the *project* level).



- *package* *name* *in* *tab*  
: to indicate if the package's name must be written in the tab which is the small rectangle on the top (note : a transparent *package* doesn't have shadow) :



- *show* *packages* *context*  
: to indicate if the context where the package is defined must be written, it is not the case just above. The context may be the “UML context” which is the path of the package in the browser (choice followed for *awt* below), or the C++ namespace specified by the package, or the Java package specified by the package (choice followed for *picture* below) or at least the Idl module :



- *write node instance horizontally* : Allows to write the the node type and the instance name on the same line (see host [below](#)) or on two lines (see laserwriter [below](#)).
- *automatic* *labels* *position*  
: To ask BOUML to move or to not move the associated labels on the relations in the diagram when the relation is edited or the linked items moved etc ...

## **Browser Menu : duplicate**

Clone the diagram in a new one in the same deployment view.

## **Browser Menu : delete**

The *delete* entry is only present when the diagram is not *read-only*.

Delete the diagram and all its representation in the opened diagrams. After that it is possible to undelete it (from the browser) until you close the project : obviously the deleted items are not saved !

## **Menu : mark**

See [mark](#)

## **Browser Menu : tools**

The menu entry *tool* is only present in case at least a *plug-out* may be applied on the diagram. The selected tool is called and applied on the current diagram.

## **Picture menu**

The diagram menu appearing with a right mouse click out off any item in the diagram picture is something like these, supposing diagram not *read-only* :

Deployment diagram menu
Edit drawing settings
Select diagram in browser
Select all (Ctrl+a)
Find selected browser element
Copy optimal picture part
Copy visible picture part
Save optimal picture part (png)
Save visible picture part (png)
Save optimal picture part (svg)
Save visible picture part (svg)
Optimal scale
Optimal window size
Set preferred size and scale
Set preferred scale (size unset)
Restore preferred size and scale
Unset preferred size and scale
Format
Undo all changes

**Picture Menu : edit drawing settings**

see [edit drawing settings](#).

**Picture Menu : select all**

Select all the items of the current diagram, may also be done through *control a*

**Picture Menu : find selected browser element**

To find in the diagram a representation of the elements selected in the browser

**Picture Menu : optimal scale**

similar to the *fit scale* button, change scale to see all the diagram's elements at great as possible

**Picture Menu : optimal window size**

Similar to the button , change the size of the windows to see all the diagram's elements with the current scale

**Picture Menu : copy optimal picture part**

Copy all the diagram elements, equivalent to a *optimal windows size* followed by a *copy visible picture part* then a restore of the original windows size

**Picture Menu : copy visible part**

Put the visible part of the current diagram in the clipboard (*copy*), it is a screen copy, the current scale have impact on the picture quality.

It is also possible to [print](#) all (not only the visible part) the current diagram on printer or a file through the button

**Picture Menu : save optimal picture part**

To save all the diagram elements in a file, equivalent to a *optimal windows size* followed by a *save visible picture part* then a restore of the original windows size

**Picture Menu : save visible picture part**

To save the visible part of the current diagram in a file, it is a screen copy, the current scale have impact on the picture quality.

**Picture Menu : set preferred size and scale**

To memorize the current diagram sub window size and the current scale. These ones will be used the next time the diagram wil opened.

**Picture Menu : set preferred scale**

To memorize the current scale. These one will be used the next time the diagram will be opened.

**Picture Menu : restore preferred size and scale**

In case a preferred sub window size and preferred scale was set, follow them.

**Picture Menu : unset preferred and scale**

To stop to define a preferred size and scale was set

**Picture Menu : format**

By default the diagram's canvas size is 840 x 1188 points corresponding to an iso A4 format (210\*4 x 297\*4), all the available formats are :

- iso A5 : 592 x 840
- iso A4 : 840 x 1188
- iso A3 : 1188 x 1680
- iso A2 : 1680 x 2376
- iso A1 : 2376 x 3364
- iso A0 : 3364 x 4756
- USA A : 864 x 1116
- USA B : 1116 x 1728
- USA C : 1728 x 2236
- USA D : 2236 x 3456
- USA E : 3456 x 4472

Each format also exists in *landscape*

**Picture Menu : undo changes**

Undo all the changes made since the diagram was opened the last time. Note that the diagram is restored independently of the th project saving. Obviously the deleted browser items was not restored !

---

**Network Menu**

The menu appearing with a right mouse click on a network line is something like these, supposing the diagram not *read-only* :

ethernet
edit
select stereotype and label
default stereotype and label position
remove from view

**Network Menu : edit**

*edit* allows to edit the network :

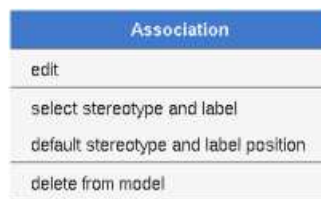


### **Network Menu : select stereotype and label**

To select the network label and stereotype. Useful when you are lost after many label movings.

## **Artifact association Menu**

The menu appearing with a right mouse click on an association between artifacts is something like these, supposing the diagram not *read-only* :



### **Association Menu : edit**

*edit* allows to edit the association :



### **Association Menu : select stereotype and label**

To select the association's label and stereotype. Useful when you are lost after many label movings. Only appears when the association have a stereotype or a label or both.

### **Association Menu : default stereotype and label position**

To place the association's label and stereotype in the default position. Only appears when the association have a stereotype or a label or both.. Useful when you are lost after many label movings.

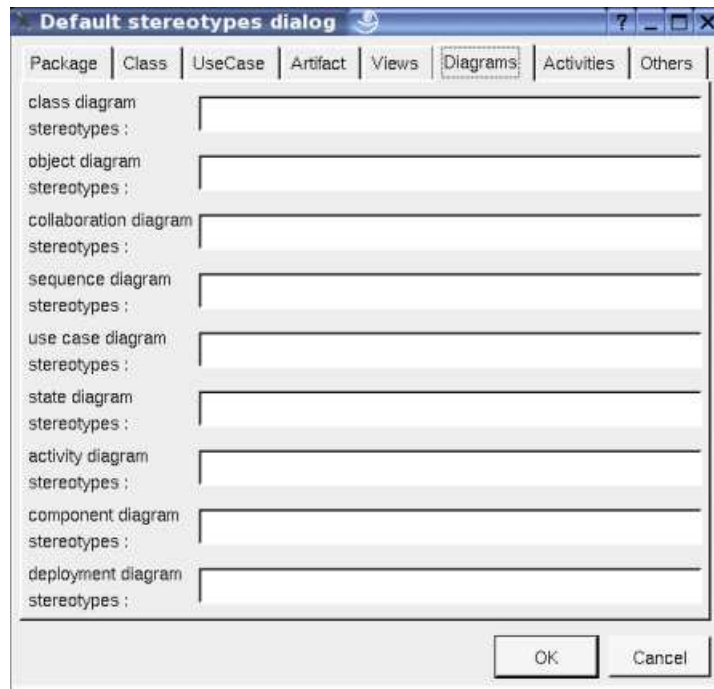
By default when you move a class or a relation point break or edit the relation, the associated labels are moved to their default position, this may be irritating. To ask BOUML to not move them automatically in a diagram, use the [drawing settings](#) of the diagrams.

### **Association Menu : delete from model**

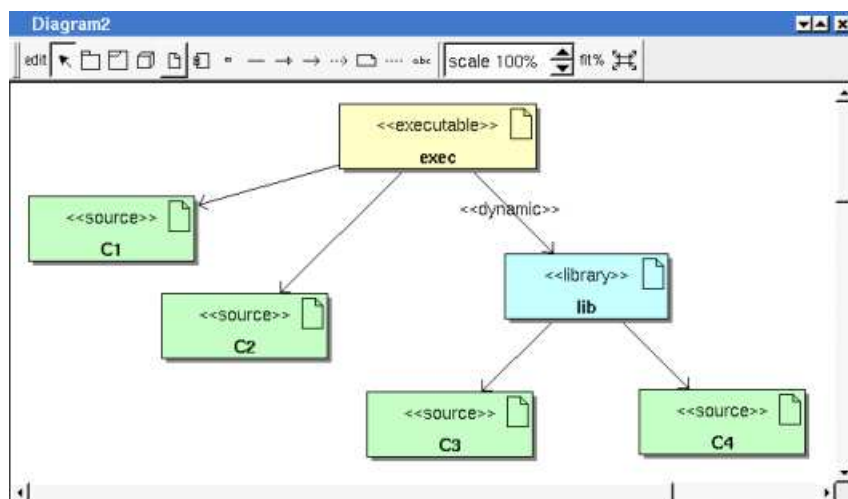
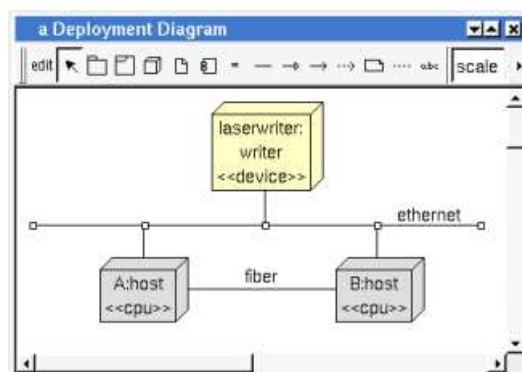
Remove the association.

## **Default stereotypes**

The [dialog](#) allowing to set the default stereotypes has a tab reserved for the diagrams :



## Drawing



A deployment diagram may contain deployment [node](#), network line, network connexion, [component](#), [fragment](#), [notes](#), [texts](#), [diagram icons](#) and the relations.

To place these elements in the diagram, use the buttons on the top (to create a new element or choose from the already defined one then click at the appropriate position in the diagram, or *grab* the element from the browser into the diagram.

The **network lines** may connect deployment nodes and/or network connexion. The **network connexions** (represented by small squares) allows to have network lines out of deployment node as it is the case in the picture above.

The

scale



may be changed from 30 up to 200 %. To allows to see the relation kinds even with a small scale, the size of the arrows is unmodified.

*fit%*  
allows to set the largest scale allowing to show all the picture, as it is possible. Because of the scrollbar management, hitting seve times on *fit%* may give several scales.

Classically, the left mouse click may be used to select an element, or several when the *control* key is down, a multiple selection may also be done defining a region using the mouse left click out of any element and moving the mouse with the click and maintaining the click down. The diagram's elements may be moved after a left mouse click on an element and maintaining the click down during mouse move. The right mouse click is used to show context dependent menus.

*control s* is a shortcut to save the project

*control a* is a shortcut to select all the diagram's elements

*control d* is a shortcut to delete the selected element(s), the model is affected

*suppr* is a shortcut to remove the selected element(s) from the diagram, these elements are not deleted

*control c* is a shortcut to copy the selected diagram's elements

*control x* is a shortcut to cut (remove from view) the selected diagram's elements

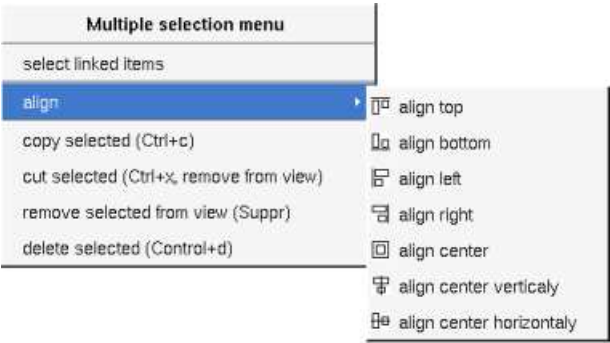
*control v* is a shortcut to paste the selected diagram's elements

*control z* and *control u* are a shortcut of *undo*

*control r* and *control y* are a shortcut of *redo*

the keyboard arrows allow to move the selected items

When several diagram's elements are selected, the menu appearing on a left mouse click allows to align them :



## Fragment

A *fragment* is resizeable. To change the fragment's name choose the *edit* entry of its menu or double click on it

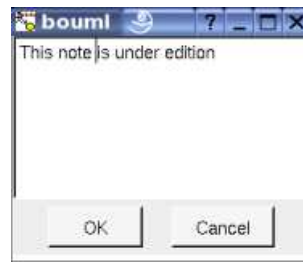
A right mouse click on a note show the following menu :



*Edit drawing settings* allows to change the fill color

## Note

A *note* may contain any text, a note is resizeable. To change the text of a note choose the *edit* entry of its menu or double click on it, this open a window containing the text, close the window to end the edition :



A right click on the edited note calls a dialog allowing some text manipulations thanks Qt.

A right mouse click on a note show the following menu :



The *font* may be changed, by default the font is the one used to write the other texts.

The color of the text may be changed, a text is black by default.

*Edit drawing settings* allows to change the fill color

*Select linked items* select all the items connected this the current one through any line, in the previous picture this select all except the package, the text “*upper diagram :*” and the use case diagram icon.

## Text

*Text* is another way to write a text, contrary to the note a *text* is written alone (see “*upper diagram :*” on the picture below), by default a text contains *blabla*: a text cannot be empty. The edition of a text is done like for a note. A text is resizeable, a resize is not automatically done when the contains is modified.

A right mouse click on a *text* show the following menu :



## Diagram icon

A diagram icon is a shortcut to a diagram, this allows to quickly navigate through the diagram without limitation (a actor or some other item kinds may be [associated](#) to only one diagram). The used icon indicates the diagram kind, but its name is not written.

A double mouse left click on a diagram icon opens the diagram, the *open* entry of the diagram icon menu does the same thing.

A right mouse click on a diagram icon show the following menu :

Overall short cut
upper
lower
open
select diagram in browser
select linked items
remove from view

The other diagram's elements have an own chapter, refer to them.

---

Previous : [component diagram](#)

Next : [C++ generator](#)

# C++ Generator

The C++ code generator is a *plug-out* directly written in C++ (the C++ code generator does not generate itself !).

The generated sources follows the definition made in BOUML at the [artifact](#) / [class](#) / [operation](#) / [relation](#) / [attribute](#) / [extra member](#) levels.

When the code generation is applied on a artifact associated to several classes, the code generation is made for all these classes. Nevertheless the C++ code generator produce first the code in memory and update the appropriate files only when it is necessary, to not change the last write date of the files for nothing. Depending on the toggle *verbose code generation* of the global menu *Languages* the code generator is verbose or not.

The C++ code generator *plug-out* may be called on :

- a [class](#) : in this case the code generation is in fact applied on the class's [artifact](#), then on all the classes associated to this [artifact](#)
- a [artifact](#) : in this case the code generation is in fact applied on all the [classes](#) associated to the [artifact](#)
- a [class view](#) : the code generation will be applied on all the sub [classes](#), then on all the [artifacts](#) associated to these [classes](#)
- a [deployment view](#) : the code generation is applied on all the sub [artifacts](#)
- a [package](#) (may be the project itself) : the code generation will be applied on all the sub [class views](#) and [deployment views](#), then on all their sub [classes](#) and [artifacts](#).

When the C++ code generator is asked through the [Tools](#) menu, it is applied on the project, then on all the [artifacts](#).

The name of the generated files depend on the artifact name, the extension depend on the language and is fixed for each by the [generations settings](#) (see below), the directory where the files are generated may be set in each package containing directly or indirectly the artifact (see [generation directory](#)).

## artifact

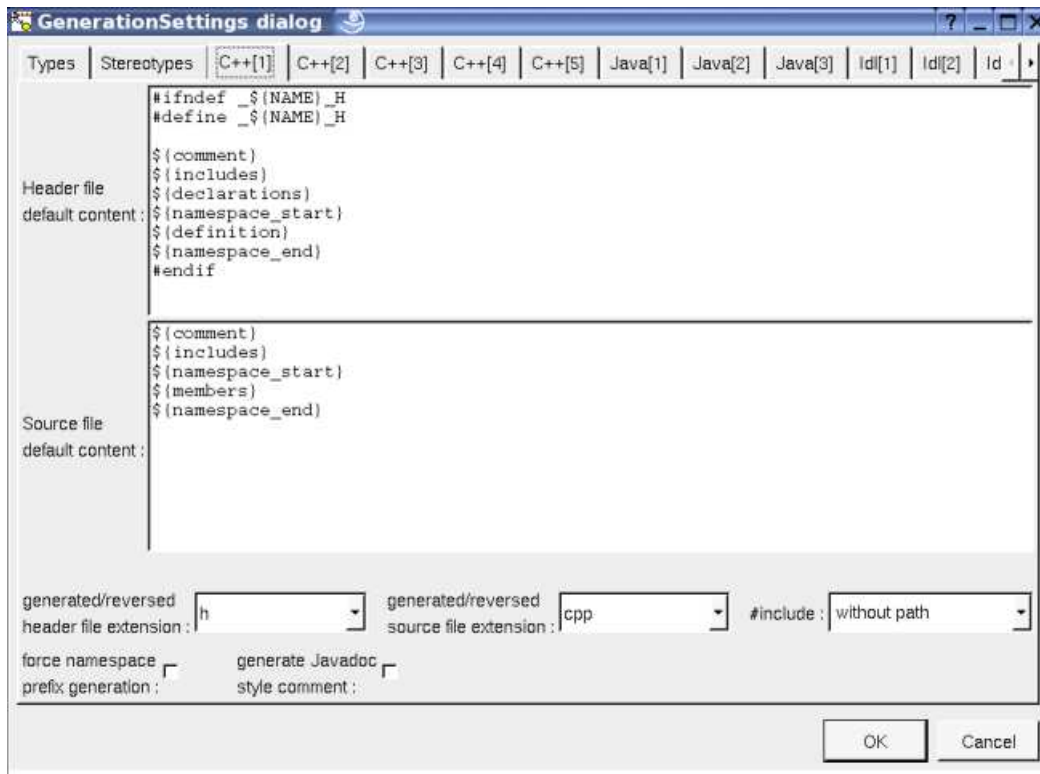
The C++ definition of a artifact is set through the [C++ header](#) and [C++ source](#) tabs of the [artifact dialog](#).

The code generation depend on the stereotype of the artifact :

- *text* : the C++ definition of the artifact is produced without changes, the name of the generated file is the name of the artifact, including the extension.
- *source* : see below.
- else nothing is generated for the artifact.

## artifact header C++

The generated file name is the artifact's name with the extension specified in the first C++ tab of the [generations settings](#) :



In BOUML the generated code is obtained by the substitution of macros in a text, the macros known by the C++ code generator are :

- **`${NAME}`** produce the artifact name capitalized, **`${Name}`** produce the artifact name with the first letter capitalized, at least **`${name}`** produce the artifact name without modification.
- **`${comment}`** is replaced by the artifact description adding //
- **`${description}`** is replaced by the artifact description without adding //
- **`${includes}`** is replaced by the automatically generated `#include` and `using` forms, it is also a good place to add your `#include` and `using` forms, when the ones produced by the code generator are not sufficient. To limit the included files number and to not produce circular `#include`, the C++ code generator does not produce an `#include` to define a type only used by pointer (\*) or reference (&) or place between <> in a template form. The C++ code generator does not look at in the operations body, only the operation's profiles, relations and attributes, classes inheritances etc ... are used to compute the needed `#include` list. You can also use dependencies between classes to add `#include`, and to choose to place them in the header of the source file or in the dependency tab C++. **`${includes}`** and **`${all_includes}`** are exclusive.

Note : the generation settings allows to choose between four ways to produce `#include` :

- *without path* : ask for the C++ code generator to generate the `#include` without relative or absolute path
- *with absolute path* : ask for the C++ code generator to generate the absolute path of the automatically included files.
- *with relative path* : ask for the C++ code generator to generate the relative path of the automatically included files, relative to the file containing the `#include`
- *with root relative path* : ask for the C++ code generator to generate the relative path of the automatically included files, relative to the directory specified by the generation settings
- **`${all_includes}`** is replaced by all the automatically generated `#include` and `using` forms, it is also a good place to add your `#include` and `using` forms, when the ones produced by the code generator are not sufficient. In this case the code generator doesn't produce declarations in the header file nor `#include` in the source file, except the ones added by you. **`${all_includes}`** and **`${includes}`** are exclusive.
- **`${declarations}`** is replaced by the class declarations when the `#include` form is not produced, it is also a good place to add your declarations, when the ones produced by the code generator are not sufficient. Produce nothing if **`${all_includes}`** is used.
- **`${namespace_start}`** is replaced by the `namespace xx {` forms, dependent on the [namespace](#) specifications associated to the package containing the [deployment view](#) where the artifact is defined.

- **`${definition}`** is replaced by the definition of the classes associated to the artifact.
- **`${namespace_end}`** is replaced by the `}` forms, dependent on the [namespace](#) specifications associated to the package containing the [deployment view](#) where the artifact is defined.
- **`@{xyz}`** is replaced by the *user property* value in case *xyz* is the key of this property. The property is searched first for the *artifact*, if not found for the *deployment view* containing the *artifact*, if not found in the *package* containing the *deployment view* etc ...

## artifact source C++

The generated file name is the artifact's name with the extension specified in the first C++ tab of the [generations settings](#).

In BOUML the generated code is obtained by the substitution of macros in a text, the macros known by the C++ code generator are :

- **`${comment}`** is replaced by the description of the artifact adding `//`
- **`${description}`** is replaced by the artifact description without adding `//`
- **`${includes}`** if **`${all_includes}`** is not used in the header definition, it is replaced by the automatically generated `#include` and *using* forms, corresponding to the declaration generated in the header file. If **`${all_includes}`** is used in the header definition, it is replaced by the `#include` of the header. It is also a good place to add your `#include` and *using* forms.
- **`${namespace_start}`** is replaced by the *namespace xx {* forms, dependent on the [namespace](#) specifications associated to the package containing the [deployment view](#) where the artifact is defined.
- **`${members}`** is replaced by the non *inline* members of the classes associated to the artifact.
- **`${namespace_end}`** is replaced by the `}` forms, dependent on the [namespace](#) specifications associated to the package containing the [deployment view](#) where the artifact is defined.
- **`@{xyz}`** is replaced by the *user property* value in case *xyz* is the key of this property. The property is searched first for the *artifact*, if not found for the *deployment view* containing the *artifact*, if not found in the *package* containing the *deployment view* etc ...

## Class

The C++ definition of a class is set through the [Uml](#), [Parametrized](#), [Instantiate](#) and [C++](#) tabs of the [class dialog](#).

A C++ type definition may be a class, a struct, an union, an enum or a typedef, depending on the stereotype and its translation in C++ (see [generation settings](#)).

In BOUML the generated code is obtained by the substitution of macros in a text, the macros known by the C++ code generator are :

- **`${comment}`** is replaced by the description of the class adding `//`
- **`${description}`** is replaced by the class description without adding `//`
- **`${template}`** produce a template declaration in case the class is a template class
- **`${name}`** is replaced by the class's name
- **`${inherit}`** is replaced by the class inheritance
- **`${members}`** is replaced by the code generated for all the class's members ([relations](#), [attributes](#), [operations](#) and [extra members](#)) following the browser order.
- **`${items}`** only for the enums, is replaced by the enum's items
- **`@{xyz}`** is replaced by the *user property* value in case *xyz* is the key of this property. The property is searched first for the class, if not found for the container of the class (an other class of a *class view*) etc ...

In the special case where the class is declared ***external***, its C++ type declaration must contain at least one line indicating how the name of the class is generated, the other lines will be added in the header file of the *components* referencing the class. By default the first line only contains `${name}` meaning that the name is produced unchanged, the only allowed keywords are `${name}`, `${Name}` and `${NAME}`. For instance you want to define the external class *string* and you want to automatically generate `#include <string>` followed by *using namespace std*; you just have to create the class *string*, to set it *external* in C++ and to have the declaration form :

```
${name}
```

```
#include <string>

using namespace std;
```

If you don't want to *use* the *namespace* and ask to write *std::string* the declaration must be :

```
std::{name}

#include <string>
```

## Operation

The C++ definition of an operation is set through the [Uml](#) and [C++](#) tabs of the [operation dialog](#).

The indentation of the first line of the declaration/definition give the indentation added to the visibility specification in the cl: definition for all the operation declaration/definition (except for the pre-processor forms beginning by # whose are placed at beginning of the line).

In BOUML the generated code is obtained by the substitution of macros in a text, the macros known by the C++ code generator are :

- ***\${comment}*** is replaced by the description of the operation adding //
- ***\${description}*** is replaced by the operation description without adding //
- ***\${static}*** produce *static* when the operation is a class operation (see the UML tab)
- ***\${const}*** produce *const* when the operation is const
- ***\${volatile}*** produce *volatile* when the operation is volatile
- ***\${friend}*** produce *friend* when the operation is friend
- ***\${virtual}*** produce *virtual* when the operation is virtual
- ***\${inline}*** produce *inline* when the operation is inline
- ***\${type}*** is replaced by the returned value type (see the UML tab)
- ***\${name}*** is replaced by the operation's name
- ***\${class}*** is replaced by the name of the class containing the operation.
- ***\${(}*** and ***\${)}*** produce ( and ), but there are also a mark for BOUML to find the parameters list
- ***\${abstract}*** produce the string = 0 when the operation is abstract (see the UML tab)
- ***\${staticnl}*** produce a line break when the operation is static, else an empty string.
- ***\${t<n>}*** and ***\${p<n>}***  
produce the type and the name of each parameter (count from 0), this allows you to remove a parameter, add type modifiers etc ...
- ***\${throw}*** is replaced by the form *throw (...)* when at least an exception is defined in the UML tab. It is also possible to produce *throw()* when there is no exception depending on the [generation settings](#).
- ***\${body}***  
is replaced by the body of the operation, this macro may also be replaced by the body itself (BOUML use this way for the *get* and *set* operations associated to a relation/attribute). The usage of ***\${body}*** has a great advantage : when you hit **Default definition** or **From declaration** the body is not cleared ! At least BOUML share the declaration and definition forms of the operations (and other object minimize the needed memory size, ***\${body}*** help for this ! The indentation of the keyword ***\${body}*** is added at the beginning of each line except the ones starting by a '#' or when the previous one ends by '\'.
  - ***@{xyz}*** is replaced by the *user property* value in case *xyz* is the key of this property. The property is searched first for the operation, if not found for the class containing the operation etc ...
  - A **template operation**  
is defined with an empty declaration (allowing Bouml to detect this case) and placing the actuals between ***\${class}*** and ***::*** in the declaration part.



If the toggle *preserve operations's body* is set through the *Languages* menu, the generators do not modify the body of the operations protected by dedicated delimiters. This means that for them the body definition set through BOUML is not used. The first time generate the code with the toggle set, because the delimiters are not yet present in the generated code, the operation's body will be updated depending on their definition under BOUML. After, while the toggle is set and the delimiters present, the bodies will not change, allowing you to modify them out of BOUML.

Notes :

- In case the file containing a body definition is not consistent with the artifact under BOUML, the body will be regenerated by the code generation, using its definition under the model. Of course an operation becoming *inline* or stopping to be *inline* is not a problem.
- When you import a project, the body of the imported operations must be the right one in the imported model. The preserved bodies of the imported operations will not be found because the identifier of an operation used to mark its body changes during the import.
- The bodies under the model are not updated by the code generation, use [roundtrip body](#) for that
- Only the operations using the keyword `#{body}` may have a preserved body.
- The only modification you can do in the lines containing the delimiters is the indent.
- The toggle is saved in the file associated to the project, be sure the save is done when you change this toggle !

## Attribute

The C++ definition of an attribute is set through the [Uml](#) and [C++](#) tabs of the [attribute dialog](#).

The indentation of the first line of the definition gives the indentation added to the visibility specification in the class definition for all the attribute definition (except for the pre-processor commands beginning by # whose are placed at the beginning of the line).

An attribute may be a standard attribute or the item of an enumeration.

### standard attribute

In BOUML the generated code is obtained by the substitution of macros in a text, the macros known by the C++ code generator are :

- `#{comment}` is replaced by the description of the attribute adding //
- `#{description}` is replaced by the attribute description without adding //
- `#{static}` produce *static* when the attribute is a class attribute (see the UML tab)
- `#{mutable}` produce *mutable* when the attribute is mutable
- `#{volatile}` produce *volatile* when the attribute is volatile
- `#{const}` produce *const* when the attribute is const (see the UML tab)
- `#{type}` is replaced by the type of the attribute (see the UML tab)
- `#{stereotype}` is replaced by the [translation in C++](#) of the relation's stereotype (see the UML tab)
- `#{multiplicity}`  
is replaced by the multiplicity of the relation (see the UML tab), must be used in case the multiplicity is a vector or dimensioning ( [ and ] are added when they are not present).
- `#{name}` is replaced by the attribute's name (see the UML tab)
- `#{value}` is replaced by the initial value of the *static* attribute in the definition part generated in the source file (see the UML tab), `#{value}` and `#{h_value}` are exclusive
- `#{h_value}` is replaced by the initial value of the *static* attribute in the declaration part generated in the header file (see the UML tab), `#{value}` and `#{h_value}` are exclusive
- `@{xyz}` is replaced by the *user property* value in case xyz is the key of this property. The property is searched first for the attribute, if not found for the class containing the attribute etc ...

### enumeration item



The macros known by the C++ code generator are :

- ***\${comment}*** is replaced by the description of the item adding //
- ***\${description}*** is replaced by the item description without adding //
- ***\${name}*** is replaced by the item's name (see the UML tab)
- ***\${value}*** is replaced by the value of the item (see the UML tab)
- ***@{xyz}*** is replaced by the *user property* value in case *xyz* is the key of this property. The property is searched first for the attribute, if not found for the class containing the attribute etc ...

## Relation

The C++ definition of a relation is set through the [Uml](#) and [C++](#) tabs of the [relation dialog](#).

The indentation of the first line of the definition gives the indentation added to the visibility specification in the class definition for all the relation definition (except for the pre-processor commands beginning by # whose are placed at the beginning of the line).

### Relation equivalent to an attribute

In BOUML the generated code is obtained by the substitution of macros in a text, the macros known by the C++ code generator are :

- ***\${comment}*** is replaced by the description of the relation adding //
- ***\${description}*** is replaced by the relation description without adding //
- ***\${static}*** produce static when the relation is a class relation (see the UML tab)
- ***\${const}*** produce const when the relation is read-only (see the UML tab)
- ***\${type}*** is replaced by the class pointed by the relation (see the UML tab)
- ***\${name}*** is replaced by the relation's role name (see the UML tab)
- ***\${value}*** is replaced by the initial value of the *static* relation in the definition part generated in the source file (see the UML tab), ***\${value}*** and ***\${h\_value}*** are exclusive
- ***\${h\_value}*** is replaced by the initial value of the *static* relation in the declaration part generated in the header file (see the UML tab), ***\${value}*** and ***\${h\_value}*** are exclusive
- ***\${mutable}*** produce *mutable* when the relation is mutable
- ***\${volatile}*** produce *volatile* when the relation is volatile
- ***\${stereotype}*** is replaced by the [translation in C++](#) of the relation's stereotype (see the UML tab)
- ***\${multiplicity}*** is replaced by the multiplicity of the relation (see the UML tab), must be used in case the multiplicity is a vector or dimensioning ([ and ] are added when they are not present).
- ***\${association}*** is replaced by the association class (see UML tab)
- ***@{xyz}*** is replaced by the *user property* value in case *xyz* is the key of this property. The property is searched first for the relation, if not found for the class containing the relation etc ...

## Dependency

A dependency having the stereotype *friend* produce a C++ friend declaration of the pointed class.

With an other stereotype the dependency may produce an *#include* of the artifact associated to the target class, and you can choose to place this *#include* in the header or source file by editing the dependency using the tab C++.

## Inheritance

The class inheritance are managed at the [class level](#).

- ***\${type}*** is replaced by the name of the inherited class more the *actuals* if the generated class instantiate a *template*.

---

## Extra member

The C++ definition of an extra member is set through the [C++](#) tabs of the [extra member dialog](#).

No macros.

---

Previous : [deployment diagram](#)

Next : [C++ reverse](#)

# C++ reverse

The *C++ code reverse* is a *plug-out* directly written in C++, as the C++ generator.

Note that the C++ *code reverse* cannot be used as a C++ code round trip and doesn't allow to update already defined classes having at least one member. Furthermore the preprocessing phase is not made by the *C++ code reverse* whose ignore all the preprocessor forms, but some substitutions may be asked, see at the end of this chapter.

The *C++ code reverse plug-out* may be applied only on packages.

When you start the *C++ code reverse*, it asks you for directories (enter the directories then choose *cancel* to start the reverse), then it reads all the sources placed under the selected directories and sub-directories. The sources read by the *C++ code reverse* are the ones having the extension specified in the [generation settings](#). The tree formed by the directories and sub-directories produce an equivalent tree of sub-[packages](#), on whose the needed [class view](#) and [artifact view](#) are made to support the [classes](#) and [artifacts](#). When a type is referenced but not defined in the reverse files, a class having the same name is defined under an additional [package](#) named *unknown*.

Obviously the *C++ code reverse* tries to produce the definitions made in BOUML at the [artifact](#) / [class](#) / [operation](#) / [relation](#) / [attribute](#) levels. When it is possible it creates a relation rather than an attribute.

Except for the special case of the templates class definition like *vector<bool>*, a [artifact](#) is created for each type definition, then a future C++ code generation may not produce a list of files similar to the reversed ones.

A C function cannot be specified in UML, the *C++ code reverse* loses them (generating a warning) except the ones declared *friend* in a class.

The C++ *code reverse* works in two steps, during the first step it scans all the files to establish the list of defined types, then the second step may reverse the files. The first phase try to replace the missing *#include* management, but it is not enough because the files whose are not under the selected directory (for instance the standard headers files !) are never taken into account and the synonym types are confusing.

When the C++ *code reverse* has a problem, for instance when it reads a C function definition, it produces a warning in the trace window. The trace window automatically opened by BOUML, it may also be manually opened through the [Tools menu](#).

To do simple substitution on the read files : edit the *package* where the C++ reverse will be started (perhaps the project itself) to add a *user property* (through the tab 'User') with the key *#file* and the value must be the absolute pathname of a file. Each line of the specified file must contain :

- just an identifier : in this case this identifier will be ignored during reverse. This is like *#define identifier*
- a form *identifier=value* : in this case the identifier will be replaced by the value during reverse. The value must be a token, for instance an other identifier float, a string etc. This is like *#define identifier value*.

Warning : the lines must not contain spaces or tabs.

---

Previous : [C++ generator](#)

Next : [Java generator](#)

# Java generator

The Java code generator is a *plug-out* directly written in C++.

The generated sources follows the definition made in BOUML at the [artifact](#) / [class](#) / [operation](#) / [relation](#) / [attribute](#) / [extra member](#) levels.

When the code generation is applied on a artifact associated to several classes, the code generation is made for all these classes. Nevertheless the the Java code generator produce first the code in memory and update the appropriate files only when it is necessary to not change the last write date of the files for nothing. Depending on the toggle *verbose code generation* of the global menu *Languages* the code generator is verbose or not.

The Java code generator *plug-out* may be called on :

- a [class](#) : in this case the code generation is in fact applied on the class's [artifact](#), then on all the classes associated to this [artifact](#)
- a [artifact](#) : in this case the code generation is in fact applied on all the [classes](#) associated to the [artifact](#)
- a [class view](#) : the code generation will be applied on all the sub [classes](#), then on all the [artifacts](#) associated to these [classes](#)
- a [deployment view](#) : the code generation is applied on all the sub [artifacts](#)
- a [package](#) (may be the project itself) : the code generation will be applied on all the sub [class views](#) and [deployment views](#), then on all their sub [classes](#) and [artifacts](#).

When the Java code generator is ask through the [Tools](#) menu, it is applied on the project, then on all the [artifacts](#).

The name of the generated files depend on the artifact name, the extension depend on the language and is fixed for each by the [generations settings](#) (see below), the directory where the files are generated may be set in each package containing directly or indirectly the artifact (see [generation directory](#)).

---

## artifact

The Java definition of a artifact is set through the [Java source](#) tabs of the [artifact dialog](#).

The code generation depend on the stereotype of the artifact :

- *text* : the Java definition of the artifact is produced without changes, the name of the generated file is the name of the artifact, including the extension.
- *source* : see below.
- else nothing is generated for the artifact.

The generated file name is the artifact's name with the extension specified in the first Java tab of the [generations settings](#) :



In BOUML the generated code is obtained by the substitution of macros in a text, the macros known by the Java code generator are :

- ***\${comment}*** is replaced by the artifact description adding */\* \*/*
- ***\${description}*** is replaced by the artifact description without adding */\* \*/*
- ***\${package}*** is replaced by the *package xx* forms, dependent on the [package](#) specifications associated to the BOUML package containing the [deployment view](#) where the artifact is defined.
- ***\${import}*** is replaced by import forms specified by dependencies stereotypes *import* from classes produced by the artifact to classes or packages.
- ***\${definition}*** is replaced by the definition of the classes associated to the artifact.
- ***@{xyz}*** is replaced by the *user property* value in case *xyz* is the key of this property. The property is searched first for the *artifact*, if not found for the *deployment view* containing the *artifact*, if not found in the *package* containing the *deployment view* etc ...

## Class

The Java definition of a class is set through the [Uml](#), [Parametrized](#), [Instantiate](#) and [Java](#) tabs of the [class dialog](#).

A Java type definition may be a class, an interface or an enum (may be defined through a class), depending on the stereotype translation in Java (see [generation settings](#)).

In BOUML the generated code is obtained by the substitution of macros in a text, the macros known by the Java code generator are :

- ***\${comment}*** is replaced by the description of the class adding */\* \*/*
- ***\${description}*** is replaced by the class description without adding */\* \*/*
- ***\${public}*** produce *public* when the class is declared public
- ***\${final}*** produce *final* when the class is declared final
- ***\${abstract}*** produce *abstract* when the class is abstract
- ***\${name}*** is replaced by the class's name
- ***\${extend}*** is replaced by the class inheritance

- ***\${implement}*** is replaced by the interface inheritance
- ***\${members}*** is replaced by the code generated for all the class's members ([relations](#), [attributes](#), [operations](#) and [extra members](#)) following the browser order.
- ***\${cases}*** only for the enums, is replaced by the enum's items definition
- ***@{xyz}*** is replaced by the *user property* value in case *xyz* is the key of this property. The property is first searched for the class, if not found for the container of the class (an other class of a *class view*) etc ...

In the special case where the class is declared ***external***, its Java type declaration must contain a line indicating how the name of the class is generated, by default ***\${name}*** meaning that the name is produced unchanged, the only allowed keywords are ***\${name}***, ***\${Name}*** and ***\${NAME}***.

## Operation

The Java definition of an operation is set through the [Uml](#) and [Java](#) tabs of the [operation dialog](#).

The indentation of the first line of the declaration/definition gives the indentation added to the class definition for all the operation definitions.

In BOUML the generated code is obtained by the substitution of macros in a text, the macros known by the Java code generator are :

- ***\${comment}*** is replaced by the description of the operation adding */\* \*/*
- ***\${description}*** is replaced by the operation description without adding */\* \*/*
- ***\${visibility}*** produce the visibility (see the UML tab)
- ***\${final}*** produce *final* when the operation is specified final
- ***\${static}*** produce *static* when the operation is a class operation (see the UML tab)
- ***\${abstract}*** produce *abstract* when the operation is abstract (see the UML tab)
- ***\${synchronized}*** produce *synchronized* when the operation is synchronized
- ***\${type}*** is replaced by the value type (see the UML tab)
- ***\${name}*** is replaced by the name of the operation.
- ***\${(}*** and ***\${)}*** produce ( and ), but there are also a mark for BOUML to find the parameters list
- ***\${t<n>}*** and ***\${p<n>}***  
produce the type and the name of each parameter (count from 0), this allows you to remove a parameter, etc ...
- ***\${throws}*** is replaced by the form *throws ...* when at least an exception is defined in the UML tab.
- ***\${staticnl}***  
produce a line break when the operation is static, else an empty string. In case you do not like this notation, change the [generation settings](#) to remove this macro.
- ***\${body}***  
is replaced by the body of the operation, this macro may also be replaced by the body itself (BOUML uses this way for the *get* and *set* operations associated to a relation/attribute). The usage of ***\${body}*** has a great advantage : when you hit ***Default definition*** the body is not cleared ! At least BOUML shares the definition forms of the operations (and other objects) to minimize the needed memory size, ***\${body}*** helps for this ! The indentation of the keyword ***\${body}*** is added at the beginning of each line.
- ***@{xyz}*** is replaced by the *user property* value in case *xyz* is the key of this property. The property is first searched for the operation, if not found for the class containing the operation etc ... If the toggle *preserve operations's body* is set through the [Languages](#) menu, the generators do not modify the body of the operations protected by dedicated delimiters. This means that for them the body definition set through BOUML is not used. The first time you generate the code with the toggle set, because the delimiters are not yet present in the generated code, the operation's body will be updated depending on their definition in BOUML. After, while the toggle is set and the delimiters present, the bodies will not change, allowing you to modify them out of BOUML.

Notes :

- In case the file containing a body definition is not consistent with the artifact under BOUML, the body will be regenerated by the code generation, using its definition under the model.
- When you import a project, the body of the imported operations must be the right one in the imported model. The preserved bodies of the imported operations will not be found because the identifier of an operation used to mark its body changes during the import.
- The bodies under the model are not updated by the code generation, use [roundtrip body](#) for that
- Only the operations using the keyword `#{body}` may have a preserved body.
- The only modification you can do in the lines containing the delimiters is the indent.
- The toggle is saved in the file associated to the project, be sure the save is done when you change this toggle !

## Attribute

The Java definition of an attribute is set through the [Uml](#) and [Java](#) tabs of the [attribute dialog](#).

The indentation of the first line of the definition gives the indentation added to the class definition for all the attribute definition.

An attribute may be a standard attribute or the item of an enumeration defined through a class.

### standard attribute

In BOUML the generated code is obtained by the substitution of macros in a text, the macros known by the Java code generator are :

- `#{comment}` is replaced by the description of the attribute adding `/* */`
- `#{description}` is replaced by the attribute description without adding `/* */`
- `#{visibility}` produce the visibility (see the UML tab)
- `#{static}` produce *static* when the attribute is a class attribute (see the UML tab)
- `#{final}` produce *final* when the attribute is final
- `#{transient}` produce *transient* when the attribute is transient
- `#{volatile}` produce *volatile* when the operation is volatile
- `#{type}` is replaced by the type of the attribute (see the UML tab)
- `#{stereotype}` is replaced by the [translation in Java](#) of the relation's stereotype (see the UML tab)
- `#{multiplicity}`  
is replaced by the multiplicity of the relation (see the UML tab), must be used in case the multiplicity is a vector or dimensioning ([ and ] are added when they are not present).
- `#{name}` is replaced by the attribute's name (see the UML tab)
- `#{value}` is replaced by the initial value of the attribute (see the UML tab)
- `@{xyz}` is replaced by the *user property* value in case *xyz* is the key of this property. The property is first searched for the attribute, if not found for the class containing the attribute etc ...

### enumeration item

The macros known by the Java code generator are :

- `#{comment}` is replaced by the description of the item adding `/* */`
- `#{description}` is replaced by the item description without adding `/* */`
- `#{name}` is replaced by the item's name (see the UML tab)
- `#{class}` is replaced by the name of the class supporting the enumeration
- `@{xyz}` is replaced by the *user property* value in case *xyz* is the key of this property. The property is first searched for the attribute, if not found for the class containing the attribute etc ...

## Relation

The Java definition of a relation is set through the [Uml](#) and [Java](#) tabs of the [relation dialog](#).

The indentation of the first line of the definition give the indentation added to the class definition for all the relation definition.

### Relation equivalent to an attribute

In BOUML the generated code is obtained by the substitution of macros in a text, the macros known by the Java code generator are :

- ***\${comment}*** is replaced by the description of the attribute adding */\* \*/*
- ***\${description}*** is replaced by the attribute description without adding */\* \*/*
- ***\${visibility}*** produce the visibility (see the UML tab)
- ***\${static}*** produce *static* when the attribute is a class attribute (see the UML tab)
- ***\${final}*** produce *final* when the attribute is final
- ***\${transient}*** produce *transient* when the attribute is transient
- ***\${volatile}*** produce *volatile* when the operation is volatile
- ***\${type}*** is replaced by the name of the class pointed by the relation (see the UML tab)
- ***\${name}*** is replaced by the role's name (see the UML tab)
- ***\${value}*** is replaced by the initial value of the relation (see the UML tab)
- ***\${stereotype}*** is replaced by the [translation in Java](#) of the relation's stereotype (see the UML tab)
- ***\${multiplicity}***  
is replaced by the multiplicity of the relation (see the UML tab), must be used in case the multiplicity is a vector or dimensioning ([ and ] are added when they are not present).
- ***\${association}*** is replaced by the association class (see UML tab)
- ***@{xyz}*** is replaced by the *user property* value in case *xyz* is the key of this property. The property is searched first for the relation, if not found for the class containing the relation etc ...

### Inheritance

The class inheritance are managed at the [class level](#).

- ***\${type}*** is replaced by the name of the inherited class more the *actuals* if the generated class instantiate a generic.

## Extra member

The Java definition of an extra member is set through the [Java](#) tabs of the [extra member dialog](#).

No macros.

Previous : [C++ reverse](#)

Next : [Java reverse](#)



# Java reverse

The *Java code reverse* is a *plug-out* directly written in C++.

Note that the *Java code reverse* cannot be used as a Java code round trip and doesn't allow to update already defined classes having at least one member.

The *Java code reverse plug-out* may be applied only on a package.

When you start the *Java code reverse*, it asks you for [java catalog](#) files, use *cancel* to finish to give the catalogs (may be immediately), then it asks for a directory, then it reads all the sources placed under the selected directory and sub-directories. The sources read by the *Java code reverse* are the ones having the extension specified in the [generation settings](#). The tree formed by the sub-directories produce an equivalent tree of sub-[packages](#), on whose the needed [class view](#) and [deployment view](#) are made to support the [classes](#) and [artifacts](#). When a type is referenced but not defined in the reverse files, a class having the same name is defined under an additional [package](#) named *unknown*.

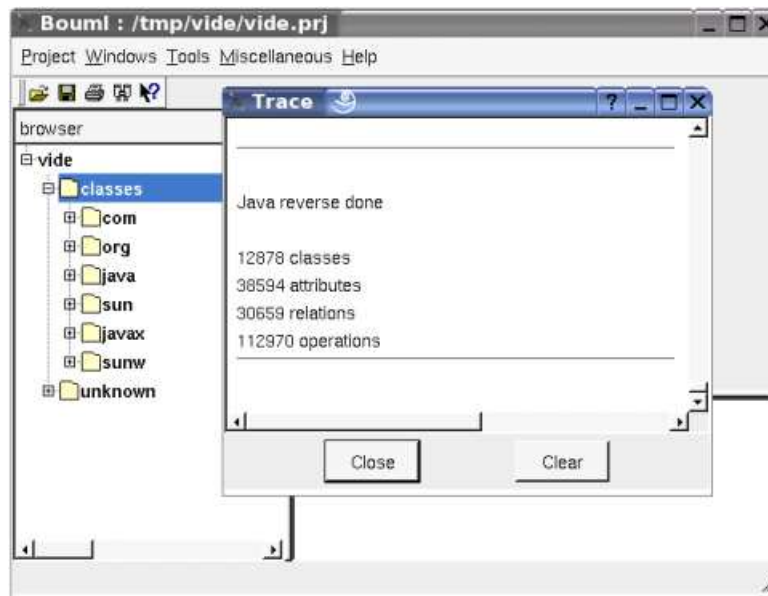
Obviously the *Java code reverse* tries to produce the definitions made in BOUML at the [artifact](#) / [class](#) / [operation](#) / [relation](#) / [attribute](#) levels. When it is possible it creates a relation rather than an attribute.

A [artifact](#) is created for each type definition, then a future Java code generation may not produce a list of files similar to the reversed ones.

The *Java code reverse* works in two steps, during the first step it scans all the files to establish the list of defined types, then the second step may reverse the files.

When the *Java code reverse* has a problem, for instance on a syntax error, it produces a warning in the trace window. The trace window is automatically opened by BOUML, it may also be manually opened through the [Tools menu](#).

When the reverse is done, the number of reversed classes, operations, attributes and relations is written, for instance in case I reverse a full JDK Java distribution :



See also [Java catalog](#).

Previous : [Java generator](#)

Next : [Java catalog](#)

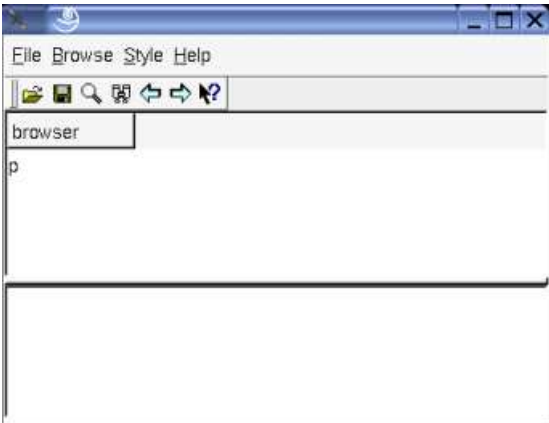
# Java catalog

Java Catalog is a *plug-out* directly written in C++.

Java is a very practical tool allowing you to import classes into BOUML. An imported class comes in BOUML without the body operations and without its *private* members : the goal is to import Java library classes in the model, not to generate them. Java catalog may also be used to browse through the Java packages, search information about a given class etc ... without sending them to BOUML.

To work, Java catalog need Java sources, this must not be a problem, these ones are generally available even for your preferred JDK version.

The Java catalog *plug-out* may be applied only on a package, it is used through a window :



The upper part is a browser like in BOUML except that only the package and classes are showed. The lower part (reduced when Java catalog starts) shows the description of the selected class interpreting the *html* forms. The top level package in the browser part (here *p*) is the name of the package in BOUML in which Java catalog is applied.

To make a catalog, use the *scan* entry of the *file* menu or the button representing a microscope, Java catalog will ask for a directory and scan all the Java files in this one and its sub-directories. For instance if *src* contains all the Java sources :



Obviously it is possible to do several scan phase in different directories to complete the catalog. At this level nothing is done on BOUML side.

To save your catalog use the *save* entry of the *file* menu or the associated button.

*Open* allows to get an already made catalog, may be to complete it.

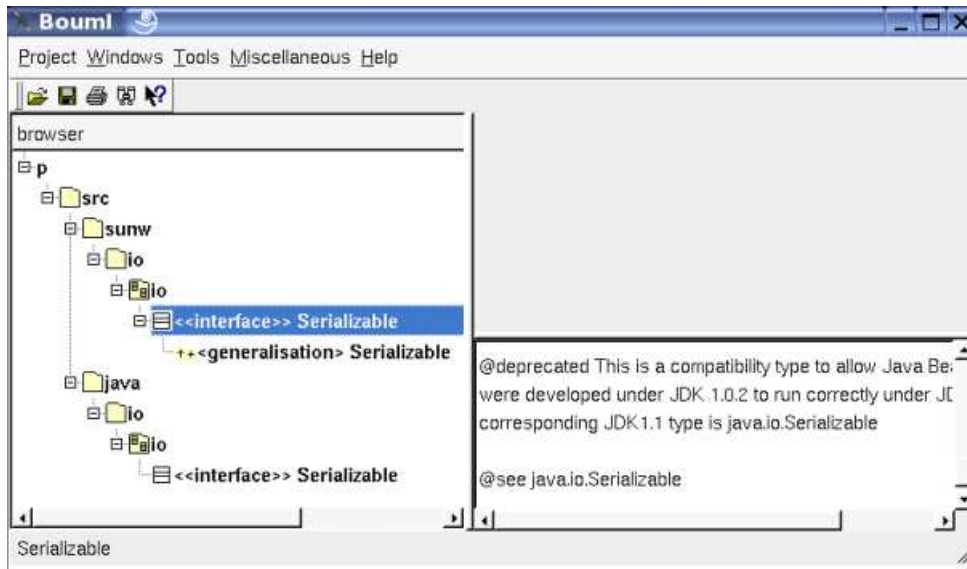
The main goal of Java catalog

is to send a class or all the classes under a package and its sub-packages to BOUML, using the menu appearing with a right mouse button click on a class or a package into the *Java catalog* browser :



In case the sent class(es) reference other classes (for instance to inherit them), these classes are also sent with the classes they reference etc ...

For instance, in case the BOUML project *p* is empty and the class *sunw.io.Serializable* is sent, the result will be :



As you can see, contrarily to the result of a standard Java reverse, the two classes do not have associated artifact, because the goal is just to allow to have a class generalizing *Serializable*, or having an attribute of this type etc ...

Thanks to *Java catalog*, you do not have to define the JDK and other classes by hand in BOUML to use them !

Note : the saving of a catalog is just a list of classes and the pathname of the file defining them, not the classes definitions. When you send a class to BOUML, *Java catalog* reverse the associated file to get the class definition, this implies that you must not delete or move the Java source files defining classes of the catalog.

Previous : [Java reverse](#)

Next : [Php generator](#)

# PHP generator

The PHP code generator is a *plug-out* directly written in C++.

The generated sources follows the definition made in BOUML at the [artifact](#) / [class](#) / [operation](#) / [relation](#) / [attribute](#) / [extra member](#) levels.

When the code generation is applied on a artifact associated to several classes, the code generation is made for all these classes. Nevertheless the the Php code generator produce first the code in memory and update the appropriate files only when it is necessary, to not change the last write date of the files for nothing. Depending on the toggle *verbose code generation* of the global menu *Languages* the code generator is verbose or not.

The Php code generator *plug-out* may be called on :

- a [class](#) : in this case the code generation is in fact applied on the class's [artifact](#), then on all the classes associated to this [artifact](#)
- a [artifact](#) : in this case the code generation is in fact applied on all the [classes](#) associated to the [artifact](#)
- a [class view](#) : the code generation will be applied on all the sub [classes](#), then on all the [artifacts](#) associated to these [classes](#)
- a [deployment view](#) : the code generation is applied on all the sub [artifacts](#)
- a [package](#) (may be the project itself) : the code generation will be applied on all the sub [class views](#) and [deployment views](#), then on all their sub [classes](#) and [artifacts](#).

When the Php code generator is ask through the [Tools](#) menu, it is applied on the project, then on all the [artifacts](#).

The name of the generated files depend on the artifact name, the extension depend on the language and is fixed for each by the [generations settings](#) (see below), the directory where the files are generated may be set in each package containing directly or indirectly the artifact (see [generation directory](#)).

---

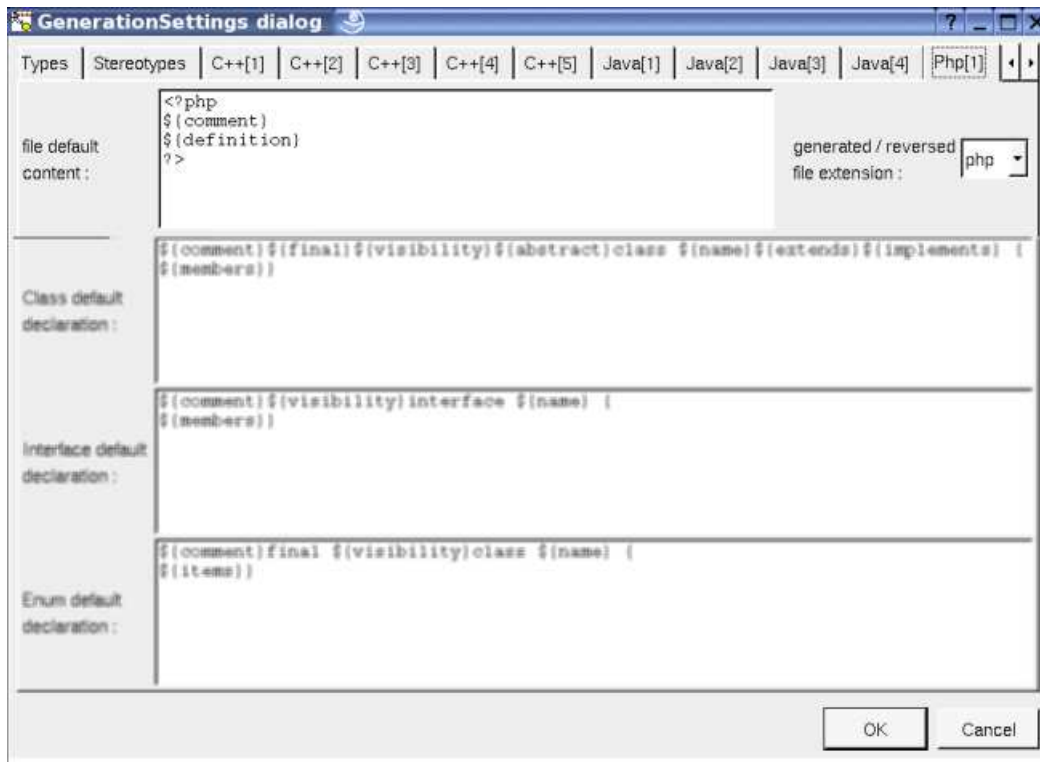
## artifact

The Php definition of a artifact is set through the [Php source](#) tabs of the [artifact dialog](#).

The code generation depend on the stereotype of the artifact :

- *text* : the Php definition of the artifact is produced without changes, the name of the generated file is the name of the artifact, including the extension.
- *source* : see below.
- else nothing is generated for the artifact.

The generated file name is the artifact's name with the extension specified in the first Php tab of the [generations settings](#) :



In BOUML the generated code is obtained by the substitution of macros in a text, the macros known by the Php code generator are :

- ***\${comment}*** is replaced by the artifact description adding */\* \*/*
- ***\${description}*** is replaced by the artifact description without adding */\* \*/*
- ***\${definition}*** is replaced by the definition of the classes associated to the artifact.
- ***@{xyz}*** is replaced by the *user property* value in case *xyz* is the key of this property. The property is searched first for the *artifact*, if not found for the *deployment view* containing the *artifact*, if not found in the *package* containing the *deployment view* etc ...

## Class

The Php definition of a class is set through the [Uml](#), [Parametrized](#), [Instantiate](#) and [Php](#) tabs of the [class dialog](#).

A Php type definition may be a class, an interface or an enum defined through a class, depending on the stereotype and its translator in Php (see [generation settings](#)).

In BOUML the generated code is obtained by the substitution of macros in a text, the macros known by the Php code generator are :

- ***\${comment}*** is replaced by the description of the class adding */\* \*/*
- ***\${description}*** is replaced by the class description without adding */\* \*/*
- ***\${public}*** produce *public* when the class is declared public
- ***\${final}*** produce *final* when the class is declared final
- ***\${abstract}*** produce *abstract* when the class is abstract
- ***\${name}*** is replaced by the class's name
- ***\${extend}*** is replaced by the class inheritance
- ***\${implement}*** is replaced by the interface inheritance
- ***\${members}*** is replaced by the code generated for all the class's members ([relations](#), [attributes](#), [operations](#) and [extra members](#)) following the browser order.
- ***\${items}*** produces the enumeration items of an *enum*. other members are not produced
- ***@{xyz}*** is replaced by the *user property* value in case *xyz* is the key of this property. The property is searched first for the class,

if not found for the container of the class (an other class of a *class view*) etc ...

In the special case where the class is declared **external**, its Php type declaration must contains a line indicating how the name of the class is generated, by default `#{name}` meaning that the name is produced unchanged, the only allowed keywords are `#{name}`, `#{Name}` and `#{NAME}`.

## Operation

The Php definition of an operation is set through the [Uml](#) and [Php](#) tabs of the [operation dialog](#).

The indentation of the first line of the declaration/definition give the indentation added to the class definition for all the operation definition.

In BOUML the generated code is obtained by the substitution of macros in a text, the macros known by the Php code generator are :

- `#{comment}` is replaced by the description of the operation adding `/* */`
- `#{description}` is replaced by the operation description without adding `/* */`
- `#{visibility}` produce the visibility (see the UML tab), except if the visibility is *package* (use this case to produce Php4 code)
- `#{static}` produce *static* when the operation is a class operation (see the UML tab)
- `#{final}` produce an empty string when the check box *final* is not checked, else produce *final*
- `#{abstract}` produce *abstract* when the operation is abstract (see the UML tab)
- `#{name}` is replaced by the name of the operation.
- `#{()` and `#{}` produce ( and ), but there are also a mark for BOUML to find the parameters list
- `#{t<n>}` and `#{p<n>}`  
produce the type and the name of each parameter (count from 0), this allows you to remove a parameter, etc ...
- `#{staticnl}`  
produce a line break when the operation is static, else an empty string. In case you do not like this notation, change the [generation settings](#) to remove this macro.
- `#{body}`  
is replaced by the body of the operation, this macro may also be replaced by the body itself (BOUML use this way for the *get* and *set* operations associated to a relation/attribute). The usage of `#{body}` has a great advantage : when you hit **Default definition** the body is not cleared ! At least BOUML share the definition forms of the operations (and other objects) to minimize the needed memory size, `#{body}` help for this ! The indentation of the keyword `#{body}` is added at the beginning of each line.
- `@{xyz}` is replaced by the *user property* value in case *xyz* is the key of this property. The property is searched first for the operation, if not found for the class containing the operation etc ... If the toggle *preserve operations's body* is set through the *Languages* menu, the generators do not modify the body of the operations protected by dedicated delimiters. This means that for them the body definition set through BOUML is not used. The first time you generate the code with the toggle set, because the delimiters are not yet present in the generated code, the operation's body will be updated depending on their definition BOUML. After, while the toggle is set and the delimiters present, the bodies will not change, allowing you to modify them out of BOUML.

Notes :

- In case the file containing a body definition is not consistent with the artifact under BOUML, the body will be regenerated by the code generation, using its definition under the model.
- When you import a project, the body of the imported operations must be the right one in the imported model. The preserved bodies of the imported operations will not be found because the identifier of an operation used to mark its body changes during the import.
- The bodies under the model are not updated by the code generation, use [roundtrip body](#) for that
- Only the operations using the keyword `#{body}` may have a preserved body.
- The only modification you can do in the lines containing the delimiters is the indent.
- The toggle is saved in the file associated to the project, be sure the save is done when you change this toggle !

## Attribute

The Php definition of an attribute is set through the [Uml](#) and [Php](#) tabs of the [attribute dialog](#).

The indentation of the first line of the definition give the indentation added to the class definition for all the attribute definition.

An attribute may be a standard attribute or the item of an enumeration defined through a class.

### standard attribute

In BOUML the generated code is obtained by the substitution of macros in a text, the macros known by the Php code generator are :

- **`${comment}`** is replaced by the description of the attribute adding `/* */`
- **`${description}`** is replaced by the attribute description without adding `/* */`
- **`${visibility}`** produce the visibility (see the UML tab), except if the visibility is *package* (use this case to produce Php4 code)
- **`${static}`** produce *static* when the attribute is a class attribute (see the UML tab)
- **`${const}`** produce *const* when the attribute is read only (see the UML tab)
- **`${var}`** produce *var* when the relation is not read only nor static(see the UML tab) and the visibility is not *package*, else produce an empty string
- **`${name}`** is replaced by the attribute's name (see the UML tab)
- **`${value}`** is replaced by the initial value of the attribute (see the UML tab)
- **`@{xyz}`** is replaced by the *user property* value in case *xyz* is the key of this property. The property is searched first for the attribute, if not found for the class containing the attribute etc ...

## Relation

The Php definition of a relation is set through the [Uml](#) and [Php](#) tabs of the [relation dialog](#).

The indentation of the first line of the definition give the indentation added to the class definition for all the relation definition.

### Relation equivalent to an attribute

In BOUML the generated code is obtained by the substitution of macros in a text, the macros known by the Php code generator are :

- **`${comment}`** is replaced by the description of the attribute adding `/* */`
- **`${description}`** is replaced by the attribute description without adding `/* */`
- **`${visibility}`** produce the visibility (see the UML tab), except if the visibility is *package* (use this case to produce Php4 code)
- **`${static}`** produce *static* when the attribute is a class attribute (see the UML tab)
- **`${const}`** produce *const* when the attribute is read only (see the UML tab)
- **`${var}`** produce *var* when the relation is not read only nor static(see the UML tab) and the visibility is not *package*, else produce an empty string
- **`${name}`** is replaced by the role's name (see the UML tab)
- **`${value}`** is replaced by the initial value of the relation (see the UML tab)
- **`${stereotype}`** is replaced by the [translation in Php](#) of the relation's stereotype (see the UML tab)
- **`@{xyz}`** is replaced by the *user property* value in case *xyz* is the key of this property. The property is searched first for the relation, if not found for the class containing the relation etc ...

## Inheritance

The class inheritance are managed at the [class level](#).

- *$\{type\}$*  is replaced by the name of the inherited class.
- 

## Extra member

The Php definition of an extra member is set through the [Php](#) tabs of the [extra member dialog](#).

No macros.

---

Previous : [Java catalog](#)

Next : [Php reverse](#)



# Php reverse

The *Php code reverse* is a *plug-out* directly written in C++.

Note that the *Php code reverse* cannot be used as a Php code round trip and doesn't allows to update already defined classes having at least one member.

The *Php code reverse plug-out* may be applied only on a package.

When you start the *Php code reverse*, it ask for a directory, then it reads all the sources placed under the selected directory and sub-directories. The sources read by the *Php code reverse* are the ones having the extension specified in the [generation settings](#). The tree formed by the sub-directories produce an equivalent tree of sub-[packages](#), on whose the needed [class view](#) and [deployment view](#) are made to support the [classes](#) and [artifacts](#). When a type is referenced but not defined in the reverse files, a class having the same name is defined under an additional [package](#) named *unknown*.

Obviously the *Php code reverse* tries to produce the definitions made in BOUML at the [artifact](#) / [class](#) / [operation](#) / [relation](#) / [attribute](#) levels. When it is possible it creates a relation rather than an attribute.

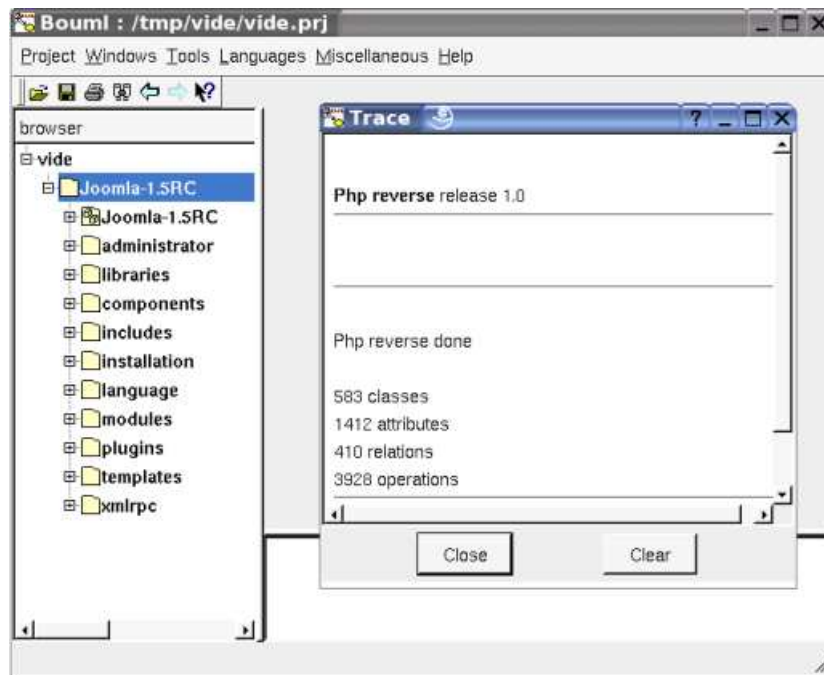
The *Php code reverse* works in two steps, during the first step it scans all the files to establish the list of defined types, then the second step may reverse the files.

When the *Php code reverse* has a problem, for instance on a syntax error, it produces a warning in the trace window. The trace window is automatically opened by BOUML, it may also be manually opened through the [Tools menu](#).

The reversed files are considered composed in three parts :

- a header without class definition and containing "<?php". This part is placed at the beginning of the generated artifact
- only classes / interfaces definition, this part start with the first class or interface and stops before the first form which is not a class nor an interface. The classes and interfaces must be at the toplevel level, this means not embedded in a *if* or other form. This part produces *\${definition}* in the definition of the artifact
- the rest of the file, probably containing the marker ">", this part is placed at the end of the definition of the artifact.

When the reverse is done, the number of reversed classes, operations, attributes and relations is written, for instance in case I reverse the sources of [Joomla 1.5RC](#) :



Previous : [Php generator](#)

Next : [Python Generator](#)

# Python generator

The Python code generator is a *plug-out* directly written in C++.

The generated sources follows the definition made in BOUML at the [artifact](#) / [class](#) / [operation](#) / [relation](#) / [attribute](#) / [extra member](#) levels.

When the code generation is applied on a artifact associated to several classes, the code generation is made for all these classes. Nevertheless the Python code generator produce first the code in memory and update the appropriate files only when it is necessary, to not change the last write date of the files for nothing. Depending on the toggle *verbose code generation* of the global menu *Languages* the code generator is verbose or not.

The Python code generator *plug-out* may be called on :

- a [class](#) : in this case the code generation is in fact applied on the class's [artifact](#), then on all the classes associated to this [artifact](#)
- a [artifact](#) : in this case the code generation is in fact applied on all the [classes](#) associated to the [artifact](#)
- a [class view](#) : the code generation will be applied on all the sub [classes](#), then on all the [artifacts](#) associated to these [classes](#)
- a [deployment view](#) : the code generation is applied on all the sub [artifacts](#)
- a [package](#) (may be the project itself) : the code generation will be applied on all the sub [class views](#) and [deployment views](#), then on all their sub [classes](#) and [artifacts](#).

When the Python code generator is asked through the [Tools](#) menu, it is applied on the project, then on all the [artifacts](#).

The name of the generated files depend on the artifact name, the extension depend on the language and is fixed for each by the [generations settings](#) (see below), the directory where the files are generated may be set in each package containing directly or indirectly the artifact (see [generation directory](#)).

The generator change automatically the indentation, adding/removing when necessary the indentation step specified through the [generations settings](#) (see dialog screen below) At least the first line of an operation's body must not be indented when you enter it through the operation dialog (except if you ask for non contextual indent), but of course you have to manage yourself the indentation change due to statement for instance.

## artifact

The Python definition of a artifact is set through the [Python source](#) tabs of the [artifact dialog](#).

The code generation depend on the stereotype of the artifact :

- *text* : the Python definition of the artifact is produced without changes, the name of the generated file is the name of the artifact, including the extension.
- *source* : see below.
- else nothing is generated for the artifact.

The generated file name is the artifact's name with the extension specified in the first Python tab of the [generations settings](#) :



In BOUML the generated code is obtained by the substitution of macros in a text, the macros known by the Python code generator are :

- ***`\${comment}`***  
is replaced by the artifact description adding `##` at the beginning of each line and a newline at end, or nothing if the description is empty
- ***`\${description}`*** is replaced by the artifact description without adding `##`
- ***`\${imports}`*** is replaced by `import` and `from ... import` forms from the dependencies between classes and between *artifacts* stereotyped *import* or *from*. A dependency stereotyped *import* between artifact produces an *import*, stereotyped *from* this produces a *from ... import \**. A dependency stereotyped *import* between classes produces an *import*, stereotyped *from* this produces a *from ... import*. Or course the packages are taken into account. You can also directly add *import* forms by hand in the artifact definition, they will be taken into account to not produce useless package path when classe names are generated.
- ***`\${definition}`*** is replaced by the definition of the classes associated to the artifact.
- ***@{xyz}*** is replaced by the *user property* value in case *xyz* is the key of this property. The property is searched first for the *artifact*, if not found for the *deployment view* containing the *artifact*, if not found in the *package* containing the *deployment view* etc ...

## Class

Bouml allows to define class for any releases of Python, to help for Python 2.2 classes generation a toggle *Python 2.2* is defined in each class, its default value is set through the [generation settings](#). When *Python 2.2* is set and a class doesn't inherit in the model, the inheritance or *object* is automatically added.

The Python definition of a class is set through the [Uml](#) and [Python](#) tabs of the [class dialog](#).

A Python type definition may be a class, or an enum defined through a class, depending on the stereotype and its translation in Python (see [generation settings](#)).

In BOUML the generated code is obtained by the substitution of macros in a text, the macros known by the Python code generator are :

- ***`\${docstring}`***  
is replaced by the description of the class placed between `"""` and followed by a newline. If the description is empty no produced. By default (set through the [generation settings](#) *`\${docstring}`* is used rather than *`\${comment}`* or *`\${description}`*.
- ***`\${comment}`*** is replaced by the description of the class adding `#`

- ***#{description}*** is replaced by the class description without adding #
- ***#{name}*** is replaced by the class's name. If needed the package path is produced, the code generator take into account the *imports* associated to the dependencies and the ones writtent by hand directly in the *artifact* definition.
- ***#{inherit}*** is replaced by the class inheritance
- ***#{members}*** is replaced by the code generated for all the class's members ([relations](#), [attributes](#), [operations](#), [extra members](#) and nested classes) following the browser order.
- ***@{xyz}*** is replaced by the *user property* value in case *xyz* is the key of this property. The property is searched first for the class, if not found for the container of the class (an other class of a *class view*) etc ...

In the special case where the class is declared ***external***, its Python type declaration must contains a line indicating how the name of the class is generated, and followed by an optional line giving the needed *import* forms. By default the specification of the name is ***#{name}*** meaning that the name is produced unchanged, the only allowed keywords are ***#{name}***, ***#{Name}*** and ***#{NAME}***.

## Operation

The Python definition of an operation is set through the [Uml](#) and [Python](#) tabs of the [operation dialog](#).

In BOUML the generated code is obtained by the substitution of macros in a text, the macros known by the Python code generator are :

- ***#{@}*** is replaced by the decorators, don't use decorator to produce *@staticmethod* and *@abstractmethod*, see ***#{static}*** and ***#{abstract}*** below
- ***#{comment}***  
is replaced by the description of the operation adding # at the beginning of each line and forcing a newline at the end. Produce nothing when the description is empty.
- ***#{description}*** is replaced by the operation description without adding #. Produce nothing when the description is empty.
- ***#{static}*** is replaced by *@staticmethod* followed by a newline if the method is declared *static* in the UML tab, else an empty string
- ***#{abstract}*** is replaced by *@abstractmethod* followed by a newline if the method is declared *abstract* in the UML tab, else an empty string
- ***#{name}*** is replaced by the name of the operation.
- ***#{()*** and ***#{)}*** produce ( and ), but there are also a mark for BOUML to find the parameters list
- ***#{t<n>}*** and ***#{p<n>}***  
produce the type and the name of each parameter (count from 0), this allows you to remove a parameter, etc ...
- ***#{body}***  
is replaced by the body of the operation, this macro may also be replaced by the body itself (BOUML use this way for the *get* and *set* operations associated to a relation/attribute). The usage of ***#{body}*** has a great advantage : when you hit ***Default definition*** the body is not cleared ! At least BOUML share the definition forms of the operations (and other objects) to minimize the needed memory size, ***#{body}*** help for this ! The indentation of the keyword ***#{body}*** is added at the beginning of each line. In the special case of the *\_\_init\_\_* operation the initialization of the instance attributes and relations is added just before the body, they are not produced if ***#{body}*** is not part of the operation definition.
- ***@{xyz}*** is replaced by the *user property* value in case *xyz* is the key of this property. The property is searched first for the operation, if not found for the class containing the operation etc ...If the toggle *preserve operations's body* is set through the *Languages* menu, the generators do not modify the body of the operations protected by dedicated delimiters. This means that for them the body definition set through BOUML is not used. The first time you generate the code with the toggle set, because the delimiters are not yet present in the generated code, the operation's body will be updated depending on their definition BOUML. After, while the toggle is set and the delimiters present, the bodies will not change, allowing you to modify them out of BOUML.

Notes :

- In case the file containing a body definition is not consistent with the artifact under BOUML, the body will be regenerated by the code generation, using its definition under the model.

- When you import a project, the body of the imported operations must be the right one in the imported model. The preserved bodies of the imported operations will not be found because the identifier of an operation used to mark its body changes during the import.
- The bodies under the model are not updated by the code generation, use [roundtrip body](#) for that
- Only the operations using the keyword `#{body}` may have a preserved body.
- The only modification you can do in the lines containing the delimiters is the indent.
- The toggle is saved in the file associated to the project, be sure the save is done when you change this toggle !

## Attribute

The Python definition of an attribute is set through the [Uml](#) and [Python](#) tabs of the [attribute dialog](#).

An attribute may be a standard attribute or the item of an enumeration defined through a class.

The static attributes are produced in order directly in the class definition. The instance attributes are produced at the beginning of the operation `__init__` when its definition contains `#{body}`. The operation `__init__` is automatically added by the code generation, even if they are no instance attribute or relation.

### Attribute corresponding to variable

In BOUML the generated code is obtained by the substitution of macros in a text, the macros known by the Python code generator are :

- **`#{comment}`**  
is replaced by the description of the attribute adding # at the beginning of each line and forcing a newline at the end. Produce nothing when the description is empty.
- **`#{description}`** is replaced by the attribute description without adding #. Produce nothing when the description is empty.
- **`#{self}`** inside the operation `__init__` this produces the name of the first parameter followed by a dot, else nothing.
- **`#{name}`** is replaced by the attribute's name (see the UML tab)
- **`#{multiplicity}`** is replaced by the multiplicity of the attribute, probably used inside a comment
- **`#{stereotype}`** is replaced by the [translation in Python](#) of the attribute's stereotype (see the UML tab), probably something like *list*
- **`#{type}`** produces the type of the attribute, probably used inside a comment
- **`#{value}`** is replaced by the initial value of the attribute (see the UML tab)
- **`@{xyz}`** is replaced by the *user property* value in case *xyz* is the key of this property. The property is searched first for the attribute, if not found for the class containing the attribute etc ...

## Relation

The Python definition of a relation is set through the [Uml](#) and [Python](#) tabs of the [relation dialog](#).

The static relations corresponding to variables are produced in order directly in the class definition. The instance relations corresponding to variables are produced at the beginning of the operation `__init__` when its definition contains `#{body}`. The operation `__init__` is automatically added by the code generation, even if they are no instance attribute or relation.

### Relation corresponding to variable

In BOUML the generated code is obtained by the substitution of macros in a text, the macros known by the Python code generator are :

- **`#{comment}`**  
is replaced by the description of the relation adding # at the beginning of each line and forcing a newline at the end. Produce nothing when the description is empty.
- **`#{description}`** is replaced by the relation description without adding #. Produce nothing when the description is empty.

- ***`#{self}`*** inside the operation `__init__` this produces the name of the first parameter followed by a dot, else nothing.
- ***`#{name}`*** is replaced by the role's name (see the UML tab)
- ***`#{multiplicity}`*** is replaced by the multiplicity of the relation, probably used inside a comment
- ***`#{stereotype}`*** is replaced by the [translation in Python](#) of the relation's stereotype (see the UML tab), probably something like *list*
- ***`#{type}`*** is replaced by the type of the relation, it is used by default when the relation is a composition to create an instance of this type
- ***`#{value}`*** is replaced by the initial value of the attribute (see the UML tab)
- ***`@{xyz}`*** is replaced by the *user property* value in case *xyz* is the key of this property. The property is searched first for the attribute, if not found for the class containing the attribute etc ...

## Inheritance

The class inheritance are managed at the [class level](#).

- ***`#{type}`*** is replaced by the name of the inherited class.

## Dependencies

The dependencies allows to add *import* forms in the *artifact*, refer to [#{import}](#) description

## Extra member

The Python definition of an extra member is set through the [Python](#) tabs of the [extra member dialog](#).

The definition of the extra members are produced without any modification, this means you are also responsible to the indent

No macros.

---

Previous : [Php reverse](#)

Next : [Python reverse](#)

# Python reverse

The *Python code reverse* is a *plug-out* directly written in C++.

Note that the *Python code reverse* cannot be used as a Python code round trip and doesn't allow to update already defined classes having at least one member.

The *Python code reverse plug-out* may be applied only on a package.

When you start the *Python code reverse*, it asks for a directory, then it reads all the sources placed under the selected directory and sub-directories. The sources read by the *Python code reverse* are the ones having the extension specified in the [generation settings](#). The tree formed by the sub-directories produces an equivalent tree of sub-packages, on whose needed [class view](#) and [deployment view](#) are made to support the [classes](#) and [artifacts](#). When a type is referenced but not defined in the reverse files, a class having the same name is defined under an additional [package](#) named *unknown*.

Obviously the *Python code reverse* tries to produce the definitions made in BOUML at the [artifact](#) / [class](#) / [operation](#) / [relation](#) / [attribute](#) levels. When it is possible it creates a relation rather than an attribute. The instance attributes and relations are created from the variable initializations made in the reverse operation `__init__`. The static attributes and relations are created from the variable initializations made at the toplevel of the class. An operation is declared static when it has the decorator `@staticmethod`, is it declared *abstract* when it has the decorator `@abstractmethod`.

The *Python code reverse* works in two steps, during the first step it scans all the files to establish the list of defined types, then the second step may reverse the files.

When the *Python code reverse* has a problem, for instance on a syntax error, it produces a warning in the trace window. The trace window is automatically opened by BOUML, it may also be manually opened through the [Tools menu](#).

The reversed files are considered composed in three parts :

- a header without class definition
- only classes definition, this part starts with the first class and stops before the first form which is not a class. The classes must be at the toplevel level, this means not embedded in a *if* or other form. This part produces *\$(definition)* in the definition of the artifact
- the rest of the file, this part is placed at the end of the definition of the artifact.

---

Previous : [Python generator](#)

Next : [Idl generator](#)



# Idl generator

The Idl code generator is a *plug-out* directly written in C++.

The generated sources follows the definition made in BOUML at the [artifact](#) / [class](#) / [operation](#) / [relation](#) / [attribute](#) / [extra member](#) levels.

When the code generation is applied on a artifact associated to several classes, the code generation is made for all these classes. Nevertheless the the Idl code generator produce first the code in memory and update the appropriate files only when it is necessary, to not change the last write date of the files for nothing. Depending on the toggle *verbose code generation* of the global menu *Languages* the code generator is verbose or not.

The Idl code generator *plug-out* may be called on :

- a [class](#) : in this case the code generation is in fact applied on the class's [artifact](#), then on all the classes associated to this [artifact](#)
- a [artifact](#) : in this case the code generation is in fact applied on all the [classes](#) associated to the [artifact](#)
- a [class view](#) : the code generation will be applied on all the sub [classes](#), then on all the [artifacts](#) associated to these [classes](#)
- a [deployment view](#) : the code generation is applied on all the sub [artifacts](#)
- a [package](#) (may be the project itself) : the code generation will be applied on all the sub [class views](#) and [deployment views](#), then on all their sub [classes](#) and [artifacts](#).

When the Idl code generator is ask through the [Tools](#) menu, it is applied on the project, then on all the [artifacts](#).

The name of the generated files depend on the artifact name, the extension depend on the language and is fixed for each by the [generations settings](#) (see below), the directory where the files are generated may be set in each package containing directly or indirectly the artifact (see [generation directory](#)).

---

## artifact

The Idl definition of a artifact is set through the [Idl source](#) tabs of the [artifact dialog](#).

The code generation depend on the stereotype of the artifact :

- *text* : the IDL definition of the artifact is produced without changes, the name of the generated file is the name of the artifact, including the extension.
- *source* : see below
- else nothing is generated for the artifact.

The generated file name is the artifact's name with the extension specified in the first Idl tab of the [generations settings](#) :





In BOUML the generated code is obtained by the substitution of macros in a text, the macros known by the Idl code generator are :

- **`${NAME}`** produce the artifact name capitalized, **`${Name}`** produce the artifact name with the first letter capitalized, at least **`${name}`** produce the artifact name without modification.
- **`${comment}`** is replaced by the artifact description adding //
- **`${description}`** is replaced by the artifact description without adding //
- **`${includes}`** is not yet managed by the Idl generator and produce an empty string.
- **`${module_start}`** is replaced by the *module* xx forms, dependent on the [module](#) specifications associated to the BOUML package containing the [deployment view](#) where the artifact is defined.
- **`${definition}`** is replaced by the definition of the classes associated to the artifact.
- **`${module_end}`** is replaced by }, dependent on the [module](#) specifications associated to the BOUML package containing the [deployment view](#) where the artifact is defined.
- **`@{xyz}`** is replaced by the *user property* value in case xyz is the key of this property. The property is searched first for the *artifact*, if not found for the *deployment view* containing the *artifact*, if not found in the *package* containing the *deployment view* etc ...

## Class

The Idl definition of a class is set through the [Uml](#) and [Idl](#) tabs of the [class dialog](#).

A Idl type definition may be a valuetype, an interface, a struct, an union, an enum, an exception or a typedef defined through depending on the stereotype and its translation in Idl (see [generation settings](#)).

In BOUML the generated code is obtained by the substitution of macros in a text, the macros known by the Idl code generator are :

- **`${comment}`** is replaced by the description of the class adding //
- **`${description}`** is replaced by the class description without adding //
- **`${abstract}`** only for the valuetypes, produce *abstract* when the valuetype is abstract
- **`${custom}`** only for the valuetypes, produce *abstract* when the valuetype is custom
- **`${name}`** is replaced by the class's name

- ***\${inherit}*** is replaced by the inheritance
- ***\${members}*** is replaced by the code generated for all the class's members ([relations](#), [attributes](#), [operations](#) and [extra members](#)) following the browser order.
- ***\${switch}*** only for the unions, is replaced by the union switch type
- ***\${items}*** only for the enums, is replaced by the enum's items definition
- ***@{xyz}*** is replaced by the *user property* value in case *xyz* is the key of this property. The property is searched first for the class, if not found for the container of the class (an other class of a *class view*) etc ...

## Operation

The Idl definition of an operation is set through the [Uml](#) and [Idl](#) tabs of the [operation dialog](#).

The indentation of the first line of the declaration/definition give the indentation added to the class definition for all the operation definition.

In BOUML the generated code is obtained by the substitution of macros in a text, the macros known by the Idl code generator are :

- ***\${comment}*** is replaced by the description of the operation adding //
- ***\${description}*** is replaced by the operation description without adding //
- ***\${oneway}*** produce *oneway* when the operation is oneway
- ***\${type}*** is replaced by the value type (see the UML tab)
- ***\${name}*** is replaced by the name of the operation.
- ***\${(}*** and ***\${)}*** produce ( and ), but there are also a mark for BOUML to find the parameters list
- ***\${t<n>}*** and ***\${p<n>}***  
produce the type and the name of each parameter (count from 0), this allows you to remove a parameter, etc ...
- ***\${raise}*** is replaced by the form *raise (...)* when at least an exception is defined in the UML tab.
- ***\${raisentl}*** produce a line break when the operation have exception. In case you do not like this notation, change the [generation settings](#) to remove this macro.
- ***@{xyz}*** is replaced by the *user property* value in case *xyz* is the key of this property. The property is searched first for the operation, if not found for the class containing the operation etc ...

## Attribute

The Idl definition of an attribute is set through the [Uml](#) and [Idl](#) tabs of the [attribute dialog](#).

The indentation of the first line of the definition give the indentation added to the class definition for all the attribute definition.

The definition of an attribute depend on the stereotype of its class.

### interface's attribute

In BOUML the generated code is obtained by the substitution of macros in a text, the macros known by the Idl code generator are :

- ***\${comment}*** is replaced by the description of the attribute adding //
- ***\${description}*** is replaced by the attribute description without adding //
- ***\${readonly}*** produce *readonly* when the attribute is read only (see the UML tab)
- ***\${attribute}*** produce *attribute*
- ***\${type}*** is replaced by the type of the attribute (see the UML tab)
- ***\${stereotype}*** is replaced by the [translation in Idl](#) of the relation's stereotype (see the UML tab)

- ***#{multiplicity}*** is replaced by the multiplicity of the relation (see the UML tab)
- ***#{name}*** is replaced by the attribute's name (see the UML tab)
- ***@{xyz}*** is replaced by the *user property* value in case *xyz* is the key of this property. The property is searched first for the attribute, if not found for the class containing the attribute etc ...

## valuetype's attribute

The macros known by the Idl code generator are :

- ***#{comment}*** is replaced by the description of the item adding //
- ***#{description}*** is replaced by the item description without adding //
- ***#{visibility}*** produce the visibility (see the UML tab)
- ***#{type}*** is replaced by the type of the attribute (see the UML tab)
- ***#{stereotype}*** is replaced by the [translation in Idl](#) of the relation's stereotype (see the UML tab)
- ***#{multiplicity}*** is replaced by the multiplicity of the relation (see the UML tab)
- ***#{name}*** is replaced by the attribute's name (see the UML tab)
- ***@{xyz}*** is replaced by the *user property* value in case *xyz* is the key of this property. The property is searched first for the attribute, if not found for the class containing the attribute etc ...

## union's attribute

The macros known by the Idl code generator are :

- ***#{comment}*** is replaced by the description of the item adding //
- ***#{description}*** is replaced by the item description without adding //
- ***#{readonly}*** produce *readonly* when the attribute is read only (see the UML tab)
- ***#{case}*** is replaced by the case associated to the attribute
- ***#{type}*** is replaced by the type of the attribute (see the UML tab)
- ***#{stereotype}*** is replaced by the [translation in Idl](#) of the relation's stereotype (see the UML tab)
- ***#{multiplicity}*** is replaced by the multiplicity of the relation (see the UML tab)
- ***#{name}*** is replaced by the attribute's name (see the UML tab)
- ***@{xyz}*** is replaced by the *user property* value in case *xyz* is the key of this property. The property is searched first for the attribute, if not found for the class containing the attribute etc ...

## enumeration item

The macros known by the Idl code generator are :

- ***#{comment}*** is replaced by the description of the item adding //
- ***#{description}*** is replaced by the item description without adding //
- ***#{name}*** is replaced by the item's name (see the UML tab)
- ***@{xyz}*** is replaced by the *user property* value in case *xyz* is the key of this property. The property is searched first for the attribute, if not found for the class containing the attribute etc ...

## constant

The macros known by the Idl code generator are :

- ***#{comment}*** is replaced by the description of the item adding //
- ***#{description}*** is replaced by the item description without adding //

- ***\_\${type}*** is replaced by the type of the attribute (see the UML tab)
- ***\_\${name}*** is replaced by the attribute's name (see the UML tab)
- ***\_\${value}*** is replaced by the initial value of the attribute (see the UML tab)
- ***@{xyz}*** is replaced by the *user property* value in case *xyz* is the key of this property. The property is searched first for the attribute, if not found for the class containing the attribute etc ...

## Relation

The Idl definition of a relation is set through the [Uml](#) and [Idl](#) tabs of the [relation dialog](#).

The indentation of the first line of the definition gives the indentation added to the class definition for all the relation definition.

### Relation equivalent to an attribute in an interface

In BOUML the generated code is obtained by the substitution of macros in a text, the macros known by the Idl code generator are :

- ***\_\${comment}*** is replaced by the description of the attribute adding //
- ***\_\${description}*** is replaced by the relation description without adding //
- ***\_\${readonly}*** produce *readonly* when the relation is read only (see the UML tab)
- ***\_\${attribute}*** produce *attribute*
- ***\_\${type}*** is replaced by the name of the class pointed by the relation (see the UML tab)
- ***\_\${name}*** is replaced by the role's name (see the UML tab)
- ***\_\${stereotype}*** is replaced by the [translation in Idl](#) of the relation's stereotype (see the UML tab)
- ***\_\${multiplicity}*** is replaced by the multiplicity of the relation (see the UML tab)
- ***@{xyz}*** is replaced by the *user property* value in case *xyz* is the key of this property. The property is searched first for the relation, if not found for the class containing the relation etc ...

### Relation equivalent to an attribute in a valuetype

In BOUML the generated code is obtained by the substitution of macros in a text, the macros known by the Idl code generator are :

- ***\_\${comment}*** is replaced by the description of the relation adding //
- ***\_\${description}*** is replaced by the relation description without adding //
- ***\_\${visibility}*** produce the visibility (see the UML tab)
- ***\_\${type}*** is replaced by the name of the class pointed by the relation (see the UML tab)
- ***\_\${name}*** is replaced by the role's name (see the UML tab)
- ***\_\${stereotype}*** is replaced by the [translation in Idl](#) of the relation's stereotype (see the UML tab)
- ***\_\${multiplicity}*** is replaced by the multiplicity of the relation (see the UML tab)
- ***@{xyz}*** is replaced by the *user property* value in case *xyz* is the key of this property. The property is searched first for the relation, if not found for the class containing the relation etc ...

### Relation equivalent to an attribute in an union

In BOUML the generated code is obtained by the substitution of macros in a text, the macros known by the Idl code generator are :

- ***\_\${comment}*** is replaced by the description of the relation adding //
- ***\_\${description}*** is replaced by the relation description without adding //
- ***\_\${readonly}*** produce *readonly* when the relation is read only (see the UML tab)

- *`\${type}`* is replaced by the name of the class pointed by the relation (see the UML tab)
- *`\${name}`* is replaced by the role's name (see the UML tab)
- *`\${stereotype}`* is replaced by the [translation in Idl](#) of the relation's stereotype (see the UML tab)
- *`\${multiplicity}`* is replaced by the multiplicity of the relation (see the UML tab)
- *`\${case}`* is replaced by the case associated to the relation
- *@{xyz}* is replaced by the *user property* value in case *xyz* is the key of this property. The property is searched first for the relation, if not found for the class containing the relation etc ...

## Inheritance

The class inheritance are managed at the [class level](#).

---

## Extra member

The Idl definition of an extra member is set through the [Idl](#) tabs of the [extra member dialog](#).

No macros.

---

Previous : [Python reverse](#)

Next : [Roundtrip body](#)

# Roundtrip body

This is a *plug-out* directly written in C++.

When you set *preserve operations's body* through the *miscellaneous* menu, you edit the operation's bodies out of Bouml, and this one doesn't contains up to date operation's bodies (except for operations setting *force body generation*). This *plug-out* updates the operation's bodies in Bouml from the sources.

*Roundtrip body* works on the [artifact](#), not at class level. So, when you ask for *roundtrip body* on an [artifact](#) (or an upper level), all the operations of all the classes having the mark in the sources associated to the [artifact](#) will be *roundtripped*.

You can ask for *roundtrip body* at [package](#), [deployment view](#) or [artifact](#) level, except in this last case, the *roundtrip* is done recursively.

---

Previous : [Idl generator](#)

Next : [State machine generator](#)

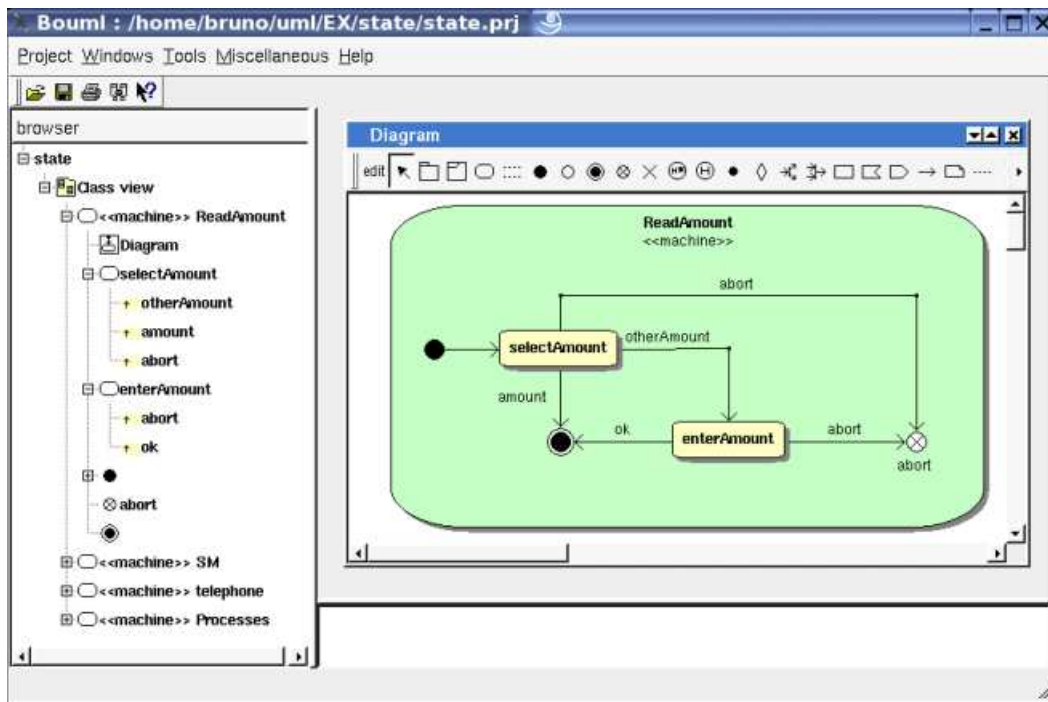
# State machine generator

The [state machine](#) generator create or update the classes implementing a behavioral state machine on which it is applied.

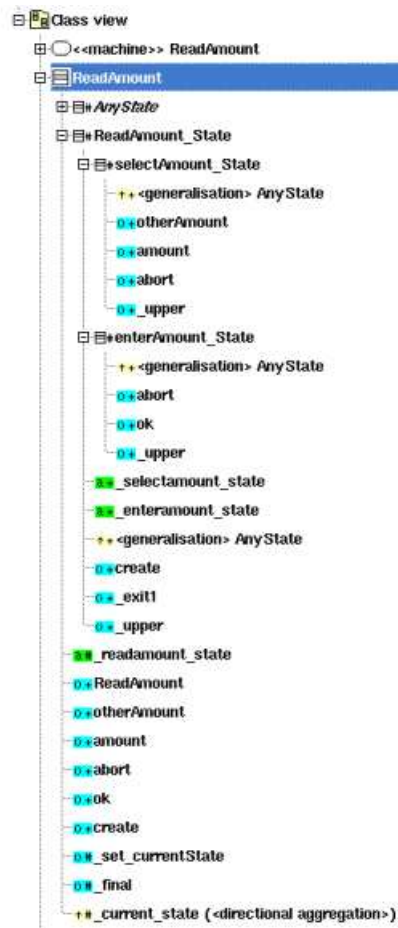
Currently the state machine generator only generates C++ definitions.

When you apply this plug-out on a state machine, a [class](#) having the same name (removing non legal characters) is created (or updated in case it already exist) in the same [class view](#) to implement it. This class doesn't contain static [attributes](#) and may have several instances. Each sub-state is managed through an own sub-class, the tree of the states and the associated classes is the same. For each trigger, whatever the state waiting for it, an [operation](#) having the same name (the C++ trigger must have a legal C++ name, the C++ guards must be legal C++ forms etc ...) is defined. Currently the triggers can't have parameters, and a transition can't have several triggers. You must apply these operations on the instance of the class implementing the state machine to simulate the event, the returned value is true until the final state is reached. The operation starting the execution is named *create* (the only legal trigger for a transition from an initial pseudo state). A trigger's name can't start by the character '\_', allowing me to proc operations and attributes starting by '\_' without collision.

For instance, in case you apply the state machine generator on the following state machine :



you will obtain these definitions :



Having an instance of *ReadAmount*, you must first call *create* to execute the initial transition, then you may call *otherAmount* etc ...

When you modify the *state machine* then you recall this *plug-out*, the classes are updated. The classes, operations and attributes produced by a previous state machine definition but now useless are deleted. Warning : if a state or other elements is renamed or moved to have a new parent, the corresponding class or class member is deleted and a new one is created. If you add classes or members, these ones will not be deleted while their container is not itself deleted, and if they don't have a user property named *STM generated* (so, don't duplicate elements created by the *state machine generator* to not have this property or remove it after).

Some parts of the generated source code are instrumented to produce a debug message using the function *puts* of *stdio*. The instrumentation is always a form :

```
#ifdef VERBOSE_STATE_MACHINE

puts("<the message>");

#endif
```

To activate these traces you must compile the code defining `VERBOSE_STATE_MACHINE`.

Previous : [Roundtrip body](#)

Next : [XMI generator](#)



# XMI generator

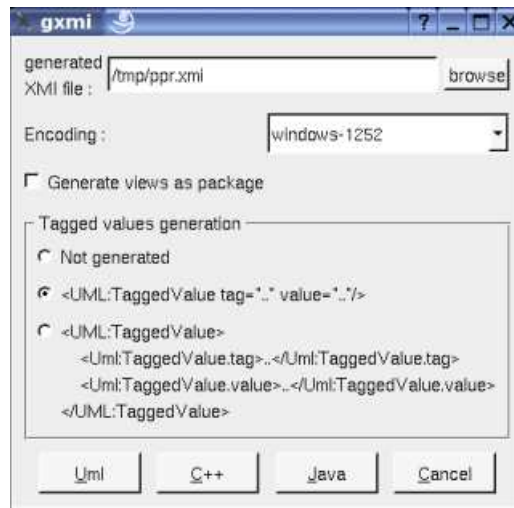
As it is signified by their name, the XMI *generators* generates the definition of the project using XMI.

They are two generators, one producing XMI 1.2 the other XMI 2.1

## XMI 1.2 generator

To be compatible with existing tools, the used format is *xml 1.0*, *xmi 1.2* for *UML 1.4*.

This *plug-out* must applied on the project itself, it asks you to specify where the *xmi* file must be generated and if you want to produce the UML, C++ or Java definitions :



Currently only the UML definition of the *packages*, *views* (generated as package or not generated depending on the toggle above), *classes*, *attributes*, *relations*, *operations*, *use cases*, *components* (without UML2.0 features) and *nodes* are saved in the produced file. Classes defined under a *use case* or a *use case view* are generated as *actor*, their operations, attributes and relations are not generated except *dependencies* and *generalizations*.

The XMI generator manages the following options (in order) :

- *-uml* or *-c++* or *-java* to specify the target language, mandatory if other arguments are given (except the port number given by Bouml itself)
- *generated XMI file path* (must not contains space)
- *encoding*, for instance windows-1252
- *-view*, optional, to generate views as package
- *-simpleTv* to generate tagged values with the simple form, *-complexTv* to generate tagged values with the complex form, else the tagged values are not generated

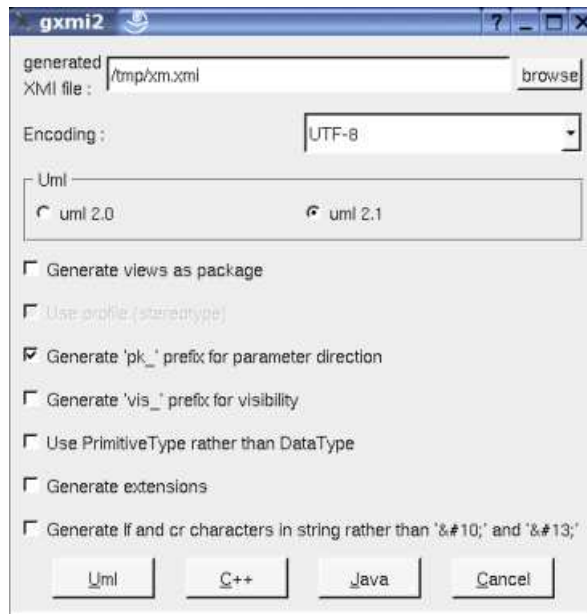
In case these arguments are given the dialog doesn't appears, this allows to export a project in XMI without any manual action.

These options may be set through the Tool setting dialog, or when you ask to start a plug-out when Bouml is lauched (see [here](#)).

## XMI 2.1 generator

The used format is *xml 1.0*, *xmi 2.1* for *UML 2.0* or *2.1*

This *plug-out* must applied on the project itself, it asks you to specify where the *xmi* file must be generated and if you want to produce the UML, C++ or Java definitions following some options :



The XMI generator manages the following options (in order) :

- `-uml` or `-c++` or `-java` to specify the target language, mandatory if other arguments are given (except the port number given by Bouml itself)
- *generated XMI file path* (must not contains space)
- *encoding*, for instance *UTF-8*
- `-view`, optional, to generate views as package
- `-uml2.0`, optional, to generate for UML 2.0 rather than UML 2.1
- `-pk`, optional, to prefix the parameter direction by *pk\_*
- `-vis`, optional, to prefix the visibility by *vis\_*
- `-primitiveType`, optional, to define primitive and derived types (like *int \**) through a *primitive* type rather than a *data type*
- `-extension`, optional, to ask the generator to produce or not through an extension the *stereotypes* and user properties (*tagged value*)
- `-lf`, optional, to produce the characters *line feed* and *carriage return* without modification, else there are replaced by `&#10;` and `&#13;`;

In case these arguments are given the dialog doesn't appears, this allows to export a project in XMI without any manual action.

These options may be set through the Tool setting dialog, or when you ask to start a plug-out when Bouml is lauched (see [here](#)).

---

Previous : [State machine generator](#)

Next : [XMI import](#)

# XMI import

The XMI *import* allows to import XMI 2.1 for *UML 2.0* or *2.1* file. The top level element in the XMI file must be a *project* or a *package*, else nothing will be imported.

This *plug-out* must be applied on the project itself or a package, it asks for a file and import it, at end the number of main imported elements is written.

This release doesn't look at the language indications, and only define elements at UML level.

---

Previous : [XMI generator](#)

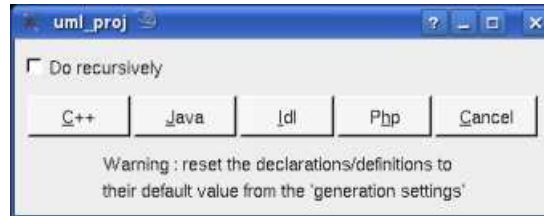
Next : [UML projection](#)

# UML projection

This *plug-out* projects the UML elements in the desired language, setting the definitions/declarations to their default value according to the *generation settings* like when you use the *button default declaration/definition* in a dialog.

This *plug-out* can be applied on the project, a *package*, a *class view*, a *class*, an *operation*, an *attribute* or a *relation* and works recursively in case you ask for.

When you launch it the following dialog appears :



Ask to also project the sub elements if desired, then hit the button corresponding to the desired language projection.

---

Previous : [XMI import](#)

Next : [Html documentation generator](#)

# HTML documentation generator

The *HTML documentation generator* is developed using BOUML and is an example of a program implemented simultaneously in C++ and Java.

As it is signified by its name, the *HTML documentation generator* generates HTML pages describing the object on which it is applied and its sub objects, with some indexes. When the generator is started some dialogs appear to configure the result, you may also see them using the options, see below

The diagrams are produced in the generated page(s), the used scale and picture size are the default ones except when there are set (for instance see [use case diagram](#)).

The HTML code generator manages the following options (in order) :

- *-flat*, optional, to produce the class definitions in index.html rather than in separate files.
- *-svg*, optional, to produce SVG images rather than PNG ones.
- *generated Html files directory*  
(must not contains space) mandatory if the next arguments are given (except the port number given by Bouml itself)
- *-del\_html*, optional, to remove already existing HTML files in the specified directory
- *-del\_css*, optional, to replace the file style.css if it already exists in the specified directory, do not use this option if you have your own styles. Of course in all the cases style.css is created if it doesn't exist

In case the generated files directory is specified the dialogs don't appears, this allows to generate the HTML documentation without any manual action.

These options may be set through the Tool setting dialog, or when you ask to start a plug-out when Bouml is launched (see [here](#)).

---

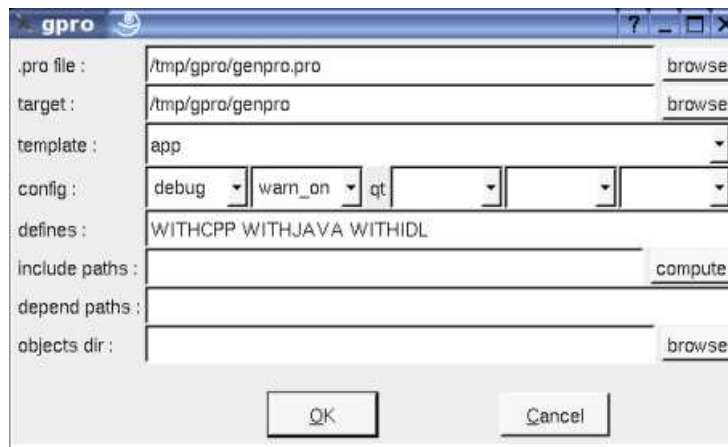
Previous : [XMI import](#)

Next : [genpro](#)

## Genpro : .pro generator

*genpro* is an other example of *plug-out developed* with BOUML, its goal is to generate the *.pro* file used by *qmake* or *tmake* to produce the *Makefile* of a C++ program using *Qt* ... for instance a *plug-out*.

*genpro* has itself a graphical interface implemented thanks to *Qt*, when it is applied on a [artifact](#) stereotyped *executable* the following window appears (here it is applied on the *gpro* executable artifact itself, the C++ generation directory is */tmp/gpro*) :



By default the *.pro* file is produced in the directory where the executable is made, the executable's name is the artifact's name except when this least is '*executable*', in this case is is the project's name. You may specify the *template* (*app*, *lib* or *subdirs*), set some flags like *debug*, the C++ *pre-processor* variable definitions (see [plug-out](#) for explanations on WITHCPP WITHJAVA and WITHIDL), ask to place the *objects* in a directory which is not the one used for the sources etc ...

The source files considered by *genpro* are the ones produced by the *source* artifacts associated to the *executable* artifact on which *genpro* is applied.

The same technique may be used to directly generate a *Makefile* for a C++ or Java program.

---

Previous : [Html documentation generator](#)

Next : [Rose project import](#)

# Rose project import

This *plug-out* is developed using BOUML in C++. As it is signified by its name, this one import a Rose model from its .mdl (and associated files) into BOUML in the package/project where it is launch.

This *plug-out* is under development, actually the imported diagrams are created empty, the BOUML's API doesn't yet allows to modify them. Actually the [reverse C++](#) doesn't allows to complete the imported model with the files generated by Rose.

BOUML manages the case of the controlled packages through a .cat or a .sub files find through a \$variable rather than a fixed path. When a variable is not an environment variable, BOUML ask for its value (only one time, or course).

BOUML write a message into the tool window each time it cannot import something.

---

Previous : [genpro](#)

Next : [Plug-out upgrade](#)

# Plug-out Upgrade

When new features are introduced in BOUML these ones are not managed by your (now old) *plus-outs*, to automatically upgrade them, the distribution of BOUML contains the *upgrade plug-out*.

Its goal is to upgrade the *plug-outs* not made from the last up to date version of the *plug-out empty*, adding the missing features.

It is applied through the entry *plug-out upgrade* of the menu *Tools* (this entry doesn't appears while a dialog is opened or when the project is not a *plug-out*) :



When it is applied, the *plug-out* ask you for a confirmation, thus it adds the needed classes etc ... When all is done a new dialog appears for information and a *save-as* is called asking you to save your *plug-out* in a new location, thus in all the cases the project is closed.

---

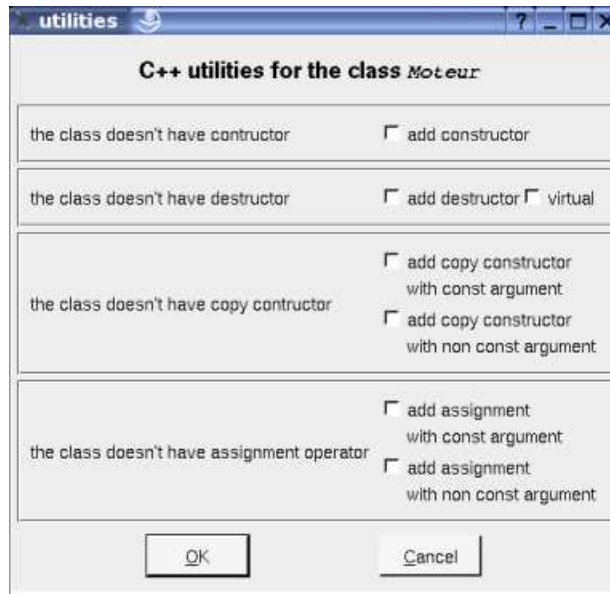
Previous : [Rose import](#)  
Next : [C++ utilities](#)



# C++ Utilities

Some utilities concerning C++.

When it is applied to a class, it allows to add specific operations not yet defined :



The content of the dialog depend on the already defined operations.

---

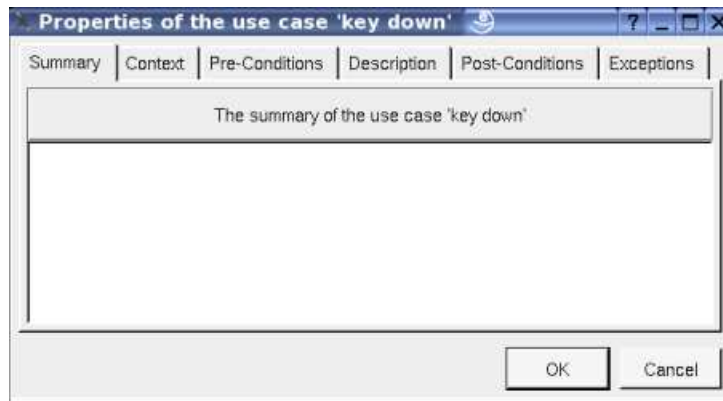
Previous : [Plug-out upgrade](#)

Next : [Use case wizard](#)

# Use case wizard

A very simple plug-out defined in C++ and Java both (like the HTML code generator) to extend [use case](#) features.

When it is applied to a use case, it allows to specify through text a summary, context, pre-conditions, description, post-conditions and exception :



Don't hesitate to modify this plug-out (saving it output of the Bouml installation directory to not loose your modifications on a f Bouml installation) for your own purposes !

---

Previous : [C++ utilities](#)

Next : [File control](#)



# Deploy classes

A very simple plug-out defined in C++ and Java both (like the HTML code generator) to automatize the creation of the [artifact](#) for all the classes of the [class view](#) on which the plug-out is applied :

- If the [class view](#) doesn't have an associated [deployment view](#), an associated [deployment view](#) having the name of the [class view](#) is created in the same package.
- An [artifact](#) with the stereotype source is created and associated to all the classes of the [class view](#) without associated [artifact](#), the name of the [artifact](#) is the name of the class.

---

Previous : [File control](#)

Next : [global change](#)

# Global change

This *plug-out* defined in C++ allows to do global modification of the [artifact](#), [classes](#), [operations](#), [attributes](#) and [relations](#) definitions, working recursively from the *browser* element on which it is applied.

When you change default definitions or declarations in the *generation settings*, these changes are not propagated in the already existing elements to not broke them. This *plug-out* allows you to apply these changes if needed on a controlled way.

This *plug-out* may be applied on a [package](#) (including the *project* itself), [class view](#), [deployment view](#) or a [class](#), and works through this dialog :

The *filters* are optional and allow to not do the change on all the potential definitions and declarations. An empty filter is ignored, else a filter may be used to consider the forms containing it (*with* is set) or not containing it (*without* is set). When you have several filters you may conjugate them with *and* or *or*, the priority of *and* and *or* are the standard ones (like in C++ Java or Php). In case you have one filter, you must use filter 1. In case you have two filters you must use filter 1 and 2.

It is also possible to filter through the *stereotype*. Choose *any* to not consider the stereotypes, *is* to consider only the elements having the specified stereotype (may be an empty one), and *is not* to not consider the elements having the specified stereotype (may be an empty one).

Several *targets* may be set, to choose on which kind of elements the change must be done.

Several *languages* may be set too, to choose for which language the definition / declarations the changes must be done.

*Current* indicates the current form to be changed, *new* the value, the substitution being done on the element definitions / declarations compatible with all the previous indications.

Notes : except for the stereotype, \n is used to indicate a line break in the inputs. The carriage return characters are removed to not pollute the filtering, searching and replacement. At most one replacement is done by form.

Examples of use :

- you want to replace everywhere *\$(comment)* by *\$(description)* : all the filters must be empty, set *any*, *artifact*, *class*, *operation*, *attribute* and *relation*, set the right languages, set *current* to *\$(comment)* and *\$new* to *\$(description)*, press *Replace*.
- In the old releases, the operation body indent must be set inside the body because the form *\$(body)* was not preceded by two spaces, to replace the old definitions by the new default one where *\$(body)* is placed after two spaces, and to keep the definition having these two spaces unchanged : the first filter must be set to *<space><space>\$(body)* (where *<space>* is a space character !) and you set *without*, the other filters are empty, set *operation* only, set the right languages, set *current* to *\$(body)* and *\$new* to *<space><space>\$(body)* (where *<space>* is a space character !), press *Replace*.

When you ask for a replacement, a dialog indicate how many changes was done, and may be how many can't be done because it is not possible to set the definition / declaration of a *read only* element.

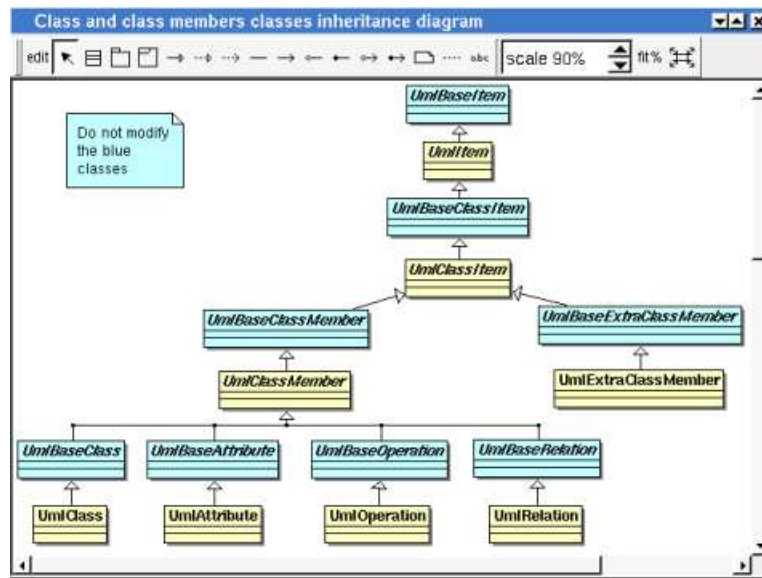
Warning, global changes may have dramatic consequences when the filtering is not sufficient !

---

Previous : [Deploy classes](#)

Next : [plug-outs](#)





In this diagram the blue classes are the *system* classes, the other classes are yours. *UmlItem* is the *user* class inherited by almost all the other classes.

The disadvantage of this choice in C++ is to recompile many files when a base class is modified, but this allows to define (*virtual*) operations managing several classes.

I do not describe in this documentation all the predefined classes and their operations, the better is to look at the model of a new created *plug-out* or to [generate](#) the html pages describing it (this is done under the [empty\\_html](#) directory, the html entry page is [index-withframe.html](#)). I also recommend you to look at the *plug-outs* given with the BOUML distribution. Nevertheless you have to know :

- the entry point of a *plug-out* in C++ and Java is placed in the artifact named *Main* under *USER API*, refer to it
- in C++ :

if the C++ pre-processor variables WITHCPP, WITHJAVA, WITHPHP and WITHIDL are defined (using *#define* or -D option of the G++ compiler, etc ...) you will get all the informations from BOUML.

if the C++ pre-processor variables WITHCPP, WITHJAVA, WITHPHP and WITHIDL are not defined you will access only to UML data, this means for instance that the operations *cppDecl javaDecl phpDecl* and *idlDecl* will not be defined for *UmlBaseClassItem*, this limits the amount of data managed by a *plug-out* and reduce the execution time and used memory space

if only WITHCPP is defined you will access only to UML and C++ data, this means for instance that the operations *javaDecl phpDecl* and *idlDecl* will not be defined for *UmlBaseClassItem*, this limits the amount of data managed by a *plug-out* and reduce the execution time and used memory space

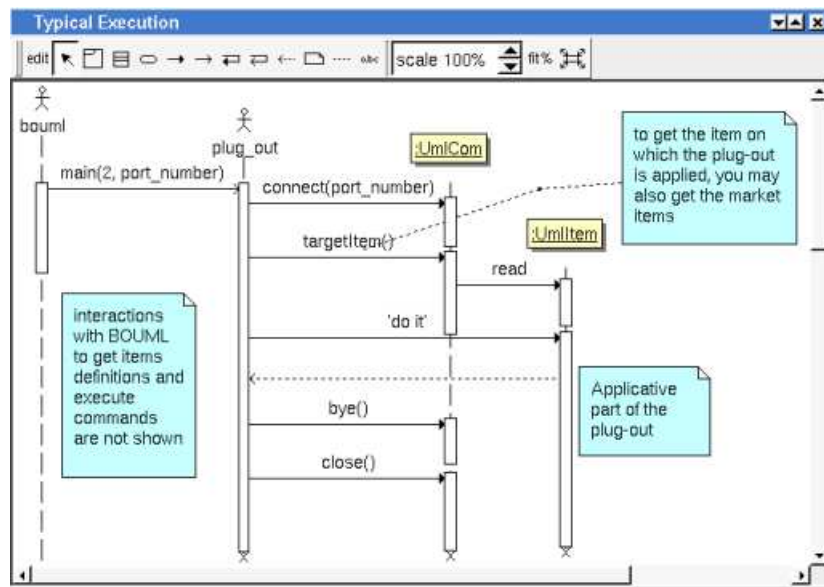
if only WITHJAVA is defined you will access only to UML and Java data, this means for instance that the operations *cppDecl phpDecl* and *idlDecl* will not be defined for *UmlBaseClassItem*, this limits the amount of data managed by a *plug-out* and reduce the execution time and used memory space

if only WITHPHP is defined you will access only to UML and Php data, this means for instance that the operations *cppDecl javaDecl* and *idlDecl* will not be defined for *UmlBaseClassItem*, this limits the amount of data managed by a *plug-out* and reduce the execution time and used memory space

if only WITHIDL is defined you will access only to UML and IDL data, this means for instance that the operations *cppDecl javaDecl* and *phpDecl* will not be defined for *UmlBaseClassItem*, this limits the amount of data managed by a *plug-out* and reduce the execution time and used memory space

the other combinations where only two of these three pre-processor variables are defined are illegal.





## Updating a *plug-out*

See [upgrade plug-out](#).

## To start a *plug-out* when Bouml is launch

It is possible to apply a *plug-out* on the project when Bouml is started through your *shell* or command line :

```
bouml <project file> -exec <plug out> <plug out options> -exit
```

- *<project file>* indicates the model to load as usual.
- *<plug out>* indicates the *plug-out* to apply on the project. The C++ generator is named *cpp\_generator*, the Java generator *java\_generator*, the Php generator *php\_generator*, the Idl generator *idl\_generator*, the HTML generator *ghhtml*, the XMI generator *gxmi* and *gxmi2*. If necessary the full path of the *plug-out* have to be done, warning : under Windows this path must not contain space.
- *<plug out options>* are the optional options you give to the *plug out*
- *-exit* is optional and allows to exit from Bouml when the *plug-out* execution is done.

For instance (HTML generator options are given [here](#), XMI generator options are given [here](#)) :

- "C:\Program Files\Bouml\bouml" "D:\...\aze\aze.prj" -exec C:\Progra~1\Bouml\gxmi -uml c:\aze.xmi windows-1252 -exit
- bouml ../qsd/qsd.prj -exec -flat /usr/lib/bouml/ghhtml /tmp/qsd\_html -del\_html -exit

## Exchanges with BOUML

The exchanges between BOUML and the *plug-outs* are supported by a TCP/IP socket. When you ask BOUML to start a *plug-out*, it search for a free port number starting at 1024. In case you have a *firewall* it may be necessary to configure it.

To reduce the number of exchanges with BOUML and run faster, a *plug-out* does not ask for BOUML each time you want to get a value. The *system* classes read and memorize all the data of a given item the first time you access to one of them (with the restrictions allowed WITHCPP, WITHJAVA, WITHPHP and WITHIDL). This is transparent for you until two *plug-outs* access to the same data, when one *plug-out* modify a value already read by the other, the second *plug-out* does not see the new value until it *unload* (an operation defined on *UmlBaseItem*) the object, then ask for the value.

## User properties

A *user property* is a couple key – value, the value may be empty.

By default the *user properties* are not used, nevertheless they exist! It is the only way to add information concerning any item (without modifying BOUML

obviously they are accessible in the *plug-outs*. The alone limitation is that the keys and the values must contain *string*, may be on several lines ... even the editor called by the dialogs is not a multi-lines editor !

The *plug-out* [genpro](#) use the properties of the artifact on which it is applied to save its data and give them is a future usage (supposing the project saved !).

To see/modify them, select the last tab named *User* present in the dialog of any kind of item.

## Compile a Java *plug-out*

To compile the java files, go in the directory where the files are and do `javac *.java` (don't compile file by file because some files associated to the API contain several classes).

## Compile a C++ *plug-out* under Windows

Qt 2.3 for Windows and Microsoft Visual C++ is available here : <ftp://ftp.trolltech.com/qt/non-commercial/>

Because this release of Qt is old, if you install Qt on a recent release of *Windows* (for instance *Xp*) the installation may be frozen at the end, in this case just kill the installation process and in fact the installation is done.

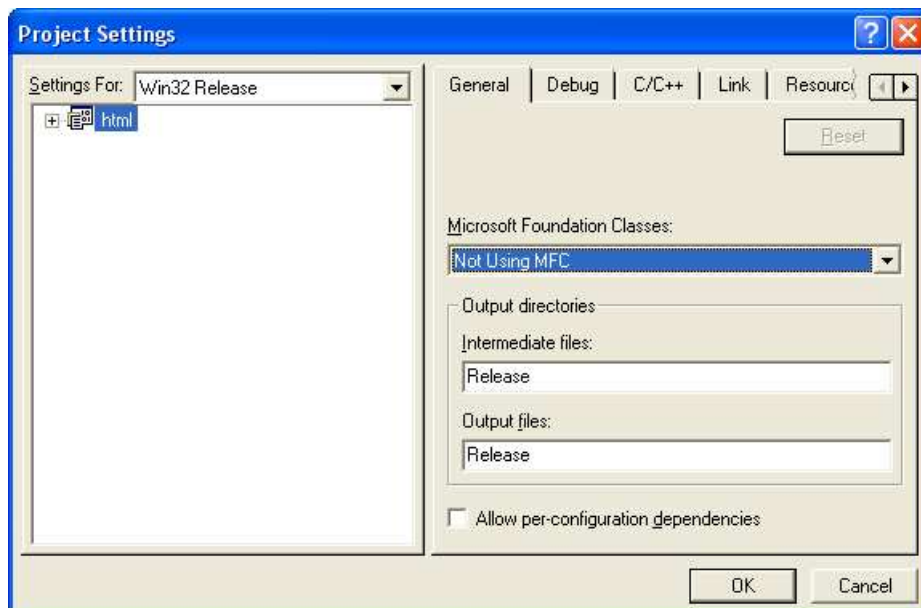
Read the Qt documents to see how to use Qt under *Microsoft Visual C++*.

On recent releases of *Windows / Visual C++* the macros allowing to automatically create a Qt project and / or to ask for the usage of *moc* on header files may not work, in this case read the following

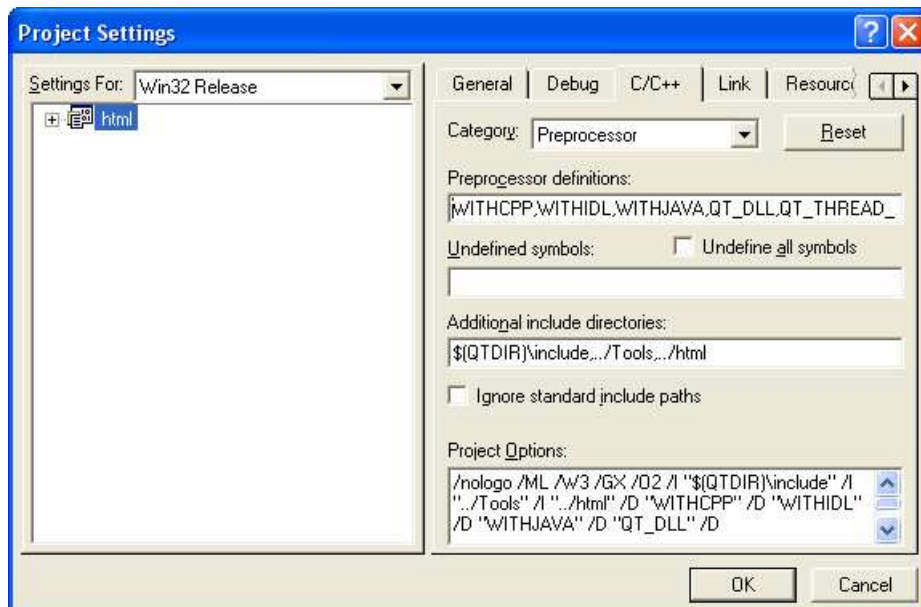
### To have a QT project

Edit project settings :

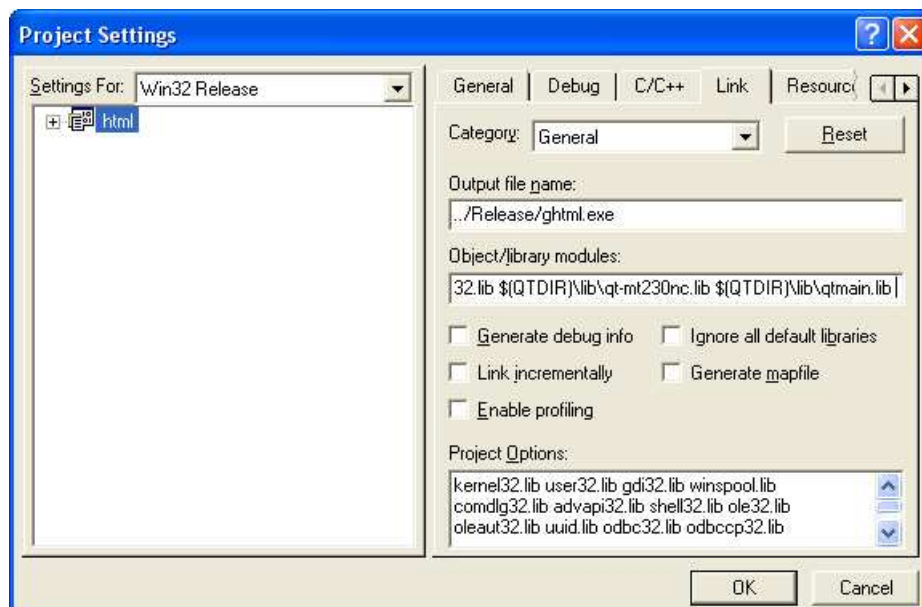
- the MFC must not be used :



- add the *preprocessor* definitions `WITHCPP WITHJAVA WITHPHP WITHIDL QT_DLL QT_THREAD_SUPPORT` (having both `WITHCPP WITHPHP` and `WITHIDL` is not mandatory, you may define only one of them, for instance `WITHJAVA` in case you are interested by the Java definition of the items, this allows to save memory space at the execution), add the *additional include directory* `$(QTDIR)\include` :



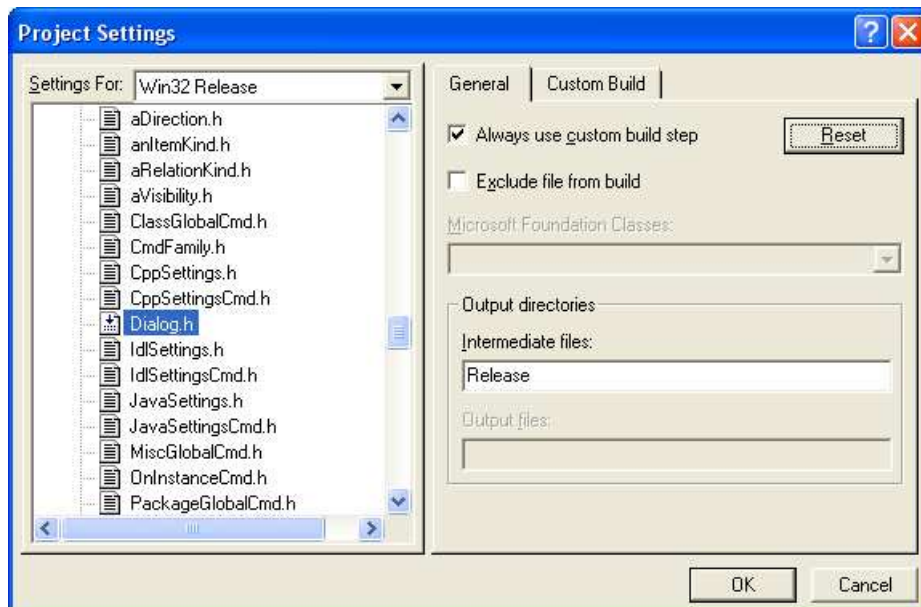
- add the *library modules* \$(QTDIR)\lib\qt-mt230nc.lib and \$(QTDIR)\lib\qtmain.lib :



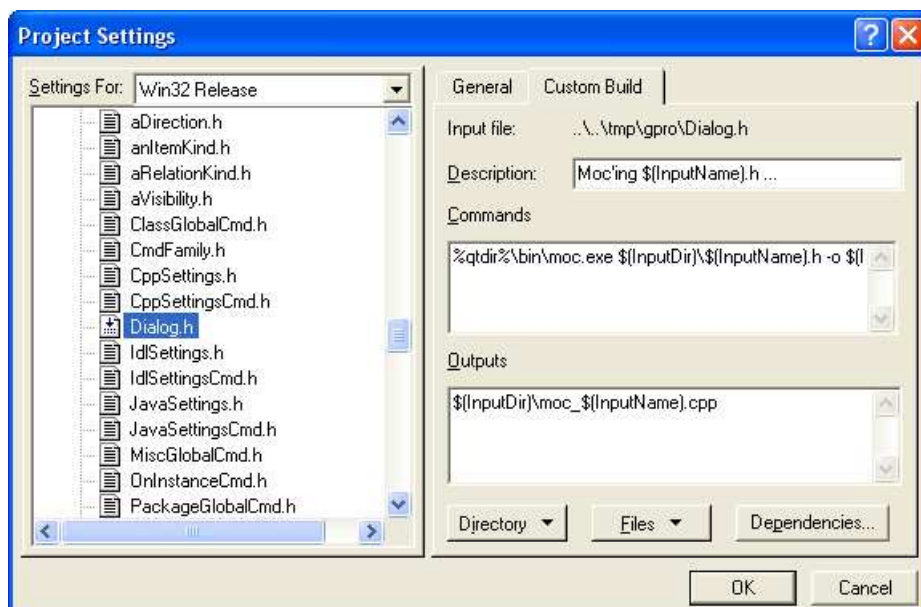
### To apply *moc* on a header

Edit the settings of the file in the following order :

- use *custom build step*:



- set *Output* to `$(InputDir)\moc_$(InputName).cpp`
- set *Commands* to `%qtdir%\bin\moc.exe $(InputDir)\$(InputName).h -o $(InputDir)\moc_$(InputName).cpp`
- set *Description* to Moc'ing `$(InputName).h`



Previous : [Use case wizard](#)

Next : [Project control](#)

# Project control

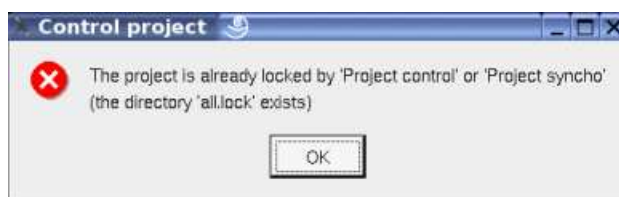
This tool manages an applicative write access to the project files without using the may be existing operating system capabilities. The goal is to give the write access of a the project files to at most one user per file, to avoid to merge modifications made by several users. The tool [Project synchro](#) associated to *project control* allows you to synchronize a repository project with the modifications made by several users (supposing each package modified by at most one user).

Because of the way the project data are saved by BOUML (see [project files](#)), the write access are managed at [package](#) level, and each information attached to a package (except its sub packages) have the write access of this package. Recall the *generation settings*, *default stereotypes* etc are associated to the *project* package, a good way is to not define views directly in the *project* package but to immediately create sub packages, each one owned by a different user or protected.

*Project Control* is not a *plug-out*, it is launch like BOUML or any tool, when you start it the following window appears :



To avoid inconsistencies, you can't apply *Project Control* several times on a given project, and you can't use it on a project while it is edited by BOUML or by *Project Synchro*, if this is the case a message warns you :



The write access are written inside the project files associated to the packages, if all the users access to the same shared image c project, the protection will be taken into account by BOUML. If there are several images of the project, after the use of *File Control* each image must be updated, or you must set the same write access using the tool by yourself.

All the operations are presented to be done for a given user, this user is indicated at the bottom of the window. Obviously the user : first the one starting the tool, the *id* is the value of *BOUML\_ID* for this user.

## Project menu

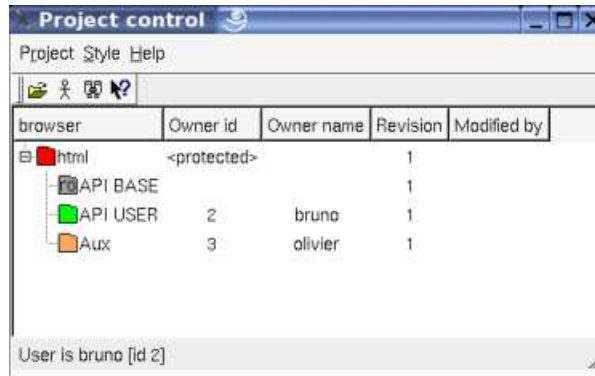


This menu allows

- to load project (may be through the historic list proposed at the bottom, this historic is the one of BOUML for the user starting *Project Control*),
- to change the current user,
- to search for a package from its name,
- or to quit the tool.

## Browser

When you load a project, only the packages are shown in the browser, for example on the project doing the HTML generation, if the *project* package is protected, the *API USER* owned by *bruno* and *Aux* by *olivier* :



The icon used to represent a package indicates the write access :

- : the package is not protected and may be modified by any user using BOUML
- : the package is protected, no users may modify it using BOUML
- : the package is own by the current user, only the current user can modify it using BOUML
- : the package is own by an other user (indicated by the columns *Owner id*, *Owner name*), only this user can modify it using BOUML
- : the package is not protected and its associated file is read-only, you can't modify the write access
- : the package is protected and its associated file is read-only, you can't modify the write access
- : the package is own by the current user and its associated file is read-only, you can't modify the write access
- : the package is own by an other user and its associated file is read-only, you can't modify the write access

## Package menu

A menu appears when you do a right mouse click on a package, its content depend on the package access, the entries are (*Ctrl* is replaced by *Apple* under MacOS X) :

- *Protect this package (Ctrl+p)* : to protect the current package, no user will be able to modify it using BOUML
- *Unassign this package (Ctrl+u)* : to not protect the current package, all the users will ne able to modify it using BOUML
- *Assign this package to <user> (Ctrl+a)* : to protect the current package except for the current user
- *Protect <user>'s packages recursively from this one* : to protect the packages own by the current user from the current package and going down recursively, the state of the packages not own by the current user is unchanged
- *Protect unassigned packages recursively from this one* : to protect all the unprotected packages from the current and going down recursively, the state of the other packages is unchanged
- *Protect all packages recursively from this one (Alt+p)* : to protect all the packages from the current and going down recursively
- *Assign to <user> unassigned packages recursively from this one* : to protect except for the current user all the unprotected packages from the current and going down recursively, the state of the other packages is unchanged
- *Assign to <user> all packages recursively from this one (Alt+a)* : to protect except for the current user all the packages from



the current and going down recursively

- *Unassign <user>'s packages recursively from this one* : to not protect the packages own by the current user from the current package and going down recursively, the state of the other packages is unchanged
- *Unassign all packages recursively from this one (Alt+u)* : to not protect the packages from the current package and going down recursively

Warning : the change is immediately made, and there is no *undo* !

## Change the current user

To change the current user, use the button representing an actor or the entry in the project menu, a dialog appears :



Choose a user from the list, set the name through the line editor when it is unknown for the selected BOUML\_ID .

---

Previous : [Plug-out](#)

Next : [Project synchro](#)

# Project synchronization

This tool synchronize several images of a project developed in parallel by several users. The synchronization is made by replacing old files by new ones. Note that the files are copied, not merged.

To use *Project synchro* several conditions *must* be followed, else the synchronization may broke you projects :

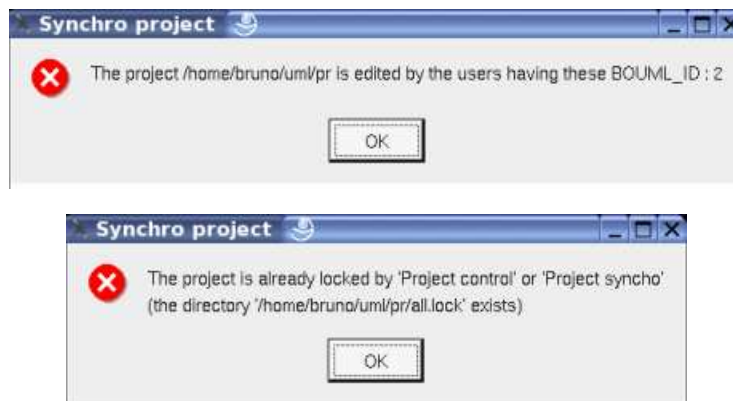
- To detect a file must be replaced by a new one, the tool uses the revision, not the date when the file was saved the last time. Because the revision is used, it is not possible to use the tool to synchronize projects modified by the releases of BOUML strictly less than 2.21 . If you already have several images of a project, you must first resynchronize them by yourself to obtain a unique image.
- Each user has his own copy of the project and works only on this copy. Obviously each user has his own BOUML\_ID.
- A given project file must be modified by only one user, for instance thanks to *Project Control*. This means that when the owner of a file change, you must perform a synchronization before (obviously it is useless to do a synchronization when a file without owner since the last synchronization become owned by someone).
- The synchronization is not possible for an image having a read-only file.

*Project Synchro* is not a *plug-out*, it is launch like BOUML or any tool, when you start it the following window appears :



The menu *Project* is used to load several images of the same project, to synchronize them or a part of them. it is also possible to start the tool given a list of project images in parameter.

To avoid inconsistencies, you can't apply *Project Synchro* several times on a given project, and you can't use it on a project while it is edited by BOUML or by *Project Control*, if this is the case a message warns you :

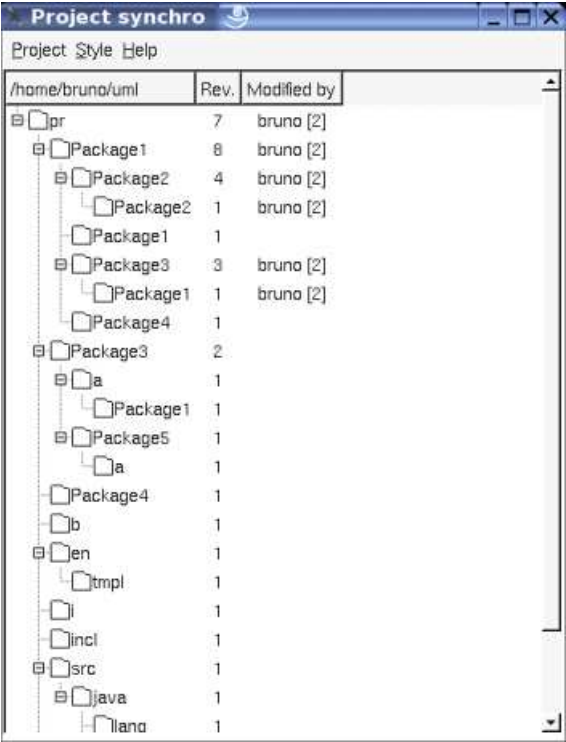


A project directory must also not contain *bak* files, these temporary files are present when BOUML or *File Synchro* crash during the modification of the files :

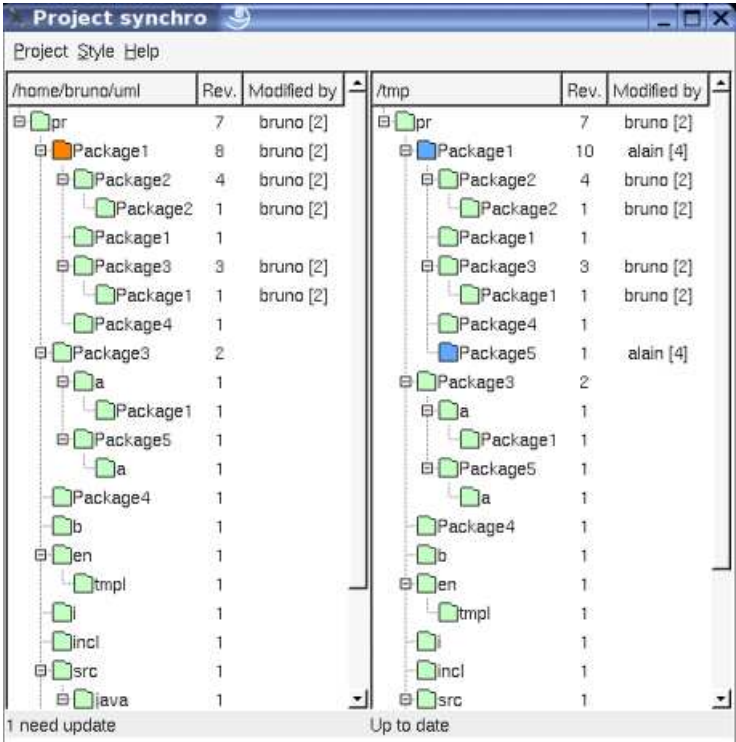


When only one project image is load the aspect is the following :



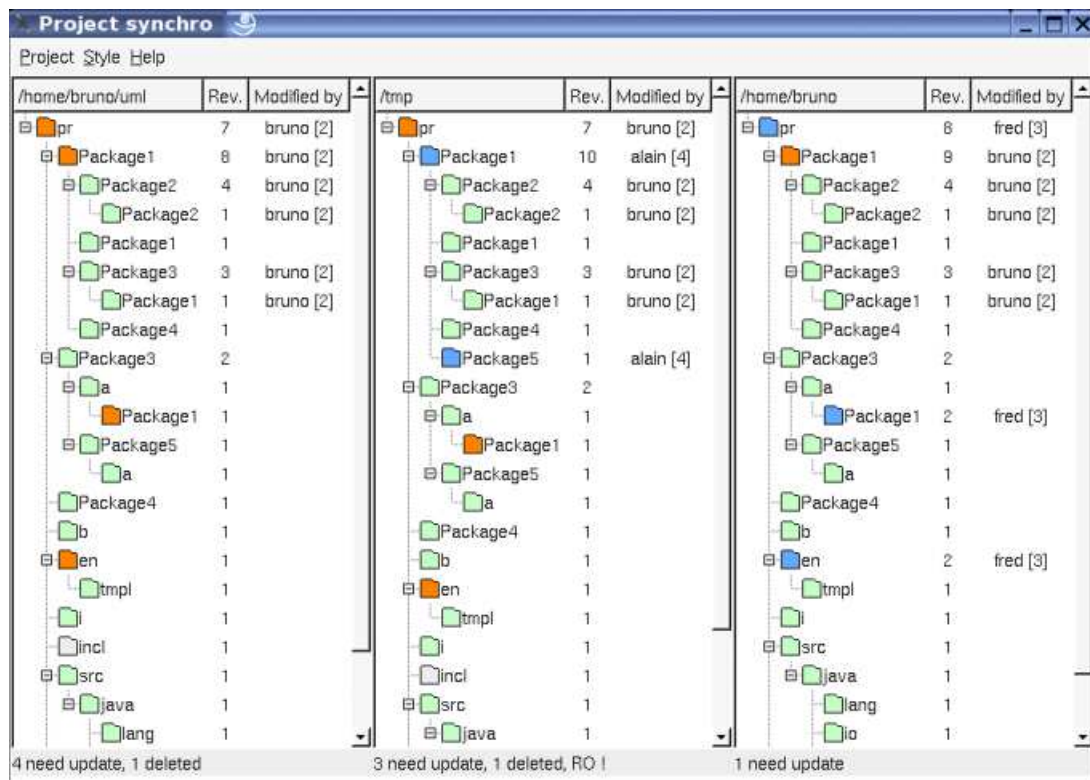


When you load a second project image, the color of the icons representing each package indicates if this one must be updated or not :



A package is green when it is up to date in all the images. It is orange when it must be updated, this means that an other image has new version of this package, and this new version is blue. A new package is also in blue. At least a deleted package is gray. In previous picture *package5* is blue because it is new (it doesn't exist in /home/bruno/uml/pr) and *package1* is blue on the right because it was modified to add the sub-package *package5*.

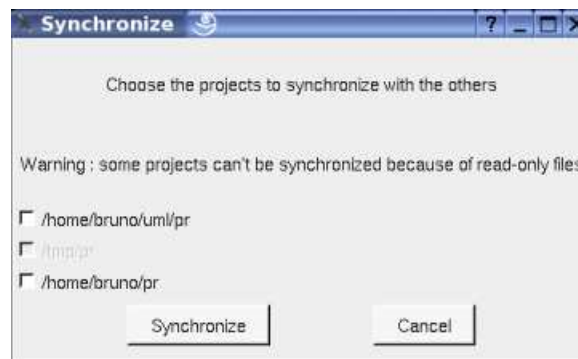
Each time a new image of the project is load, the colors are computed (obviously it is not possible to load several times the sam project image) :



As you can see *incl* is gray indicating it was deleted, and the project itself was modified, at least to remove *incl*.

On the bottom of each image you have the number packages whose need to be updated or deleted. The indication *RO* means that at least one file of the project image is read-only, in this case the image can't be synchronized, even if the file correspond to an up to date package.

When enough images was load, you can ask for the synchronization through the menu *Project* or the shortcut *control-S*. A dialog appears :

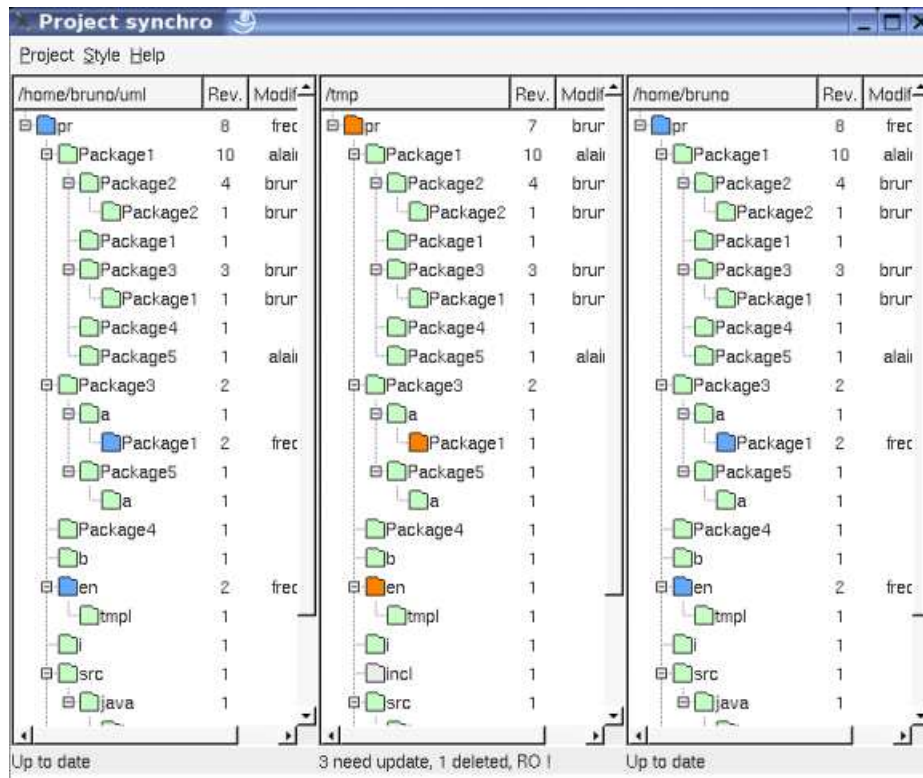


As you can see only two of the three images may be synchronized because of read only files in the second image. However the second image will be used during the synchronization to get the new version of *package1* and the new package *package5*.

After selecting at least one image and hitting on *synchronize* the synchronization the selected images are updated, then a dialog indicating the end of the synchronization appears :



When you confirm the execution of *Project Synchro* is done. Supposing the synchronization done for the first and third image, restarting *Project Synchro* this the same images, one obtains :



Obviously the first and third images have the same aspect and don't contain old packages.

As I said it is possible to start the tool giving the project images in parameter, for instance under Linux :

```
projectControl ~bruno/uml/pr/pr.prj /tmp/pr/pr.prj ~bruno/pr/pr.prj
```

Previous : [Project control](#)

Next : [Multi user considerations](#)

# Multi users considerations

BOUML attributes an identifier to the main objects to resolve the references to the not already read objects when you load a project. This identifier is established during the object creation and will not change later, to allow several users to work simultaneously on the same project, this object identifier contains the identifier of the user creating the object. This explains why BOUML asks you to have an own identifier between 2 and 127 defined through the environment variable **BOUML\_ID** (see [Define an environment variable](#)).

These identifiers are also used to name the files memorizing a project, refer to [Project files](#). The *plug-out* API defines the *UmlBaseItem::supportFile* to know whose file(s) support a given item.

Notes : When it loads a project, BOUML checks the write permission of the file done through the OS or the applicative permission indicated inside the file. In the *browser* the writable objects are written in **bold**. When you load a project, this one must be consistent, else the references to the unknown items are removed and these unknown items are considered deleted.

When you are several working on the same project you have to choose between two ways :

## To share the same project files

In this case all the users work on the same project files, this means that no one has a copy of the project files. This is possible because when you do a save only the modified files are written. However this supposes at most one user may modify a given project; modifications made by several will quickly be done.

There are two ways to allow at most one user to modify a file : to use the write permission set using your OS (difficult under Windows) with the help of *File Control*, or to use *Project Control*.

### File Control

[File control](#) is a *plug-out* mainly dedicated to be an interface between BOUML and a file control like Clearcase, CVS or Subversion etc ..., however to use a file control may affect the write permission of the files.

### Project control

[Project control](#) is a tool managing an applicative write permission optionally memorized in the project files associated to the *packages*. This allows to set the write permission independently of the OS you use.

## To use project images

In this case each user has an own copy of the project, and sometimes all the modifications done in parallel are grouped. You may adopt two strategies : to allow any user to modify any part of the project, in this case you will have to merge the modification yourself, or to allow only one user to modify a given project file.

Even if the project files are ASCII files, it may be difficult to merge several versions of the same file, I know several companies use this way, but it is possible to not have to merge the modifications using the write protection through [File control](#) or [Project control](#) and to resynchronize the model using [Project Synchro](#).

### To work without merge

Let's suppose Bruno, Olivier and Annie work on the same project and decide to use *Project Control* and *Project Synchro*.

At the beginning a unique image of the project exists. If the project already exists, using *Project Control* you give the write access of the package to at most one person. If you start from scratch with an empty project, immediately create at least a sub-package for each, this may be changed later. Because the project itself is used to memorize *generation settings* and other global information, I propose you to decide immediately what are the settings and to protect the package supporting the project, this means it is not owned by Bruno, Olivier or Annie.

When the write accesses are set, duplicate the project files (in fact the directory of the project) to have three images, and perhaps a fourth image placed under a file controller. Bruno, Olivier and Annie work each on its private image and don't change the write accesses, except to loose the write access, or to get the write access to a protected package(s) not owned by the others since synchronization.

Sometimes, for instance because Annie defines a class used by Olivier, you have to synchronize the parallel developments. In this case using *File Synchro*, Annie, Olivier and maybe Bruno get the modifications done by the others. When the project is synchronized it is also possible to change the write access from a user to another one, obviously this must be made on all the images, or on one image and copying it in the others.

Obviously like in any development, when an element is removed from the model by Bruno, for instance a class, he has to ask for

Olivier and Annie except if this element was not yet delivered to them. To move an element inside the browser is not a problem element is referenced through an internal key, not through its location. In a similar way an element may be renamed because the doesn't contains the name, but this may have impact in the body of the operations where the renaming is not taken into account BOUML itself.

---

Previous : [Project synchro](#)

Next : [Project files](#)

# Project files

A project is saved in an own directory through several files, all of them are text files, this may help you to do some searches or changes directly in the files (after a backup !), or of course to use a software configuration management. The files are :

- *<project name>.prj* : contains the top level package and project level data, for instance the generation directories.
- *<number>* : contains the data of a package (an *uml* package, not a *java* package !) and its sub-elements, the package's name is the second saved data, after a format indication. The number is the package's identifier (see [Multi users considerations](#)).
- *<number>.diagram*  
: contains a diagram definition. For time and memory save reason, BOUML does not load a diagram until it is edited. The number is the diagram's identifier (see [Multi users considerations](#)).
- *<number>.bodies*  
: contains the C++ and Java bodies of all the operations of a class. For time and memory save reason, BOUML does not load body until it is edited. The number is the class's identifier (see [Multi users considerations](#)).
- *generation\_settings* : contains the *generation settings*, this allows you to quickly set them after creating a new project copying the *generation\_settings* file of an old project.
- *tools* : contains the [plugins-outs](#) declaration, this allows you to quickly set the tools list after creating a new project copying the *tools* file of an old project.
- *cpp\_includes*  
: contains the C++ include & using specifications for the non defined types, used by the C++ generator and set through the [C++ generation](#) settings.
- *java\_imports*  
: contains the Java imports associated to the non defined types, not already used by the Java generator and set through the [Java generation](#) settings.
- *idl\_includes*  
: contains the Idl imports includes to the non defined types, not already used by the Idl generator and set through the [Idl generation](#) settings.
- *<user id>.session* : contains the session state (opened diagrams, browser view ...), set when the project is closed
- *<number>\_<user id>.d* : contains the new definition of a diagram until a save.
- *<number>\_<user id>.b* : contains the new definitions of a class's operations until a save.
- *<user\_id>.lock* : a directory created when you load the project, and deleted when you close the project (see [Multi users considerations](#)).

Note : in a *plug-out* the *supportFile* operation defined on *UmlBaseItem* then applicable on all the browser items returns the absolute pathname of the file memorizing the item. In general an item is saved in a file associated to its package, the exceptions are :

- the body of the operations of a class which definition contains the keyword '*body*'. This path name is returned when you apply *supportFile* on any operation of the class.
- the drawing of the diagrams, this path name is returned when you apply *supportFile* on the diagram.
- the configuration of the tools edited throw the entry '*Tools Settings*' of the menu [Tools](#) is saved in the file 'tools'
- the configuration of the *#include* and *using* forms associated to the external type and edited through the last C++ tab of the 'Generation Settings' dialog is saved 'cpp\_includes'
- the configuration of the *imports* forms associated to the external types and edited through the last Java tab of the 'Generation Settings' dialog is saved 'java\_imports'
- the configuration of the *#include* forms associated to the external type and edited through the last Idl tab of the 'Generation Settings' dialog is saved in 'idl\_includes'

---

Previous : [Multi users considerations](#)

Next : [Troubleshootings](#)



# Troubleshootings

## The plug-outs start but do nothing

The exchanges between BOUML and each *plug-out* are supported by a TCP/IP socket. When you ask BOUML to start a *plug-out*, it search for a free port number starting at 1024 on *localhost* (127.0.0.1), this port is given in argument to the called *plug-out*. In case you have a *firewall* it may be necessary to configure it to allow BOUML and the *plug-outs* to use a socket.

## A modification made by a plug-out is not see by the other ones

To reduce the number of exchanges with BOUML and run faster, a *plug-out* does not ask for BOUML each time you want to get a value. The *system* classes read and memorize all the data of a given item the first time you access to one of them (with the restriction allowed by [WITHCPP](#), [WITHJAVA](#) and [WITHIDL](#)). This is transparent for you until two *plug-outs* access to the same data, when one *plug-out* modify a value already read by the other, the second *plug-out* does not see the new value until it *unload* (an operation defined on *UmlBaseItem*) the object, then ask for the value.

## A class is defined with operations, attributes ... but the C++/Java/Idl generator doesn't generate it

The class doesn't have an associated artifact, or the class/artifact is not defined for the language. Note that a definition (set through the generation settings) may be done at the creation for each language if you ask for it through the [Languages menu](#)

## I cannot edit a browser item, the edit menu entry or button does not appear :

you do not have the write permission for the associated files or the item is a *system* item supporting the *plug-out* API. It is also not possible to edit/delete a class when an other item is already edited.

## I use a multiple monitor (side by side) configuration and the dialogs are placed partially on each

The [environment variable](#) BOUML\_LIMIT\_DESKTOP is for you !

This optional variable allows to specify a sub-part of all the desktop, its value must be *x0,y0,x1,y1* without any spaces, where *x0,y0* specify the top-left corner and *x1,y1* the bottom-right corner. These limits are taken into account only for the [initial](#) size and position of the BOUML's windows and when a dialog is opened, after that you are able to go out of these limits up to the true desktop limits.

## After editing a description or other part the font is modified or several characters are replaced by a space

During the edition the characters encoded with a 16 bits unicode character set, but out of the editions the characters are saved on only 8 bits. When you use non ISO\_8859-1 (latin1) characters you must specify how to do the convers through the environment variable BOUML\_CHARSET.

When this variable is undefined or has an empty value, the character set is ISO\_8859-1. If its value is not a kn character set's name, a dialog appears when you start Bouml giving the available names. So, to know the possil character sets, set BOUML\_CHARSET to wrong for instance and call Bouml.

Examples (the list depends on the release of Qt) :

- ISO\_8859-1 (latin1) for English, French, Italien, German ...
- ISO\_8859-7 for Greek
- KOI8-R or KOI8-RU or CP\_1251 to allow Cyrillic
- JIS7 for Japan



- GB2312, GBK or GB18030 for Chinese

The specified encoding is used for :

- the description of every elements,
- the user properties and associate values of every elements,
- the stereotype of every elements,
- the definition of the *artifacts*, *classes*, *operations*, *attributes*, *relations* and *extra members*,
- the *state action* behavior
- the *state entry / exit / do behavior*
- the *transition trigger / guard / do behavior*
- the *attribute's* value in an *object diagram*
- the *notes* and *text* in the diagrams

It is not used else where, for instance you can't use non ISO\_8859-1 characters to name the elements or in a message of a sequence or collaboration diagram.

The specified encoding is followed in the project's files and for the exchanges with the plug-out, this means for instance that Cyrillic characters may be used in the comments and are generated unchanged by the code generators.

---

Previous : [Project files](#)

Next : [GNU public license](#)

# GNU public license

The BOUML Toolkit is Copyleft 2004-2008 Bruno PAGES

You may use, distribute and copy the BOUML Toolkit under the terms of GNU General Public License version 2, which is displayed below.

---

## GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

### Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software

Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

## GNU GENERAL PUBLIC LICENSE

## TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in

the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under the conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on

a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

- a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any additional restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot

distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this license incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of our free software and of promoting the sharing and reuse of software generally.

## NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

## END OF TERMS AND CONDITIONS

Previous : [Troubleshooting](#)

Next : [Miscellaneous](#)

# Define an environment variable

To define an environment variable under **Windows** click on :

start > settings > control panel > system > advanced > environment variables

To define an environment variable under **Linux** (t)csh, place the following line in your .cshrc

```
setenv <var> <value>
```

To defined an environment under **Linux** bash, place the following line in your .bashrc

```
export <var>=<value>
```

---

Previous : [GNU public license](#)