

# **Safety Designer User's Guide BPA SD9 Delivery 8 for V5R20**



---

# Contents

<b>CONTENTS .....</b>	<b>3</b>
<b>INTRODUCTION .....</b>	<b>5</b>
<b>GETTING STARTED.....</b>	<b>6</b>
SYSTEM REQUIREMENTS .....	6
INSTALLING .....	6
LAUNCHING SAFETY DESIGNER.....	6
OPENING A WORKSPACE .....	7
SWITCHING TO ANOTHER WORKSPACE .....	10
<b>ORGANIZING A MODEL .....</b>	<b>12</b>
GENERAL .....	12
WORKING WITH FAMILIES.....	12
WORKING WITH PROJECTS .....	16
<b>CREATING A MODEL .....</b>	<b>17</b>
WORKING WITH MODEL CLASSES .....	17
WORKING WITH SYSTEMS, EQUIPMENTS AND COMPONENTS .....	49
WORKING WITH OPERATORS .....	59
WORKING WITH TYPES .....	65
SETTING LINK COLORS .....	70
SEARCHING FOR A MODEL.....	72
SETTING PAGE LAYOUT.....	74
PRINTING A MODEL .....	74
<b>EXPORTING &amp; IMPORTING .....</b>	<b>77</b>
EXPORTING .....	77
IMPORTING .....	77
<b>MANAGING PLUG-INS .....</b>	<b>80</b>
INTRODUCTION .....	80
GENERAL .....	80
SETTING PLUG-INS PREFERENCES.....	81
EDITING ACTIONS.....	83
INSERTING PLUG-IN ACTIONS IN MENUS.....	85
<b>SETTING PREFERENCES .....</b>	<b>89</b>
OPENING THE PREFERENCES PANEL .....	89
SETTING ENVIRONMENT PREFERENCES .....	89
SETTING DESKTOP PREFERENCES .....	90
SETTING TOOLBARS PREFERENCES .....	90
SETTING EDITION PREFERENCES .....	91
SETTING INPUT/OUTPUT PREFERENCES .....	93
SETTING SIMULATION PREFERENCES .....	94
SETTING PLUG-IN PREFERENCES .....	94
<b>INTERFACE DESCRIPTION.....</b>	<b>97</b>
MENU BAR .....	97
TOOLBARS .....	100
<b>GLOSSARY .....</b>	<b>102</b>
<b>APPENDIX: ALTARICA LANGUAGE SPECIFICATION .....</b>	<b>103</b>
GENERAL .....	103
LEXICAL CONSIDERATIONS .....	103
EXPRESSIONS .....	104
TRANSITIONS .....	106
ASSERTIONS .....	107
<b>APPENDIX: USER MANAGEMENT .....</b>	<b>109</b>
ADMINISTRATION WINDOW .....	109
USER MANAGEMENT .....	109

WORKGROUP MANAGEMENT.....	111
PRINTING ADMINISTRATION INFORMATION .....	114
MANAGING PERMISSIONS.....	114
<b>APPENDIX: PROBABILITY LAWS IN ARALIA FORMAT .....</b>	<b>122</b>

---

# Introduction

This user's guide describes Safety Designer.

The main features of Safety Designer are:

- Describing the behavior of systems, under nominal condition as well as under failure conditions thanks to the AltaRica language,
- Organizing models in libraries for reuse,
- Simulating models using the integrated AltaRica simulator,
- Generating fault trees by selecting an unexpected event from the model,
- Generating failure scenarios leading to a selected unexpected event.

# Getting Started

## System Requirements

### Windows

- Operating system version: Windows XP 32bits
- Intel Pentium or equivalent processor
- 50MB of available hard disk space,
- 512MB of RAM (1Gb recommended)

## Installing



Please refer to the installation manual.

## Launching Safety Designer

1. Double-click on the shortcut added on the desktop or on the shortcut in the start menu. The Splash screen appears.



*Figure 1: Splash screen*

-  By default Safety Designer opens the most recently used workspace.
-  At the very first launch, no workspace is known, and the selection workspace panel opens up; see Opening a Workspace for further information.

---

# Workspace Management

Safety Designer allows users to work with several workspaces and to switch easily between them. Workspaces are stored in MS Access© or Oracle© 9 databases.

## Opening a Workspace

A workspace is a place where models are stored and versioned. A workspace can be based on:


- An MS Access database;
- An Oracle database;
- An ODBC data source.

The Workspace selection panel allows user to specify which workspace to open and to create a new one.



*Figure 2: Workspace selection panel*

### Selecting a workspace based on an MS Access database

1. Select “Access” in the list of database types.
2. Enter path of the Access database file in the edition area or click on button Browse  to search it from an explorer.

 The file extension of an MS Access database is “.mdb”.

3. Click on “Validate” to open the Workspace connection panel.

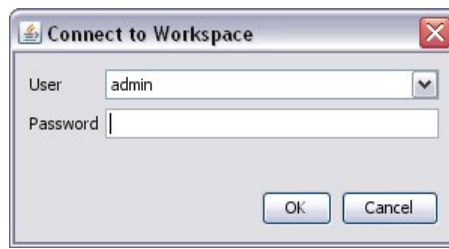


Figure 3: Workspace connection panel

4. Set the login and password, click on “OK”

## Creating a new workspace based on an MS Access database

To create a new, empty workspace based on an MS Access database:

1. From the Workspace selection panel
2. Select “Access” in the list of database types
3. Click on the “Create new database” button
4. Set the destination directory and a file name to be used for the new MS Access database (Figure 4)

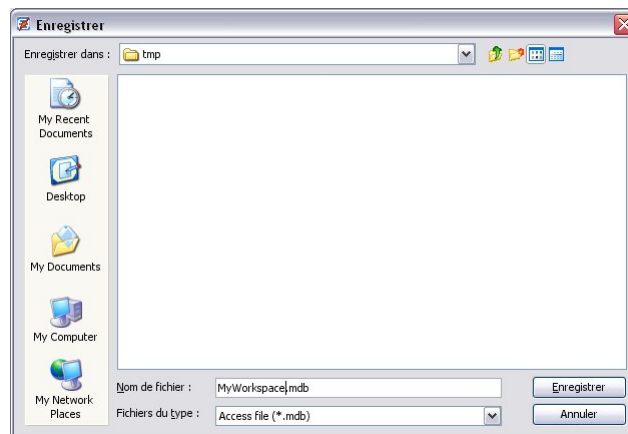


Figure 4: Creation of a new Access database

5. Click on Save (Figure 4) to validate the creation of the new database
6. Click on “Validate” to open the new workspace



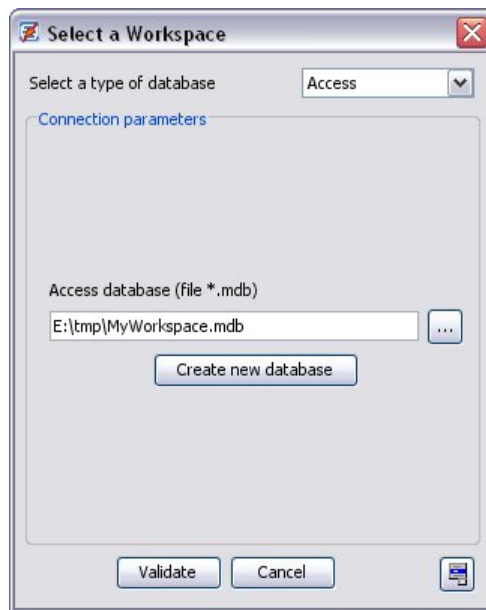



Figure 5: Validating creation of a new workspace

To complete the workspace switch, the Workspace connection panel opens, asking for the appropriated login and password.

## Selecting a workspace based on an Oracle database

 The setup of an Oracle database is not explained in this manual

1. Select “Oracle” in the list of database types; the Oracle database connection parameters tab shows up.

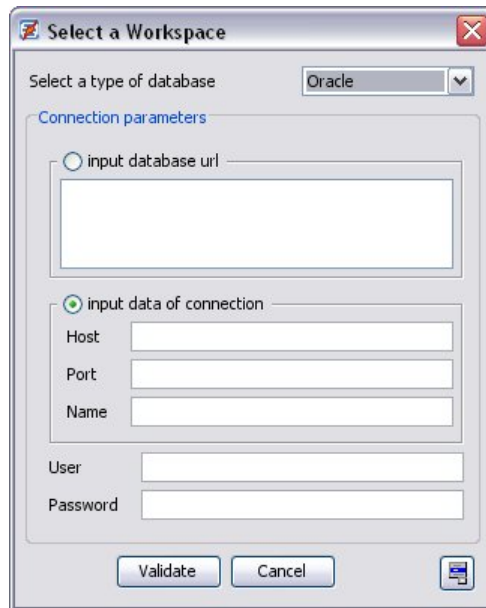



Figure 6: Oracle database connection parameters tab

2. Fill in the connection parameters to the Oracle database appropriately; ask your Oracle database administrator for them.

3. Click on “Validate” to open the Figure 3: Workspace connection panel.
4. Set the login and password, click “OK”.

## Selecting a workspace stored in an ODBC data source

 The creation of an ODBC data source is not explained in this manual.

1. Select “ODBC” in the list of database types; the ODBC data source parameters tab shows up.
2. Enter the name of the data source in the corresponding area.

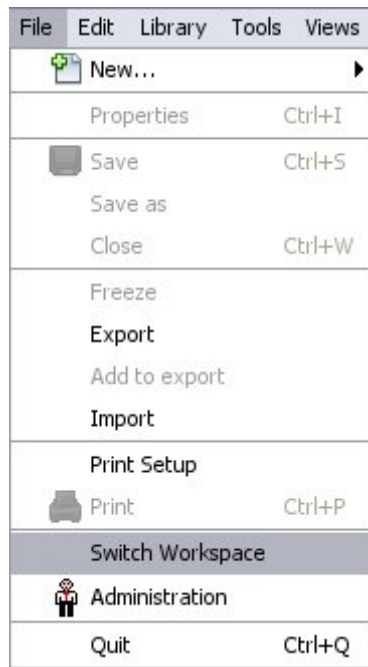


*Figure 7: ODBC data source parameters tab*

3. Click on “Validate” to open the Figure 3: Workspace connection panel.
4. Set the login and password, click “OK”.

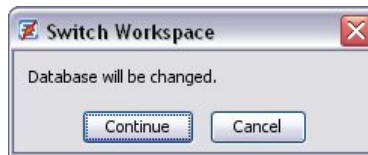
## Switching to another Workspace

1. Select “File” -> “Switch Workspace”.



*Figure 8: File / Switch Workspace menu*

2. Click on “Continue” in the Workspace switch confirmation dialog.



*Figure 9: Workspace switch confirmation dialog*

The Figure 2: Workspace selection panel opens up; see the Opening a Workspace section for further information.

# Organizing a Model

## General

Models can be organized in folders and sub-folders. Two kinds of folders are available:

- Families, dedicated to the organization of model classes, components, equipments, operators and types.
- Projects, dedicated to the organization of systems.

## Working with Families

Models can be organized at will in hierarchies of families. The Figure 10 presents an example of families and sub families of model classes.

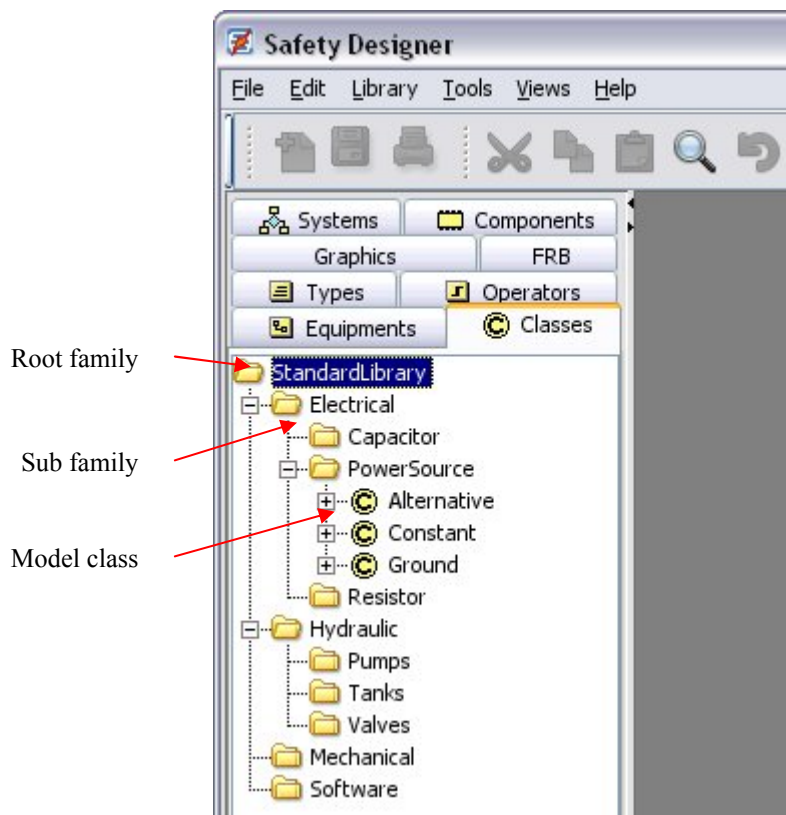


Figure 10: Example of component organization

## Creating a new root family

To create a new root family of model classes, equipments or components, operators or types in the current workspace:

1. Select corresponding tab in the model tree
2. Right-click in an empty area (no existing family must be selected); the Figure 11: Model edition and organization contextual menu opens

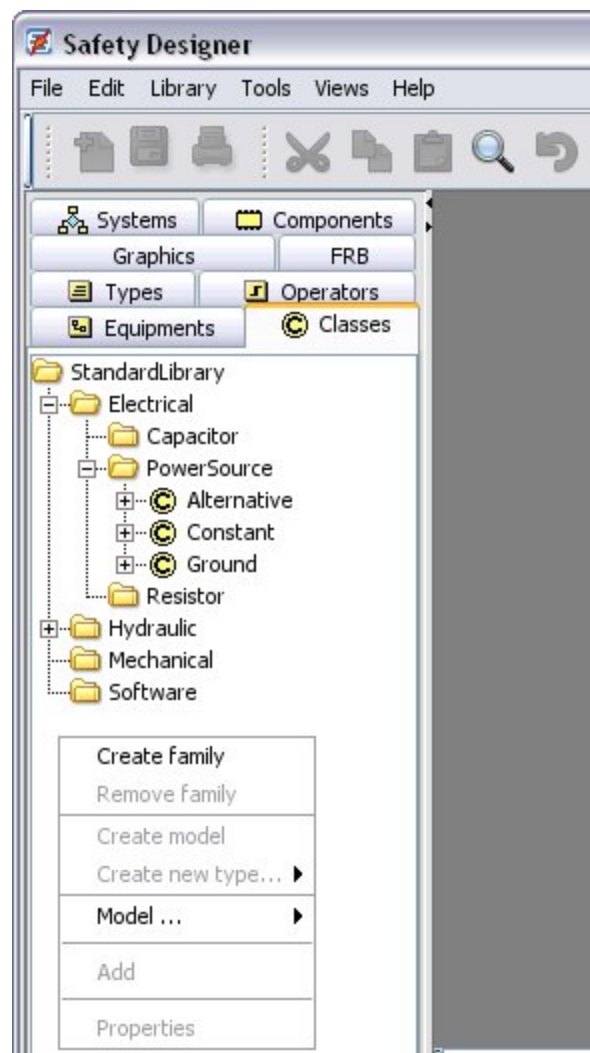



Figure 11: Model edition and organization contextual menu

3. Select “Create family”; the Family creation panel opens.

 Select “Library” -> “Create family” to access the same functionality.

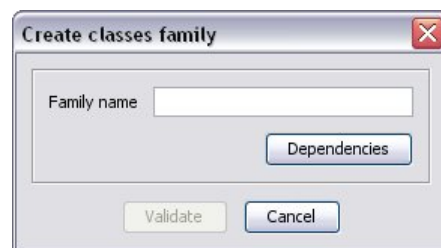


Figure 12: Family creation panel

4. Enter the name of the new family to be created

5. If the owner belongs to several workgroups, he has to specify the owning group of the family; click on “Dependencies” to select the group. Click on “Close” to validate.



Figure 13 : Selection of dependencies

6. Click on “Validate” to accept the creation of the new family.

A new root family is created in the workspace.

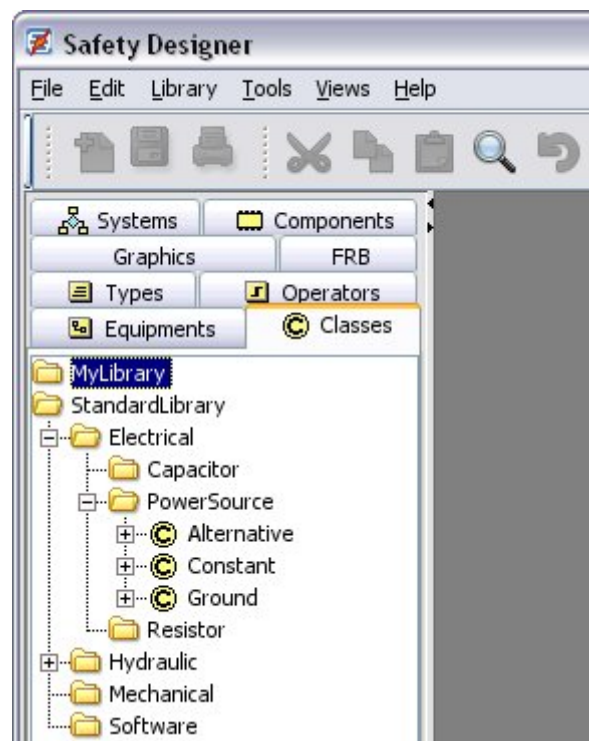


Figure 14: A new root family

## Creating a new sub-family

To create a new sub family of model classes, equipments or components, operators or types in the current workspace:

1. Select corresponding tab in the model tree
2. Select the existing family in which the new sub-family must be created
3. Right click on the existing family to display the Figure 11: Model edition and organization contextual menu
4. Proceed as described in Creating a new root family

A new sub-family is created under the selected existing family.

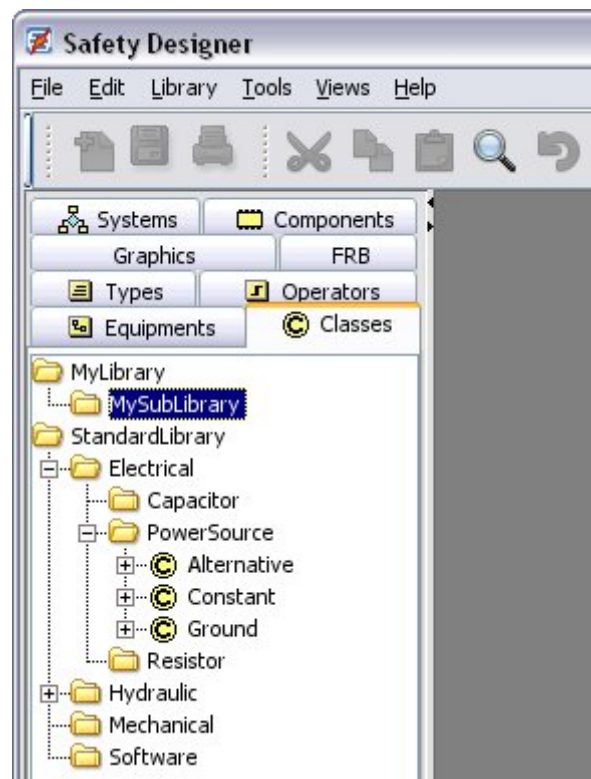


Figure 15: A new sub-family

## Removing a family

✎ It is not possible to remove a family that is not empty; first move or remove any contained sub-families and models.

1. Click on the family to remove to select it
2. Right click on the family to remove to display the Model edition and organization contextual menu
3. Click on the “Remove family” item; the Family removal confirmation dialog opens.

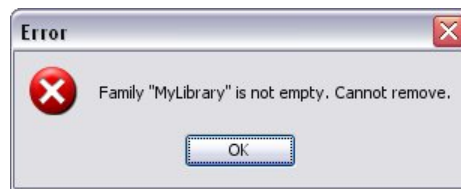


Figure 16: Family removal confirmation dialog

4. Click on “Continue” to confirm the removal of the family


The family or the project is removed from the workspace.

✎ When attempting to remove a non-empty family, an error message is displayed (Error when attempting to remove a non-empty family).



*Figure 17: Error when attempting to remove a non-empty family*

## Working with Projects

 This feature is only relevant when using the system models

A project is a kind of family that is dedicated to the organization of system models (system tab). Working with projects is completely identical to working with families, only the wording changes.



# Creating a Model

## Working with Model Classes

A model class corresponds to an AltaRica “node”. An AltaRica “node” is made of several clauses, which can be grouped as follow:

- A declarative part (“flow”, “state”, “event”, “sub”, “init” and “extern” clauses);
- A behavioral part (“trans” and “assert” clauses)

### Accessing an Existing Model Class

#### Opening an Existing Model Class

1. Access the Model class tab (Figure 19: Model Class Editor)
2. Select a model class in the family tree (“BackupCoolingSystem/BackupCoolingSystem”)
3. Double-click on a version (“Version 1”); the Model Class Editor opens.

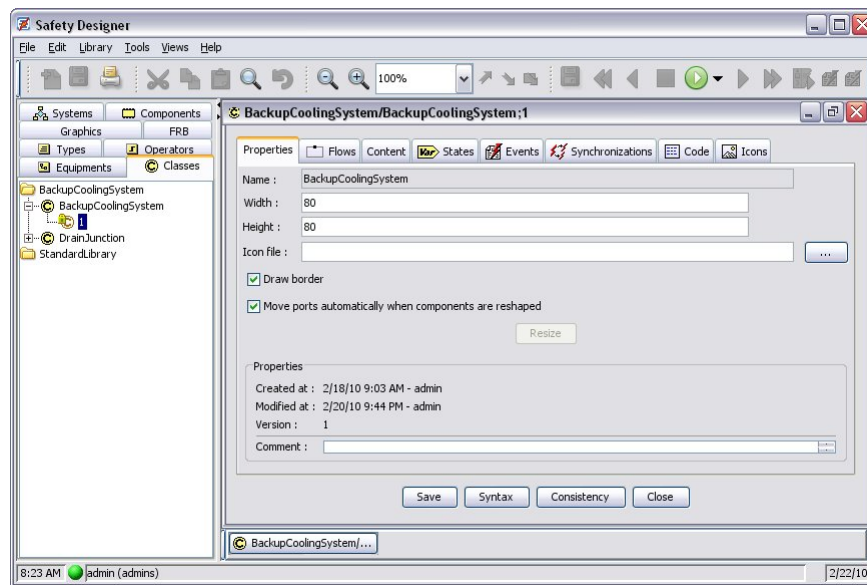


Figure 19: Model Class Editor

- ✎ When a model is opened for edition, it is automatically locked so that several users cannot edit the same model at the same time.
- ✎ A model class can remain locked after a system crash; access its properties to remove the lock manually


### Checking a Model Class for Consistency

1. Click on the “Syntax” and “Consistency” buttons at the bottom of the Model Class Editor

### Saving an Existing Model Class

1. Click on the “Save” button at the bottom of the Model Class Editor.

## Closing an Existing Model Class

1. Click on the “Close” button at the bottom of the Model Class Editor, or on “” on its top right corner
2. If the model class has been modified, a confirmation dialog opens

Editing Flow Variables, Editing State Variables, Editing Events, Editing content are convenient ways to edit the declarative part. The behavioral part is written by hand in a dedicated AltaRica editor, as described in the Editing AltaRica code section.

## Creating a new Model Class

1. Click on the Class tab of the model browser
2. Right click on the family where to add a new model class to open the Model edition and organization contextual menu
3. Select “Create model”; the Model creation dialog opens up.

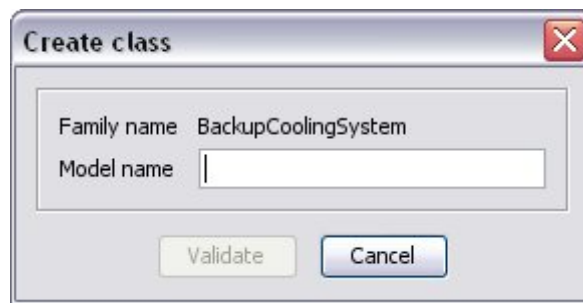



Figure 18: Model creation dialog

4. Edit the name of a new model, press enter.

 The name of a model class must respect the constraints on identifiers described in the Appendix: AltaRica Language Specification.

## Accessing an Existing Model Class

### Opening an Existing Model Class

4. Access the Model class tab (Figure 19: Model Class Editor)
5. Select a model class in the family tree (“BackupCoolingSystem/BackupCoolingSystem”)
6. Double-click on a version (“Version 1”); the Model Class Editor opens.

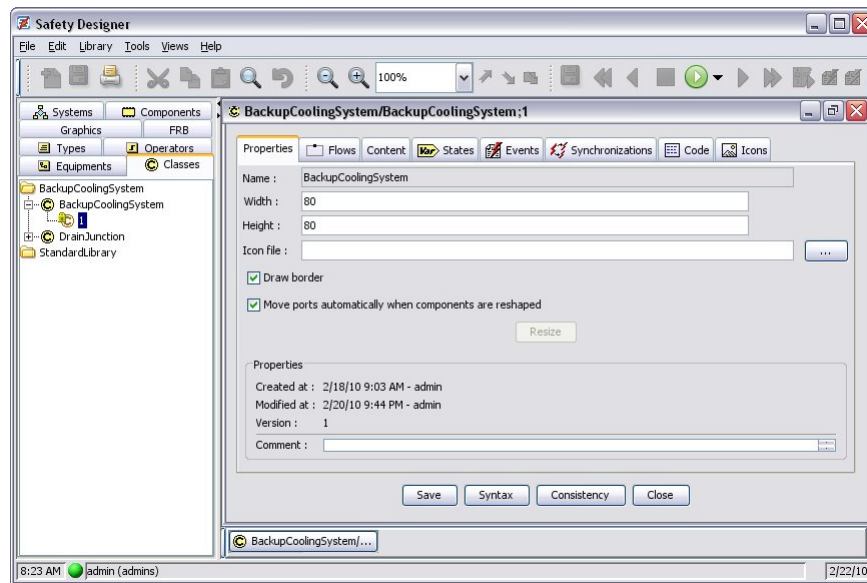


Figure 19: Model Class Editor

- ✎ When a model is opened for edition, it is automatically locked so that several users cannot edit the same model at the same time.
- ✎ A model class can remain locked after a system crash; access its properties to remove the lock manually


## Checking a Model Class for Consistency

2. Click on the “Syntax” and “Consistency” buttons at the bottom of the Model Class Editor

## Saving an Existing Model Class

2. Click on the “Save” button at the bottom of the Model Class Editor.

## Closing an Existing Model Class

3. Click on the “Close” button at the bottom of the Model Class Editor, or on “” on its top right corner
4. If the model class has been modified, a confirmation dialog opens

## Editing Flow Variables

- ✎ Refer to Appendix: AltaRica Language Specification for more information on the semantics of flow variables

Flow variables of a model class can be created, removed and edited with the Figure 20: Flow variables edition tab.

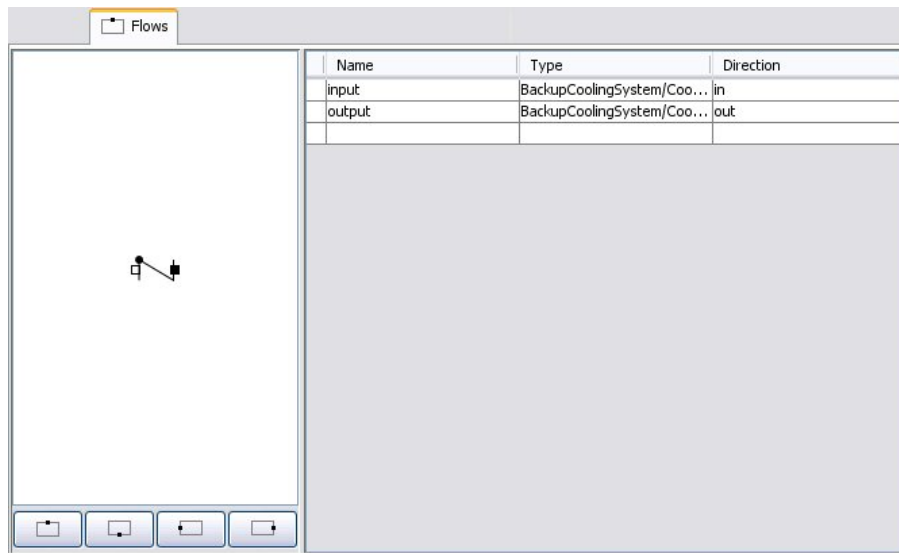




Figure 20: Flow variables edition tab

## Creating a new flow variable

1. Click on the “Name” column of the last blank line of the flow table
2. Edit the name of the flow variable to be created (see Modifying the name of an existing flow variable below); entering a valid name immediately creates a new flow variable with a type defaulting to “bool” and a direction defaulting to “in”
3. Set the type of the flow variable (see Modifying the type of an existing flow variable)
4. Select the direction of the flow (see Modifying the direction of an existing flow variable)

Once a flow variable is created, an associated port is displayed on the graphic panel.

-  An input flow variable is identified by a white rectangle; an output variable is identified by a black rectangle
-  By definition, internal variables are private and cannot be linked to variables of other elements; hence no port is created for them.


## Modifying the position of a port

1. Select the port to be moved:
  - Select the associated flow variable in the flow table, or
  - Click on the shape of the port
2. Move the port:
  - Use the mouse to drag the shape of the port, or
  - Click on the top center button, or
  - Click on the bottom center button, or
  - Click on the left middle button, or
  - Click on the right middle button.


## Editing an existing flow variable

### Modifying the name of an existing flow variable

1. Click on the “Name” column of the row describing the flow variable to be edited
2. Enter a new name
3. Press “Enter”


- 
-  The name of a flow variable must respect the constraints on variable identifiers described in the Appendix: AltaRica Language Specification.

## Modifying the type of an existing flow variable

1. Click on the “Type” column of the row describing the flow variable to be edited
  2. Set the new type, either by writing it directly or by selecting it among the proposed ones:
    - “bool” for the Boolean type (possible values are “true” and “false”)
    - “int” for the integer type
    - “float” for the real type
    - “Range...” for opening a wizard to write an integer range type
    - “Predefined...” for selecting a predefined type.
-  Enumerated types can only be written manually, by listing the possible values separated by commas (whitespaces are forbidden), e.g.: “null,low,high”.

## Modifying the direction of an existing flow variable


1. Click on the “Direction” column of the row describing the flow variable to be edited
2. Select the direction of the flow (see Modifying the direction of an existing flow variable):
  - “in” for input flows
  - “out” for output flows
  - “local” for internal flows

-  Internal flow variables are typically used to store intermediate results in complex assertions

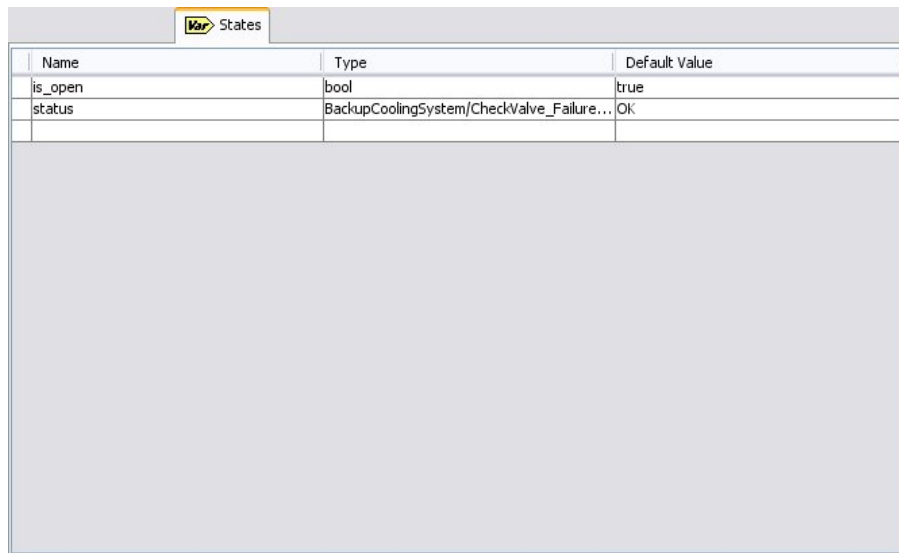
## Deleting an existing flow variable

1. Select the flow variable to be deleted:
  - Select the row describing it in the flow table, or
  - Click on the shape of its associated port if any
2. Press the “Delete” key.

## Editing State Variables

-  Refer to Appendix: AltaRica Language Specification for more information on the semantics of state variables

State variables of a model class can be created, removed and edited with the Figure 21: State variables edition tab.



Name	Type	Default Value
is_open	bool	true
status	BackupCoolingSystem/CheckValve_Failure...	OK

Figure 21: State variables edition tab


## Creating a new state variable

1. Click on the “Name” column of the last blank line of the state table
2. Edit the name of the state variable to be created (see Modifying the name of an existing); entering a valid name immediately creates a new state variable with a type defaulting to “bool” and a default value defaulting to “false”
3. Set the type of the state variable (see Modifying the type of an existing state variable)
4. Set the default value of the state variable (see Modifying the default value of an existing state variable)

## Editing an existing state variable


### Modifying the name of an existing state variable

1. Click on the “Name” column of the row describing the state variable to be edited
2. Enter a new name
3. Press “Enter”

 The name of a state variable must respect the constraints on variable identifiers described in the Appendix: AltaRica Language Specification.



### Modifying the type of an existing state variable

1. Click on the “Type” column of the row describing the state variable to be edited
2. Set the new type, either by writing it directly or by selecting it among the proposed ones:
  - “bool” for the Boolean type (possible values are “true” and “false”)
  - “int” for the integer type
  - “float” for the real type
  - “Range...” for opening a wizard to write an integer range type
  - “Predefined...” for selecting a predefined type.

 Enumerated types can only be written manually, by listing the possible values separated by commas (whitespaces are forbidden), e.g.: “null,low,high”.

### Modifying the default value of an existing state variable


1. Click on the “Default value” column of the row describing the state variable to be edited
2. Set the default value, either by writing it directly or by selecting it among the suggested ones

-  The suggestion panel only appears for variables with finite domains (Boolean, enumerations, integer ranges, and possibly predefined types).
-  Default values are used at the beginning of a simulation (in the general sense of the term, i.e. including fault tree generation, event sequence generation, etc.) to initiate state variables when no initial configuration has been specified.

## Deleting an existing state variable

1. Select the row describing the state variable to be removed in the state table
2. Press the “Delete” key.

## Editing Events

-  Refer to Appendix: AltaRica Language Specification for more information on the semantics of events

Events of a model class can be created, removed and edited with the Event edition tab.

Events			
Name	Law	Comment	Attributes
fail_to_open	exponential(0.0010)	The valve cannot be opened	
fail_to_reseat	exponential(0.0010)	The valve cannot be closed	
open	Dirac(0.0)	Do open the valve!	
close	Dirac(0.0)	Do close the valve!	

Figure 22: Event edition tab


## Creating a new event

1. Click on the “Name” column of the last blank line of the event table
2. Edit the name of the event to be created (see ); entering a valid name immediately creates a new event with no probability distribution, no comment and no attributes
3. Set the probability distribution of the event (see)
4. Optionally fill a comment (see)
5. Optionally add attributes

## Editing an existing event

### Modifying the name of an existing event

1. Click on the “Name” column of the row describing the event to be edited
2. Enter a new name
3. Press “Enter”

 The name of an event must respect the constraints on variable identifiers described in the Appendix: AltaRica Language Specification.

## Modifying the probability distribution of an existing event

1. Click on the “Law” column of the row describing the event to be edited; the Event properties panel opens.

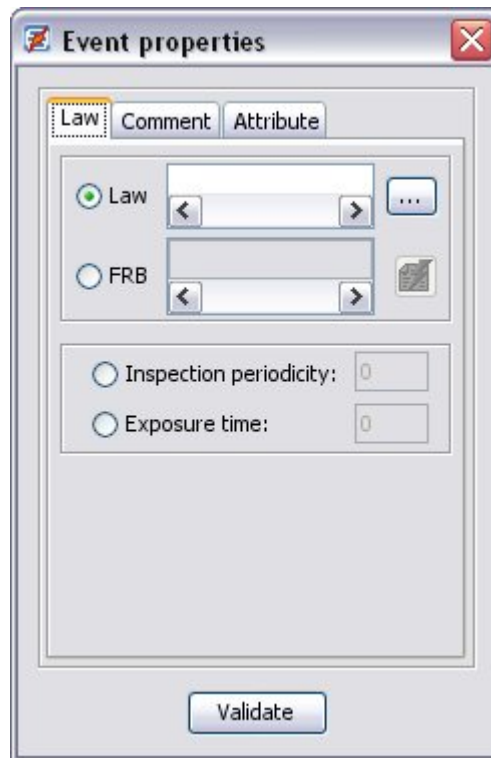



Figure 23: Event properties panel

2. Set the probability distribution of the event, either by writing it by hand or by clicking “...” to open the Probability distribution wizard

The following probability distributions are supported:

Probability Distribution	Syntax and Parameters	Example
<b>Constant</b>	Constant(time)	Constant(0.5)
<b>Exponential</b>		
<b>Gamma (exponential)</b>	GLM(gamma, lambda, mu)	GLM(0.5, 1e-6, 1e-3)
<b>Gamma (asymptotic)</b>	GLM_asymptotic(lambda, mu)	GLM_asymptotic(1e-6, 1e-3)
<b>Weibull</b>	Weibull(alpha, beta)	Weibull(0.5, 1.5)
<b>Periodic test</b>	Periodic_test(lamnda, period, t0)	Periodic_test(1e-3, 10, 0)
<b>Constant Mission Time</b>	CMT(lambda, duration, probability)	CMT(1e-3, 72, 0)
<b>Dirac</b>	Dirac(time)	Dirac(0.0)

 Failure Rate Banks, Inspection periodicity and Exposure time are not described in is manual



---

## Using the probability distribution wizard




*Figure 24: Probability distribution wizard*

The Probability distribution wizard is split into two parts; the first part let the user select a probability distribution among the supported ones, the second part contains fields to fill in the parameters on the selected distribution.

### Modifying the comment of an existing event

1. Click on the "Comment" column of the row describing the event to be edited
2. Enter the comment
3. Press the "Enter" key.

 Comments on events are included in the fault trees generated by Safety Designer when selecting the Aralia file format; they can be seen in fault tree editors supporting the Aralia file format.

### Deleting an existing event

1. Select the row describing the state variable to be removed in the state table
2. Press the "Delete" key.

### Editing content

A model class can compose – instantiate – other model classes. Such constituents can be added, removed and linked with the Content edition tab.

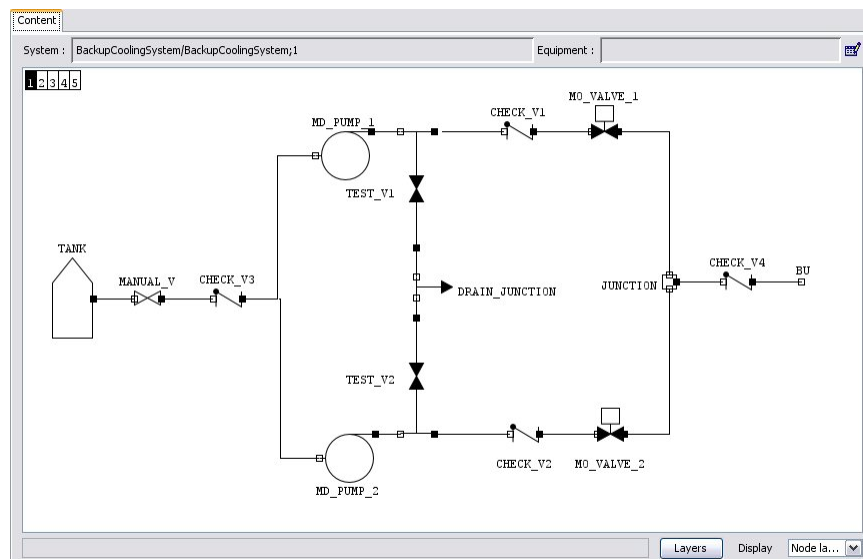


Figure 25: Content edition tab

The Content edition tab consists in:

- A main central frame where the content edition takes place
- A layer manager
- A filter on elements to display

## Inserting a new model class instance

1. In the model class tree, click on the model class and version to instantiate to select it (Figure 26: Selecting a model class)

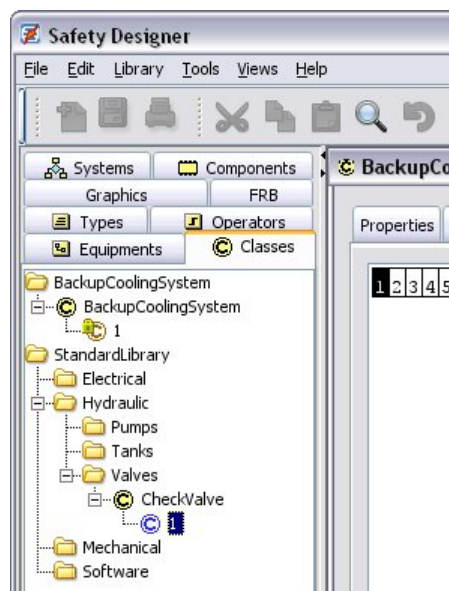


Figure 26: Selecting a model class

2. Do instantiate it:

- drag the selected element and drop it into the main frame (Figure 27: Dragging a model class for instantiation)

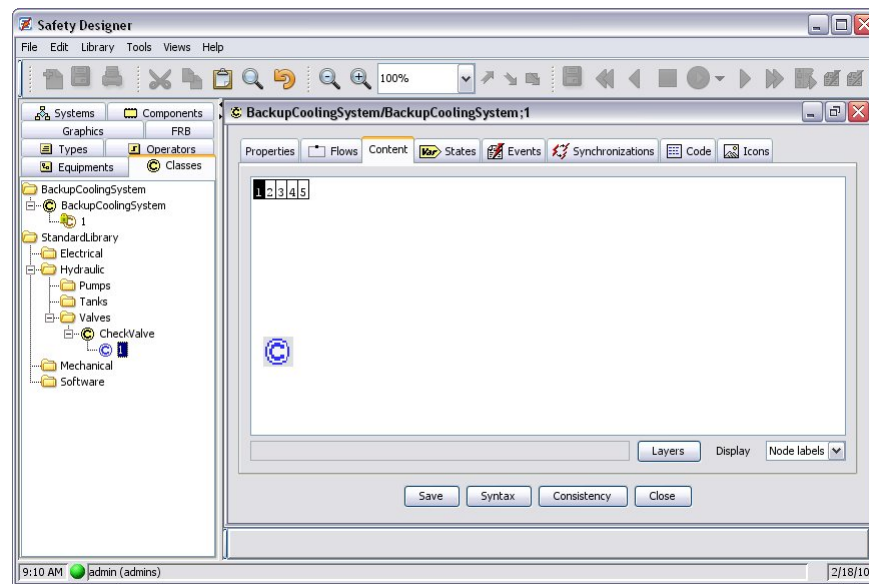


Figure 27: Dragging a model class for instantiation

- ✎ A forbid sign is displayed when trying to drop a model class that cannot be instantiated in the current one (this is relevant when using the system/equipment/component paradigm, as in Figure 28: Cannot drop for instantiation)

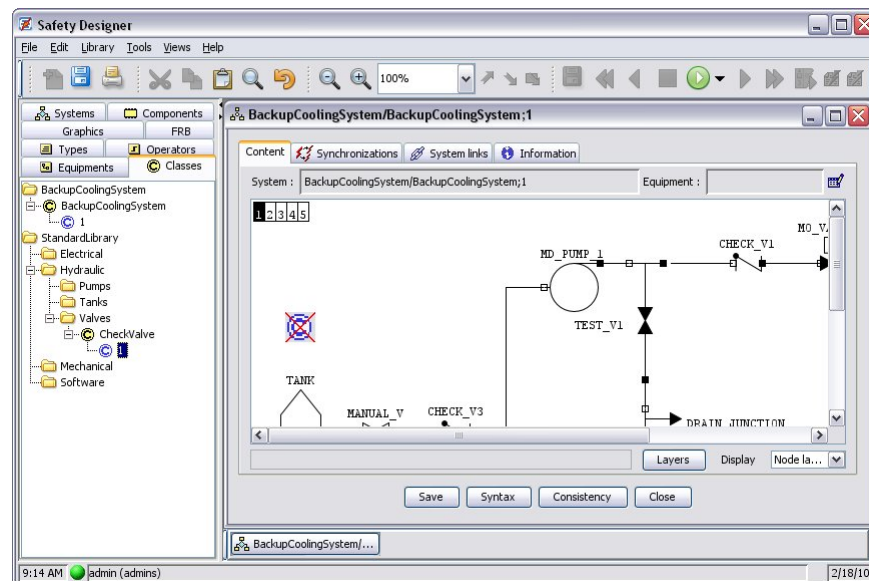


Figure 28: Cannot drop for instantiation

- right-click on the selected element and click “Add” (Figure 29: Instantiating through the "Add" menu item)

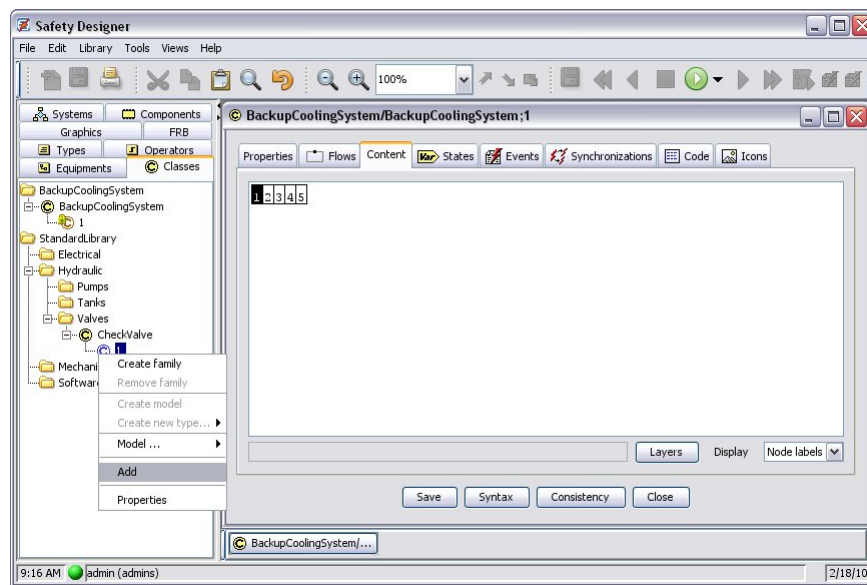


Figure 29: Instantiating through the "Add" menu item

The specified model class is instantiated with a default name (Figure 30: Instantiated model class).

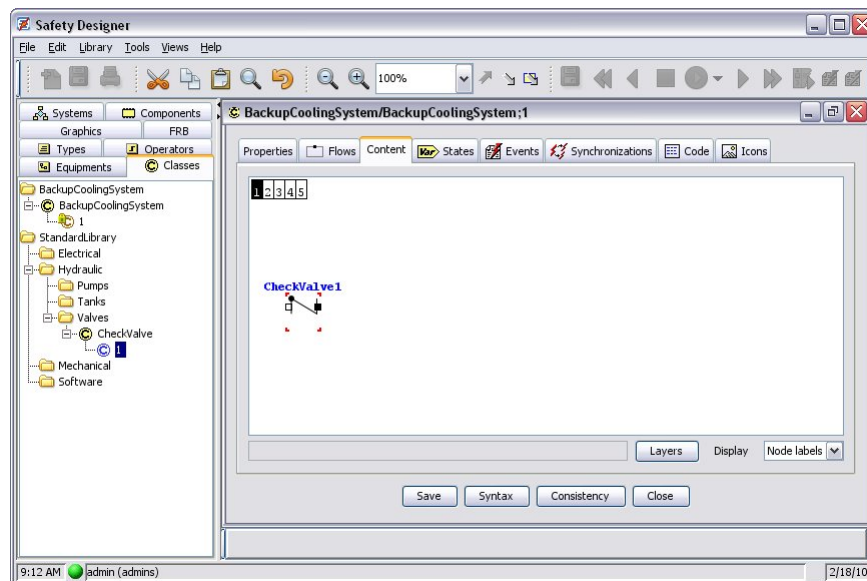


Figure 30: Instantiated model class

## Modifying an existing instance

### Renaming an existing instance

- Double-click on the model instance to rename (Figure 31: Renaming an instance in content frame) ; enter new name and press enter

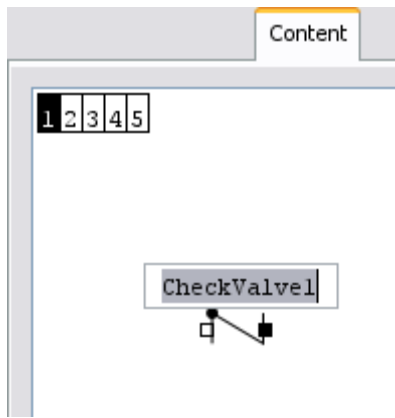


Figure 31: Renaming an instance in content frame

- Right click on the model instance to rename; the Figure 32: Content contextual menu opens. Click on “Rename” item (Figure 31: Renaming an instance in content frame) ; enter new name and press enter.

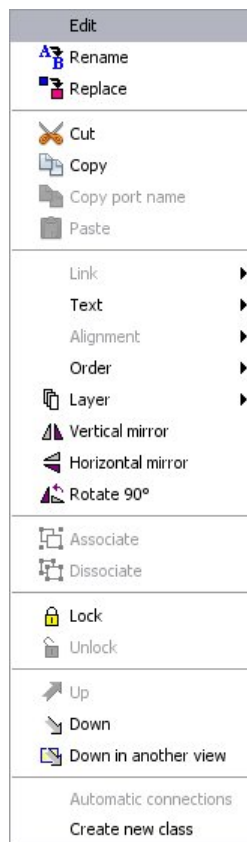





Figure 32: Content contextual menu


## Rotating the graphical representation of a model instance

Use the following commands of the Figure 32: Content contextual menu or the corresponding icons of the Design toolbar to rotate the graphical representation of a model instance:

- Vertical mirror: 

- Horizontal mirror: 
- 90° rotation: 

## Replacing a model instance by another

- Click on the “Replace” item of Figure 32: Content contextual menu
-  If the new model class has the same interface as the previous one (same input and output flow variables, i.e. with same names and types), the links present will be conserved, otherwise they are deleted.

## Moving the graphical representation of a model instance

1. Click on the graphical representation of a model instance to select it
2. Drag it around, or move it using the arrow keys

## Accessing the model class from an instance

- Double-click on the graphical representation of a model instance to open its class
- Click the graphical representation of a model instance to select it, click on “Down” in the Content contextual menu

## Editing links

Links can be drawn between two flow ports, to represent the fact that one end will “emit” a flow into the other end, as in Figure 33: The output of the tank is linked to the input of the manual valve.

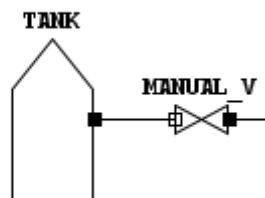




Figure 33: The output of the tank is linked to the input of the manual valve







## Creating a new link between two flow ports

1. Click on the first flow port to connect
2. Maintain click and drag a link to the second flow port
3. Release mouse button

-  Two flow ports can be connected only if they have the same type and if their directions are opposite

## Using predefined link routes

1. Double-click on the link route that must be modified to select it
2. Right-click on the selected link route; the Content contextual menu opens
3. Click on the “Link” item, then on the “Route” item, then click on one of the following items:
  - : to make a straight line between the two ports

- : to make a two corners route between two ports
- : to make a two corners route between two ports
- : to make a route from a port on a right edge to another port on a bottom edge
- : to make a route from a port on a bottom edge to another port on a left edge
- : to make a route from a port on a right edge to another port on a top edge
- : to make a route from a port on a top edge to another port on a left edge


## Inserting a breakpoint into a link route

1. Click on the link route, where a breakpoint must be inserted; maintain click
2. Drag at will

## Moving a breakpoint of a link route

1. Double click on the link route that must be modified; the route is displayed in blue and the breakpoints are shown
2. Click on the breakpoint to move to select it; drag it at will

## Displaying arrows along a link route

1. Double-click on the link route that must be modified to select it
2. Right-click on the selected link route; the Content contextual menu opens
3. Click on the “Link” item, then on the “Arrows” item, then on the “ ” item


 To remove the arrows, proceed as described but click on the “ ” item instead


The status line located at the bottom of the screen displays the “synthesis\_3.e\_anna (bool)” information which gives the following indication:

Equipment name: synthesys\_3,


Port: e\_anna,


Type: bool.


 When the mouse cursor is on a link, the display area located at the left lower part of the architecture view indicates the link description: e.g. a.b <->c.d.


 The architecture view displays in the bottom left window some information about models when moving the mouse cursor on them: the generic name in the library in brackets, and the instance name.




 When an existing architecture is opened or if modifications are carried out in the library and these modifications concern an opened architecture, the tool executes a consistency control on all links in the system. If a consistency error is detected (consistency between the architecture and the library: model, type used in architecture), the link is deleted.

 When a link is deleted in architecture, a message indicates its deletion.

 Click on the OK button.

 Consult the system Information window

 Click with the mouse right button on the button corresponding to an opened architecture at the bottom of the work area; the following contextual menu (Figure 34) is displayed:

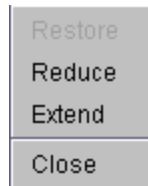


Figure 34 : Contextual menu

## Using automatic connections

The user can automatically connect communication ports of components or equipments based on the name of the port.

This functionality is available through the contextual menu by right clicking on the work area with no-object selected (see **Error! Reference source not found.**):

This functionality asks first for the kind of link to use, then reports the non-ambiguous connections and after validation by user do the non ambiguous connections. It reports also the following lists (with the possibility to locate the communication port involved):

1. List of connections with problem about type
2. List of outputs connected to the same input (fan-in problem)
3. Inputs not connected without any related output found
4. List of non-connected outputs which can connect to input already connected

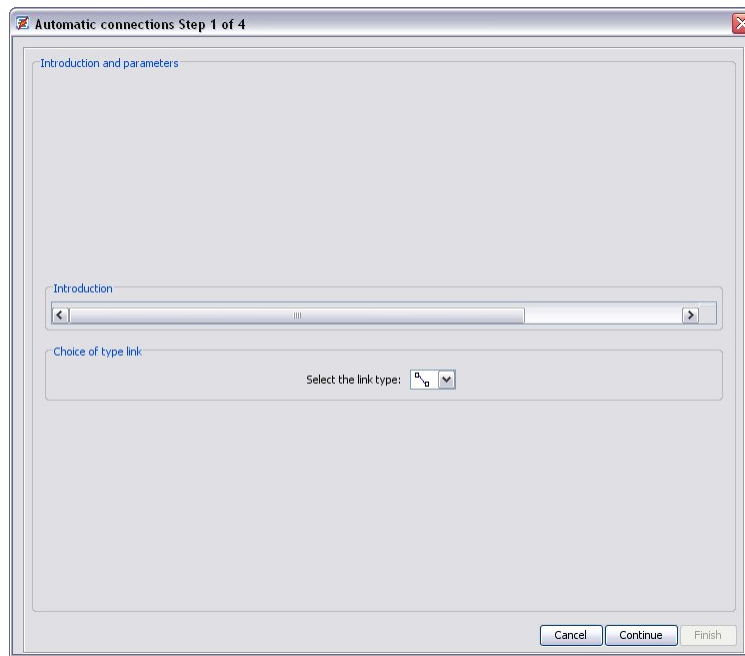
To end, the user can interactively resolve non-connected inputs for the three first items listed above (we list outputs with the same port name for the second item and outputs with the same port type for the third item).

Thus, the functionality 'automatic connections' processes in four steps:

### Step 1

A first panel describes the main rules about the port connections. Use can also select the link type which will be used for automatic connections:

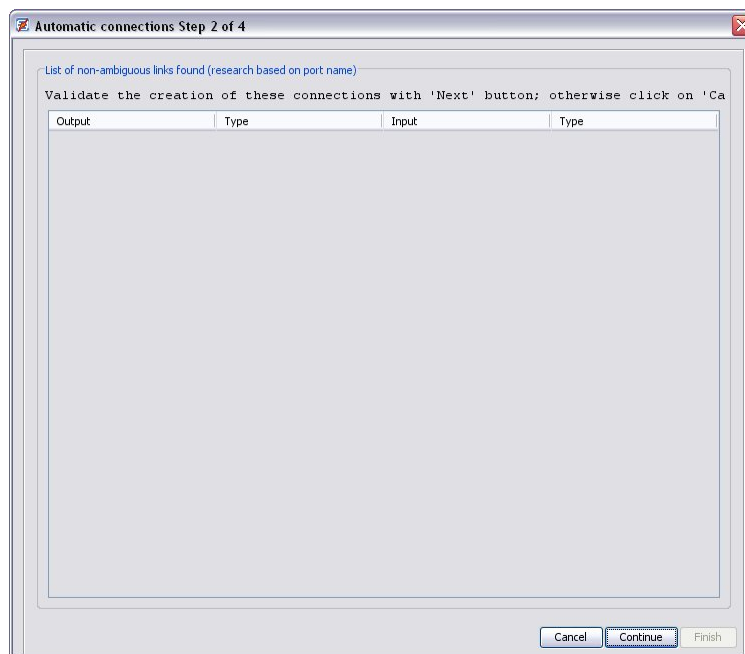




*Figure 35: Automatic connections: step 1 of 4*

## Step 2

The second panel displays the list of non-ambiguous links (same name and same type):



*Figure 36: Automatic connections: step 2 of 4*

## Step 3

The third panel displays a report about potential problems on automatic connections:

1. List of connections having a same port name, but a different type,
2. Multiple outputs which can be connected (same name) to a same non-connected input,
3. List of Non-connected inputs because no output was found with a same port name,
4. List of Non-connected outputs which can connect to an input already connected.

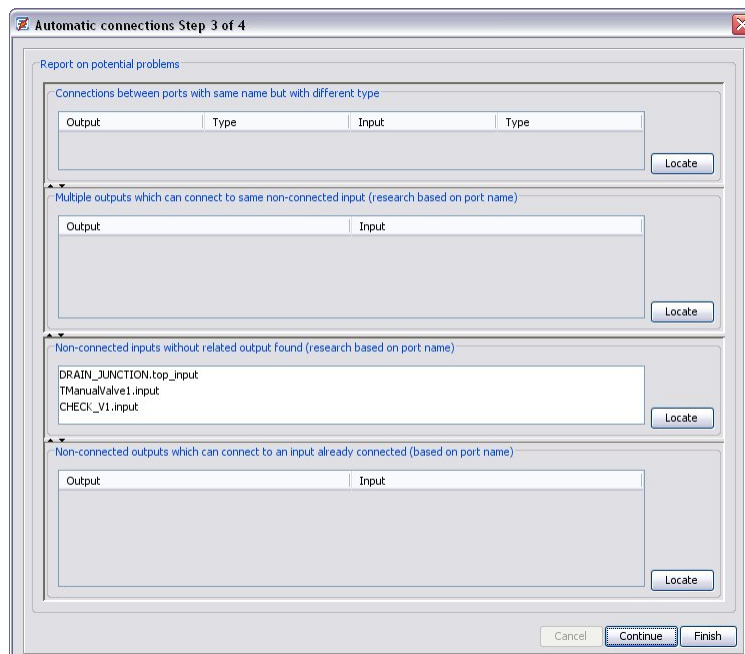
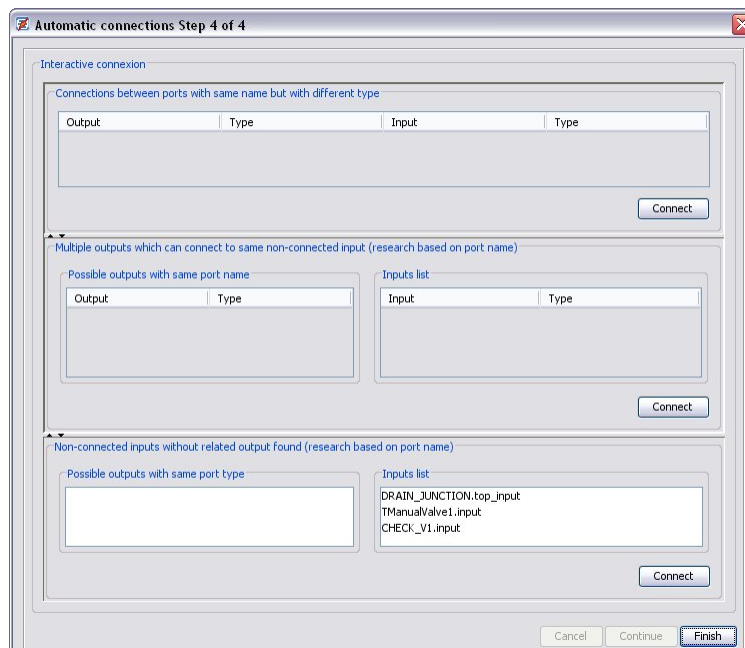


Figure 37: Automatic connections: step 3 of 4

You can select a port name in a list and click on button Locate to identify graphically the model in the current view.

## Step 4

The last step consists in proposing user to connect interactively the remaining ports identified in step 3.

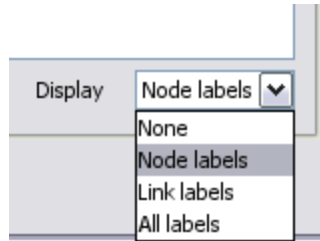


*Figure 38: Automatic connections: step 4 of 4*

Select ports to connect, and click on button Connect to create the link.

## Filtering the labels to be displayed







To select the labels to be displayed, use the Display filter from the Content edition tab.



*Figure 39: Display filter*

- Click on “All labels” to display instances names and links names
- Click on “Node label” to display instances names only
- Click on “Link labels” to display links names only
- Click on “None” to hide all labels.

## Aligning elements

1. Select a group of components with the mouse
2. Right-click to open the Figure 32: Content contextual menu:
  - Click on the  item to align elements on their left,
  - Click on the  item to align elements on their right,
  - Click on the  item to align elements on their top,
  - Click on the  item to align elements on their bottom,
  - Click on the  item to align elements on their horizontal middle,
  - Click on the  item to align elements on their vertical center.

## Using layers

Graphical elements (model instances and links as well as pure graphical elements) can be dispatched onto 5 different layers. The Figure 41: Layer manager can then be used to set the visibility and the depth of these layers.

## Sending graphical elements to a specific layer

1. Right-click on a graphical element; the Content contextual menu opens
2. Click on the “Layer” item, then click on one of the 5 layers to send the element to it (see Figure 40: Layer/Layer number menu).

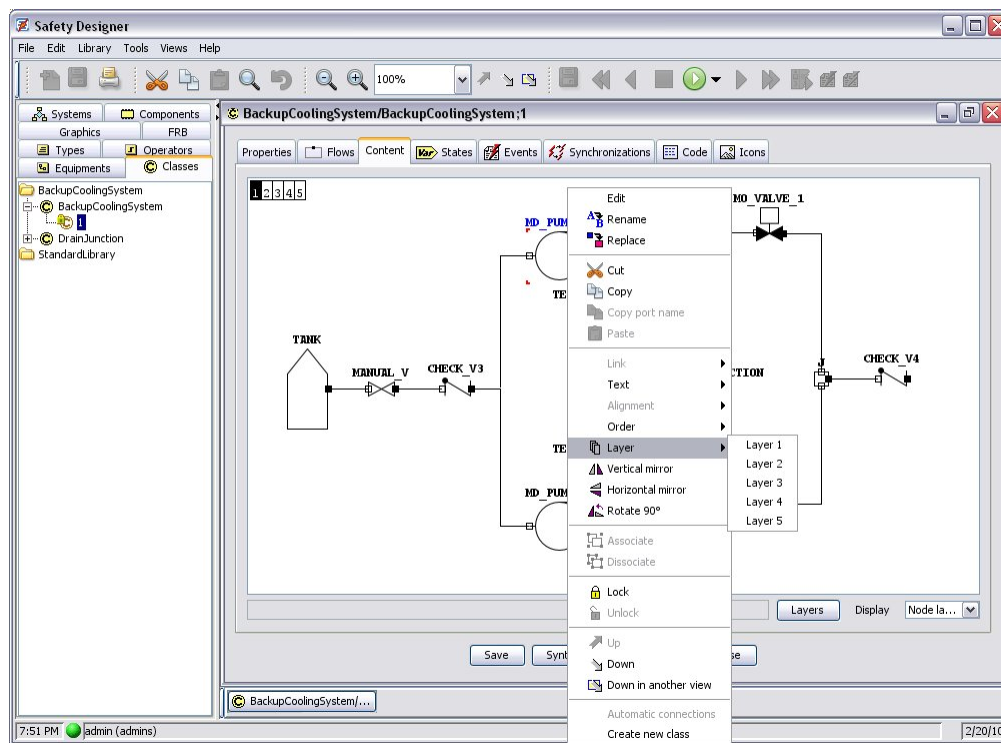


Figure 40: Layer/Layer number menu

## Selecting the active layer

When dropped, graphical elements are added to the current active layer; proceed as follow to change it:



1. Click on the “Layers” button from the Content edition tab; the Layer manager opens. The active layer has a “” next to it:





Figure 41: Layer manager

2. Click on “” next to a layer to make it the active one

## Hiding and showing layers

1. Click on the “Layers” button from the Content edition tab; the Layer manager opens.

- 
2. Click on “” next to a layer to show it, “” to hide it

## Organizing layers by depth

Layers are drawn one after the other; proceed as follow to change the depth of layers:

1. Click on the “Layers” button from the Content edition tab; the Layer manager opens.
2. Set the depth of a layer using the drop down list next to it; layer with depth “1” is drawn first.

## Editing Synchronizations

Several events can be synchronized with the Synchronization tab.

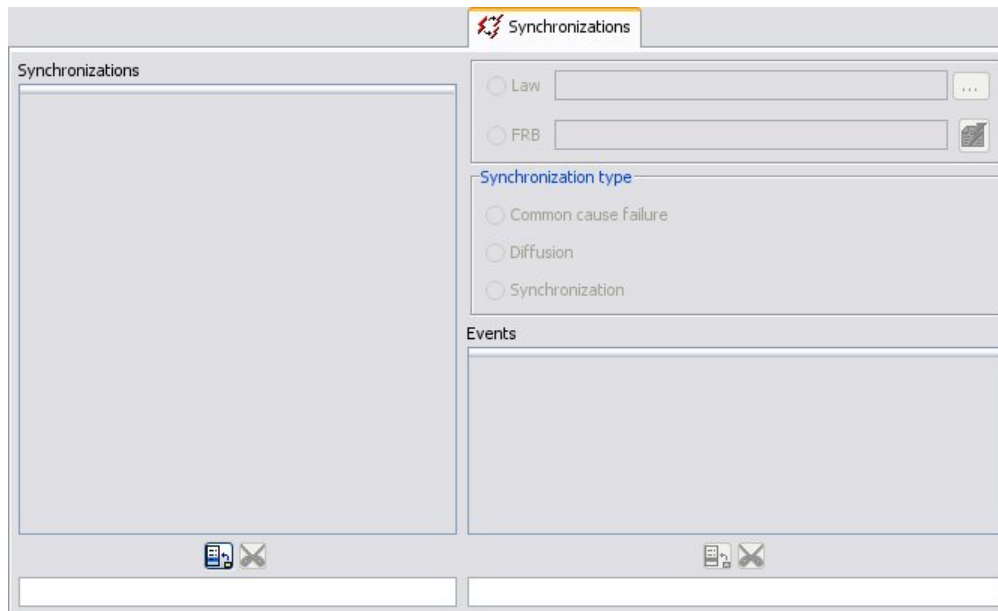






Figure 42: Synchronization tab

## Creating a new synchronization between events

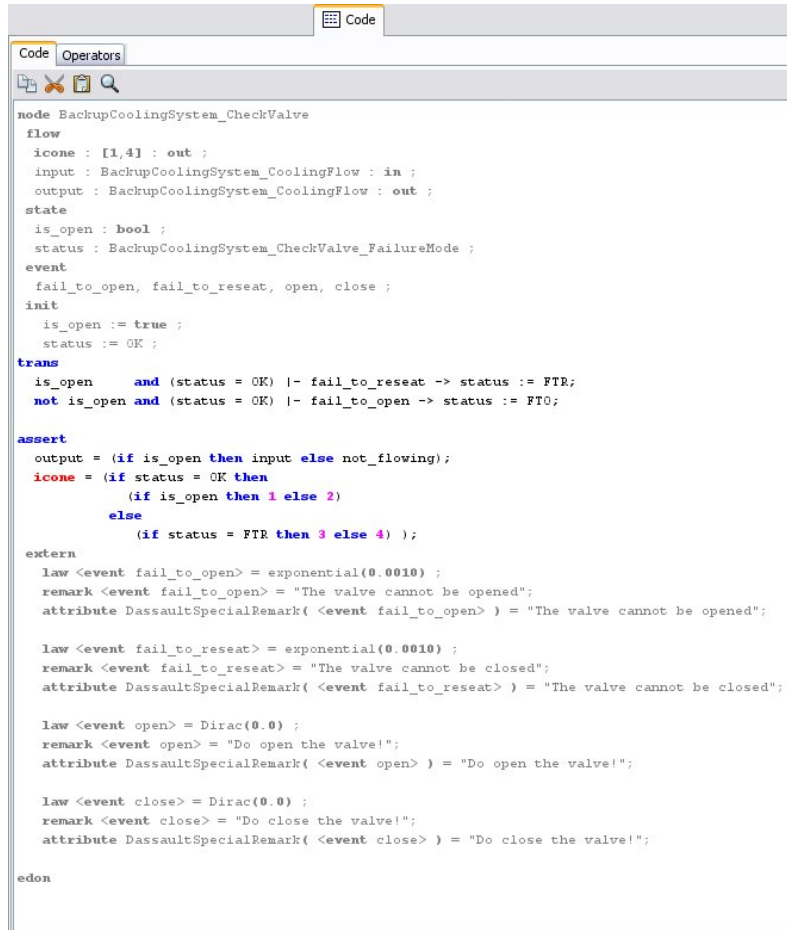
1. Enter the name of the synchronization in the left text field
2. Click on the left  to create a new empty synchronization
3. Edit the vector of synchronized events:
  - Enter the name of an event in the right text field (press “Ctrl+Space” to open the name completion panel); click on the right  to add the event to the synchronization vector
4. Select the synchronization type among the following ones:
  - Common Cause Failure (CCF)
  - Diffusion
  - Synchronization
5. Set the probability distribution using the “Law” field (see Modifying the probability distribution of an existing event)

 At simulation time, synchronizations can be fired as any event.

## Editing AltaRica code

 Refer to Appendix: AltaRica Language Specification for more information on the AltaRica language

The AltaRica code edition tab is used to edit the behavior of a model class using AltaRica language statements.



```

node BackupCoolingSystem_CheckValve
flow
  icone : [1,4] : out ;
  input : BackupCoolingSystem_CoolingFlow : in ;
  output : BackupCoolingSystem_CoolingFlow : out ;
state
  is_open : bool ;
  status : BackupCoolingSystem_CheckValve_FailureMode ;
event
  fail_to_open, fail_to_reseat, open, close ;
init
  is_open := true ;
  status := OK ;
trans
  is_open and (status = OK) |- fail_to_reseat -> status := FTR;
  not is_open and (status = OK) |- fail_to_open -> status := FTO;
assert
  output = (if is_open then input else not_flow);
  icone = (if status = OK then
    (if is_open then 1 else 2)
  else
    (if status = FTR then 3 else 4) );
extern
  law <event fail_to_open> = exponential(0.0010) ;
  remark <event fail_to_open> = "The valve cannot be opened";
  attribute DassaultSpecialRemark( <event fail_to_open> ) = "The valve cannot be opened";

  law <event fail_to_reseat> = exponential(0.0010) ;
  remark <event fail_to_reseat> = "The valve cannot be closed";
  attribute DassaultSpecialRemark( <event fail_to_reseat> ) = "The valve cannot be closed";

  law <event open> = Dirac(0.0) ;
  remark <event open> = "Do open the valve!";
  attribute DassaultSpecialRemark( <event open> ) = "Do open the valve!";

  law <event close> = Dirac(0.0) ;
  remark <event close> = "Do close the valve!";
  attribute DassaultSpecialRemark( <event close> ) = "Do close the valve!";
edon

```



Figure 43: AltaRica code edition tab

## Copying AltaRica code

1. Select the characters to be copied in the text editor
2. Click on the “Copy” button, or press “Ctrl + C”

## Cutting AltaRica code



1. Select the characters to be copied in the text editor
2. Click on the “Cut” button, or press “Ctrl + X”

 Cutting text copies it to the clipboard and suppress it  
 The generated part (in gray) cannot be cut

## Pasting AltaRica code

1. Move the caret to the point after which the text must be pasted from the clipboard

2. Click on the “Paste” button, ore press “Ctrl + V”

-  The clipboard must contain text for this operation to be performed
-  The generated part (in gray) cannot be pasted in

## Using the code completion

### Inserting a variable or an event

1. Move the caret to the desired position
2. Press on “Ctrl + Space”; the Atom insertion panel opens.

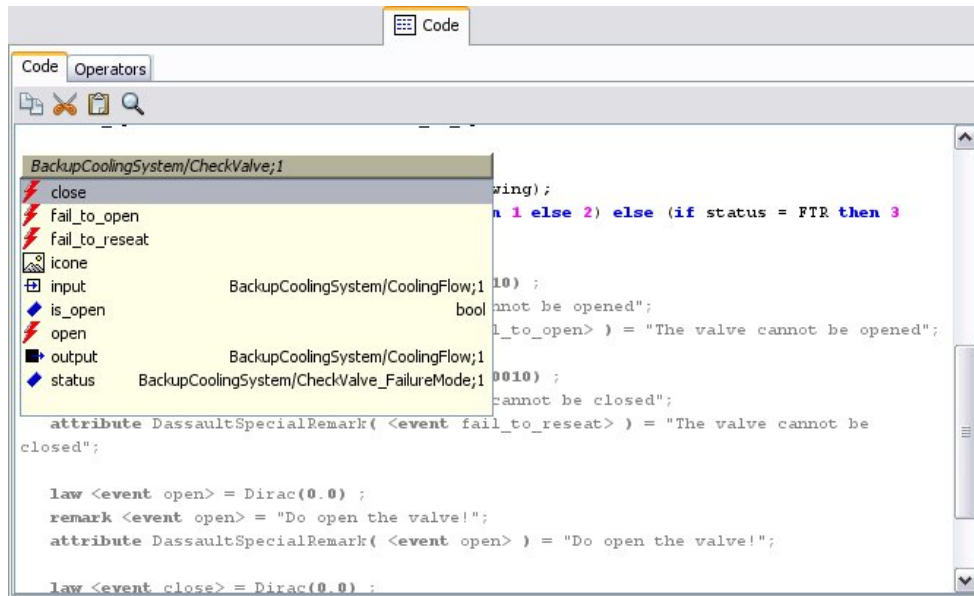


Figure 44: Atom insertion panel

3. Double-click on the element to be inserted

### Inserting a record field

1. Type the character “^” after a variable name; the Field completion panel opens:

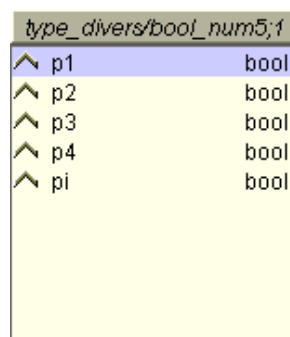



Figure 45: Field completion panel


2. Double-click on a field in the list to insert it

## Understanding the AltaRica syntax highlighting

 The AltaRica syntax highlighting cannot be setup

The syntax of AltaRica code is highlighted as follows:

AltaRica construction	Style
<b>Keywords (“if”, “then”, “else”, “tran”, etc.)</b>	Bold blue
<b>Numerical values</b>	Purple
<b>Operator names</b>	Green
<b>“icone” built-in variable</b>	Bold red

 If operators are used in the AltaRica code, the Operators tab becomes available. It allows selecting operator family and version, if different operators have the same name. If there is no ambiguity, the only family containing the operator is selected, with its last version.

## Searching in AltaRica code

1. Move the caret to the point from where the search must start
2. Click on the “Search” button, or press “Ctrl + F”.

## Checking AltaRica code

### Checking AltaRica code syntax

1. Click on the “Syntax” button
2. If a syntax error is detected, an error window is displayed, describing the error and the line containing the error.

The most common errors are:

- Incorrect syntax in transition declaration (e.g. mixing up the “=” comparison operator and the “:=” assignment operator)
- Assignments between variables of incompatible type (e.g. assigning an integer to a Boolean variable)
- Missing “trans” or “assert” keywords
- Unbalanced parenthesis (not the same number of opening and closing parenthesis)

### Checking AltaRica code semantics

1. Click on the “Consistency” button
2. If a consistency error is detected, an error window is displayed describing the error and the line containing the error.

## Using Icons

Icons can be associated to model classes with the Icon management tab, so that at simulation time, a model instance can have a particular graphical representation depending on its internal state.



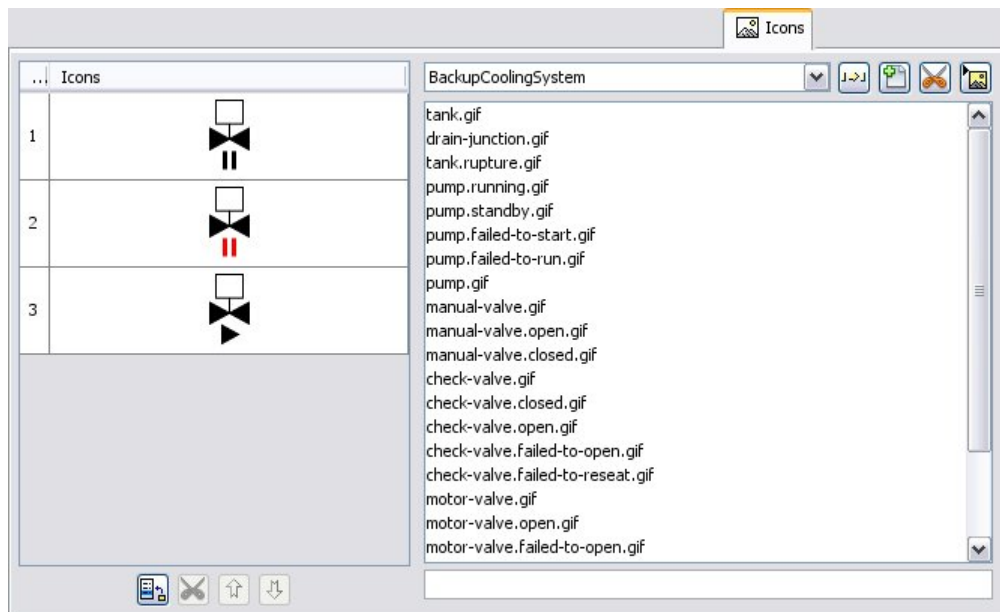



Figure 46: Icon management tab

## Creating a new icon library

1. Click on “” in the Icon management tab.
2. Enter the name of the new library; press enter.

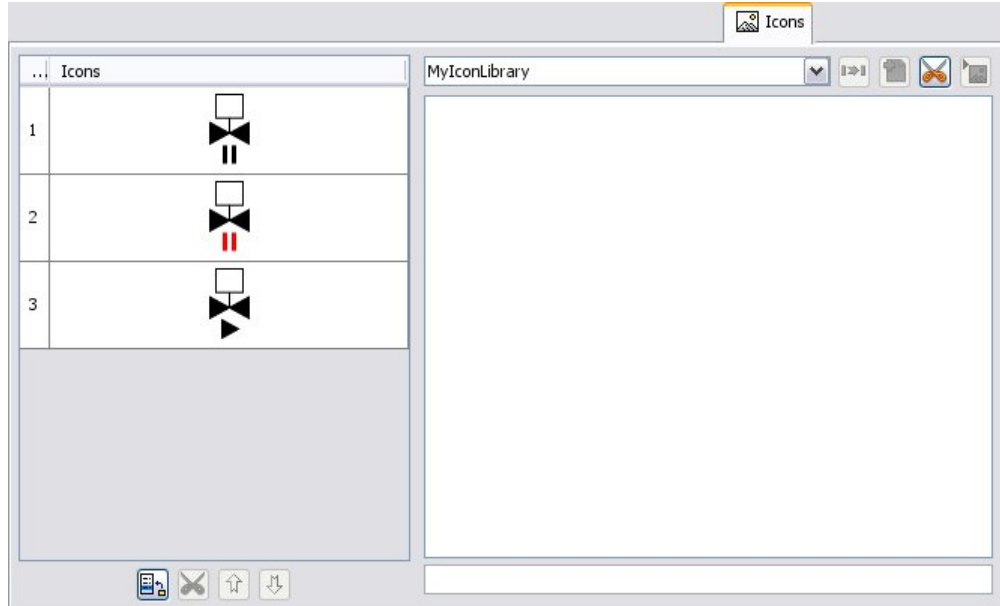


Figure 47: Creating a new icon library

A new icon library is available in the Icon library selector.

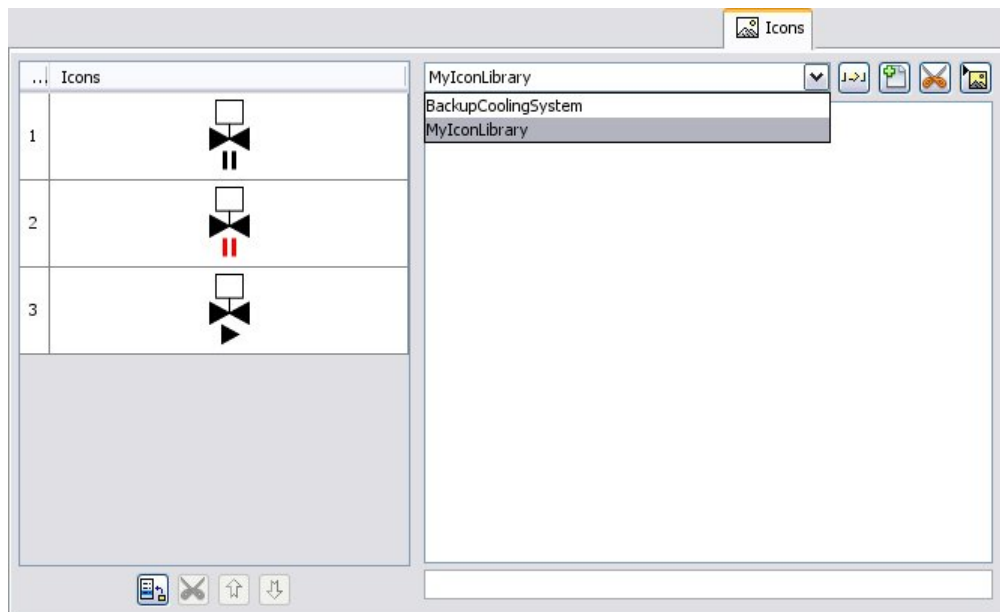



Figure 48: Icon library selector

## Adding icons to a library

1. In the icon library selector of the Icon management tab, select the library where to add a new icon
2. Click on “”; theImage file selection dialog opens.

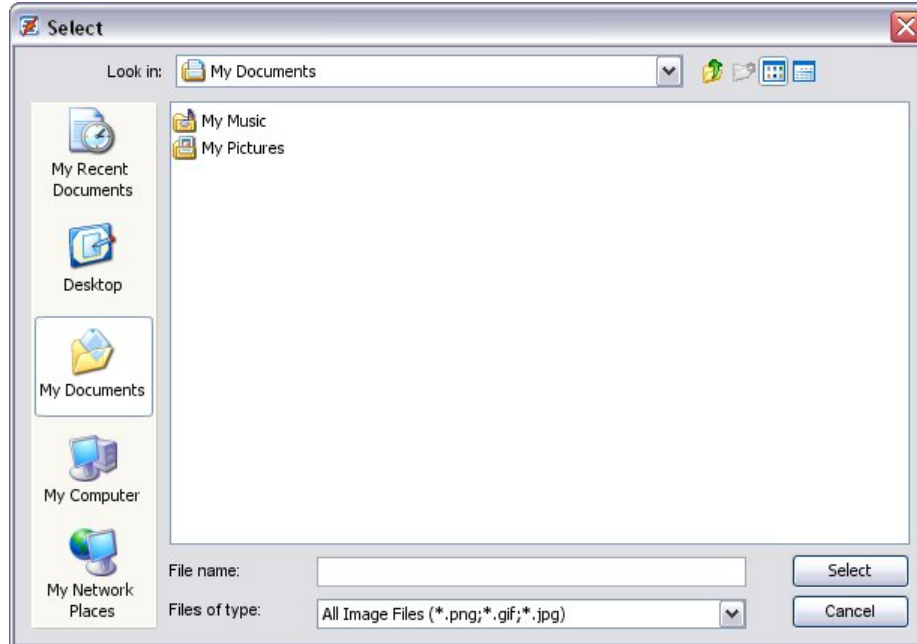



Figure 49: Image file selection dialog


3. Select the image file to add to the library; click on “Select”




---



## Removing icons from a library

1. In the icon library selector of the Icon management tab, select the library to modify
2. Select an icon
3. Click on the right “” to remove it from the library


## Adding icons to a model class

-  To set the graphical representation of a model class independently of the model instance state, edit the general properties of the class instead

1. In the icon library selector of the Icon management tab, select the source library
2. Select an icon
3. Click on “” to append it to the icon vector used by the class
4. Click on “” to move up the icon in the vector, “” to move it down

-  In a class, an icon is associated to a number (its position in the icon vector) which is meant to be used in the AltaRica code of the class.
-  Using icons of the same size is a good practice to ensure that the model instance evolves nicely during simulation

## Removing icons from a model class

1. In the icon library selector of the Icon management tab, select the source library
2. Select an icon
3. Click on the left “” to remove it from the icon vector used by the class

## Editing general properties

The general properties of a model class can be edited with the General properties tab

The screenshot shows a software interface for editing a component's properties. The main window is titled 'Properties' and contains several input fields and checkboxes. The 'Name' field is set to 'MotorValve'. The 'Width' field is set to '33' and the 'Height' field is set to '54'. The 'Icon file' field is set to 'BackupCoolingSystem/motor-valve.gif' and has a browse button (three dots) to its right. Below these fields are two checked checkboxes: 'Draw border' and 'Move ports automatically when components are reshaped'. A 'Resize' button is located to the right of the second checkbox. At the bottom of the window, there is a sub-section also titled 'Properties' containing a 'Created at' field (2/18/10 2:52 PM - admin), a 'Modified at' field (2/18/10 2:55 PM - admin), a 'Version' field (1), and a 'Comment' field with a large text area below it.

Properties	
Name :	MotorValve
Width :	33
Height :	54
Icon file :	BackupCoolingSystem/motor-valve.gif
<input checked="" type="checkbox"/> Draw border	
<input checked="" type="checkbox"/> Move ports automatically when components are reshaped	
Resize	
Properties	
Created at :	2/18/10 2:52 PM - admin
Modified at :	2/18/10 2:55 PM - admin
Version :	1
Comment :	

Figure 50: General properties tab

## Duplicating a Model Class

1. Click on the Class tab of the model browser
2. Select a model class
3. Right click on it; the Model edition and organization contextual menu opens up.
4. Select "Model..." -> "Duplicate"

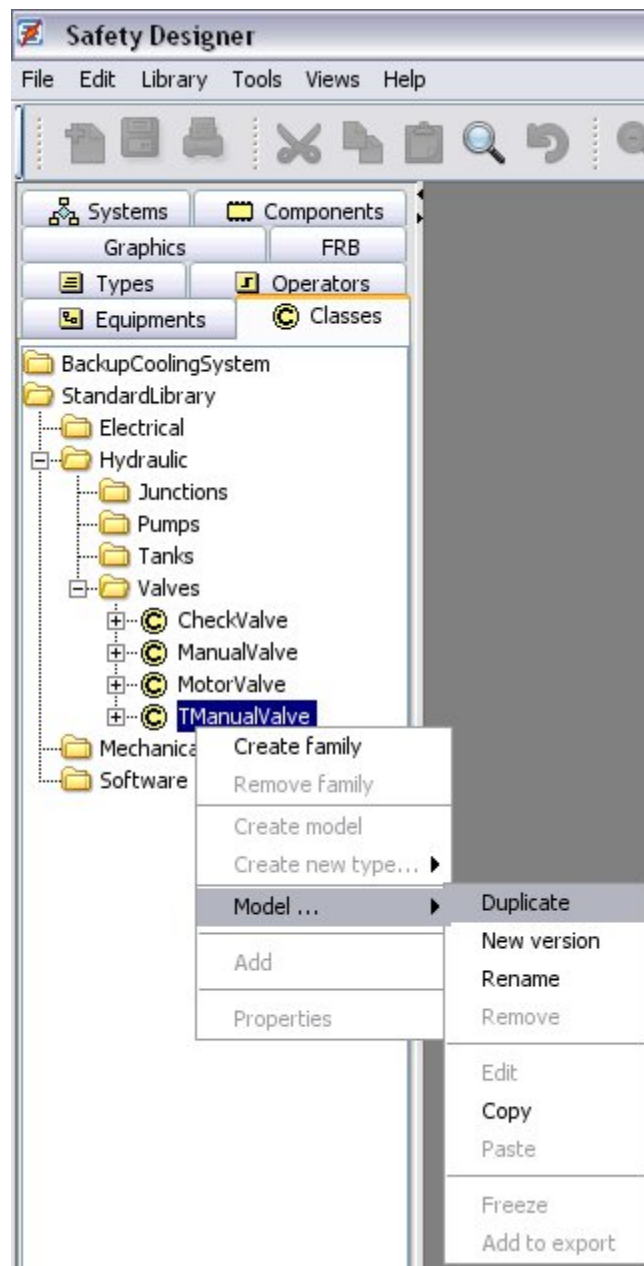



Figure 51: "Model..." -> "Duplicate"

-  If the model has several versions, the last version is duplicated. To duplicate another version of the component, select the version in the model tree structure and proceed as described

## Creating a New Version of a Model Class

1. Click on the Class tab of the model browser
2. Select a model class
3. Right click on it; the Model edition and organization contextual menu opens up.
4. Select "Model..." -> "New version"

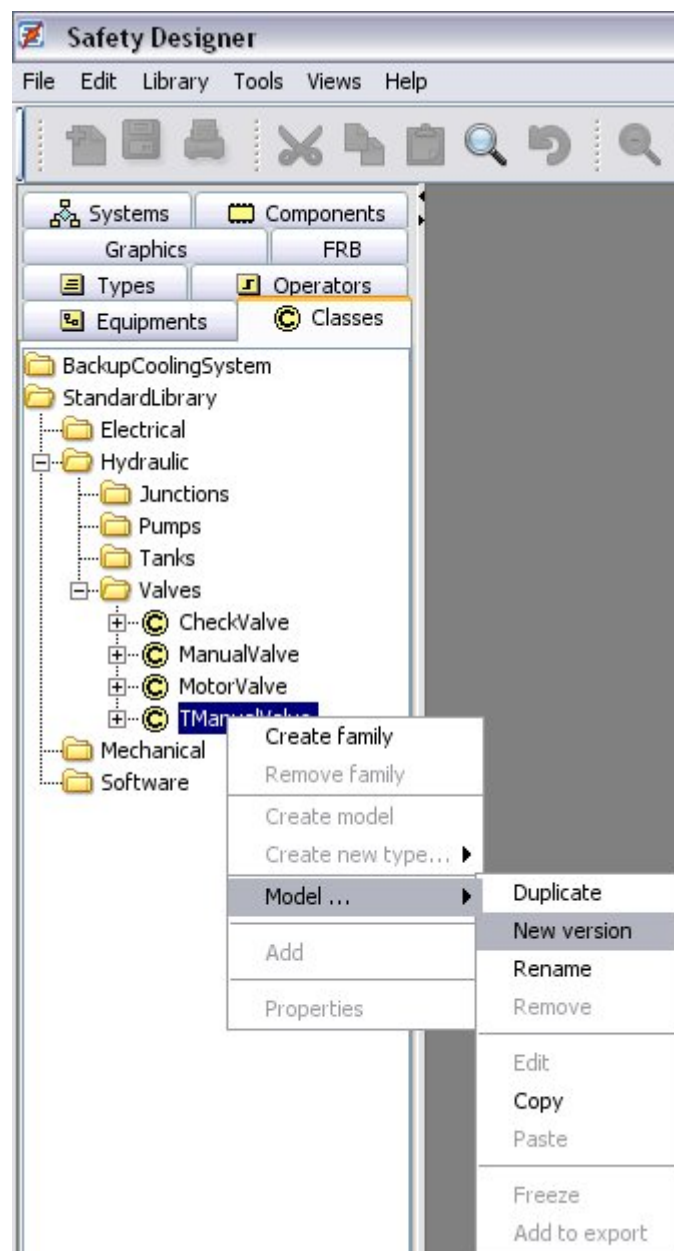


Figure 52: "Model..." -> "New version"

5. Select the version to derive in the New version creation dialog.



Figure 53: New version creation dialog

The new version can be seen in the model structure (Figure 54: A new version).

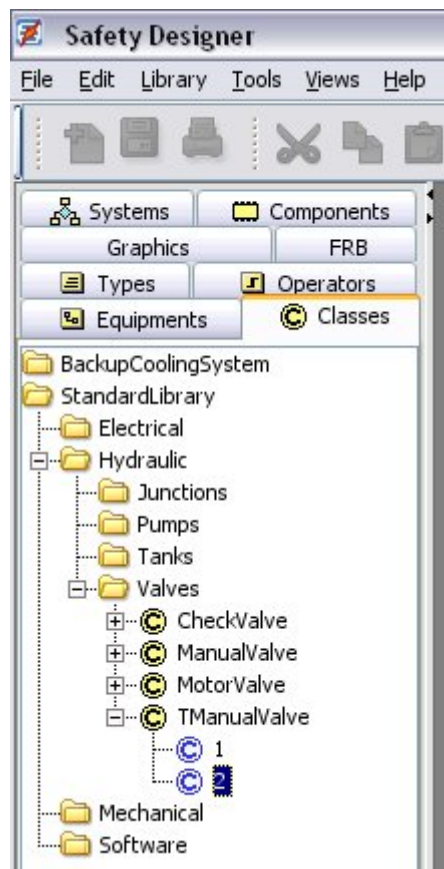


Figure 54: A new version

## Renaming a Model Class

★ Use this function with extreme care, as it can broke the models using the model class

1. Click on the Class tab of the model browser
2. Select a model class
3. Right click on it; the Model edition and organization contextual menu opens up.
4. Select "Model..." -> "Rename"

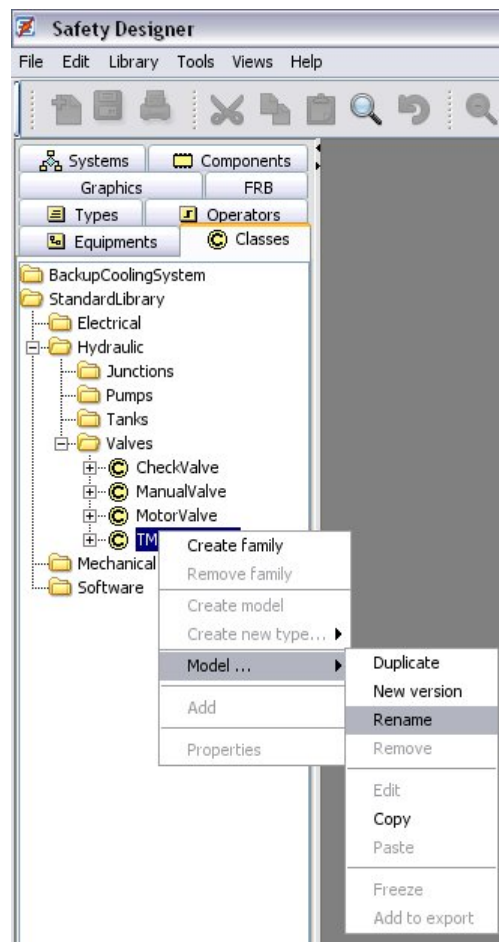



Figure 55: "Model..." -> "Rename"

5. Enter a new name and press "Enter".

 If the name already exists, an error message is displayed. Click "OK" and enter another name

## Removing a Model Class Version

★ Use this function with extreme care, as it cannot be undone; the class and its instances are deleted, as well as the links pointing at them.

1. Click on the Class tab of the model browser
2. Select a model class
3. Select a version
4. Right click on it; the Model edition and organization contextual menu opens up.
5. Select "Model..." -> "Remove"



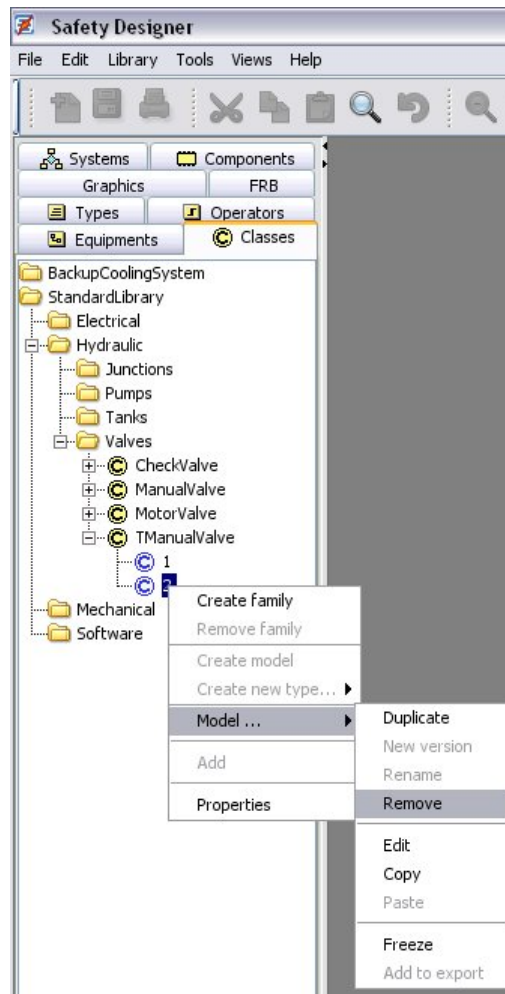



Figure 56: "Model..." -> "Remove"

6. Click on "Continue" to confirm deletion

 Remove all the versions of a model class to remove it entirely

## Freezing a Model Class

★ Use this function with extreme care, as it cannot be undone; a frozen model cannot be modified or duplicated.

1. Click on the Class tab of the model browser
2. Select a model class
3. Select a version
4. Right click on it; the Model edition and organization contextual menu opens up.
5. Select the "Model..." -> "Freeze"

# Working with Systems, Equipments and Components

## Working with Component Models

A component model is a particular kind of model class, which can have flow and state variables, events, icons and AltaRica code, but cannot compose instances of other models, hence cannot have synchronizations. A component model can be considered as a flow processor which has an internal state, can react to events, and transform its inputs accordingly.

## Creating a new component model

Open the Component model tab, and proceed as in Creating a new Model Class.

## Editing flow variables

## Proceed as in Accessing an Existing Model Class

## Opening an Existing Model Class

7. Access the Model class tab (Figure 19: Model Class Editor)
8. Select a model class in the family tree (“BackupCoolingSystem/BackupCoolingSystem”)
9. Double-click on a version (“Version 1”); the Model Class Editor opens.

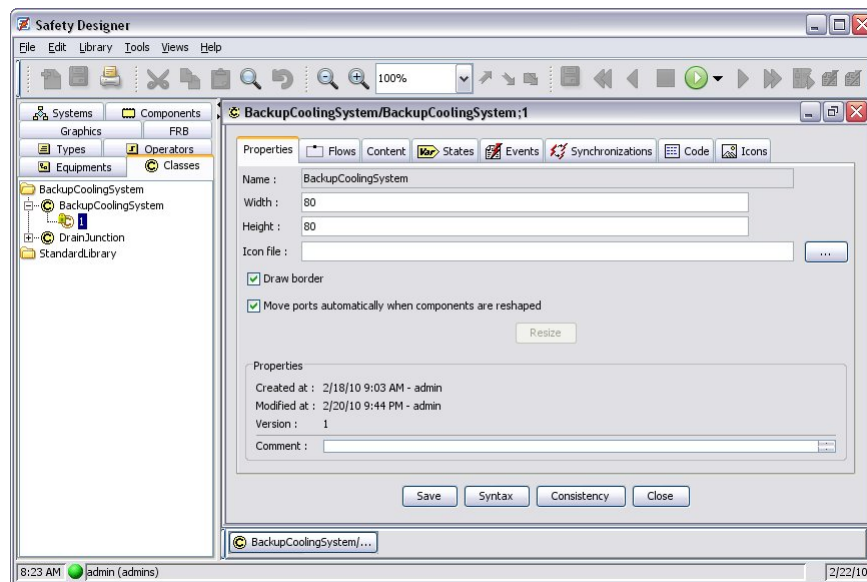


Figure 19: Model Class Editor

- ✎ When a model is opened for edition, it is automatically locked so that several users cannot edit the same model at the same time.
- ✎ A model class can remain locked after a system crash; access its properties to remove the lock manually

## Checking a Model Class for Consistency


3. Click on the “Syntax” and “Consistency” buttons at the bottom of the Model Class Editor

## Saving an Existing Model Class

3. Click on the “Save” button at the bottom of the Model Class Editor.

---

## **Closing an Existing Model Class**

5. Click on the “Close” button at the bottom of the Model Class Editor, or on “” on its top right corner
6. If the model class has been modified, a confirmation dialog opens

Editing Flow Variables.

### **Editing state variables**

Proceed as in Editing State Variables.

### **Editing events**

Proceed as in Editing Events.

### **Editing AltaRica code**

Proceed as in Editing AltaRica code.

### **Using icons**

Proceed as in Using Icons.

### **Editing general properties**

Proceed as in Editing general properties.

### **Duplicating a component model**

Proceed as in Duplicating a Model Class

### **Creating a new version of a component model**

Proceed as in Creating a New Version of a Model Class

### **Renaming a component model**

Proceed as in Renaming a Model Class

### **Removing a component model version**

Proceed as in Removing a Model Class Version

### **Freezing a component model**

Proceed as in Freezing a Model Class

## Working with Equipment Models

An equipment model is a particular kind of model class, which can have flow variables, icons and AltaRica code, can compose instances of other models and synchronize events, but cannot have state variables and independent events.

### Creating a new equipment model

Open the Equipment model tab, and proceed as in Creating a new Model Class.

### Editing flow variables

## Proceed as in Accessing an Existing Model Class

### Opening an Existing Model Class

10. Access the Model class tab (Figure 19: Model Class Editor)
11. Select a model class in the family tree (“BackupCoolingSystem/BackupCoolingSystem”)
12. Double-click on a version (“Version 1”); the Model Class Editor opens.

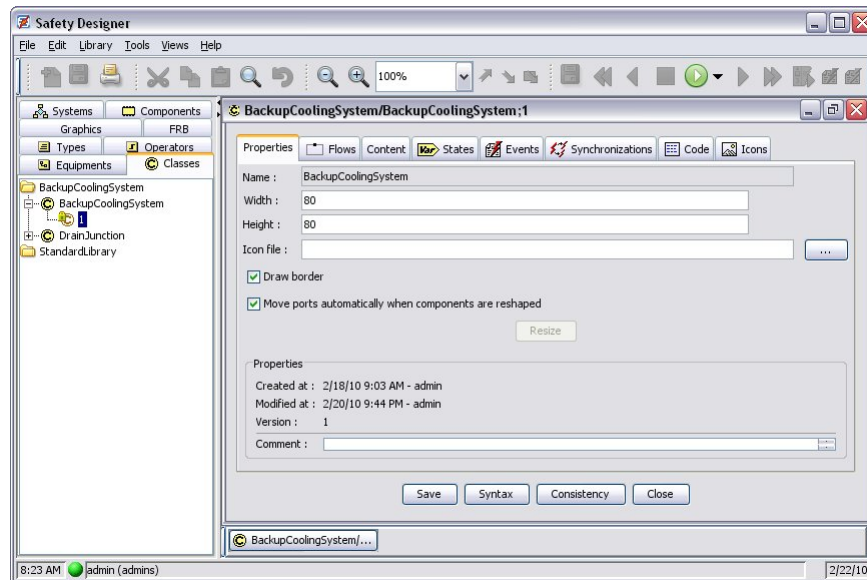




Figure 19: Model Class Editor

-  When a model is opened for edition, it is automatically locked so that several users cannot edit the same model at the same time.
-  A model class can remain locked after a system crash; access its properties to remove the lock manually

### Checking a Model Class for Consistency


4. Click on the “Syntax” and “Consistency” buttons at the bottom of the Model Class Editor

### Saving an Existing Model Class

4. Click on the “Save” button at the bottom of the Model Class Editor.

---

## Closing an Existing Model Class

7. Click on the “Close” button at the bottom of the Model Class Editor, or on “” on its top right corner
8. If the model class has been modified, a confirmation dialog opens

Editing Flow Variables.

## Editing AltaRica code

Proceed as in Editing AltaRica code.

## Using icons

Proceed as in Using Icons.

## Editing general properties

Proceed as in Editing general properties.

## Duplicating an equipment model

Proceed as in Duplicating a Model Class

## Creating a new version of an equipment model

Proceed as in Creating a New Version of a Model Class

## Renaming an equipment model

Proceed as in Renaming a Model Class

## Removing an equipment version

Proceed as in Removing a Model Class Version

## Freezing an equipment model

Proceed as in Freezing a Model Class

## Working with System Models

A system model is a particular kind of model class, which can compose instances of equipment and component models, can synchronize events, and can have system assertions (made of assertions expressed in AltaRica code), but cannot have state or flow variables, independent events or icons.

## Creating a new system model

Open the System model tab, and proceed as in Creating a new Model Class.

## Duplicating a system model

Proceed as in Duplicating a Model Class

## Creating a new version of a system model

Proceed as in Creating a New Version of a Model Class

## Renaming a system model

Proceed as in Renaming a Model Class

## Removing a system version

Proceed as in Removing a Model Class Version

## Freezing a system model

Proceed as in Freezing a Model Class

## Editing system assertions

A system assertion is a link which has no graphical representation.

1. Click on the **System assertions** tab; the following window (Figure 57) is displayed.
2. In the left part of the window, click with the mouse left button on the “main” hierarchical level, then double click to display the hierarchical levels; select a hierarchical level.
3. The *Edition* area displays the variable assignment in the **System assertions**.

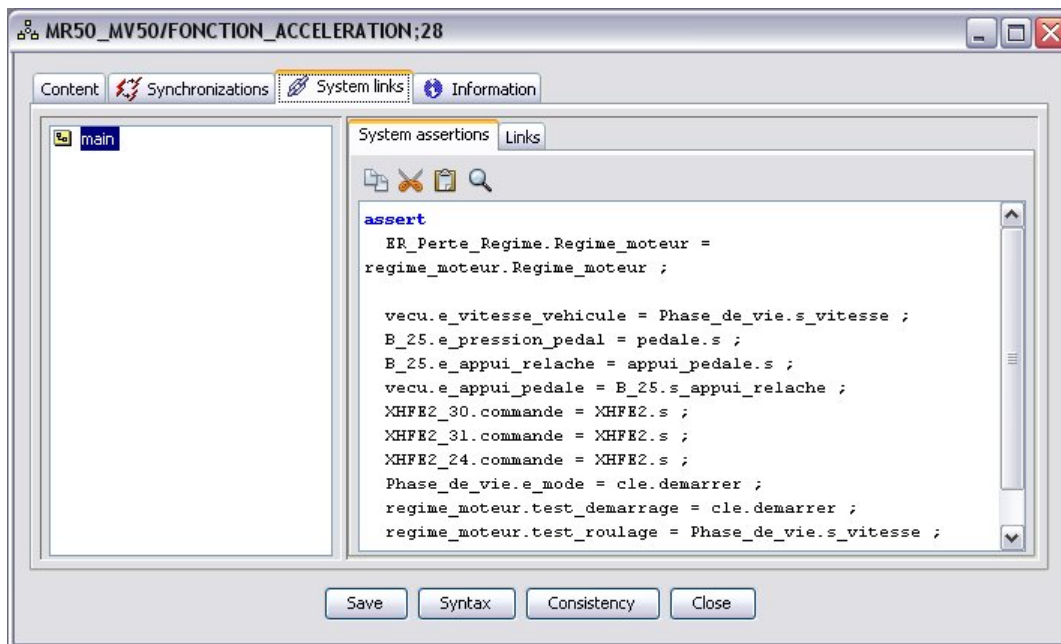


Figure 57 : System assertions tab

- 
- ✎ System assertions are links which exist but are not displayed in the graphical view; they concern only flow assignments.

4. The syntax for system assertions is:

Simple assignment between “flow in” and “flow out” with same type:

```
<component_i>.<In_Name> = < component j>.<OutName>;
```

Simple assignment between “flow in” and a specific value defined with a same type:






```
<component_i>.<In_Name> = true;  
<component_i>.<In_Name> = low;
```

Assignment using operators:

```
<component_i>.<In_Name> =  
  (<component_j>.<OutName> or <component_k>.<OutName>);  
  
<component_i>.<In_Name> = my_operator (<component_j>.<OutName>,  
                                       <component_k>.<OutName>);
```

5. It is possible to get the port name in the clipboard by selecting a port in the system model: right click to display the contextual menu and choose **Copy port name**. The name of this port can be pasted in the equipment Altarica code.

Five icons are available to edit system assertions:

- : Copy,
- : Cut,
- : Paste,
- : Backward zoom,
- : Forward zoom.

- ✎ The validity of these system assertions will be checked by the system consistency control (activated with the command Check properties from the System menu).

- ★ **In system architecture, assertions can only be written at the system level (main node). For hierarchical level associated to equipments, only a viewing of assertions defined in library is possible (in equipment model editor).**

## Editing system links

A link defines an interconnection between two objects in architecture.

- Click on the **Links** tab; the following window (Figure 58) displays an exhaustive list of all the links which have a graphical representation in a given hierarchical level.
- In the left part of the window, click with the mouse left button to display the “main” hierarchical level, and then double click to display the hierarchical levels; select a hierarchical level.

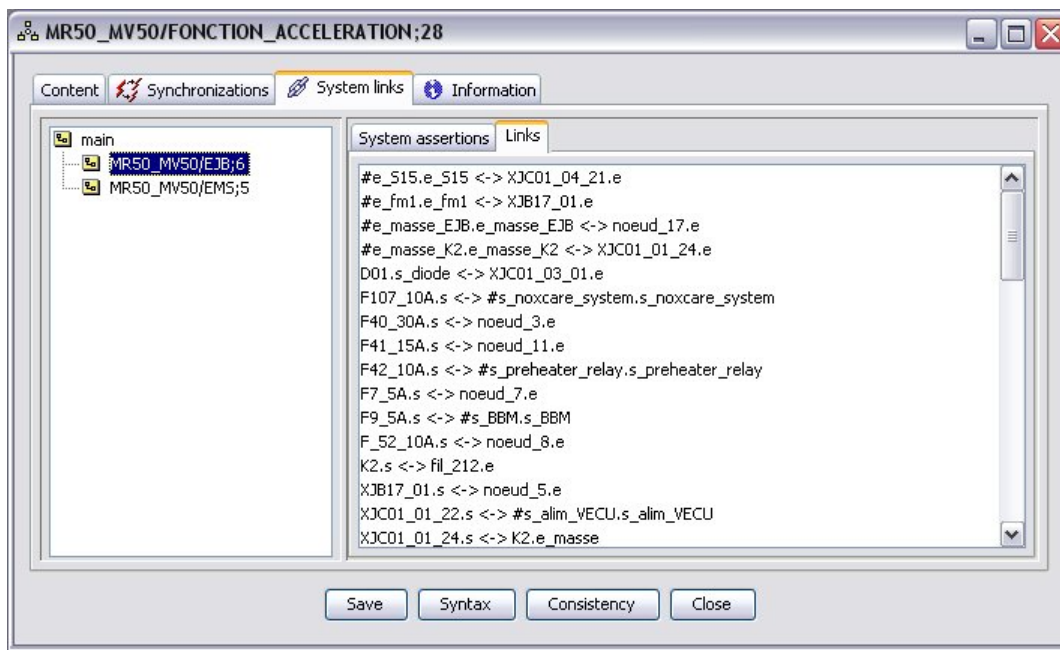


Figure 58 : System assertions – Links tab

## Reading information on models

### Unknown models tab

Click on the Unknown models tab (Figure 59) the window displays the list of unknown models.

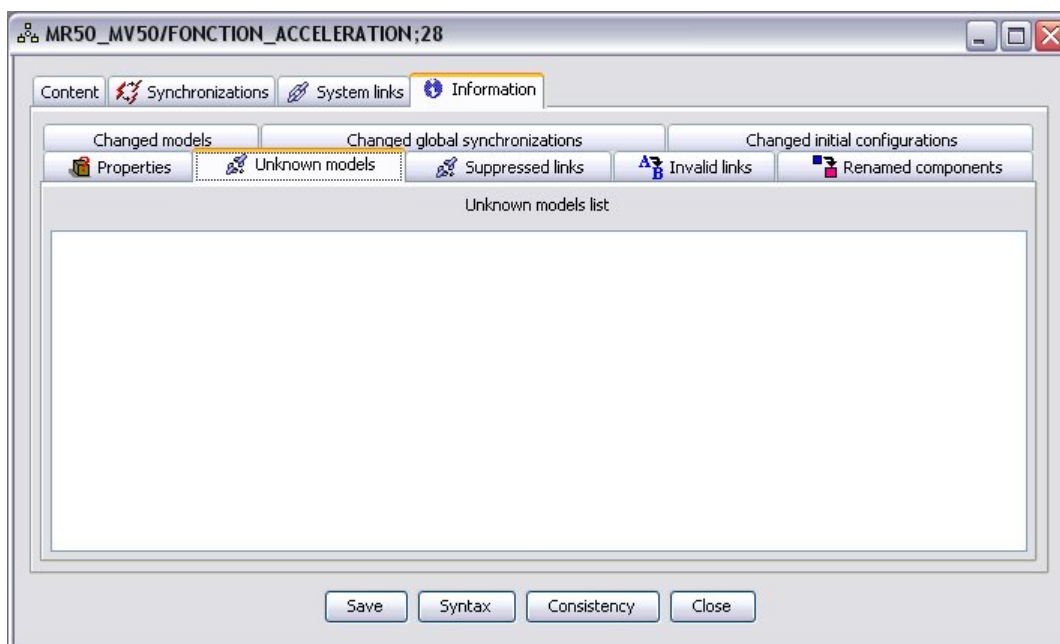
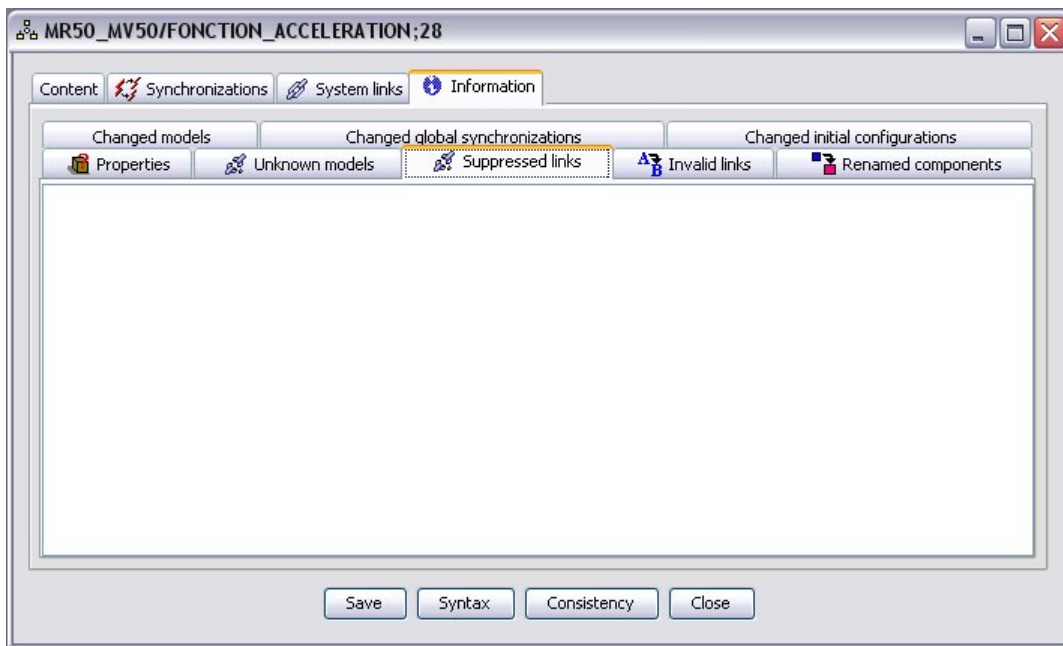


Figure 59 : Information –Unknown models tab

### Suppressed links tab



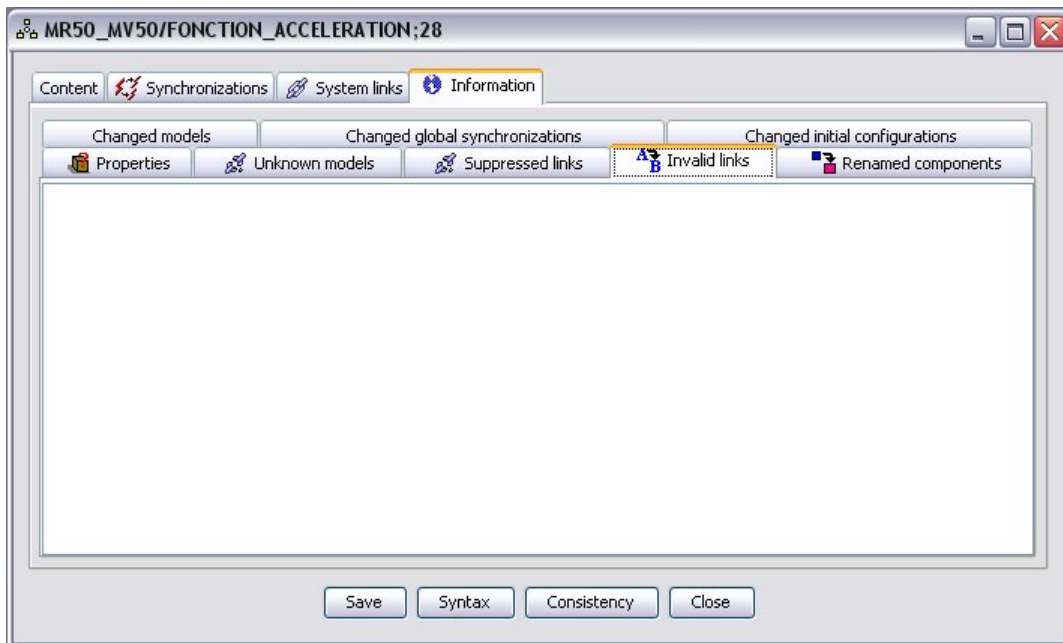
Click on the Suppressed links tab (Figure 60); it indicates information for each of the following elements: component, component model, port, port type and field of type.



*Figure 60 : Information – Suppressed links tab*

### Invalid links tab

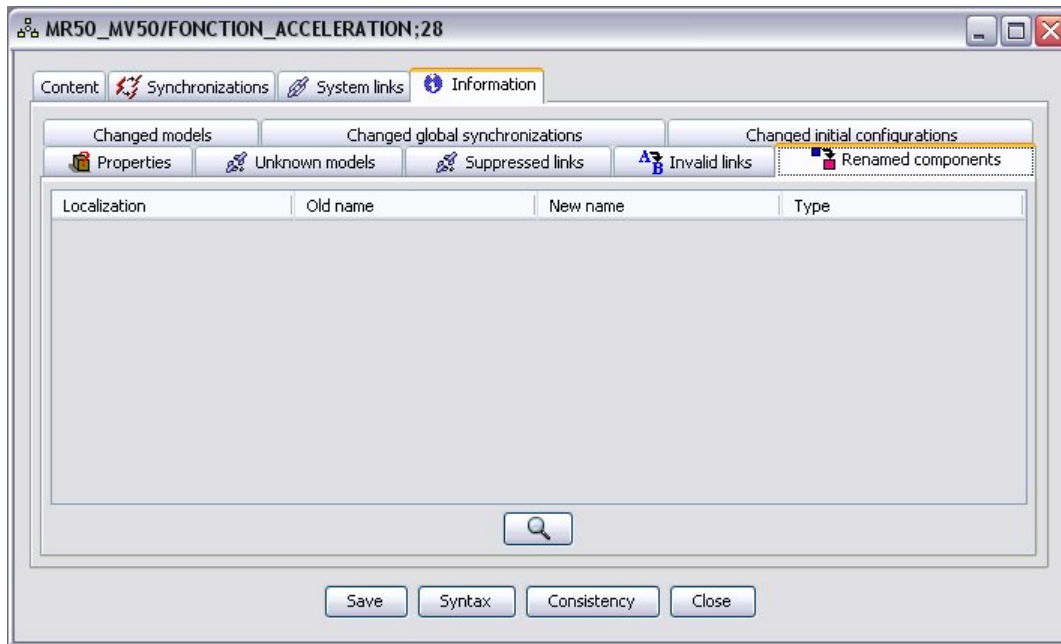
Click on the Invalid links tab (Figure 60); it indicates information for each of the following elements: component, component model, port, port type and field of type.



*Figure 61 : Information – Invalid links tab*

### Renamed components tab

Click on the Renamed components tab (Figure 62).



*Figure 62 : Information – Renamed components tab*

The window displays the following information concerning components which have been renamed in an architecture:

- Localization,
- Old name,
- New name,
- Type.

### Changed models tab

Click on the Changed models tab; the window (Figure 63) indicates the models, the ports and the states.

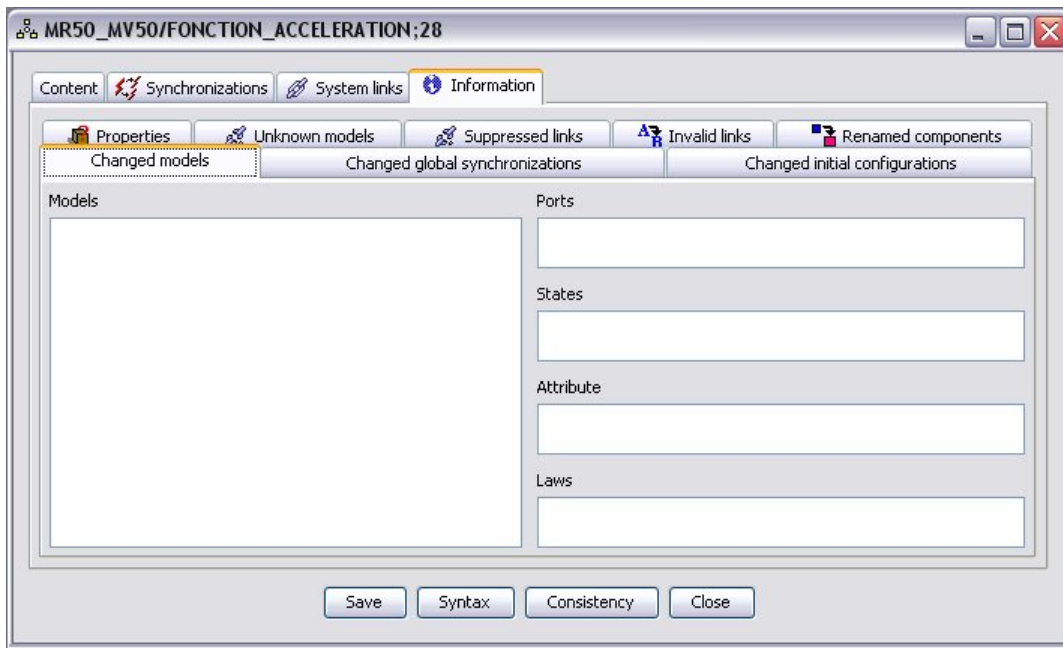



Figure 63 : Information – Changed Models tab

## Working with Operators

### Creating a new operator

#### Definition

Operators allow simplifying the writing of assertions in AltaRica code. An operator corresponds to a transfer function between output flow(s) and input flow(s). Contrary to a component model, an operator doesn't include behaviour (represented by transitions).

-  The operators can be used only within component (or equipment) assertion. Using operators in transitions is currently forbidden.


To create a new operator model:

1. Click on **Operators** tab to display the list of operator families in the library.
2. Select an operator family with left click,
3. Display the contextual menu with right click or open the **Library** menu.
4. Select **Create new model** command, the following bar graph is displayed (**Error! Reference source not found.**) :
5. The window *Operator editor* (Figure 64) is displayed; it is composed of three tabs :
6. General,
7. Properties,
8. AltaRica code.

#### General tab

The General tab (Figure 64) allows to filling in the following information:

1. **Name** (in the top edit area): enter the operator name, e.g. o2by3.

- ✎ In the Operands area, the first line corresponds to the operator result. The default name of the operator result is the name of the name. It cannot be changed. The first line cannot be removed.
  - 2. **Name** (in the bottom edit area): enter the operand name.
  - 3. Use **Type** menu to define the operand type.
  - 4. Click on  icon or use the **Enter** key to add the operand in the list.
  - 5. To modify the type of an existing operand, select it and click on **Assign** button.
- ✎ Type of an operator/operands can be a record, but the orientation must be “normal” and without crossing.

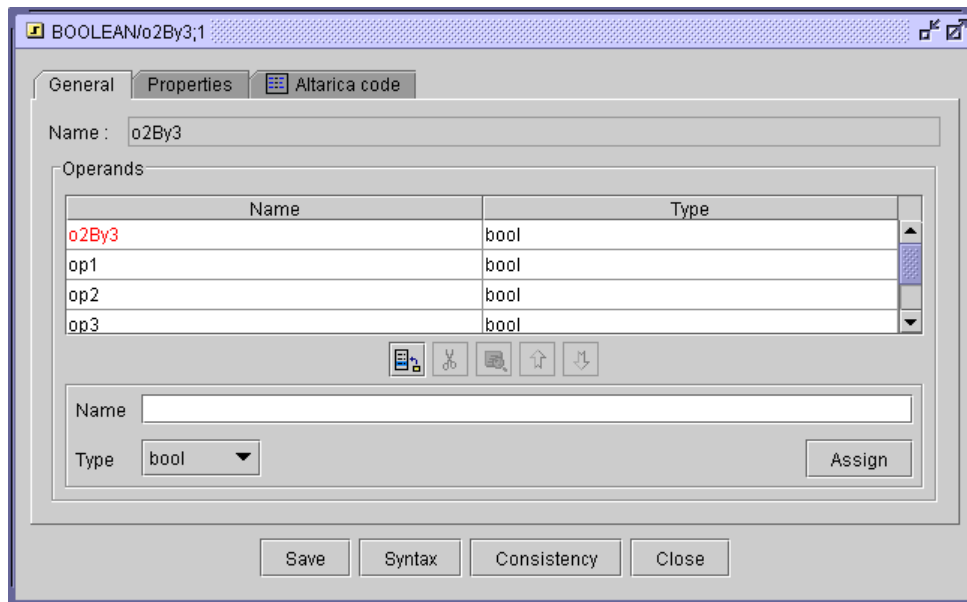


Figure 64 : Operator editor - General tab


## Properties tab

The Properties tab display:

- Creation date,
- Last modification date,
- Version.
- The **Comment** field allows adding a comment describing the operator.

## AltaRica code tab

The AltaRica code tab contains:

- A shortcut bar (Copy, Cut, Paste, Backward zoom, Forward zoom, Display extern clauses).
- **Extern clauses** is a no-editable area and the icon  is used to display/mask it.
- A *variable declaration* area below the edition bar displays information about the current operator: function name, flow variables (I/O) and their type. The content of this area is generated automatically:

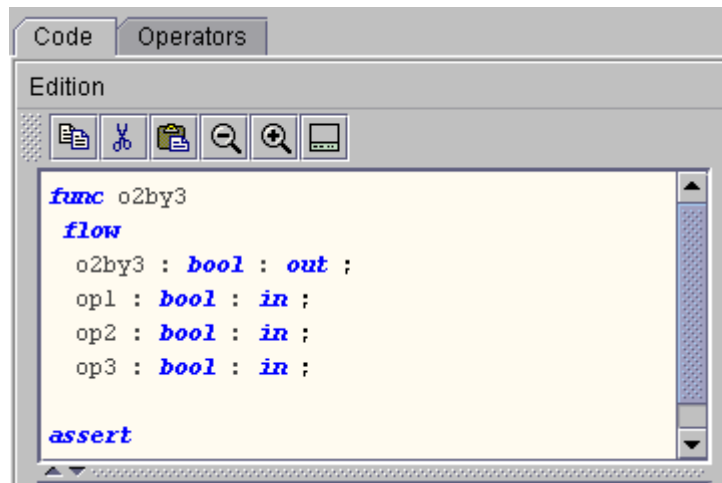


Figure 65 : Operator – AltaRica code – Operands

- The AltaRica code area (assertions [**assert**]). The AltaRica code syntax of an operator is identical to component assertions:

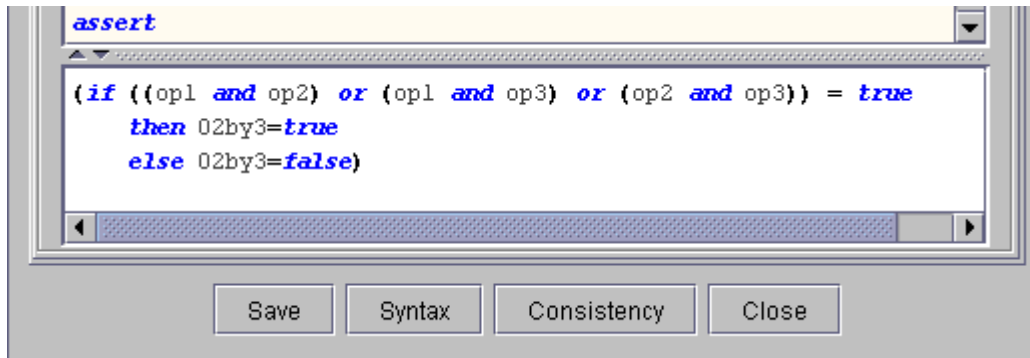
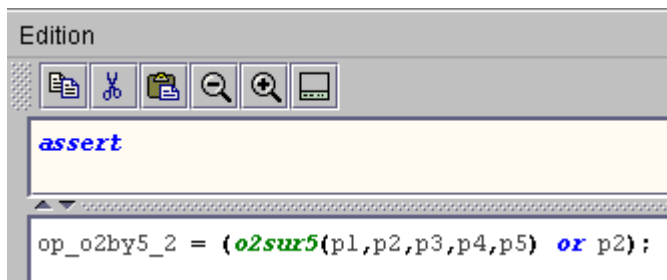


Figure 66 : Operator editor - AltaRica code tab

Assertions allow specifying the values of the output flow (identified by the operator name: O2by3) according to the input flows (operands: op1, op2, op3).

- ✎ in the previous example, the assertion can be defined by the following expression: O2by3= ( (op1 and op2) or (op1 or op3) (op2 or op3)).
- ✎ Operators can be used to define other operators:



If operators are used in AltaRica code, Operators tab is accessible (Figure 67 : AtlaRica Code - Operator Tab). It allows selecting the family and the version of the operator, if different operators have the same name. If there is no ambiguity, the only family containing the operator is selected with the last version.

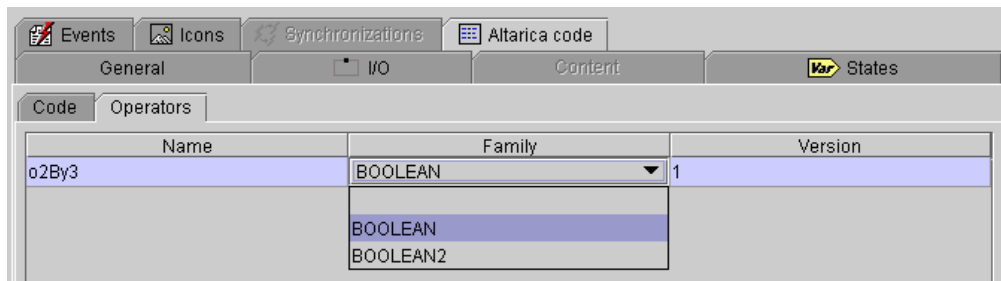


Figure 67 : AtlaRica Code - Operator Tab

1. Click on button **Syntax** (Figure 66), to check syntax of AltaRica code. If a syntax error is detected, an error window is displayed and describes precisely the kind of error and the line containing the error.
2. The most common syntax errors are:

Assertions define affectation with variables having a type incompatible (enumerate with Boolean, ...)

Syntax of expressions between brackets is not respected.


3. Click on button **Consistency (Error! Reference source not found.)** to check consistency of AltaRica code. If a consistency error is detected, an error window is displayed and describes precisely the kind of consistency error and the line containing the error.
4. The most often consistency errors detected are:

Existence of a combination of operand values (input flow variables) for which the result of the operator is not defined.

Existence of a combination of operand values (input flow variables) for which the result of the operator (output flow variable) has two different values.

- ✎ When the new operator is saved, it is saved in the version 1 and it depends on the user group of the family it belongs to.
- ✎ When the new operator is saved, it is automatically inserted in the operator library (by alphabetical order).

## Editing an operator

1. To edit an operator model proceed as follows:
2. Click on the Operators tab in the left part of the screen to display the operator family list,
3. Select the model (left click on the mouse),
4. Click  to access the tree structure of versions (Figure 68),

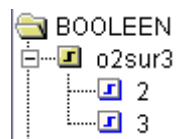


Figure 68 : Operators – Versions tree structure

5. An operator can be edited by two ways :
  - Double click on the version number
  - Select the command Edit model in the menu Library or use Edit model command in the contextual menu with a right click.
6. The operator edition window is displayed (Figure 64).


---

## Renaming an operator

1. In the **Operators** tab, click on the model to rename.
2. In the contextual menu, click on **Rename model** command; the name field of the operator becomes available.
3. Enter the new model name and validate with the **Enter** key.

★ **When an operator is renamed, the Altarica code must be modified because the output flow is identified by the operator name.**

Moreover, modifying name of an operator can have an impact on component models using this operator.

-  If the name already exists, an error message is displayed. Click on OK button and enter another name, then validate with the Enter key.

## Duplicating an operator

1. Click on the **Operators** tab in the left part of the screen to display the operator family list.
2. Select the model to duplicate with left click,
3. Open the contextual menu. with a right click on the model,
4. Select the command **Model ... → Duplicate**
5. The default name can be possibly changed

★ **If the model has several versions, only in the last version is duplicated. To duplicate another version of the operator, it is necessary to select the version in the model tree structure (Figure 84) before launching the command Model ... → Duplicate.**

To change the family of an operator model:

1. Copy the operator : command Model ... → Copy,
2. Paste the operator in the destination family : command Model ... → Paste,
3. Restore the original name of the operator (remove the extension « \_copy ») in the destination family.

## Creating a new version of an operator


To create a new version of an operator model:

1. Click on **Operators** tab in the left part of the screen to display the operator family list,
2. Select the model with left click,
3. Click on the operator name with right click to display the contextual menu
4. Select Model ... → New version command.

When the new operator version has been created, it is inserted in the versions tree structure (**Error! Reference source not found.**).

## Freezing a version of an operator

★ **This operation is very risky and must be used with extreme care, because it is irreversible: any ulterior modification becomes impossible on a frozen model.**


1. Click on the **Operators** tab in the left part of the screen to display the operator family list,
2. Select the operator model (left click),
3. Click  to display the versions in the tree structure of the selected model,
4. Select the version number (left click),

5. Display the contextual menu with a right click,
6. Select the **Model ... → Freeze** command.

 The frozen versions are represented by the symbol:  in the model library. This Frozen attribute can be also visualized in the property editor (**Error! Reference source not found.**): Properties command in the contextual menu or from the general menu File.

## Deleting an operator version

★ **This operation is very risky and must be used with extreme care, because it is irreversible.**

1. Click on **Operators** tab in the left part of the screen, to display the operator family list.
2. Select the model (left click),
3. Click on icon  to display the version list of the model (Figure 68),
4. Select the version number,
5. Display the contextual menu with right click,
6. Select **Delete model** command; a window indicates that the operator will be removed.
7. Click on **Continue** to confirm deletion, or on **Cancel** to abort deletion.




---


# Working with Types

## Creating a type

### Creating an enumerated type

Creating an enumerated type in library allows avoiding defining several times a same type for state variables and flowing variables

1. Select in library the command **Create new type... Enumerate** or,
2. Click on **Types** tab to display the list of types in the left part of the screen.
3. Right click to display the contextual menu or select the **Library** menu.
4. Select **Create new type...- Enumerate** command.
5. Enter a name for the new type in the **Type name** field (Figure 69)
6. In the **Name** field at the bottom of the window, enter a string value.
7. Click on  icon or use the **Enter** key, the variable is added in the list.
8. Click on **Save** and **Close** buttons.

 By default, a new type is created in version 1 and is linked to the family user group.

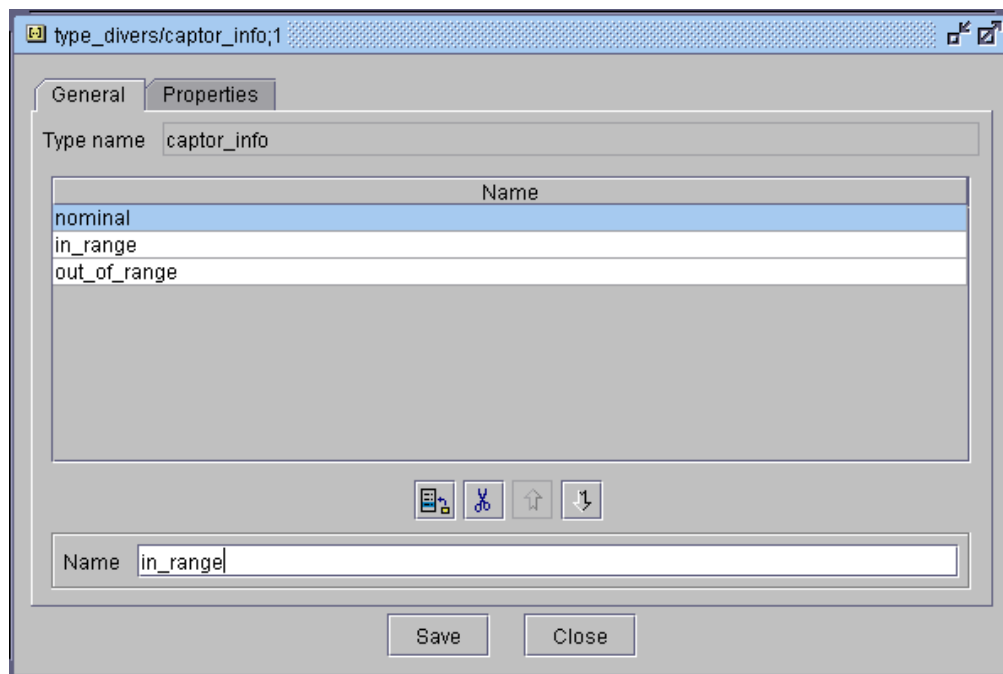




Figure 69 : Create enumerate type


 In the Types tab in the left part of the screen, the enumerate type is represented by the symbol .


## Creating a structured type

The structured type (or Record) allows including several flow information (record field) in a same link. In a model, the enumerate type is a list of possible values for a variable (state or flow) whereas a structured type is a list of flow variables.

A structured type can be compared to a bus allowing reducing the number of graphic links in equipments and architecture systems. Only the flow variables (**in** or **out**) can be declared with a structured type

To create a new record type:

1. Click on **Types** tab to display the list of types in the left part of the screen.
2. Select a family
3. Right click to display the contextual menu or use the **Library** menu.
4. Select Create new type...- Record command.
5. Enter a name for the new type in the **Type name** field.
6. In the **Name** area at the bottom of the window, enter a name for the record field.
7. Select the **Type** menu and choose a type.
8. Click on the  icon or use the **Enter** key, to add the record field in the list.
9. Click on the **Assign** button to modify the type, if necessary; the type is displayed in the **Type** column.
10. Click in **Orientation** column; select the orientation: **normal** or **inverse**.
11. Click in the **Crossfield** column to define a flow crossing (Figure 71)
12. Enter the cross field and validate with the **Enter** key.

 A new type is created in version 1 and is linked to the family user group.

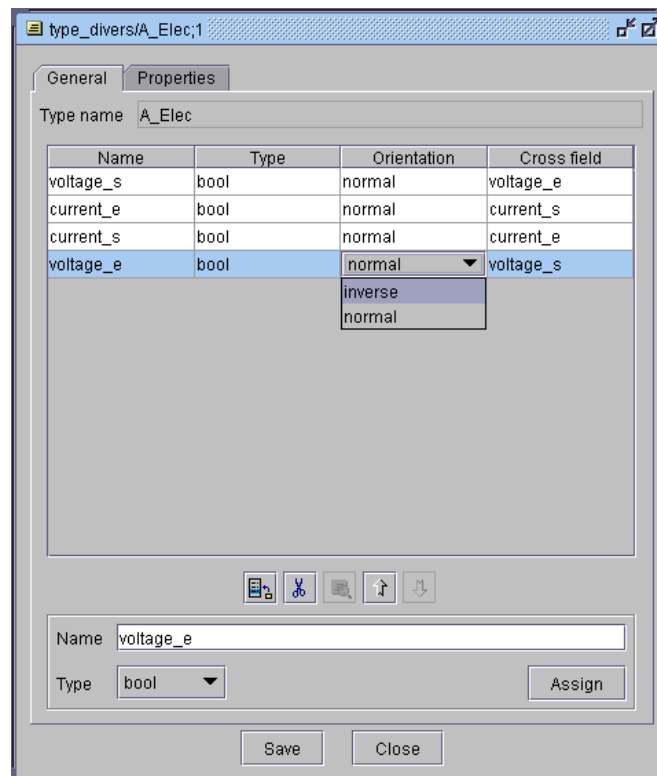


Figure 70 : Create flow structured type

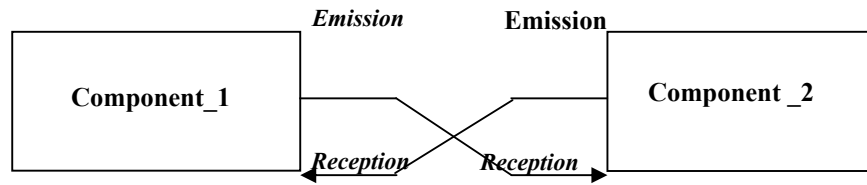


Figure 71 : Cross field for Inputs/Outputs

The definition of the structured type flow generates two connection studs:

1. a port in,
2. a port out.

Each connection stud is composed of n fields (flow parameters=field).

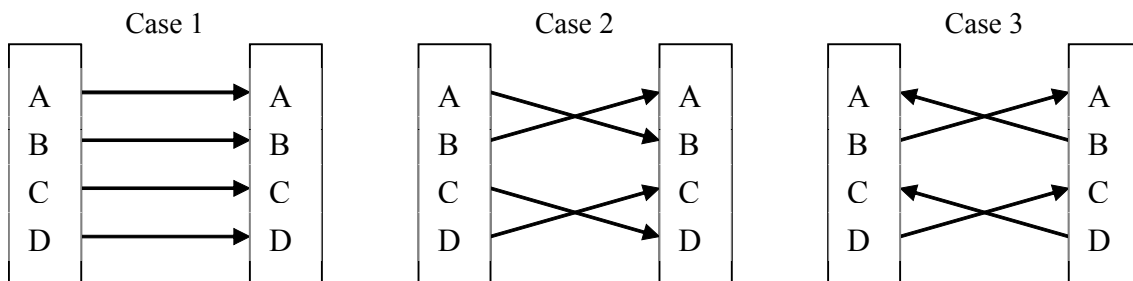
In general the field has the same orientation as its reference stud. The clause `inverse` allows specifying if the fields of the stud `in` or stud `out` have an inversed orientation.

The figure below shows the various relations of possible equality between the fields of the stud `out` and stud `in` according to the cross fields and orientations defined for the flow parameters (A, B, C, D).

Case 1 : A data structure with direct assignments,

Case 2 : A data structure with cross field assignments,

Case 3 : A data structure with inversed and cross field assignments.



In order to simplify the code interpretation presented below, the equality relations are oriented and thus defined using the assignments.

To access a field, use the character '^'.

```





link
  flow A,B,C,D : int ;
  assert
    in^A := out^A;
    in^B := out^B;
    in^C := out^C;
    in^D := out^D;
knil

link
  flow A,B,C,D : int ;
  assert
    in^A := out^B;
    in^B := out^A;
    in^C := out^D;
    in^D := out^C;
knil




link
  flow A,B,C,D :
  int ;
  inverse out^A;
  out^C;          in^B;
  in^D;
  assert
    in^A := out^B;
    out^A := in^B;
    in^C := out^D;
    out^C := in^D;
knil

```

The following icons are used for creating a type:

1.  : to delete an Input/Output,
2.  : to edit a parameter type if it is predefined (or double click on the **Type** cell),
3.  : to move up the selected line in the list of Inputs/Outputs,
4.  : to move down the selected line in the list of Inputs/Outputs.

These commands are also available in the contextual menu with a right click.

-  In the Types tab in the left part of the screen, the record type is represented by the symbol .
-  In the list of “predefined” types, only the list of enumerate types is displayed (the list of record types is not displayed) because a record type cannot contain another record type.

## Editing a type


1. Click on the **Types** tab in the left part of the screen to display type family list,
2. Select the chosen model (click left),
3. Click  to access the tree structure of versions (Figure 72),



Figure 72 : Types – Tree structure of versions

4. A type can be edited by two ways :
  - Double click on the version number
  - Select the command Edit model in the menu Library or use Edit model command in the contextual with a right click.
5. The type edition window is displayed (Figure 69).

## Renaming a type

- ★ **This function is very risky and must be used with extreme care, because the renaming leads to a reference change of the model in models using the type.**

- 
1. In the **Types** tab, click on the model to rename.
  2. In the contextual menu, click on **Rename model** command; the name field of the type becomes available.
  3. Enter the new model name and validate with the **Enter** key.
- ✏ If the name already exists, an error message is displayed. Click on OK button and enter another name, then validate with the Enter key.

## Duplicating a type

1. Click on the **Types** tab in the left part of the screen to display the type family list.
  2. Select the model to duplicate with left click,
  3. Open the contextual menu with a right click on the model,
  4. Select the command **Model ... → Duplicate**
  5. The default name can be possibly changed.
- ★ **If the model has several versions, only in the last version is duplicated. To duplicate another version of the type, it is necessary to select the version in the model tree structure (Figure 72) before launching the command Model ... → Duplicate.**

To change the family of a type model:

1. Copy the type : command **Model ... → Copy**,
2. Paste the type in the destination family : command **Model ... → Paste**,
3. Restore the original name of the type (remove the extension « \_copy ») in the destination family.

## Creating a new version of a type

1. Click on **Types** tab in the left part of the screen to display the type family list,
2. Select the model with left click,
3. Click on the type name with right click to display the contextual menu (Figure 56)
4. Select **Model ... → New version** command.
5. A bar graph (**Error! Reference source not found.**) indicates that the new version is being created.

When the new type version has been created, it is inserted in the tree structure of versions (Figure 72).

## Copying a type


1. Click on the **Types** tab in the left part of the screen to display the type family list.
2. Select the model to copy with left click,
3. Open the contextual menu with a right click on the model,
4. Select the command **Model ... → Copy**
5. A bar graph indicates that the type is being copied;
6. Select the new target family for the copy.
7. Select the **Paste** command.



- ★ **If the type has several versions, the copy is possible only for the last version.**

## Freezing a version of a type

- ★ **This operation is very risky and must be used with extreme care, because it is irreversible: any ulterior modification becomes impossible on a frozen model.**


1. Click on the **Types** tab in the left part of the screen to display the type family list,
2. Select the type model (left click),


3. Click  to display the versions in the tree structure of the selected model,
4. Select the version number (left click),
5. Display the contextual menu with a right click,
6. Select the **Model ... → Freeze** command.

 The frozen versions are represented by the symbol:  2 in the model library. This Frozen attribute can be also visualized in the property editor: Properties command in the contextual menu or from the general menu File.

## Deleting a type

★ **This operation is very risky and must be used with extreme care, because it is irreversible. The component is deleted and all its associated links in equipments and architectures using it.**

1. Click on **Types** tab in the left part of the screen, to display the type family list.
2. Select the model (left click),
3. Click on icon  to display the version list of the model
4. Select the version number,
5. Display the contextual menu with right click,
6. Select **Delete model** command; a window indicates that the type will be removed.
7. Click on **Continue** to confirm deletion, or on **Cancel** to abort deletion.

1. Select the model, click on the second icon , the search model and the number of found instances appear at the bottom (result of search).
2. Double click on the model to unroll the list of model instances in the system. It provides information about localisation (in H\_ELEC/Poste\_EDF; 1).
3. Select an instance, click on the **Locate** button in order to view on this instance.

## Setting Link colors

1. Open a system architecture.
2. Use the **Views – Colours** menu to define the links colour in a system during simulation.
3. The Links colours window (Figure 73) is displayed: it contains in its right part a colour palette, and in the left part, an area displaying the link colours according to their type.
4. When a new architecture is created, the boolean type is the default type with two colours (Figure 73):

False: red,

True: green



Figure 73 : Boolean variable colors

If a new architecture contains links (non-boolean) which must be coloured, follow the procedure:

Enter a type in the text edit zone (enum, bound or predefined).

Click on the  icon or use the Enter key to add the type in the list.

E.g.: To select a predefined type variable, enter *pressure* in the enum field; the following window is displayed (Figure 74) :




Figure 74 : Link color

Double click on the type in the list to display the possible values, e.g., *pressure*; the four possible values for pressure are displayed:




- High,
- Low,
- Normal,
- Null,

Select a value, e.g. Normal.

In the colour palette, select a colour; the grid on the right indicates the last selected colours.


Click on the  icon to assign the color to the value;

Repeat the same steps to define the colours for the three other values.


-  The color assignment is specific for a system; the colors are saved with the system.
-  An existing enumerated type can be deleted by clicking on the corresponding icon.
-  When a link is of the record type, the colors displayed during the simulation correspond to the first flow field.

## Searching for a Model

Search a component as follows:

1. Use the **Edition – Search** menu or type the **CTRL + B** command on the keyboard or use the  icon in the *Standard* tool bar; the *Search* window (Figure 75) is displayed.
2. Click on the **Component** tab and in the *Filter* field, enter the component name as follows :  
<Family\_Name>/<Model\_Name>

NB: the character « \* » can be used to decrease the search space (example: H\*/val\*) or to complete a partially-known-name (examples: HYD\*/val\*, \*/valve).

3. Click on the  icon, the component found (or the components having a name corresponding to the filter used) is displayed in the middle part of the window



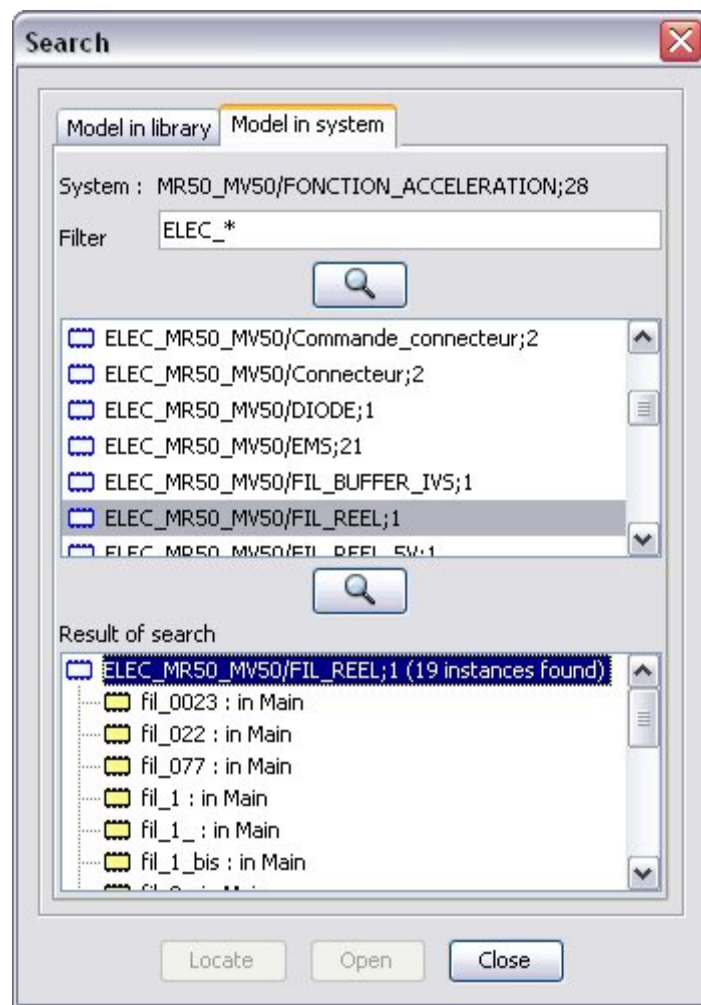


Figure 75 : Search window

# Printing

## Setting Page Layout

1. Select “File” -> “Print Setup”; the Page Setup opens.
2. The displayed; in the Paper area, select:
  - The size (A4 or A3),
  - The source (manual feeding or automatic selection).
3. In the Orientation area, tick the *Portrait* or *Landscape* mode.
4. Each format or orientation modification automatically updates the scale factor of the page displayed.
5. In the Margin area, enter the margins (left, right, up or down).
6. Click on “OK” to save the new printing format, or click on “Cancel”

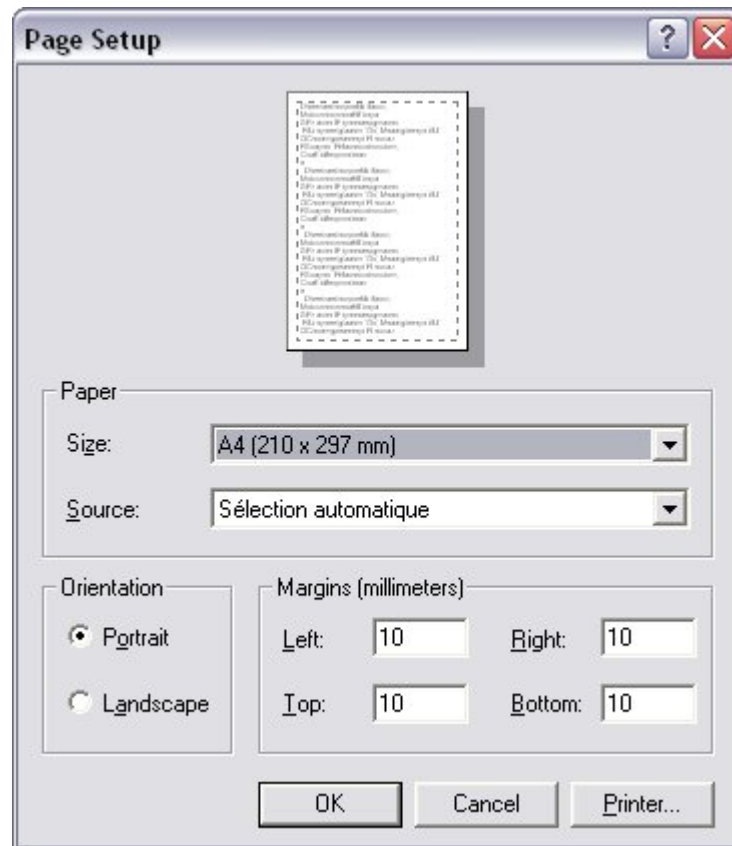
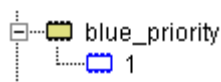



Figure 76: Page Setup

## Printing a Model


1. Select the model version you want to print in the library:

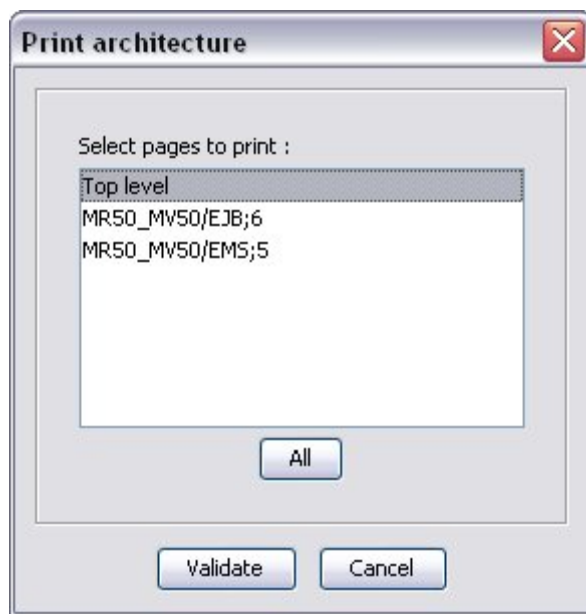


2. Edit model by right-clicking on the version, select **Model – Edit** in contextual menu
3. Select the **File – Print** menu or click on icon .

---

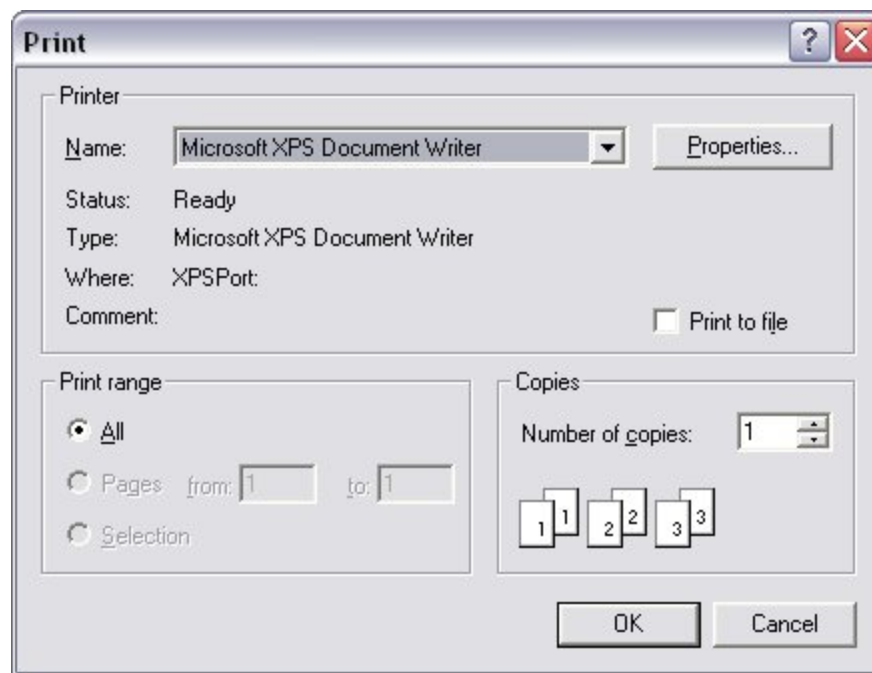
Print a system as follows:

1. Select the system version you want to print.
2. Open **System** (double-click)
3. Select the **File – Print** menu or click on icon  to print selected system (Figure 77);
4. Select the pages to be printed.



*Figure 77 : Print architecture window*

5. Click on the **Validate** button, or click on the **Cancel** button to abort the printing.
6. The *Printing* window is displayed (Figure 78).
7. In the *Printing area*, select the area to be printed (all, pages *n* to *m* or selection).



*Figure 78 : Printing window*

- ✎ In the Preferences window – Options tab, it is possible to select a file containing a logo which will be added.

---

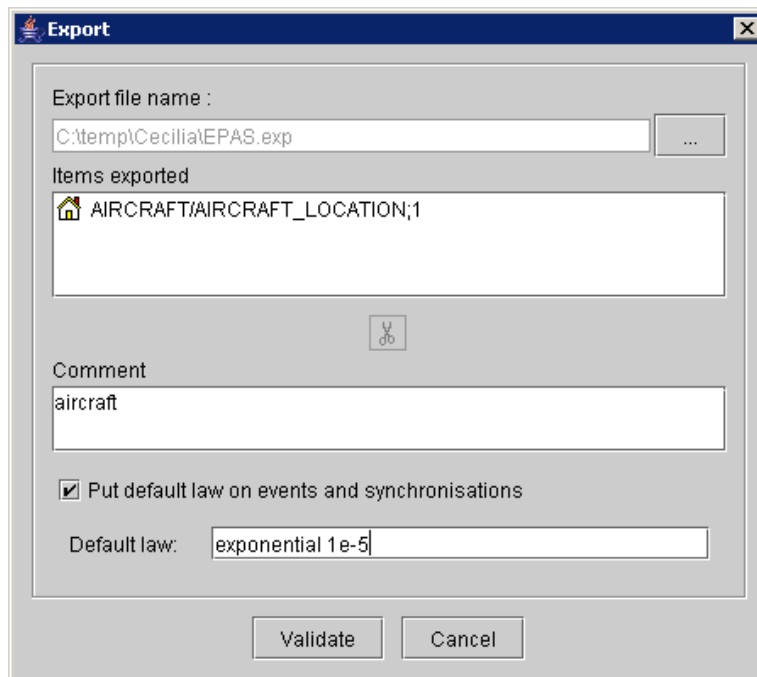
# Exporting & Importing

In order to facilitate data exchanges, Safety Designer allows export and import of objects (family, project, models, and systems) in XML format.

## Exporting

To export data, proceed as follows:

1. Activate the export setup window (Figure 79) with the **Export** command in **File** menu.



*Figure 79 : XML export setup*

2. Click on “...” button in **Export file name** area,
3. Chose the export file name and the directory,
4. add elements to export. Select an element in the library and use the **File/Add to Export** menu, or drag and drop the object in **Items exported** area.
5. The **Comment** area allows to add a description.
6. Tick the box **Put a default law on events and synchronizations** to add a default law on events and synchronizations that have no law. Use the **Default law** area to specify the default law
7. Click on **Validate** button to export or on **Cancel** to abort exportation.

★ **Frozen and locked models cannot be exported.**

## Importing

To import data from an XML import file (.exp extension) generated by Safety Designer, proceed as follows:

1. Display the import window with the **Import** command of **File** menu.

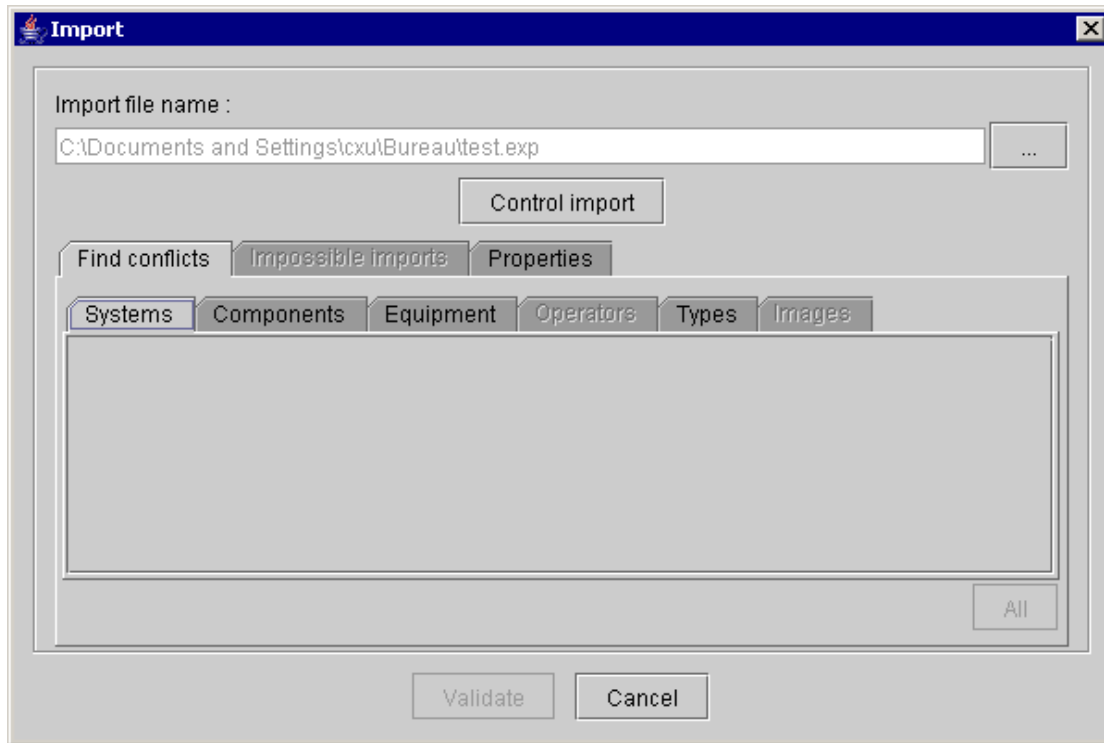


Figure 80 : XML Data Import



2. Click on the browser button “...” in **Import file name** area,
3. Choose the export file (extension .exp) containing data. The **Properties** tab display properties associated to the imported data (creation date, comment, and owner).
4. Click on **Control Import** to launch validity check. This check do the following operations:
  - XML Format validity Check. Detected errors are displayed as “impossible imports”.

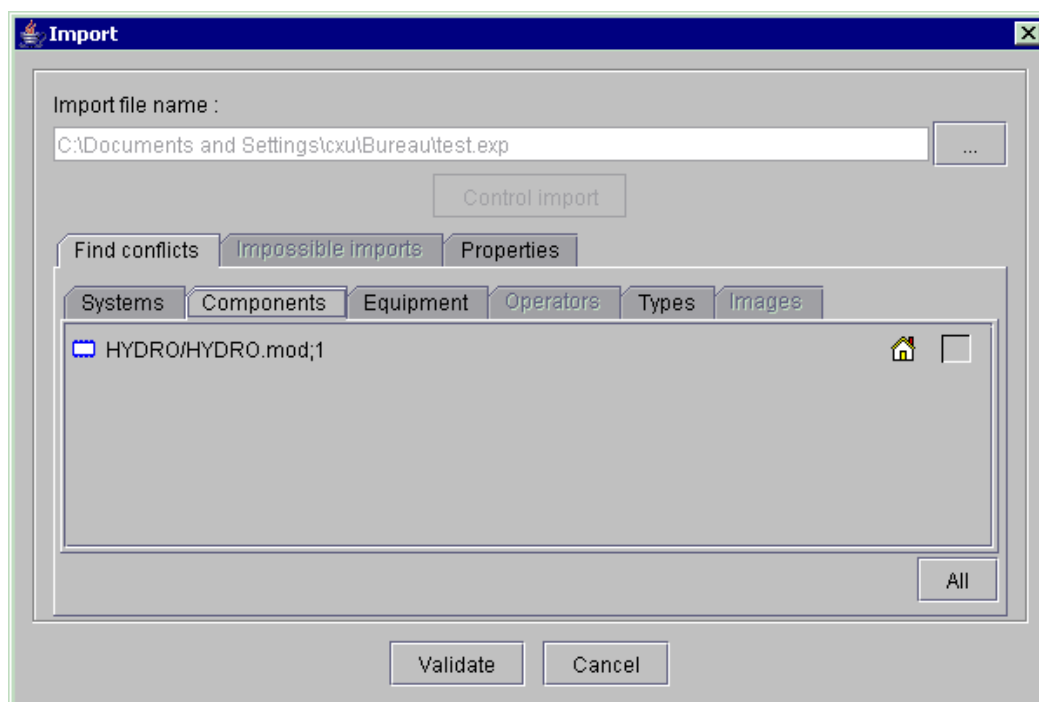
★ **Only ascendant compatibility is ensured.**

## Detecting conflicts when importing data

This detection is done essentially by comparing the last modification date of each imported element with the model in database.

Detected conflicts are displayed in the following tabs: Systems, Components, Equipments, Operators, Types, and Images. By default, data in database are not updated during data importation.

Nevertheless, the user can specify, for each conflict, a possible update of the database by clicking on   (Figure 81). The All button allows updating (importing) all models in conflict displayed in the tab.



*Figure 81 : XML Import – Detected conflict*

1. Click on **Validate** to import or **Cancel** to abort importation.

# Managing Plug-ins

## Introduction

The plugin manager allows the user to manage the compute plugins. The compute plugins are processing units taking in input a model generated in Safety Designer. According to the treatment made, the user will obtain on output event minimal cuts, event sequences, a step by step simulator by inference rules or a Java step by step simulator.

A user can create new plugins and add them inside Safety Designer by using the plugin manager. This chapter does not show the plugin creation process. This theme is developed in details in another document.

From their use point of view, the user sees the plugins like treatment units contained in JAR files (in Java). A JAR file can contain several plugins. These files are read by the plugin manager in order to load the plugins into the application during its launch.

## General

The plugin manager window contains three tabs which are: Plugins, Actions and Items (Figure 82):

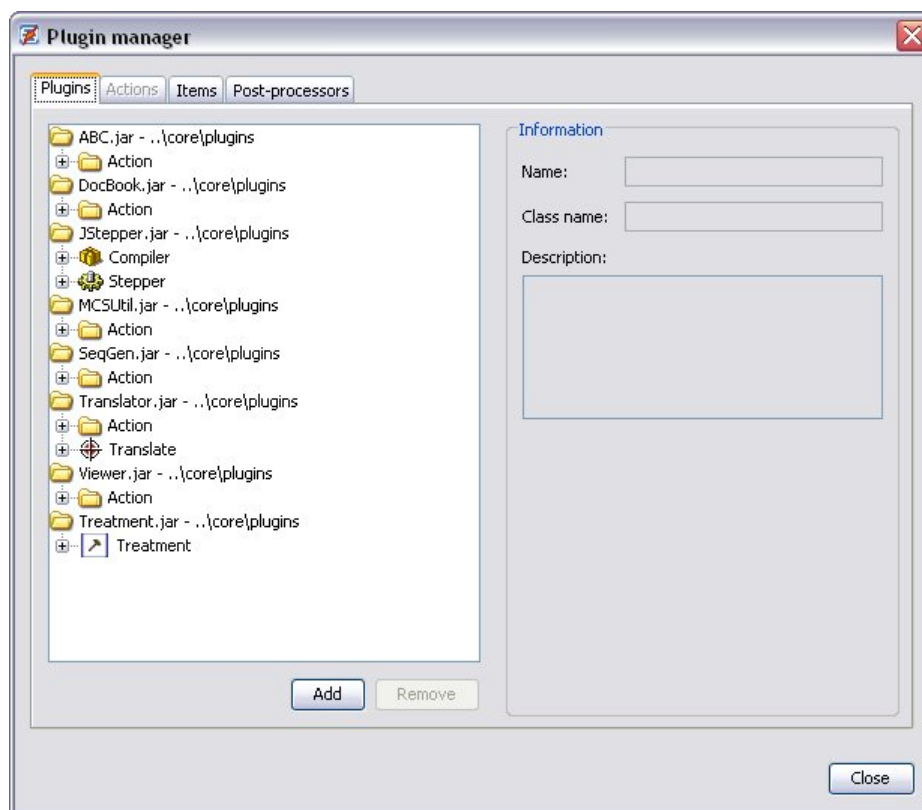


Figure 82 : Plugin manager

The Plugins tab allows the user to manage the plugins. The Actions tab allows adding actions associated to the plugins and the Items tab allows inserting inside the application user interface, plugin linked action launch point.



---

# Setting Plug-ins Preferences

The Plugins tab allows the plugin management inside Safety Designer. Through this tab the user can add, remove plugin to/from the application and also view the plugin properties.

## Plugin organization

Plugins are organized in a tree structure. The first level shows the different JAR files from which the plugins are loaded into the application. The plugin paths are expressed in relative from the location of the file named “plugins.xml” (Figure 83):

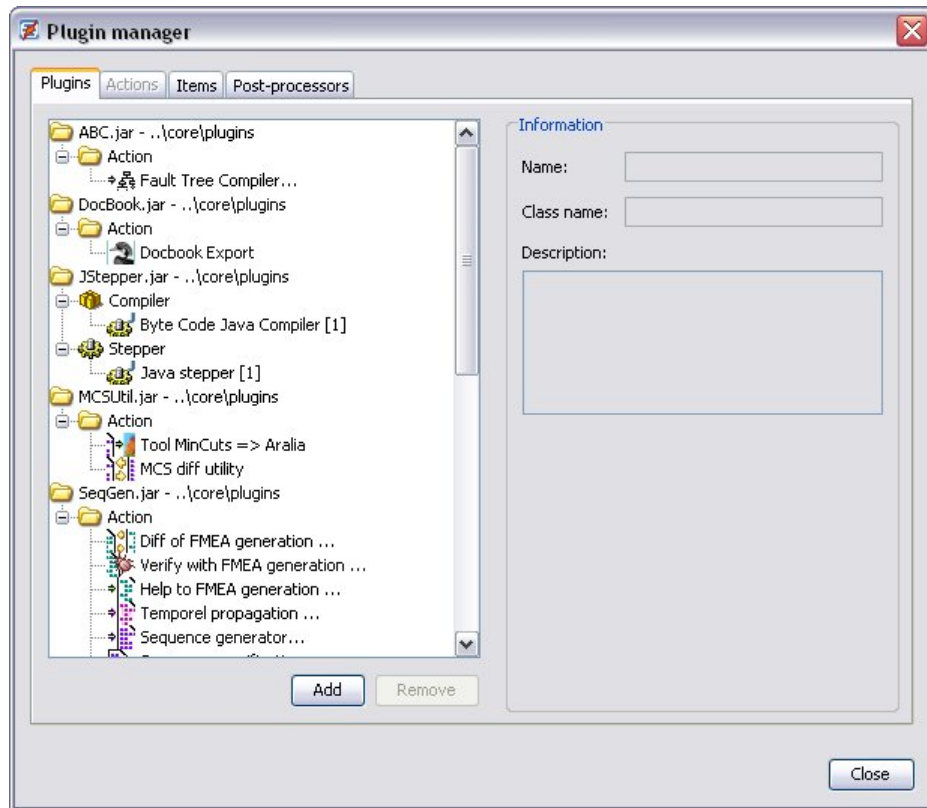


Figure 83 : Plugin manager – plugin tree structure

The second level contains Action, Translate, Compiler and/or Stepper nodes. The nodes allow classifying plugins into categories.

The Action nodes gather plugins that can be directly executed by the user if associated to an action and a launch point in the application user interface.

The Translate nodes gather plugins used in treatment flows and are dedicated to the translation of some languages into other languages.

The Compiler nodes gather plugins allowing compilation of models which accelerate its use by the simulation engine.

The Stepper nodes gather the plugins that implement simulation engines.

## View plugin informations

In order to visualize plugin related information, the user selects a plugin in the tree structure. The information then appears in the right part of the tab (Figure 84):

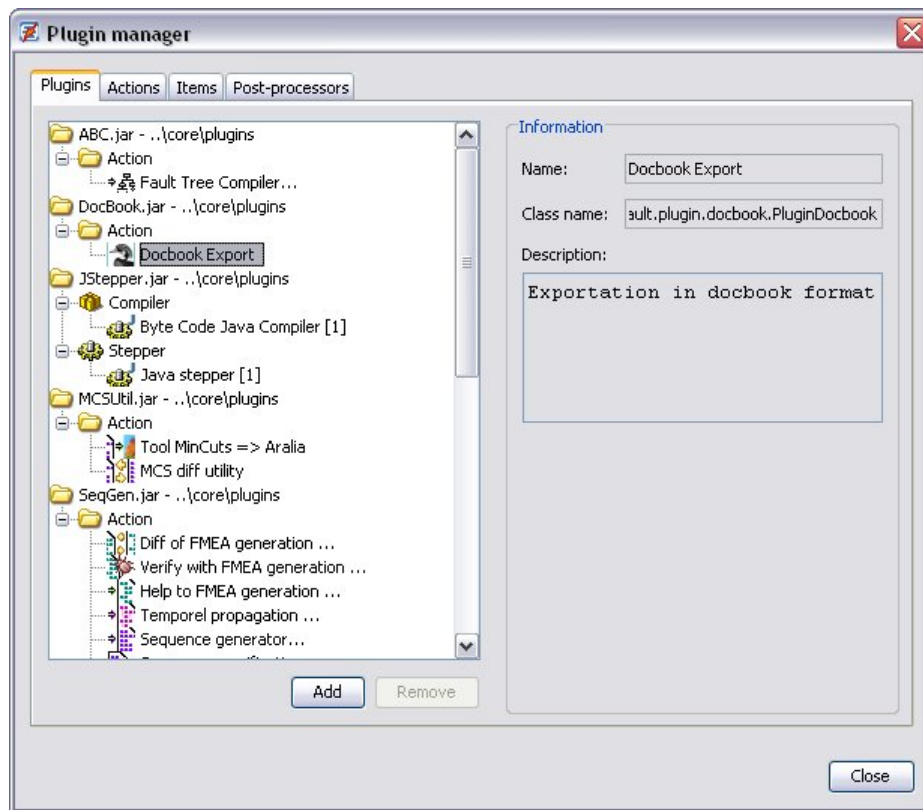


Figure 84 : Plugin manager – Plugin information area

A plugin has a name, a class name and a description. This information is accessible to the user but is not editable for write because it is to the plugin designer responsibility.

The class name is the complete name of the implementation Java class. It is the plugin entry point.

## Add JAR files

Plugins are only accessible throughout JAR files. The user load JAR files containing plugins and not plugins directly.

1. Click on the **Add** button; a file explorer dialog opens
2. Select the JAR files containing the plugins that are to be integrated into the application,
3. Click on the **Open** button.

The plugins are registered in the application and then can be referenced by plugin actions.

## Remove JAR files

1. Select JAR files nodes that are to be deleted,
2. Click on the **Remove** button; a confirmation dialog opens

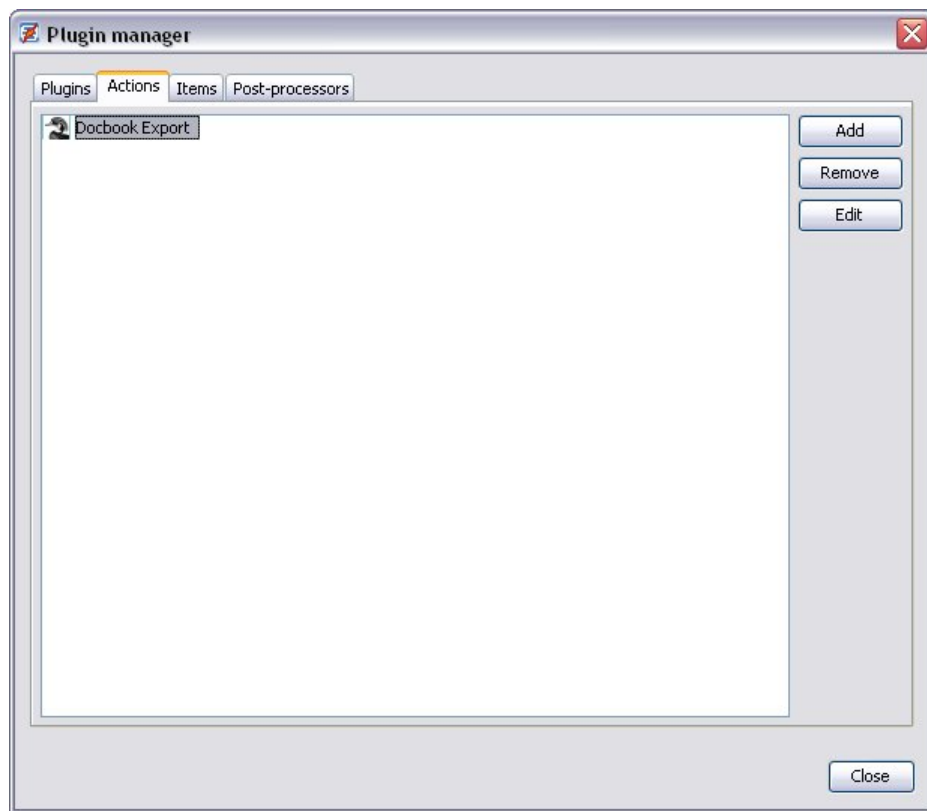
3. Click on the **Continue** button to confirm the JAR files deletion or click on the **Cancel** button to cancel the action.

If the user clicks on the Continue button, the selected JAR files nodes and all their content are removed from the tree structure.

## Editing Actions

The actions are plugin instantiations in the sense that they allow using the plugins with distinct parameters. The actions only apply to the Action plugins and not to the others categories. When the user select a plugin of Translate, Compiler or Stepper type, the Action tab become inaccessible.

The Action tab appears as follows (Figure 85):



*Figure 85 : Plugin manager –Actions tab*

## Adding plug-in actions

1. Select if necessary the **Plugin** tab and then, in the tree structure, the plugin of **Action** type on which the new action is to be created,
2. Select the **Action** tab,
3. Click on the **Add** button; the following window then appears (Figure 86):

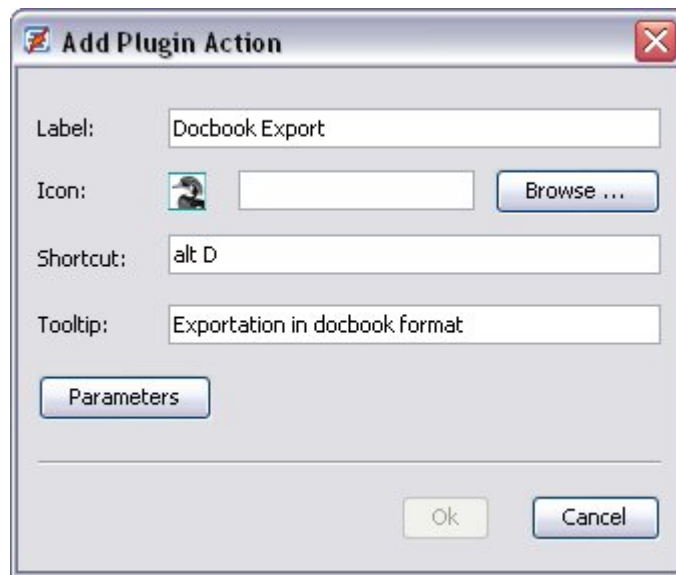



Figure 86 : Add a plugin action

4. Fill the dialog box fields in,
5. Click on the **Ok** button to confirm the plugin action creation or on the **Cancel** button to do nothing.

If the user clicks on the Ok button, the new action is added to the already existing actions list.

-  A default label fill the corresponding field when the Add a plugin action dialog box is shown to the user. This label is provided by the plugin. It is the same for the Icon, Shortcut and Tooltip text fields.

Two different actions associated to the same plugin cannot have the same label. When an already existing label is entered in the corresponding field, the Ok button is then disabled.

The Parameters button give write access to the plugin parameter list when the plugin has parameters. These parameters and their viewing are provided by the plugin designer.

## Removing plug-in actions

1. Select the **Actions** tab if needed,
2. Select the action nodes ,
3. Click on the **Remove** button; a confirmation dialog opens
4. Click on the **Continue** button to confirm the actions deletion or on the **Cancel** button to do nothing.

If the user clicks on the Continue button, the selected actions then disappear from the action list.

## Editing plug-in action

1. Select if necessary the **Action** tab and the action to edit,
2. Click on the **Edit** button; the following window then appears (Figure 87):



*Figure 87 : Edit a plugin action*

3. Modify the fields in the dialog box,
4. Click on the **Ok** button to confirm the modification or on the **Cancel** button to cancel the operation.

## Inserting Plug-in Actions in Menus

### Plugin items overview

The Items tab allows the plugin item management. Note that the plugin items are user interface graphical objects like buttons or menu items that provide a launch point for the Action typed plugins. By the other, a plugin item action is an action defined or predefined in the Action tab. When created on a tool bar, the plugin item is a button and created on a menu bar, the plugin item is a menu item.

The Items tab is shown bellow (Figure 88). The Items tab is divided in two parts: a first part that shows all the application menu items and toolbar buttons in a tree structure view and a second one that gather the commands accessible on the plugin items.

By navigating in the tree structure representation, the user can for instance reach a sub-menu in order to add a plugin launch point. The user has also the possibility to create other type of objects like separators or new menu in existing ones. All the supplementary graphic objects appear in the tree structure with bold text font.

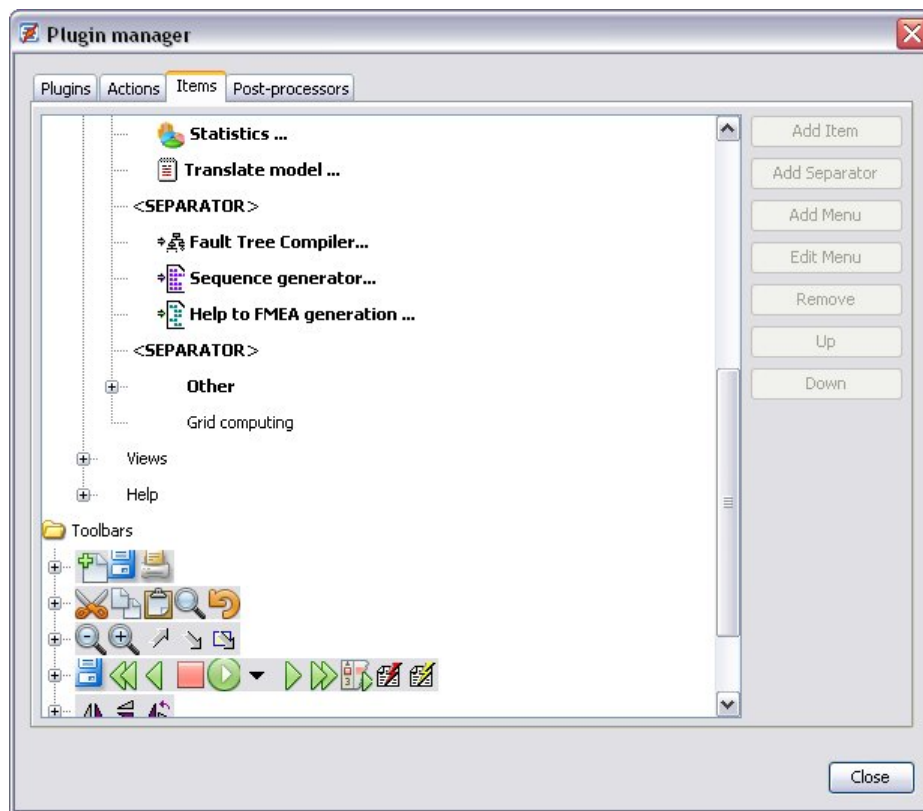


Figure 88 : Plugin manager –Items tab

## Adding plugin item

To add a plugin item, the user proceeds as follows:

1. Select the **Plugins** tab then a plugin of Action type if necessary,
2. Select the **Action** tab then an action in the list if necessary,
3. Select the **Items** tab if necessary,
4. Navigate in the tree structure until the insertion location,
5. Click on the **Add Item** button.

The plugin item is then created and added to the tree structure and in the application menubar or toolbars. When the selected object for the item insertion is a menu, the item is inserted at the last position in that menu, when it is a menu item the insertion is done at the preceding position of that menu item.

## Adding separators

1. Select the **Items** tab if necessary,
2. Navigate in the tree structure until the insertion location,
3. Click on the **Add Separator** button.

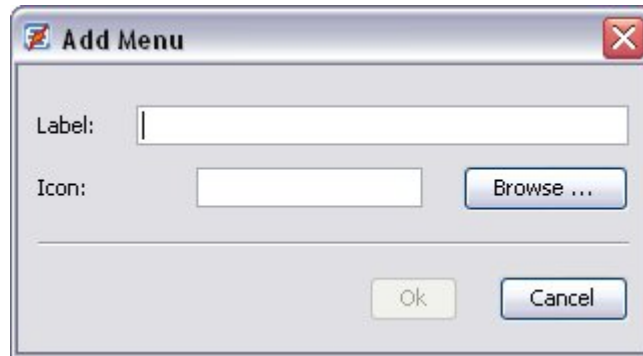
The separator is then created and inserted in the selected location. When the selected object for the insertion is a menu, the separator is inserted at the last position in that menu, when it is a menu item the insertion is done at the preceding position of that menu item. Separators created by the user have all the same name:

<SEPARATOR>

---

## Adding customized menus

1. Select the **Items** tab if needed,
2. Navigate in the tree structure until the insertion location,
3. Click on the **Add Menu** button, the following dialog box then appears (Figure 89):



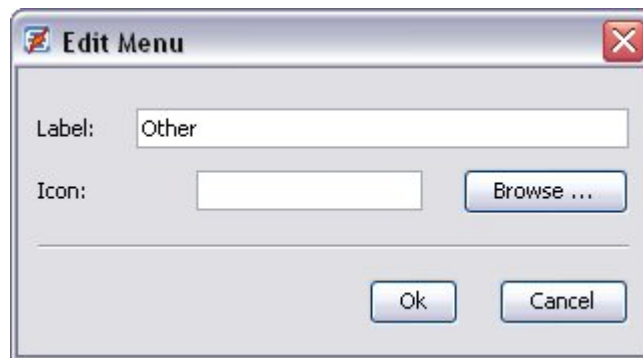
*Figure 89 : Add menu*

4. Fill the label field in and if needed an icon path in the corresponding field,
5. Click on the **Ok** button to validate the creation or on the **Cancel** button to cancel the operation.

If the user clicks on the Ok button, the newly created menu is added at the selected location. When the selected object for the insertion is a menu, the newly created menu is inserted at the last position in that menu, when it is a menu item the insertion is done at the preceding position of that menu item.

## Editing customized menus

1. Select the **Items** tab if needed,
2. Select the menu to edit in the tree structure,
3. Click on the **Edit Menu** button, the following dialog box then appears (Figure 90):



*Figure 90 : Edit menu*

4. Modify the fields,
5. Click on the **Ok** button to validate the modification or on the **Cancel** button to cancel the operation.

If the user clicks on the Ok button, the menu is then modified.

## Remove graphic item

Only the graphic elements added by the user are removable. It can be plugin item, supplementary separator or supplementary menus.

★ **When a menu is removed, all the graphic elements under this menu are also removed.**

1. Select the **Items** tab if needed,
2. Select the graphic elements in the tree structure,
3. Click on the **Remove** button; a confirmation dialog opens
4. Click on the **Continue** button to confirm the graphic element deletion or on the **Cancel** button to do nothing.

If the user clicks on the Continue button, the selected graphic elements then disappear from the tree structure and from the application user interface (menu bar and toolbar).



---

# Setting Preferences

## Opening the Preferences Panel

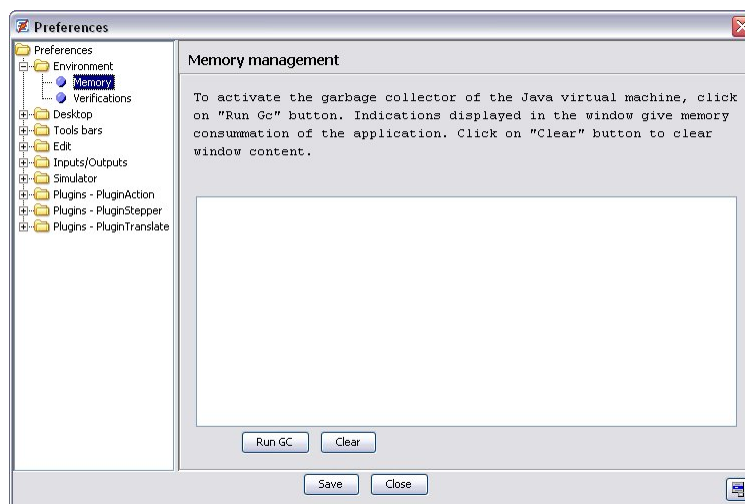
1. Select “Views” -> “Preferences...”; the Figure 91: Preferences Panel opens



*Figure 91: Preferences Panel*

## Setting Environment Preferences

- ✎ Sub-rubric Memory (Figure 92) is for maintenance only.



*Figure 92 : Memory management*

- ✎ The Save button has no effect in this rubric.

The Verification sub-rubric allows the user to choose the syntactic and consistency checking (Figure 93).

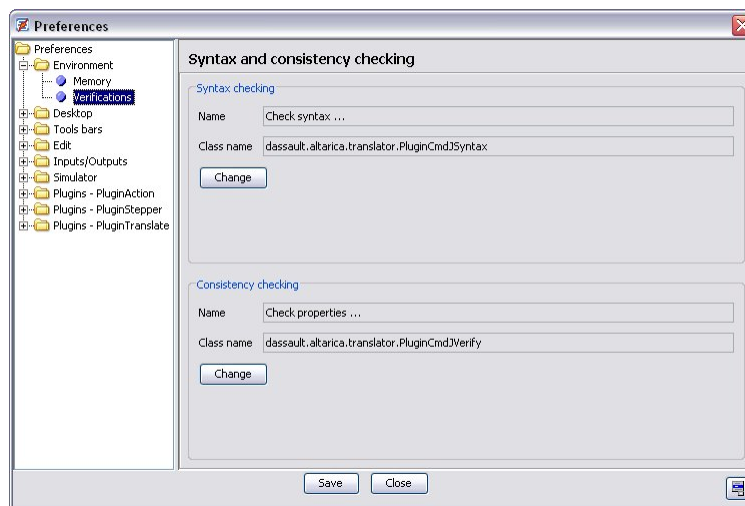


Figure 93 : Syntax and consistency checking

## Setting Desktop Preferences

The Desktop Options sub-rubric allows background choice:

1. Display time and date,
2. Display tools bars,
3. Display task bar,

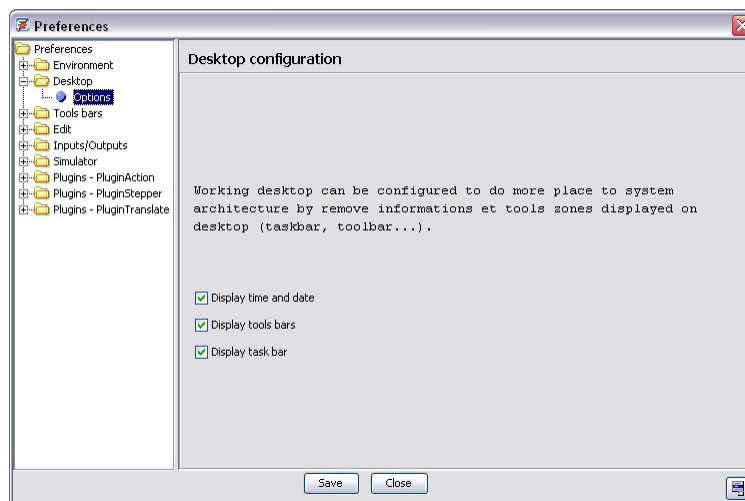


Figure 94 : Desktop options tab

## Setting Toolbars Preferences

The Tools bars tab (Figure 95) allows customization of the tools bars with the following bars:

- Standard,
- Edition,
- Design,
- Links,

- Alignment,
- Navigation,
- Simulation.

Validate by clicking on Save and then Close.

- ✎ If the number of tools bars exceeds the window width, buttons located at both ends enable access to the tools bars which are not displayed on the screen.

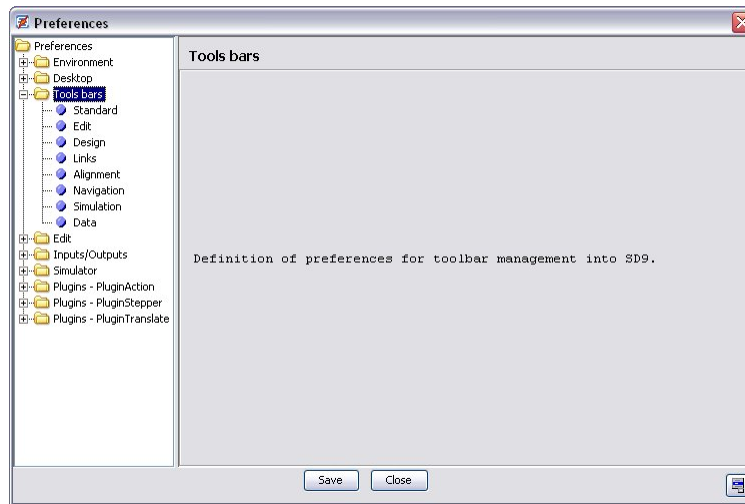


Figure 95 : Toolbars tab

## Setting Edition Preferences

### Setting Editor Font

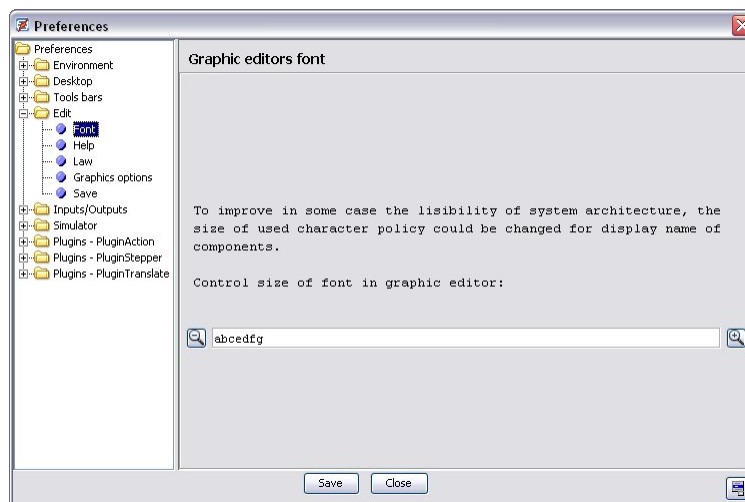




Figure 96 : Graphic editors font

In the Adjustment of font size in architecture view field, type any font size and adjust this size by means of the  and  icons.

## Setting Code Completion

Safety Designer has an edition help for writing AltaRica code by given, during data acquisition, the list of component which are present at corresponding level.

Check “activate edition helper” to activate it.

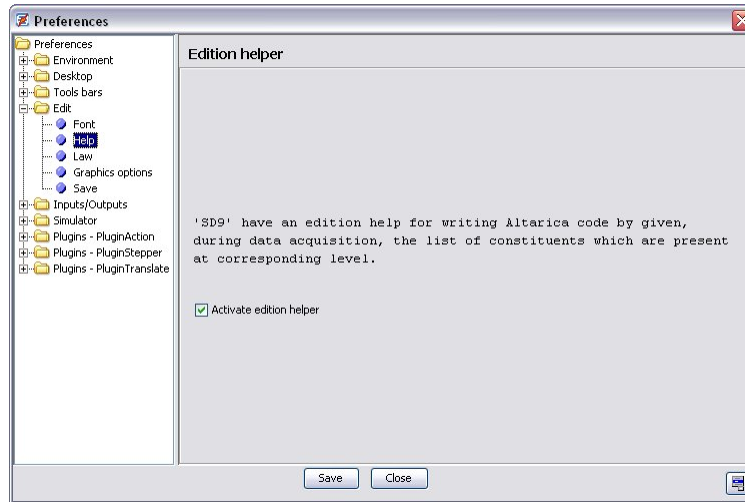


Figure 97: Activating Code Completion

## Setting Graphic Options

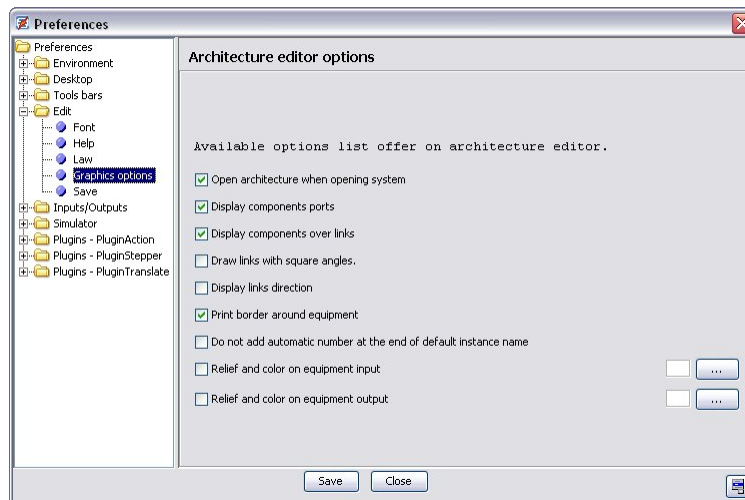


Figure 98: Graphics options sub-rubric

- The Graphics options sub-rubric (Figure 98) allows customization of the tool with the following options:
- Open architecture when opening project.
- Display components ports
- Display components over links
- Draw links with square angles
- Display links direction
- Print border around equipment.
- Relief and color on equipment input

- Relief and color on equipment output

## Setting Save Option

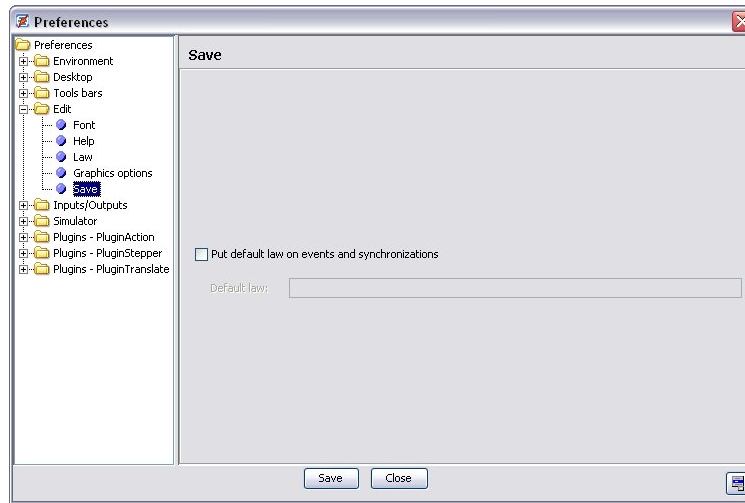


Figure 99 : Save

Check the box “Put default law on events and synchronizations” to save events and synchronizations with a default law. The default law is only put on events and synchronizations that have no law. Use the “default law” area to define this law.

## Setting Input/output preferences


The Inputs/Outputs rubric contains one sub-rubric: Printing logo (Figure 100)

In the File name field (Figure 100), type the pathname to a logo which will be printed in the page top left corner (the file format containing the logo is of the “\*.gif” or “\*.jpg” type) or use the Select button to indicate the file containing the logo.



Figure 100 : Printing Logo

Validate by clicking on the Save and Close buttons.

-  The Save button must be used only if the modifications are to be saved in the “preferences.dat” file.

## Setting Simulation Preferences

The Simulator rubric allows choosing simulator options.

Simulator behaviour depends on options that can be activated or not (Figure 101):

- Trigger Automatically event and change state off
- Number of automatic execution: to avoid endless simulation due to automatic triggering of instantaneous transition, user must specify the maximum number of automatic.

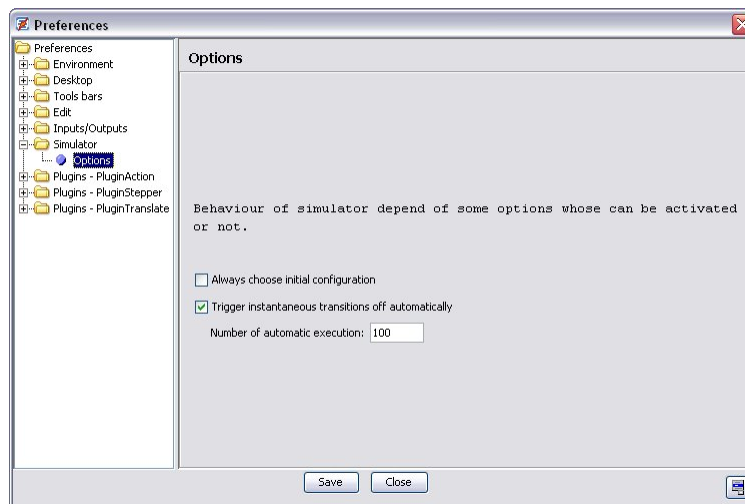
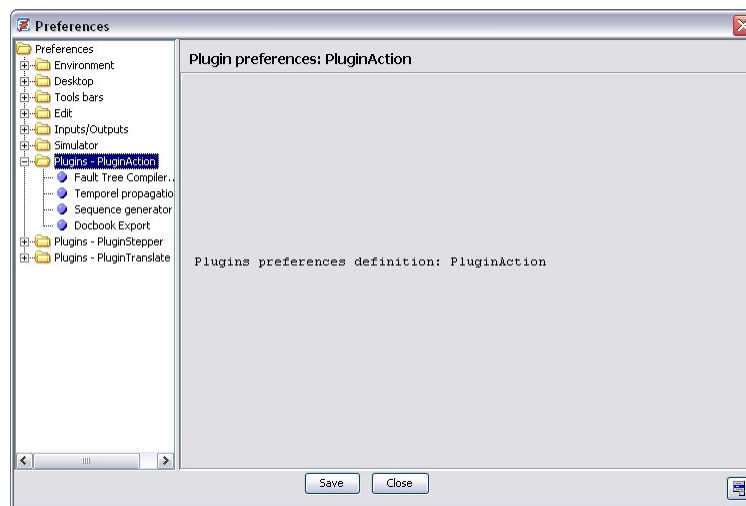


Figure 101 : Simulator Options

## Setting Plug-in Preferences

The Plugins – Plugin Action and Plugins - PluginTranslate rubrics (Figure 102) allows to choose options concerning the plugins.

For detailed explanations about these options, please see the plugin appendices.

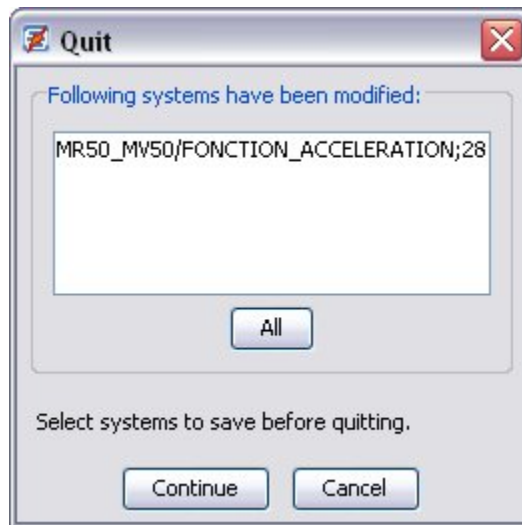


*Figure 102 : Plugin option rubrics*

# Quitting Safety Designer

- Select “File” -> “Quit”.
- Press “Ctrl+Q”.

If models have been modified, a confirmation window opens



*Figure 103 : Saving modified models*

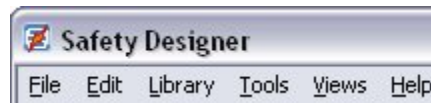


---

# Interface Description

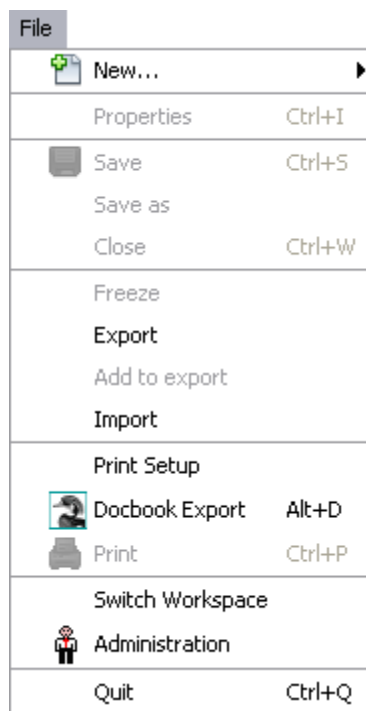
## Menu Bar

This section describes the Figure 104: Menu bar that is visible at the top of Safety Designer.



*Figure 104: Menu bar*

## File Menu



*Figure 105: File menu*

## Edit Menu

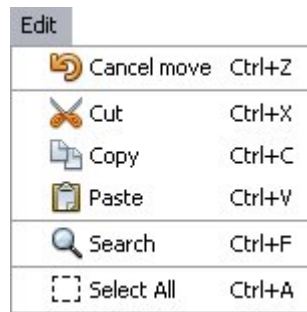


Figure 106: Edit menu

## Library Menu

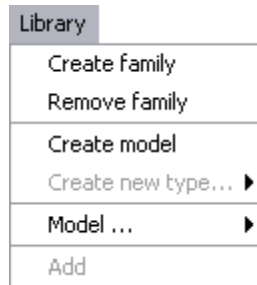


Figure 107: Library menu

## Tools Menu

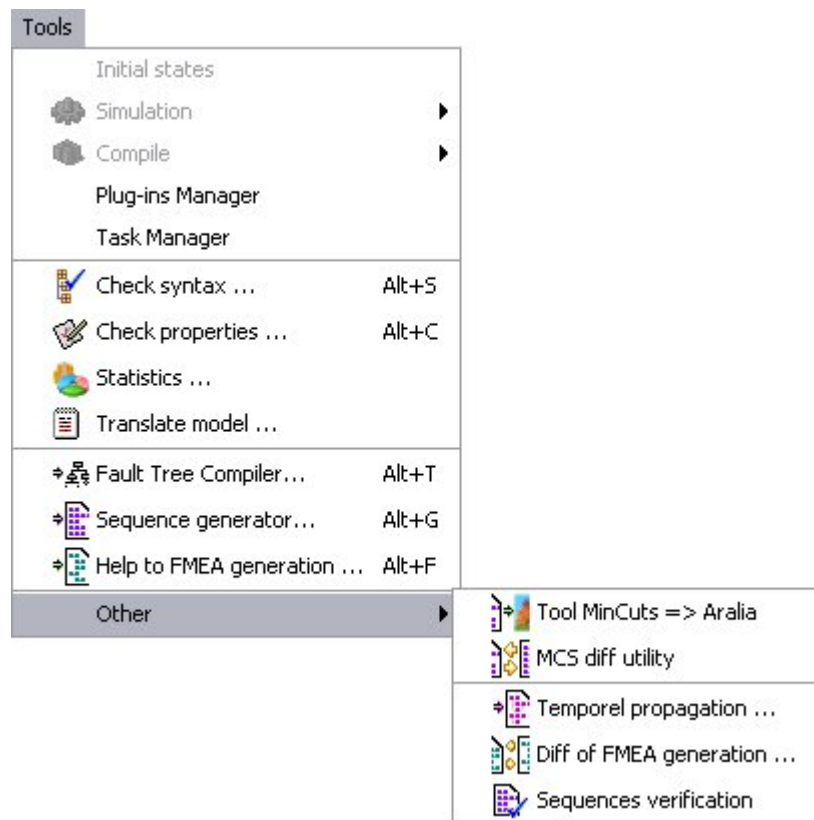


Figure 108: Tools menu

## Views Menu

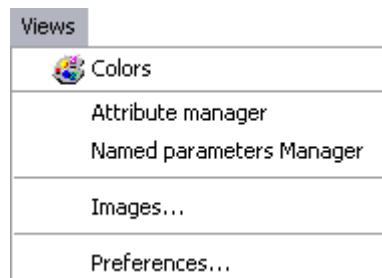


Figure 109: Views menu

## Help Menu



Figure 110: Help menu

# Toolbars

## Standard Toolbar



Figure 111: Standard Toolbar

## Edition Toolbar



Figure 112: Edition Toolbar

## Navigation Toolbar

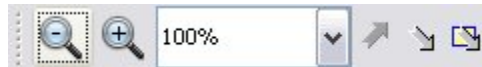


Figure 113: Navigation Toolbar

## Simulation Toolbar



Figure 114: Simulation Toolbar

## Data Toolbar



Figure 115: Data Toolbar

## Design Toolbar



Figure 116: Design Toolbar

## Alignment Toolbar

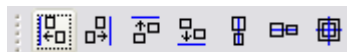


Figure 117: Alignment Toolbar

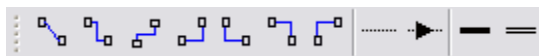


Figure 118: Link Style Toolbar

# Glossary

Assertion, assert	AltaRica code allowing variation of the output flow variables of components
Base	It is made up of three directories (icons, models and project) and contains all the systems, equipment, components, operators and types of the tool library
Boolean constant	“true” or “false”
Component	It is an elementary behavioural object which can be represented graphically. It is a component type model.
Constant of an enumerated type	It is an identifier previously declared in an enumeration field.
Enumerated type	It consists of a single variable which can take several values, e.g. power supply: {null, nominal}
Equipment	It is a hierarchical behavioural object which can be represented graphically. It is an equipment type model
Event	It is an external action which conditions a component state variable.
Family	It enables classification of models in the library.
Flow variable	It is a component input or output variable.
Identifier	String of characters containing letters (uppercase or lowercase), numbers or underscore character ( _ ). The string must start with a letter.
Integer constant	It is either zero or a string of numbers which do not start with zero.
Keyword	It is an identifier reserved for exclusive use in the AltaRica language, e.g. if, then, etc.
Lexemes	They are identifiers, keywords, integer constants, operators and separators in the AltaRica code.
Link	It defines a link between two components or equipment in an architecture.
Model	It is a generic term which describes an object in the library.
Object	It is a generic term which denotes a component, an equipment, an operator or a type.
Operator	It is a generic elementary behavioural object without graphical representation which can be used to model a component.
Project	It is a generic term representing a set of systems.
Transitions	AltaRica code which enables evolution of the state variables of components
State variable	It is a variable identifying a component internal state (e.g. a variable with the following values “open/blocked”, “ok/hs”...).
Structured type	It is constituted of many variables, and each variable can have many values, for example voltage : { null, nominal} and short-cut : { yes, no}
System	It is a component of a project which is represented by an architecture.
System assertions	System assertions are links which are not displayed in the hardware architecture; they represent only flow assignments and do not use operators (e.g., cst1.e = true).

---

# Appendix: AltaRica Language Specification

## General

With the “trans” clause of an AltaRica “node”, one can describe how a model react when a particular event occur, in terms of a transition from one state to another. With the “assert” clause, one can describe how the output flow variables are computed in function of the values of state variables and/or input flow variables.

## Lexical considerations

### Whitespaces

Blank characters (spaces, tabulation, etc.) and comments are ignored by the compiler. They can be used to make a program more readable but do not modify the semantics of the model.

### Comments

Two types of comments are available in AltaRica: long comments, and short ones.

#### Long comments

The `/*` character sequence opens a long comment; it must be terminated by the `*/` character sequence. Long comments cannot be nested.

```
/*
  This is a long comment
  *****
  on several lines.
*/
```

#### Short comments

The `//` character sequence indicates the start of a one-line, short comment; it is delimited by the end of the line.

```
// This is a comment on one line.
// This is another one.
```

### Identifiers

An identifier is a made of uppercase and lowercase alphabetical letters, digits and ‘\_’, and cannot start by a number. Identifiers are case sensitive.

```
my_motor
my_Motor
valve_9
```

#### Qualified names

Qualified names are used to access elements of hierarchies; they are made of identifiers separated by dots “.”:

```
block_1.p.v
```

## Keywords

The following words are reserved words and have a specific meaning; they cannot be used as identifiers:

and	else	imply	max	sub
assert	event	in	min	sync
bool	extern	init	node	term
case	false	int	not	then
cnuf	float	inverse	or	trans
const	flow	knil	out	true
domain	func	link	private	#
edon	if	local	state	

## Literal constants

### Boolean literals

true and false are the two Boolean constants.

### Enumeration literals

Members of enumerations are enumeration literals; they can be used as constants, and cannot be used to identify anything else (flow or state variable, event, etc.).

### Integers literals

Integers are written as sequences of digits, which cannot start by 0, except 0 itself.

```
0
42
2097
```

## Expressions

This section presents the syntax of AltaRica expressions, which may appear in assertion and transition clauses.

The expressions below are presented in decreasing priority order (The logic OR is the lowest order priority).

- The parenthesized expressions.
- The variables.
- The atomic expressions.
- The unary operators.
- The relational operators.
- The equality operators.
- The logic AND operator.
- The logic OR operator.





---

## Parenthesized expressions

In order to facilitate interpretation of the expressions If-Then-Else and If-Then, the latter must be mandatory between parentheses.

```
parenthesized-expression ::=
    "(" expression ")"
| "(" expression ? expression : expression ")"
| "(" "if" expression "then" expression [ "else" expression ] ")"
```

For an If-Then-Else expression, the first sub-expression is mandatory boolean; it is called the operation condition. The two next sub-expressions make up the parts Then and Else of If-Then-Else.

-  The expressions authorized in the parts Then and Else are the same as those authorized at the first level of an assertion.
-  ( if [expression](#) then [expression](#) else [expression](#) ) and ( [expression](#) ? [expression](#) : [expression](#) ) are equivalent.

## Atomic expressions

Atomic expressions are either pathnames to variables or constants or parenthesized expressions.

Depending on the context, a path can be interpreted as a variable identifier or an enumeration member or also a declared constant identifier. In these conditions, a same identifier cannot be used for a declared constant, a variable or an enumerate constant.

```
atomic-expression ::=
    parenthesized-expression
| hierarchy-path
| true
| false
```

## Unary operators

The unary operations are evaluated from right to left.

```
unary-expression ::=
    ("not" | "~") unary-expression
| atomic-expression
```

The NOT operator has two identifiers, the not word or the tilde ~. The NOT operand must be of the boolean type and the result of the expression is the inverted value of this operand.

## Relational operators

The relational operators are evaluated from left to right. Their operands must be of an arithmetic type because, unlike some programming languages, neither the enumerate constants nor the boolean variables, are assimilated to integers.

The operators < (less than), > (greater than), <= (less or equal to) and >= (greater or equal to) give the value false if the given relation is not satisfied by the operands; otherwise, the value of the expression is true.

```
relational-expression ::=
    relational-expression < additive-expression
| relational-expression > additive-expression
| relational-expression <= additive-expression
```

```
| relational-expression >= additive-expression
| additive-expression
```

## Equality operators

These operators are = (equal to) and != (not equal to).

The operands of the = and != operators are boolean, arithmetic or enumerate. The value of these expressions is true if the expressed relation is satisfied by the values of the operands and false in the contrary. In the boolean case, the equality corresponds to the equivalence between the operands and the difference with the Exclusive or (XOR).

```
equality-expression ::=
    relational-expression = relational-expression
| relational-expression != relational-expression
| relational-expression
```

## Logic AND operator

The logic AND operator has two identifiers: & and *and*. Its operands must be of the boolean type. It is evaluated from left to right. It is true if its two operands are true and otherwise it is false.

```
and-expression ::=
    and-expression ( "and" | "&" ) equality_expression
| equality-expression
```

## Logic OR operator

The logic OR operator has two identifiers: | or *or*. Its operands must be of the boolean type. It is evaluated from left to right. It is false if its two operands are false and otherwise it is true.

```
or-expression ::=
    or-expression ( "or" | "|" ) and-expression
| and-expression
```

## Transitions

Transitions allow evolution of the state variables of components. Transitions are guarded by Boolean expressions; when the guard is evaluated to true, the transition may be fired. It is effectively fired when an associated, probabilistic event occurs.

The declaration of a model transitions starts with the `trans` keyword. This keyword is followed by a non-empty list of transition specifications separated by semi-colons. The transitions declaration may eventually end by a semi-colon.

```
transitions-definitions ::=
    "trans" transition { ";" transition }
```

A transition specification always starts with a boolean expression called the transition *condition*. The variables which appear in these expressions are the model flow variables or state variables.

```
transition ::=
    boolean-expression transition-target { ";" transition-target }
```

---

The transition condition is then followed by a non-empty list of actions associated to events.

An action starts by the separator `| -` followed by a non-empty list of events previously declared in the model.

The list ends with the `->` separator followed by an assignment list for the variables.

```
transition-target ::=
    "|-" event-name-list "->" assignment { "," assignments }
```

Assignments of variables are separated by commas (,).

An assignment consists in an identifier, typically the name of a state variable, followed by the assignment operator `:=`, followed by an expression of the same type as the state variable. The expression assigned to the state variable can contain flow variables and state variables.

```
assignment ::=
    variable-name "==" expression
```

## Example 1

```
trans
    state = ok |- fail -> state := failed;
```

## Example 2

```
trans
    state = not_distributed |- fail_locally -> state := failed;
    state = not_distributed |- fail_locally_untimely -> state := unstable;
```

## Example 3

```
trans
    state = open and blocked = no
    |- unexpected_close -> state := closed, blocked := yes;

    state = closed and blocked = no
    |- unexpected_open -> etat := open, blocked := yes;
```

# Assertions

The computation of output flow-variables is described in assertion clauses.

Assertion clauses starts with the `assert` keyword. This keyword is followed by a non-empty list of Boolean expressions separated by semi-colons (;). The declaration may eventually end with one semi-colon.

```
assertions-definition ::=
    "assert" assertion { ";" assertion }

assertion ::=
    boolean-expression
```

The variables which appear in an assertion are the flow variable or state variables of the model in which the assertion is defined.

Boolean expressions authorized at the first level in an assertion are the following:

- The parenthesized expressions such as if then else
- The assignments of the variables:

An assignment is composed of the identifier of an output flow variable (`out`) for the model followed by the separator `=` and ends with an expression of the same type as the flow variable. The expression assigned to the flow variable can contain constants and input flow variables (`in`) or state variables for the model.

### Example 1

```
assert
  (if state = ok and
    (true = (e_v1 and e_v2) or (e_v1 and e_v3) or (e_v1 and e_v4) or
      (e_v1 and e_v5) or (e_v2 and e_v3) or (e_v2 and e_v4) or
      (e_v2 and e_v5) or (e_v3 and e_v4) or (e_v3 and e_v5) or
      (e_v4 and e_v5)) then
    s = true, validity = true);
```

### Example 2

```
assert
  (if state = not_distributed or state = distributed then
    (if e1 = normal or e2 = normal then
      s = normal
    else
      (if (e1 = untimely) or (e2 = untimely) then
        s = untimely
      else
        s = absent
      )
    )
  );
```

### Example 3

```
assert
  (if (right_engine_failure and (~auto_right_flag or ~oversupply_left)) or
    (left_engine_failure and (~auto_left_flag or ~oversupply_right)) then
    s = true
  else
    s = false);

  (if state=out_of_order then
    s = false, validity = false);
```

---

# Appendix: User Management

## Administration Window

The user management window is available only for the administrator user or any user having administrator rights.

Each administrator can manage access rights for users:

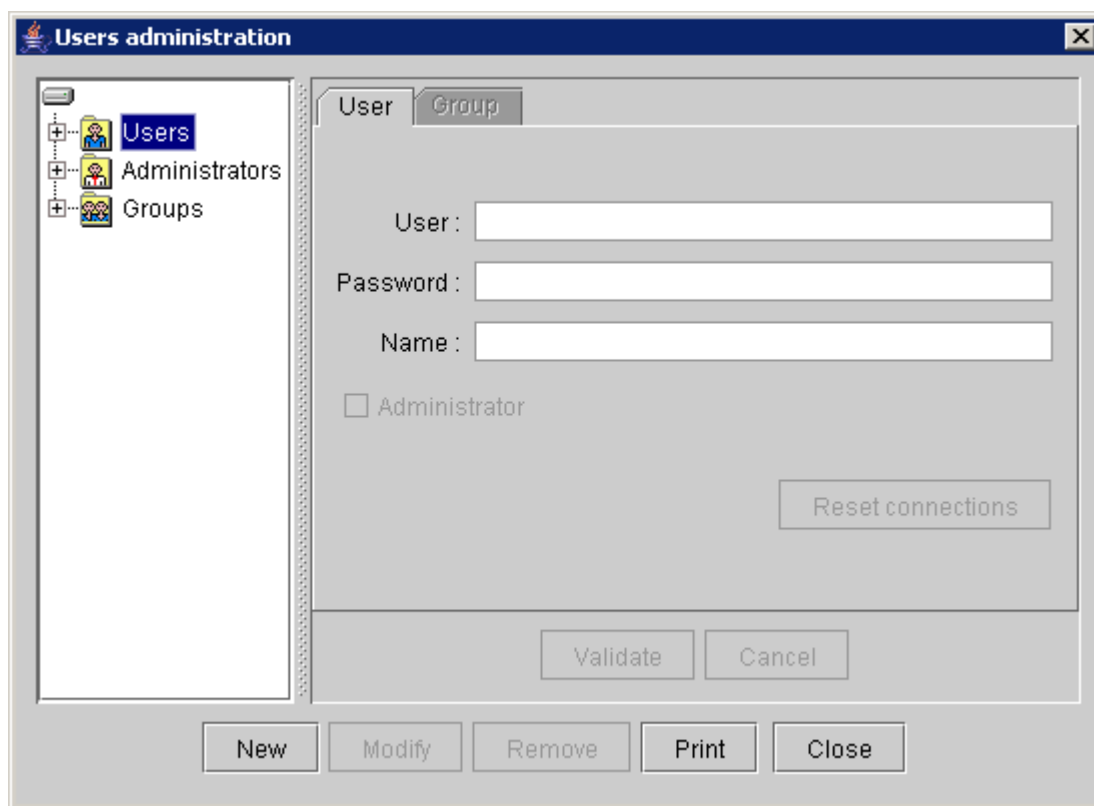
- New user creation,
- Definition of user profile (user, administrator),
- Creation of User group.

Access on data (components, equipments, types, systems, operators) depends on user rights.

Select “File” -> “Administration” to open the administration console.

## User Management

User management is available through the tab User in User Administration window (Figure 119).

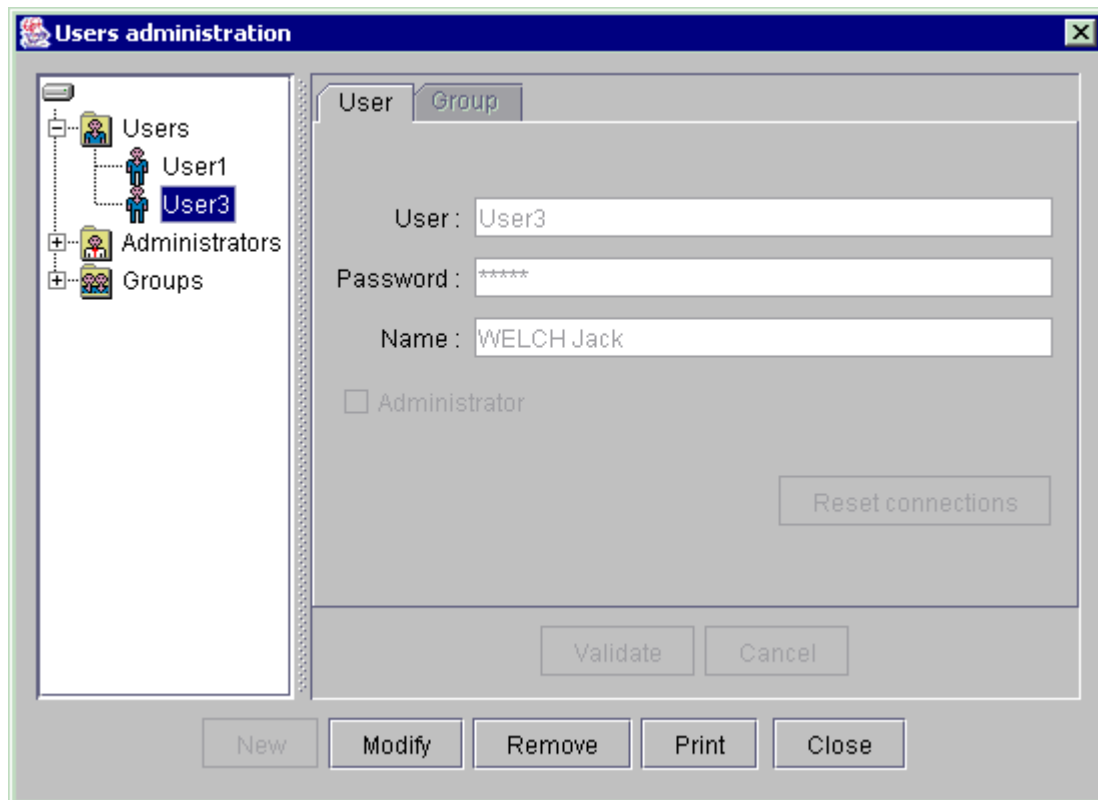


*Figure 119 : Users Administration window – User tab*

To create a new user:

- Select Users or Administrators in the left part of the window,
- Click on button New,
- Enter the logging name of the new user in the field User,
- Choose a password for the new user in the field Password,
- Enter a user name in the field Name,
- Tick the box Administrator to allow this user to access user administration functions (Administrator privileges).
- Click on button Validate to validate creation of the new user profile, or on button Cancel to cancel the creation of this profile.

When the user is created, he is inserted in the Administration tree structure, either in the Users *list* as a common user (Figure 120), or in Administrators list if he benefits of administrator privileges.



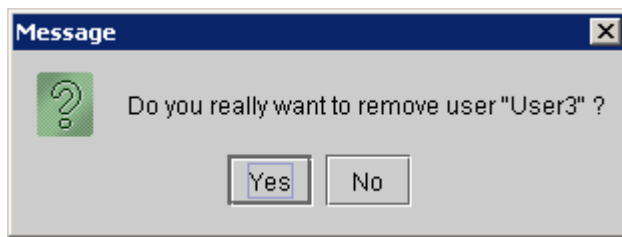
*Figure 120 : Creation of a new user*

To modify an existing user:

- Select (left click) in Users or Administrators list, the user to modify,
- Click on button Modify,
- Make all the necessary adjustments on user profile (User, password, name, rights),
- Click on button Validate to validate the modifications, or on button Cancel to cancel modifications on this profile.

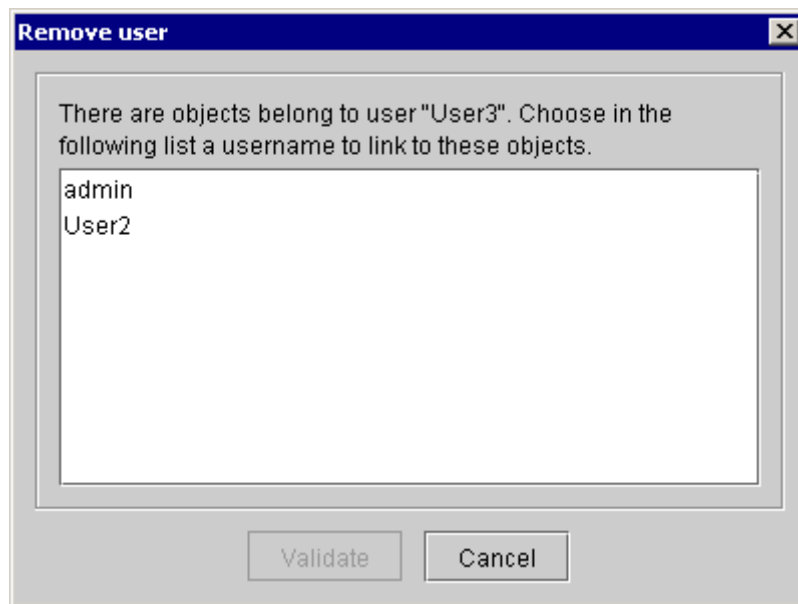
To remove an existing user:

- Select (left click) user to be removed from the Users or Administrators list.
- Click on button Remove, the following window is displayed (Figure 121):



*Figure 121 : Message for removing a user*

- Click on button Yes to delete the user profile or on button No to cancel the destruction.
- If the removed user is the owner of object (models, systems, etc ...), the administrator must specify the user who will be the new owner of this object. During removal, a dialog box (Figure 122) asks Administrator for the new owner.



*Figure 122 : New Owner*

Note: Only the Master Administrator profile created during the initialization of database (Access©, Oracle©) can not be deleted.

## Workgroup Management

The user workgroup management is available through the tab Groups from the Users administration window (Figure 123).

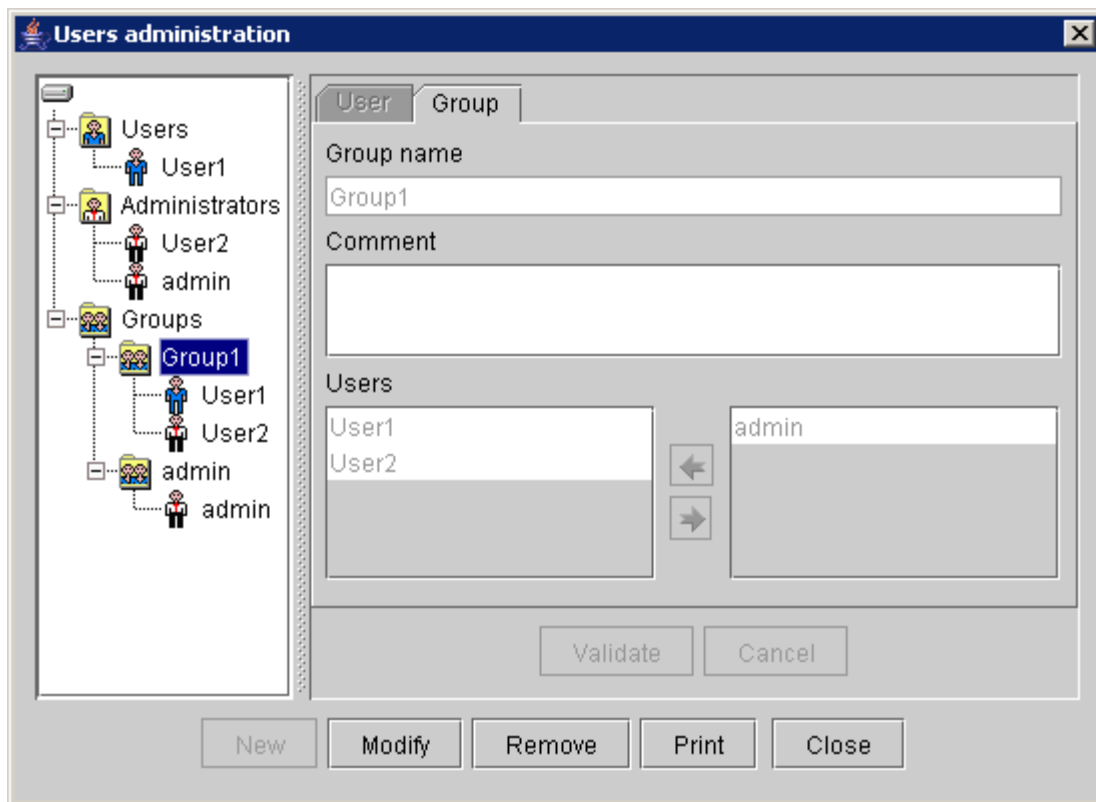



Figure 123 : Users administration window - Tab Groups

To create a new workgroup:

- Click on button New,
- Enter a group name in the field Group name,
- In the tab Users, select one by one users belonging to this group. Use the arrow  to move on the left list,
- Enter if required a comment for this group in the field Comment,
- Click on button Validate to validate the creation of this group, or on button Cancel to cancel the creation.
- When the group is created, it is displayed in the Groups list (Figure 124).



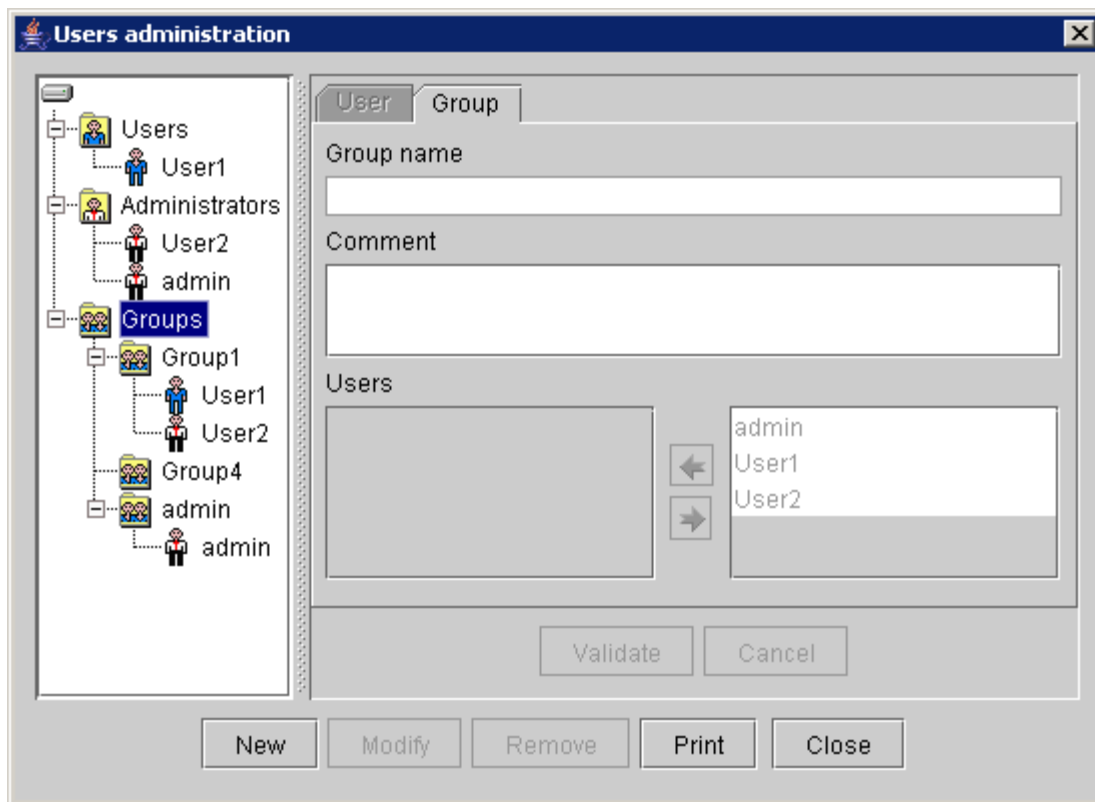


Figure 124 : Users group creation

To modify an existing group:

- Select the group to modify in the Groups list (left click),
- Click on button Modify,
- Make the required adjustments on this group. To remove a user, select in Users area the user in the left list and use the arrow → to move it in right list. To add a user, proceed as described for the creation of a group,
- Click on button Validate to validate the modifications, or on button Cancel to cancel the modification on this group.

To delete an existing group:

- Select the group to be deleted in the Groups list (click left),
- Click on button Remove, the following window is displayed (Figure 125) :



Figure 125 : message of the user group destruction

- Click on button Yes to valid, or on button No to cancel the deleting of this group.

- ★ Only empty groups can be deleted.

Note: Only the Master Administrator profile created during the initialization of database (Access©, Oracle©) cannot be deleted.

## Printing Administration Information

The administrator can print information contained in the Administration database by clicking on button Print from the User administration window (Figure 126):

- The user list,
- The workgroups.

Users list	
Administrators	
Login	Name
User2	Durant Jean
admin	Master administrator
Users	
Login	Name
User1	DUPONT Pierre
Groups	
Group name	Users
admin	admin
Group1	User1
	User2

Figure 126 : Printing Administration information

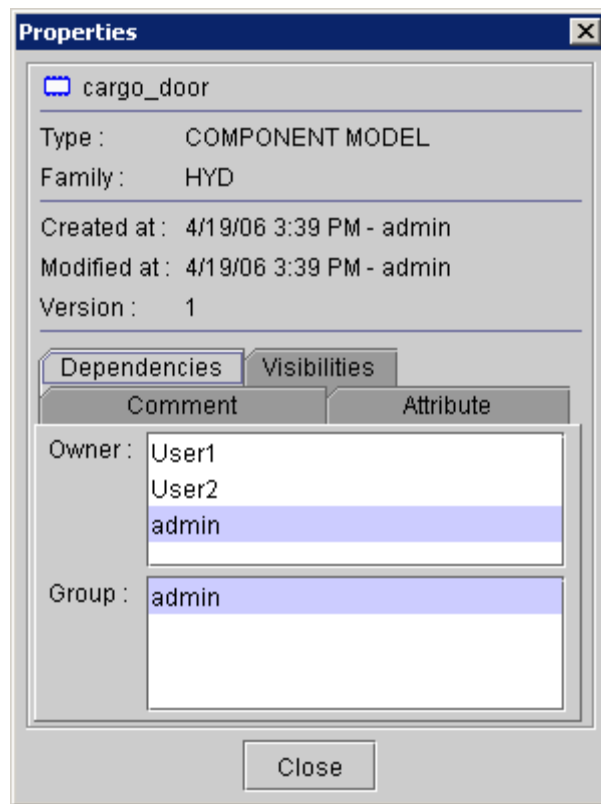
## Managing Permissions

Safety Designer allows the access right management on data (generic models in library and System architectures) by dependencies and visibility attributes defined on each object.

### Dependencies

Any object, generic model in library or system architecture, owns to the user who created it. If this user belongs to several Workgroups, this one must specify during the creation of an object, the Workgroup to which this object must be linked.

Only the administrator of database or the owner of a given object can modify the dependencies by using the property window (Figure 12).



*Figure 127 : Editor of Properties – Dependencies tab*

To modify the owner of an object:

- Select a **Error! Reference source not found.** object (model or system) in the tree.
- Displays Property Editor with the File/Properties menu or with contextual menu.
- Click on the tab **Dependencies**,
- In the area **Owner**, select the user who must be the new owner of this object.
- If the new owner of this object belongs to several Workgroups,
- In the **Group** area, select the workgroup to which this object must be linked
- Moreover, if the **Error! Reference source not found.** object Is a project or a family, user can specify if the rights must be propagated to each included object with “Apply to included objects...”

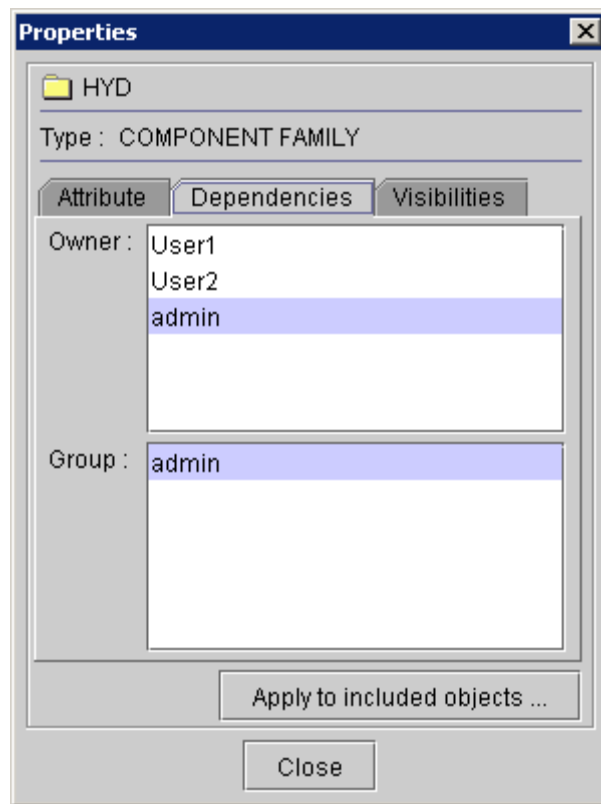


Figure 128 : Apply Dependencies to included objects

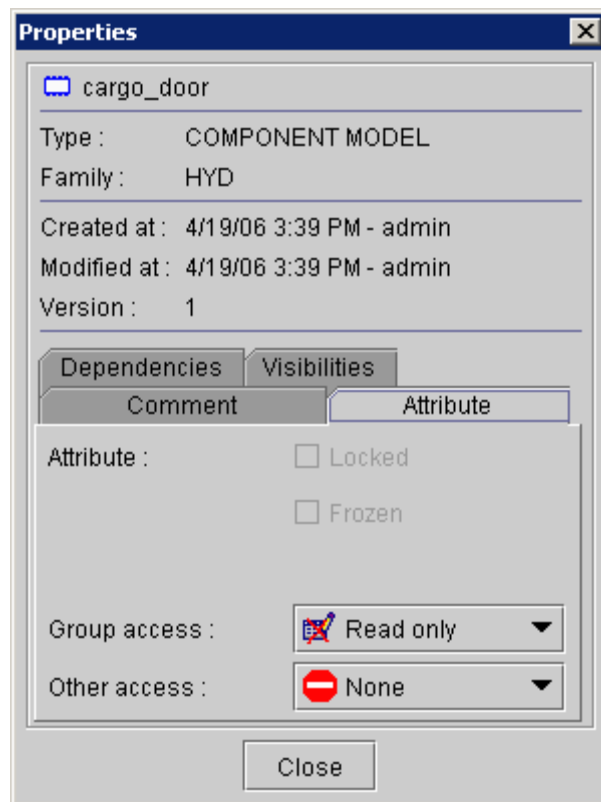
## Access Rights (Attributes)

During the creation of a **Error! Reference source not found.** object, the default access rights are:

- Read Access Right for users of the workgroup to which this object is linked (only the owner of this object has an Read/Write Access Right),
- No access for users of the other workgroups.

Only the administrator of a database or the owner of a given object can modify its attributes of access rights by using the property editor (Figure 131).

- To modify the access rights to a n object for workgroups:
- Select the object
- Open the property editor by clicking on Properties in File Menu
- Click on the **Attribute** tab,
- In the Group access area, choose in the popup menu the new access rights for the workgroup to which this object is linked,
- In the Other access area, choose in the popup menu the new access rights for the other workgroups.



*Figure 129 : Property edition - Attribute*

- If you edit a Project or a Family of Models, you can specify if the rights must be applied to all included elements by clicking on Apply to included objects (Figure 130).

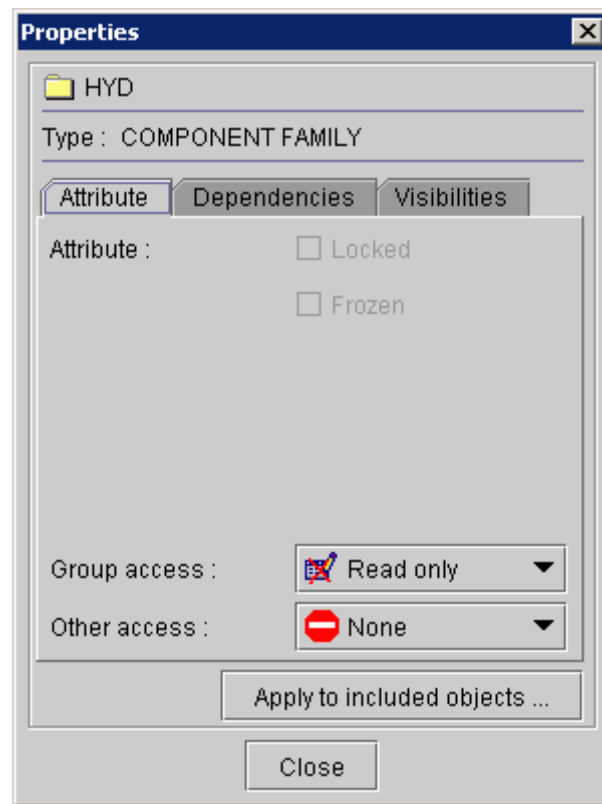


Figure 130 : Apply change to included Objects

## Visibility Definition

The Visibility tab in Properties editor allows user to specify access rights to a **Error! Reference source not found.** Object (generic models in library and system architectures) for each workgroup defined in the administration module (See § *Administration Window*). The access rights defined by default in the Visibilities tab are defined in the Attribute/Dependencies tabs.

Only the administrator of **Error! Reference source not found.** database or the owner of a given object can modify attributes of access rights in the property editor (Figure 131).

To modify the access rights to a **Error! Reference source not found.** object for a workgroup:

- Select the **Error! Reference source not found.** Object (model or architecture)
- Launch the Editor of properties by clicking on Properties in File Menu
- Click on the **Visibilities** tab,
- In the Access Right area for this workgroup, select in the popup menu the new access rights for this workgroup (no access, read only, read/write).

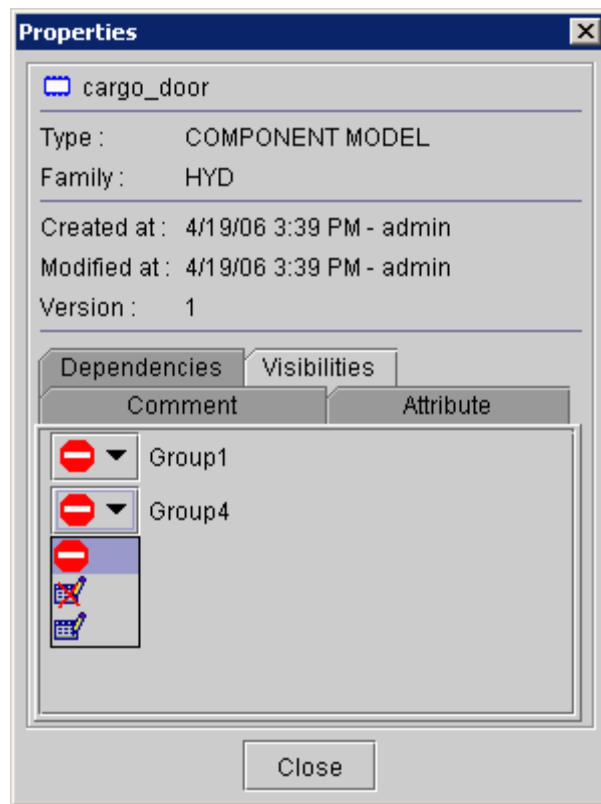
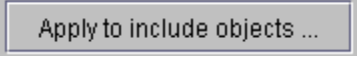


Figure 131 : Properties edition – Visibilities

- If you edit a Project or a Family of Models, you can specify if the visibilities must be applied to all included elements by clicking on .

## Configuration Definition Example

The following diagram (Figure 132) displays:

- Users with their rights
- User Groups
- **Error! Reference source not found.** model families and attributes (dependencies, rights, visibilities)

The definition of FAMILY\_B attributes (Figure 132) associated with this configuration, are shown Figure 133.

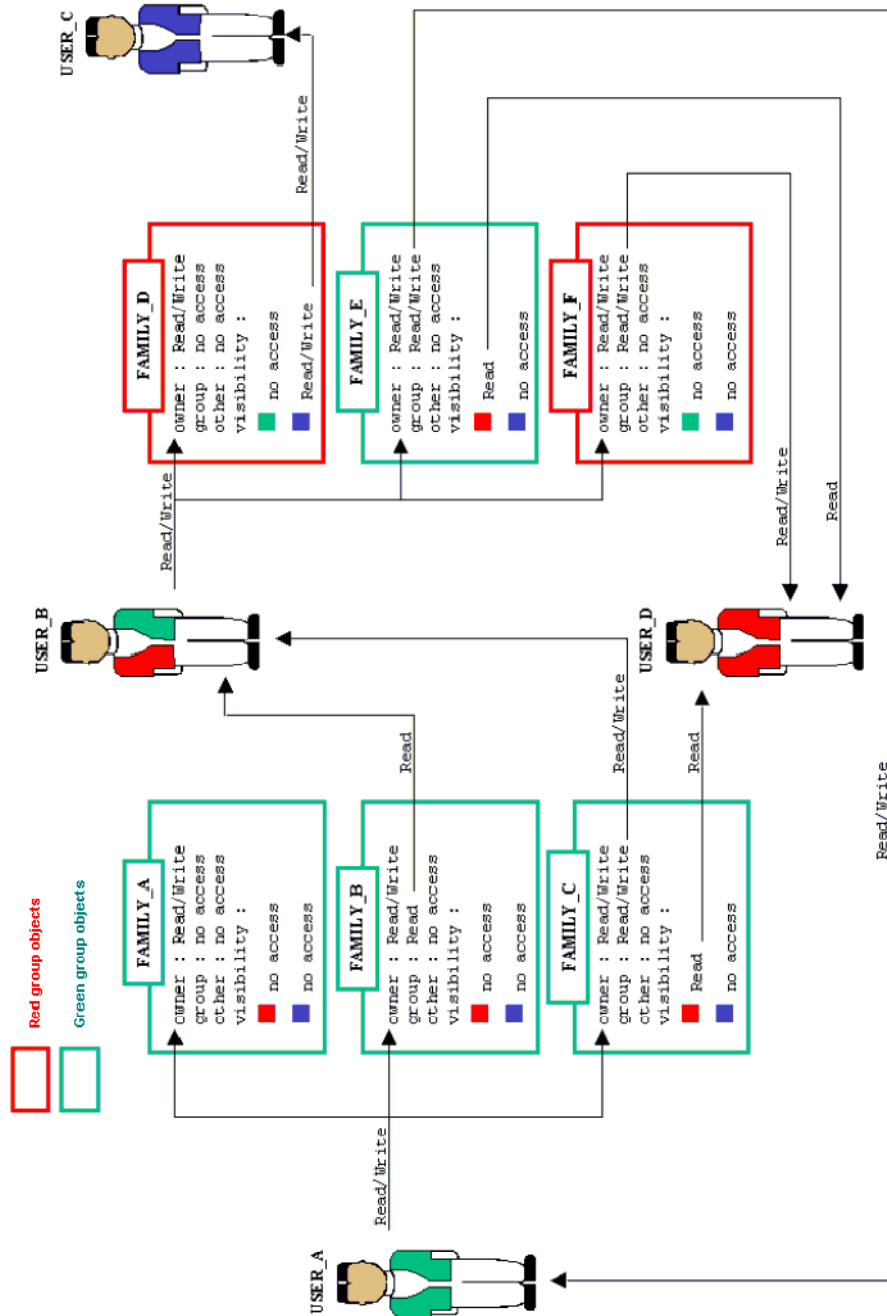


Figure 132 : Rights, Dependencies, Visibilities



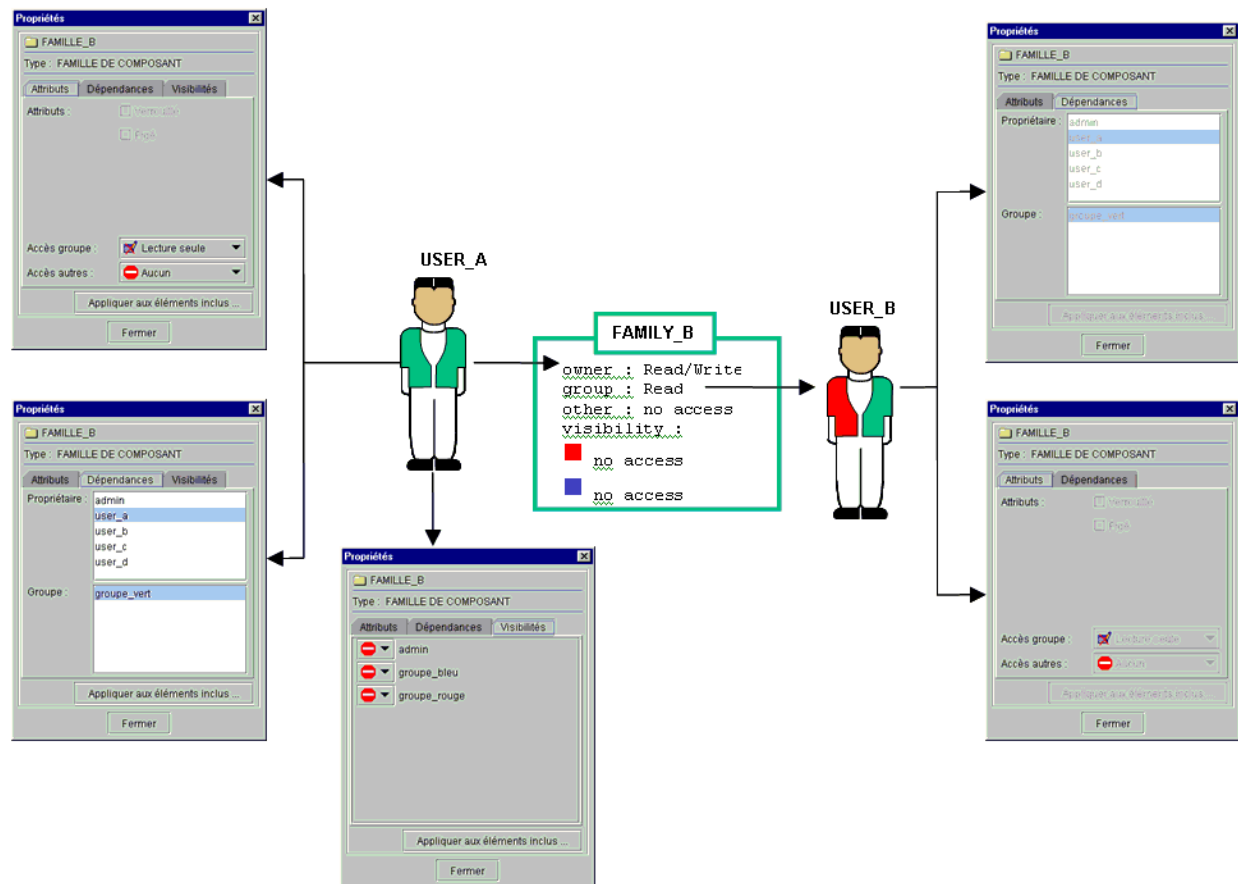


Figure 133 : Example: FAMILY\_B

# Appendix: Probability Laws in Aralia Format

Probability laws for Basic Events				
Law	Expression	Par.	Definition	Limits
constant	$A(t) = q$	$q$	Failure probability	$0 \leq \text{Probability} \leq 1$
exponential	$A(t) = 1 - \exp^{-\lambda \cdot t}$	$\lambda$	Failure rate	Rate $\geq 0$
GLM	$A(t) = \gamma \exp^{-(\lambda+\mu) \cdot t} + \lambda / (\lambda+\mu) \exp^{-(\lambda+\mu) \cdot t}$	$\gamma$	Probability of initial start-up refusal	$0 \leq \text{Probability} \leq 1$
		$\lambda$	Failure rate	Rate $\geq 0$
		$\mu$	Repair rate	Rate $\geq 0$
GLM-asymptotic	$A(t) = \lambda / (\lambda+\mu)$	$\lambda$	Failure rate	Rate $\geq 0$
		$\mu$	Repair rate	Rate $\geq 0$
Weibull	$A(t) = 1 - \exp^{-\phi(t)}$ avec $\phi(t) = ((t-\tau)/\alpha)^\beta$	$\alpha$	Scale parameter	Factor $> 0$
		$\beta$	Form parameter	Factor $> 0$
		$\tau$	Location parameter ( $\alpha \leq 0$ as $t-\alpha$ must always be greater than 0)	Time $> 0$
periodic-test	$A(t) = \begin{cases} 1 - \exp^{-\lambda \cdot t} & \text{si } t \leq \tau \\ 1 - \exp^{-\lambda \cdot ((t-\tau) \cdot \theta)} & \text{si } t > \tau \end{cases}$	$\lambda$	Failure rate of the component in operation or on stand-by	Rate $\geq 0$
		$\tau$		Time $> 0$
		$\theta$	First interval between tests	Time $> 0$
periodic-test-2	Numeric integration	$\lambda$	Functioning failure rate	rate $\geq 0$
		$\lambda^*$	Failure rate during test	rate $\geq 0$
		$\mu$	Repair rate (once failure is detected)	rate $\geq 0$
		$\tau$	First interval between tests	Time $\geq 0$
		$\theta$	Interval between two tests	Time $\geq 0$
		$\gamma$	Failure probability because of test (0 if test can't create failure)	$0 \leq \text{Proba} \leq 1$
		$\pi$	Test duration	Time $\geq 0$
		$X$	Availability during test (= 0, if unavailable; = 1, if available)	0 or 1
		$\sigma$	Test coverage rate (the probability that test detect failure)	$0 \leq \text{Proba} \leq 1$
		$\omega$	reSetUp forgetfulness probability	$0 \leq \text{Proba} \leq 1$
dormant			(after test and after repair)	
		$\lambda$	Failure rate	rate $\geq 0$
		$MTTR$	Medium time to repair	Time $\geq 0$
CMT	$A(t) = Q + 1 - \exp^{-\lambda \cdot T}$	$T$	interval between two consecutive tests	Time $\geq 0$
		$\lambda$	Failure rate	rate $\geq 0$
		$T$	Mission duration	Time $\geq 0$
NRD	$A(t) = \exp^{-\mu \cdot d}$	$Q$	Probability	$0 \leq \text{Proba} \leq 1$
		$\mu$	Repair rate	rate $\geq 0$
		$d$	Delay	Time $\geq 0$