Archived UML / Diagramming

Go with the arrows to the desired class name and press Enter.
It is not recommended to use the following fonts for diagram saving as .eps, .emf, and .wmf:

Default, Dialog, DialogInput, Monospaced, SansSerif, and Serif.
This is a know problem of Mac Word. Still we don't know when it will be fixed.
Such problem really exists and will be fixed in future releases.

As a workaround we suggest to save your diagrams as SVG and then use free Inkscape editor (http://www.inkscape.org/) to export diagrams as hi-res png.
There are several ways to change the symbol style. For this purpose you can use:

- Buttons on the Diagram toolbar
- **Project Options** dialog
- **Symbol Properties** dialog

Using buttons on Diagram toolbar

There are buttons in the diagram toolbar designed to modify a symbol style.

blocked URL

Set Selected Symbol Style as Default. If you select a symbol, for example, class A and click this button, a current class symbol style will be set as default for all new classes in the project.

blocked URL

Apply Default Symbol Style. If you select a symbol and click this button, a default style will be applied to the symbol.

blocked URL

Select All of the Same type. If you want to select on a diagram all symbols of the same type, click only one symbol as an example of the type and then click this button. All symbols of this type will be selected. Once all symbols of the same type are selected, you could apply changes, for example, a new default style to all of them at one go.

NOTE: If you can not see these buttons, please, switch your perspective to the Expert mode (on the main menu, click **Options > Perspectives > Perspectives** and select the **Expert** check box in the **Select Perspective** dialog).

Using Project Options dialog

1. On the main menu, click **Options > Project**. The **Project Options** dialog will open.
2. In the Project Options tab tree, select **Symbols properties styles > Default**, expand **Default** if needed, and then click the appropriate node: **Shapes, Paths, Diagram**, or **Stereotypes**. If you want to change style of a particular shape or path, expand the **Shapes** or **Paths** node, and select the desired symbol, for example, the actor or link.
3. Edit property values in the pane on the right.
4. When you have finished, do any of the following:
- If you need to apply changes to the project, click the **Apply** button. The list of diagrams wherein the symbol style changes can be applied will open. Select diagrams (use CTRL or SHIFT keys for multiple selection) you need and click **OK**. Then click **OK** to close the **Project Options** dialog.
- If you do not need to apply changes right now, click **OK** to close the **Project Options** dialog. The new style properties will be saved and set as default.

Using Symbol Properties dialog

1. Select a symbol and open the **Symbol Properties** dialog (right-click on the symbol and select **Symbol(s) Properties** or press ENTER+ALT). 2. Edit symbol property values (see figure bellow). 3. Select the **Make Default**check box before closing the dialog (see figure bellow), and all symbols of this type in the same diagram will be in the new style.

blocked URL
To visualize relationships between elements do the following:

1. Select one or more elements in the diagram.

2. Right-click selected elements and, from the shortcut menu, select **Related Elements** > **Display Paths**. To see related elements of the selected element do the following:

1. Drag an element into a diagram.
2. Right-click this element and, from the shortcut menu, select **Related Elements** > **Display Related Elements**. The **Display Related Elements** dial og will open.
3. In the dialog, choose settings you need and click OK. We suggest using the Layout feature to arrange elements automatically for a better view in a diagram. You can find this feature in the main menu, in the diagram toolbar, or in the diagram shortcut menu. For more information about related elements please refer to Section **Displaying Related Elements** in MagicDraw UserManual.pdf. To open the manual, in the main menu, select **Help** > **MagicDraw UserManual**. Also you can find the manual in the < MagicDraw install root > \manual folder.

If you are using MagicDraw 16.8 or the later version, you can use a Relation Map for the relationship visualizing. To create the Relation Map do the following:

1. Select one or more elements in the diagram.

2. Right-click selected elements and, from the shortcut menu, select **Related Elements** > **Create Relation Map**. A new Relation Map will be created. For more information about the Relation Map creation please refer to Section **Relation Map** in MagicDraw UserManual.pdf. To open the manual, in the main menu, select **Help** > **MagicDraw UserManual**. Also you can find the manual in the < MagicDraw install root > \manual folder.

You can easily find elements that are not used in any diagram using MagicDraw Find functionality.

Select command Find... from the main Edit menu to invoke Find dialog. Then specify search scope (we do not recommend to search in the whole Data package, as elements from modules will be found), and select checkboxes "Search data unused in diagrams" and "Load diagrams and unloadable modules".

Elements that are not used in any diagram will be displayed in the Search tab in model browser.
To automatically discover relations on elements drag'n'drop - go to Options -> Environment -> Diagram -> Editing -> and change Display paths on element drop to true.
"Type Modifier" property should be used. Open property specification window, switch to the "Expert" property displaying mode, and define value (for example - 14) there.
You may model loops and conditions using fragments.

Use Combined fragment with operator "loop" for iteration notation and add another one Combined Fragment with operator "alt" for conditional messages grouping inside loop.
MagicDraw allows you to show navigability arrow only on one end of an association path. When both ends are "navigable," MagicDraw simply suppresses that information and arrows are not visible.
There are two ways:

1. Drag and drop a state diagram on the selected class in the browser.

2. Choose the "State Diagram" command from the class shortcut menu in the browser tree.
Port must have type specified. Only after that provided interfaces can be specified. This is because port can't provide interfaces itself, port type does (Interface Realization relation is created between type of the port and Interface, it can't be connected to Port, because port is not Classifier). Port type can be also class like port_impl.
Class must implement all operations from interface first. From the class shortcut menu, choose Tools->Implement/Override Operations or implement all operations manually.
Please use Related Elements > Display Paths shortcut menu command.
The name of the transition (the value of the name property) is never shown in diagram - this has no semantic meaning. Instead, you can specify e.g. a signal, that triggers the transition. In this case the name of the trigger is displayed. If trigger name is not specified, signal name is displayed. If start typing on a transition, the signal with the name is typed will be created Automatically and the trigger will reference that signal.

A trigger defines types of events that can initiate a transition between states. An event is anything that can happen in a system: signal sent by some behavior, a call to a specific operation, reaching a point in time, a change in values within the system, etc. In other words, the formally defined list of possible events is enough for modeling state machine transitions using UML. A transition has to know of some event (indirectly), it cannot be fired by a trigger alone.

Transition signature defined in section 15.3.14 of UML 2.2 is as follows:

::= [[',']* ['[' ']'] ['/']]

However, trigger is a non-terminal and its production rule is described in section "13.3.31 Trigger" (UML 2.2):

::= | | | |

Production rules for call-event, signal-event, any-receive-event, time-event and change-event are described in sections 13.3.6, 13.3.25, 13.3.1, 13.3.27, 13.3.7 respectively:

::= ['(' [ ] ')']

::= ['(' [ ] ')']

::= 'all'

::= |

::= 'after'

::= 'at'

::= 'when'

As can be seen, trigger names are never used in transition labeling. Instead, names of the referenced elements are used.
It is considered that an element is used in a diagram, when there is a dedicated symbol representing it on the diagram. States and transitions have dedicated symbols for them. When a signal is assigned to the transition via trigger, it does not have a dedicated symbol on the diagram. It is only a part of textual representation of transition signature.

Let's do the following: assign a type X to an attribute on a diagram. Though we see X on the diagram as an attribute type, it is not considered as used in the diagram, because it is displayed only as a part of the attribute notation.

If you want to find out whether a particular element is needed in the model or not, use the Usages/ Dependencies functionality. Select an element in the Model Browser and from its shortcut menu select Used By. This way you may follow the chain of elements in the model and decide whether a given element is rubbish, or it makes sense.
All of the variants, defined in OCL 2.0 spec can be modeled (inv:, def:, init:, derive:, pre:, post:, body:), but each is modeled in a slightly different way. Correspondingly, the header line in the OCL expression editor is generated according to model situation.*

inv:* is easiest - any simple constraint having an OCL2.0 expression language is treated as invariant. They should have a field constrainedElement filled in, pointing to some class (constrainedElement=SomeClass). To be extra thorough in following the OCL spec, you should also apply the <<invariant>> stereotype on the constraint, but this is not necessary - it is implied.

Note that any other OCL constraints/expressions, that the handling code can not classify into the more specific categories (as defined below) are treated as inv: constraints.*

def:* - definition constraints are modeled in the same way as invariant constraints, but the constraint must have <<definition>> stereotype applied. The concrete element to be defined (additional field or additional operation) is placed inside of the body.*

init:* - this has to be an expression (not a constraint!) placed in some property's defaultValue field. So, you need to create some property in your class, then open specification of that property, then rightclick the Default Value field, go to >Value Specification>Opaque Expression. After this you can enter the OCL into the field in the same manner as you enter OCL into the specification field of the constraint*.

derive:* - is modeled the same way as init, but the property has to be derived (isDerived=true).*

pre:, *post: - these are constraints on some operation of the class. However these must be placed NOT as a simple constraints(rightclick>New Element>Constraint) BUT in a special designated fields(metaproperties) of the operation. Open operation specification window, switch into Expert mode, see the Precondition, Postcondition fields. Click the necesary field, click [+], then choose Constraint. Fill in the OCL constraint as necessary.*

body:* - body condition is filled in the same way as pre and post conditions, but there is an additional hoop to jump through - for some reason, I do not know why, Body Condition field is marked as not shown in Expert mode (neither in Standart). So in specification window of Operation, click Customize button, find Body Condition field, and flip the radio button to Expert, and OK the dialog. After this, Body Condition field will appear in Expert mode specification window of operation in the same way Precondition and Postcondition fields do.

Small nuance: operation can have multiple pre and post conditions but only one body condition.

All these things are described/mandated in the OCL spec (06-05-01.pdf for OCL2.0), chapter 12: The Use of OCL Expressions in UML Models.

Modeling of these expression types is also briefly mentioned in MagicDraw UserManual.pdf. See chapter 8 Model Analysis>Validation>Advanced Topics>Modeling other types OCL2.0 constraints/expressions