`uml.sty`, a package for writing UML diagrams in LaTeX

Ellef Fange Gjelstad

March 17, 2010

Contents

1 Syntax for all the commands

Most of the commands in `uml.sty` have the syntax `\uml`⟨*Construct*⟩`[`⟨*Named options*⟩`]`⟨*Arguments*⟩.

⟨*Construct*⟩ can be `Diagram`, `Class`, `Method` or other things.

⟨*Named options*⟩ is a comma-separated list of ⟨*key*⟩`=`⟨*value*⟩ pairs. This is implemented by `keyval.sty`. The entire `[`⟨*Named options*⟩`]` part is optional.

In this documentation, the `=` often is replaced with a `-`. This is due to an error in the index generation process, which does not accept.

⟨*Arguments*⟩ a number of arguments, typically enclosed in brackets (`{}`). Typical arguments are a name and some interior of a box.

1.1 Lengths

Legal *lengths* are all PSTricks lengths, including TEX lengths. In PSTricks lengths, the unit is optional; if no unit is present, the current PSTricks `unit` is used.

1.2 Angles

We distinguish between *angles* and *rotations*. An angle should be a number giving the angle in degrees, by default. You can also change the units used for angles with the PSTricks `\degrees[num]` command.
A rotation also expresses a direction, but can have different syntax forms:

An angle i.e. a number

An angle preceded by an asterisk This means the angle is to be interpreted absolute (relative to the page) and not to the underling text direction. This is useful when getting a relation label right up, not in the direction of the relation

A letter One of

letter	*Short for*	*Equiv. to*	*letter*	*Short for*	*Equiv. to*
U	Up	0	N	North	*0
L	Left	90	W	West	*90
D	Down	180	S	South	*180
R	Right	270	E	East	*270

When rotating along lines e.g. relations, you can precede an angle or one of `ULDR` with a colon. This is to orientate the text relative to the line. This is useful when getting a relation along the relation, not right up.

1.3 Node names

A *node name* is a name of a location on the page. Such a name must contain only letters and numbers, and must begin with a letter. Also, the node names should be unique (at least, to the page).

Node names are typically placed by means of the `reference` named option, but could be placed using any PSTricks node command.

1.4 Reference points

A *reference point* is the point in a box which is to be placed. E.g., if the reference point is `tl`, meaning top left, the top left corner is placed relative to whatever.

A reference point can be any of `l` (left), `r` (right), `t` (top), `b` (bottom) or `B` (baseline) or reasonable combinations of them.

1.5 Colors

This is as in PSTricks. E.g., you can say `\red Red` gives Red and use the color `red` as linecolor or fillcolor.

See also section 25 on page 30.

1.6 Line styles

As in PSTricks, e.g., solid, dashed, dotted or none.

2 uml.sty options

2.1 debug

2.2 index

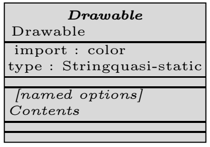
`\umlIndexOn`
`\umlIndexOff`

By default, every named uml.sty construct (e.g., each Class and Attribute) makes an entry in the index of the form ⟨*name*⟩!⟨*type*⟩. This feature can be turned off with the option `noindex` or on with `index`. The feature can also be turned on with `\umlIndexOn` and off with `\umlIndexOff`.

Tip: It is possible to change the index entry format by changing `\umlIndex`. `\umlIndex` should take two arguments: The name and the type. E.g., `\renewcommand\umlIndex[2]{\index{#1 (#2)}}`.

3 Object-oriented approach

When making an uml package for LaTeX, it seems natural to do the programming object-oriented. While LaTeX has no direct support for object-oriented programming, it can be simulated using command calls and keyval.sty.



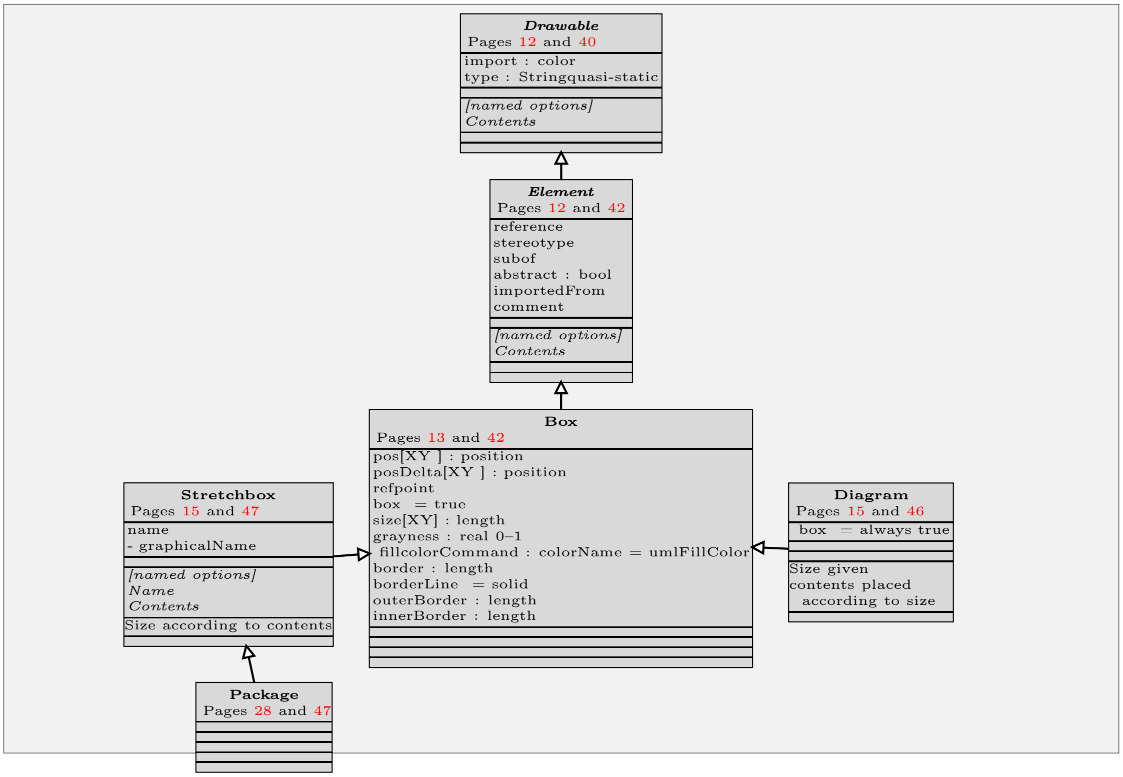


Figure 1: Main commands in uml.sty. This and the following class diagrams work as a short-form documentation of uml.sty.

Each box in figure 1–5 corresponds to one LaTeX command (e.g., `\umlDrawable` and `\umlElement`). Each attribute correspond to one LaTeX variable (e.g., `\umlReference`) and often one named option (e.g., `reference=`).
The arguments of the commands are shown in the fourth compartment. Characteristics of each command are shown in the sixth.
The subclass relation (e.g., between Box and Element) is implemented this way:

Command call The subclass-commando calls the superclass-commando. E.g., `\umlBox` calls

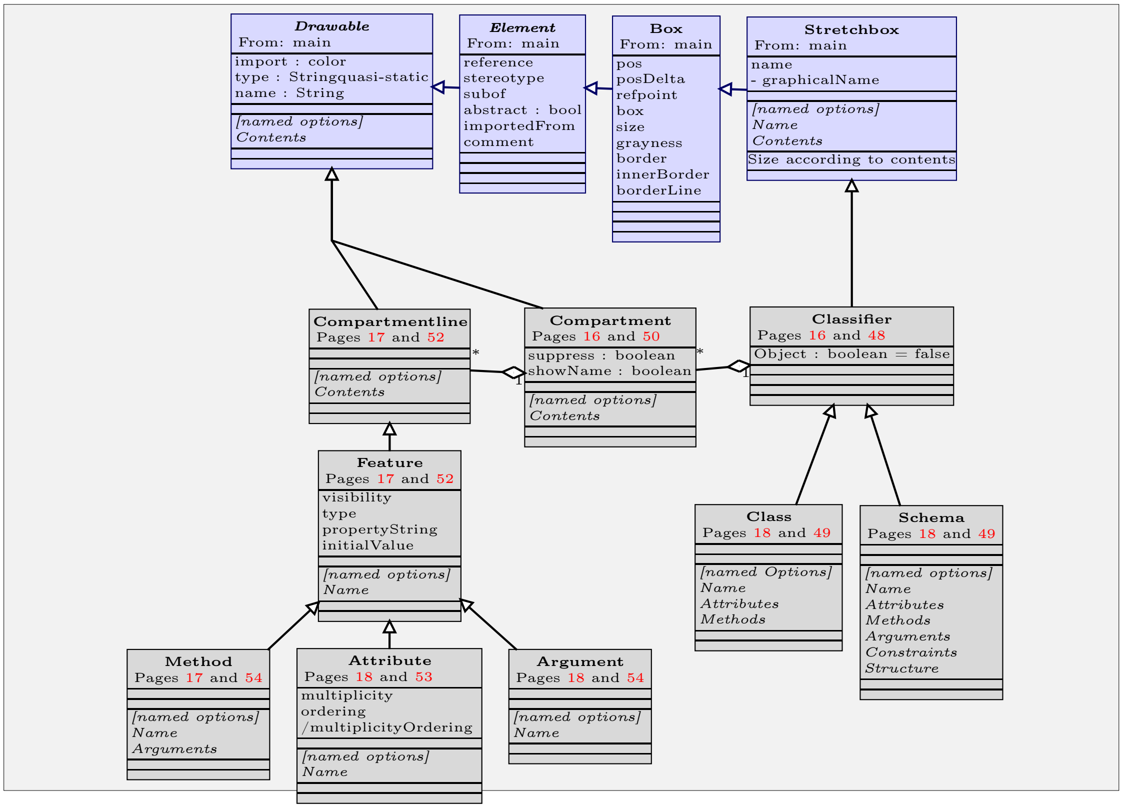


Figure 2: Classifier commands

`\umlElement`.

Variable sharing The variables of the superclass are also used by the subclass. `\umlReference` is used by `\umlBox` as well as `\umlElement`. Of course, there is no support for namespaces in LaTeX, so this is easy and requires discipline at the same time.

Use of named options A list of named options is sent from the subclass-command to the superclass-command. Ultimately, it is evaluated in `\umlElement`. Each command on the road can add its own values in the named options; if it is placed last, it takes precedence, if it placed first, it does not.

E.g., `\umlClassifier` sets `grayness=0.85` first in its named options, meaning that the default grayness is 0.85.

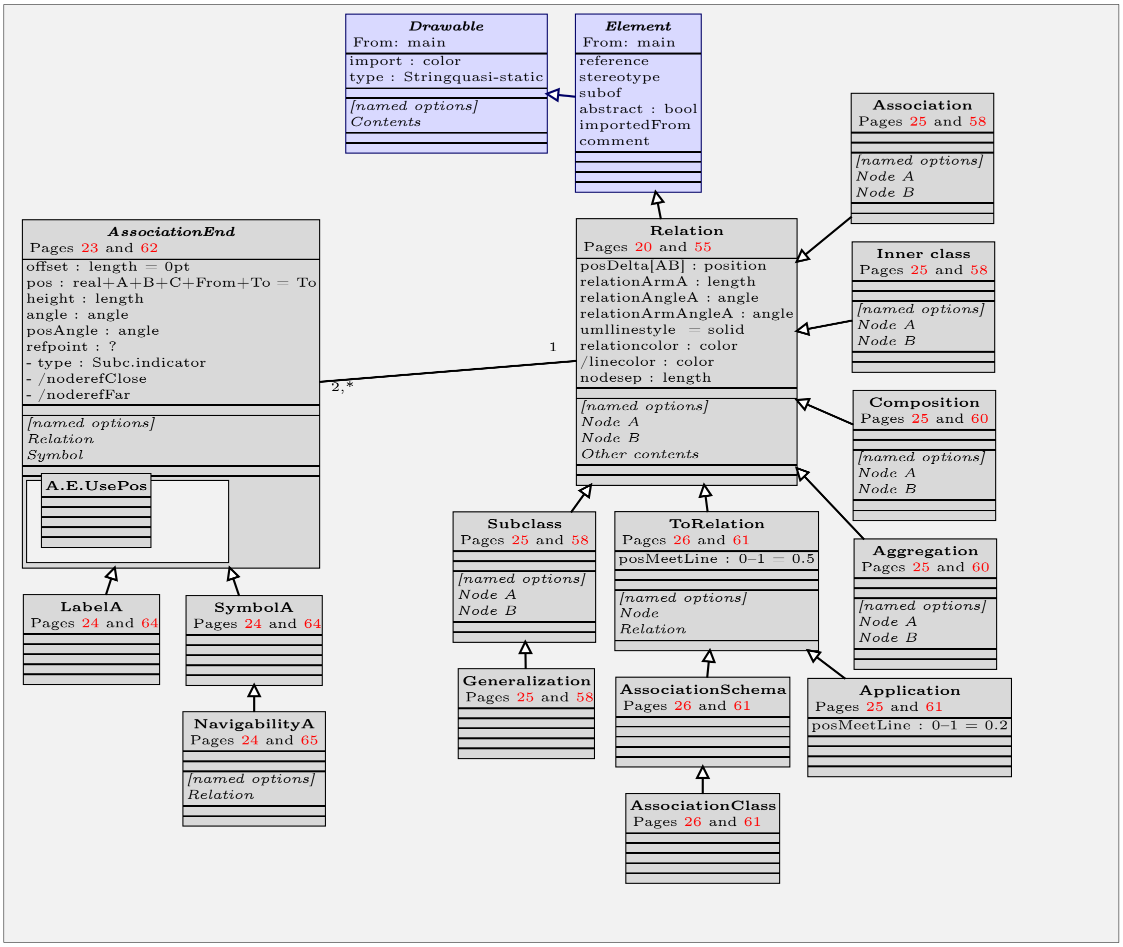
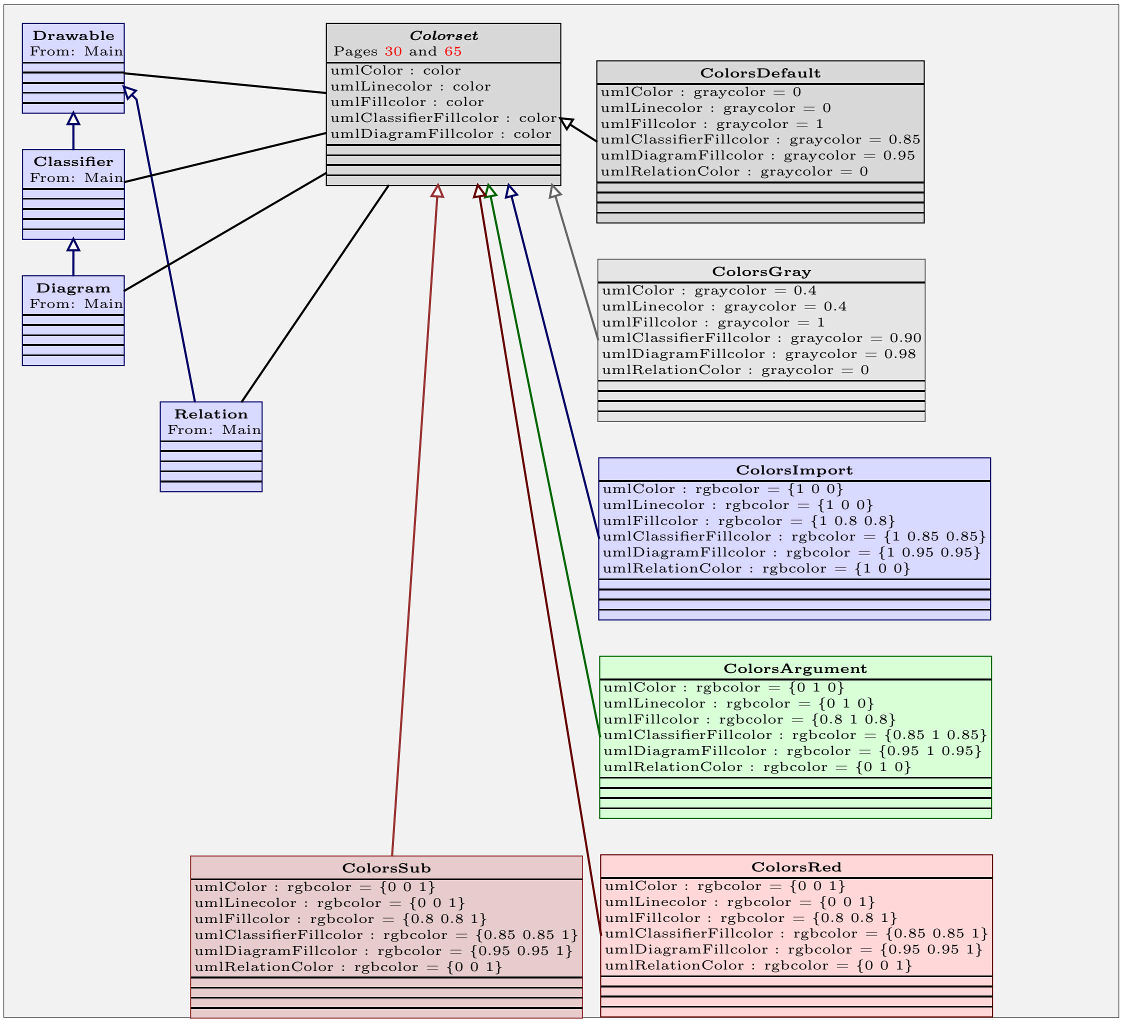


Figure 3: Relation commands in uml.sty.

3.1 Colors

Colors are explained in figure 4. For more comments on that figure, see section 25 on page 30.

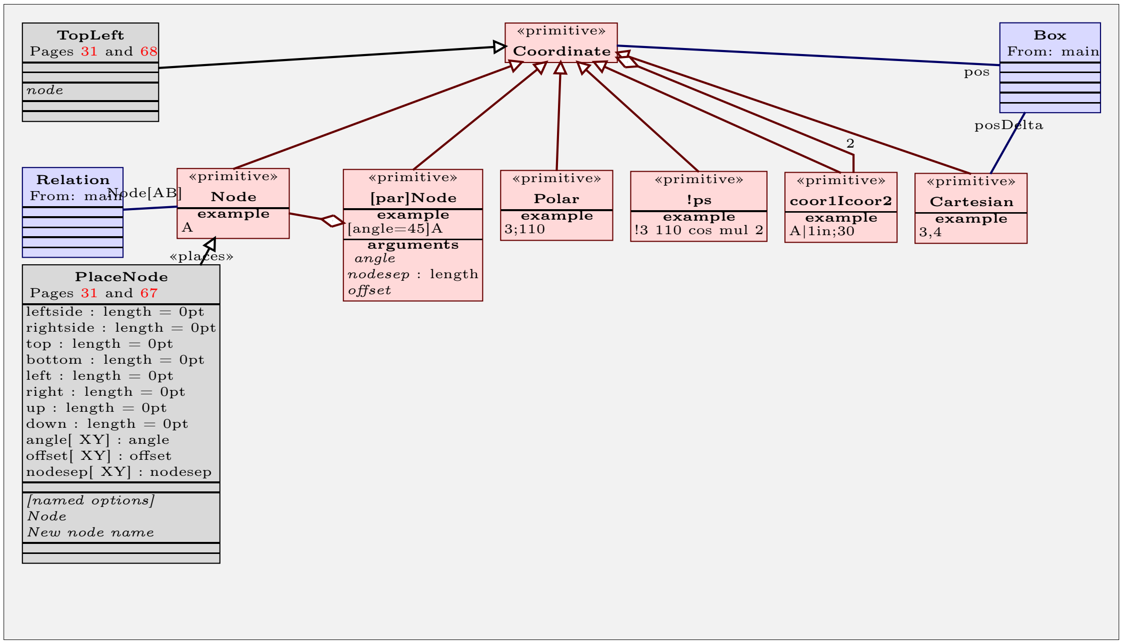


Drawable
From: Main
Classifier
From: Main
Diagram
From: Main
Relation
From: Main
Colorset
Pages 30 and 65
umlColor : color
umlLinecolor : color
umlFillcolor : color
umlClassifierFillcolor : color
umlDiagramFillcolor : color
ColorsDefault
umlColor : graycolor = 0
umlLinecolor : graycolor = 0
umlFillcolor : graycolor = 1
umlClassifierFillcolor : graycolor = 0.85
umlDiagramFillcolor : graycolor = 0.95
umlRelationColor : graycolor = 0
ColorsGray
umlColor : graycolor = 0.4
umlLinecolor : graycolor = 0.4
umlFillcolor : graycolor = 1
umlClassifierFillcolor : graycolor = 0.90
umlDiagramFillcolor : graycolor = 0.98
umlRelationColor : graycolor = 0
ColorsImport
umlColor : rgbcolor = {1 0 0}
umlLinecolor : rgbcolor = {1 0 0}
umlFillcolor : rgbcolor = {1 0.8 0.8}
umlClassifierFillcolor : rgbcolor = {1 0.85 0.85}
umlDiagramFillcolor : rgbcolor = {1 0.95 0.95}
umlRelationColor : rgbcolor = {1 0 0}
ColorsArgument
umlColor : rgbcolor = {0 1 0}
umlLinecolor : rgbcolor = {0 1 0}
umlFillcolor : rgbcolor = {0.8 1 0.8}
umlClassifierFillcolor : rgbcolor = {0.85 1 0.85}
umlDiagramFillcolor : rgbcolor = {0.95 1 0.95}
umlRelationColor : rgbcolor = {0 1 0}
ColorsSub
umlColor : rgbcolor = {0 0 1}
umlLinecolor : rgbcolor = {0 0 1}
umlFillcolor : rgbcolor = {0.8 0.8 1}
umlClassifierFillcolor : rgbcolor = {0.85 0.85 1}
umlDiagramFillcolor : rgbcolor = {0.95 0.95 1}
umlRelationColor : rgbcolor = {0 0 1}
ColorsRed
umlColor : rgbcolor = {0 0 1}
umlLinecolor : rgbcolor = {0 0 1}
umlFillcolor : rgbcolor = {0.8 0.8 1}
umlClassifierFillcolor : rgbcolor = {0.85 0.85 1}
umlDiagramFillcolor : rgbcolor = {0.95 0.95 1}
umlRelationColor : rgbcolor = {0 0 1}

Figure 4: Color commands in uml.sty. The colors in this figure is for demonstration, and not necessarily correct. See also 25 on page 30.

3.2 Positions

Positions are explained in figure 5. For more comments on that figure, see section 26 on page 31. In figure 5, the blue stuff is imported from diagram 1. The red stuff is derived from the PSTricks manual. Only the black classes is defined here.



End of diagram

Figure 5: Position commands in uml.sty. See also 26 on page 31.

4 Drawable

A Drawable is an unspecified UML construct. It has no graphical appearance itself, but is a common superclass for all uml.sty constructs.

`\umlDrawable` The syntax of `\umlDrawable` is, as indicated in figure 1 on page 7, `\umlDrawable[`⟨*named options*⟩`]`⟨*Contents*⟩.

`\umlDrawable` is an "abstract command", not ment to be used directly by users. But `\umlDrawable[import=]{Contents}` would yield Contents (the border around is added here to emphasize the example).

4.1 Named options

The named options `import=` and `noimport=` are described under the next section.

4.1.1 import

`import-`
`noimport-` The named options *import* and *noimport* indicate whether this construct are defined in another diagram, and only imported here.

They take no options.

They make `\umlColorsImport` and `\umlColorsDefault`, *henholdsvis*, take effect on the construct.

See also the variable Element.importedFrom (section 5.1.4).

5 Element

An element is an construct which can be referenced.

`\umlElement` The syntax of `\umlElement` is exactly the same as that of `\umlDrawable`. However, `\umlElement` has more named options.

`\umlElement` is abstract too. It does not itself use much of the information supplied by the named options. `\umlElement{Contents}` would yield Contents.

5.1 Named options

5.1.1 Reference

`reference-` Each element has a $reference$. $reference$ is used among other things when placing nodes.

Legal values are valid PSTricks node names (1.3 on page 6).

Default values differ between the constructs.

$reference$ is also used by some commands to make several nodes with derived names. E.g., `Aa`$reference$ of a relation is one of the endpoints.

5.1.2 Stereotype

`stereotype-` The variable *stereotype* indicates the stereotype of the construct. The value is not used by every construct. Typically, the name is placed in «guillements» around or in the construct.

Legal values is any string. Default is no stereotype.

5.1.3 Subof

`subof-` The variable *subof* indicates the superconstruct (in standard UML parlance, the *generalization*) of this construct. The value is not used by every construct, but is typically placed around or in the construct.
Legal values is any string. Default is nothing.
The *subof* variable and the subclass relation (section 23.0.2) are different ways of expressing the same thing.

5.1.4 ImportedFrom

`importedFrom-` The variable *importedFrom* indicates which package or digram this construct is imported from. The value is not used by every construct, but is typically placed around or in the construct.
Legal values is any string. Default is nothing.

5.1.5 Comment

`comment-` The variable *comment* is a comment. The value is not used by every construct, but is typically placed around or in the construct.
Legal values is any string. Default is nothing.

6 Box

Box is an UML construct drawn as a box, i.e. with some areal. Boxes has size, position and possibly a border.
The syntax is `\umlBox[`⟨*Named options*⟩`]{`⟨*Contents*⟩`}`.

`\umlBox[box=,border=2pt,innerBorder=2pt]{Contents}` would be typeset as Contents
(The inner border is made by the box).

6.1 Named options concerning location

Boxes are positioned by various named options. The positioning mechanism starts at (*pos*), given by the `pos` or `posX` and `posY` named potions. Then, it moves to (*posDelta*), given by the `posDelta` or `posDeltaX` and `posDeltaY` keywords. The *refpoint* of that class (e.g. top left corner), given by `refpoint` is placed at this point.
All the named options with names starting with `pos` have legal values coordinate pairs, as described on page 32.

`pos-` **`pos`** sets *pos*. Default is to look at the values set by `posX` and `posY`.

`posX-` **`posX`** sets *pos* horizontally. Default is `0,0`.

`posY-` **`posY`** sets *pos* vertically. Default is `0,0`.

`posDelta-` **`posDelta`** sets *posDelta*. Default is to look at the values set by `posDeltaX` and `posDeltaY`.

`posDeltaX-` **`posDeltaX`** sets *posDelta* horizontally. Default is `0,0`.

`posDeltaY-` **`posDeltaY`** sets *posDelta* vertically. Default is `0,0`.

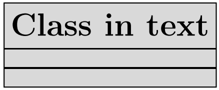
`refpoint-` **`refpoint`** which point in the class to be placed. Can be any of `l`, `r`, `t`, `b` and `B`, meaning left, right, top, bottom and baseline, or any reasonable combination of them.

E.g., if the named options are `[pos=Car, posDelta={1,-1}, refpoint=tl]`, the top left corner of the class are placed one unit to the right and one down from the node Car.

6.2 Boxes in text

Default, an box is placed inside a zero size hbox. While used in text, you want the class to be put inside a hbox with the natural size of the box. This is done by means of the named option `box`. Note that a `=` must be present, even if nothing follows it before the next `,`. This option overrides the location named options.

`box-` **`box`** normally, the class is put in a hbox of size (0,0), suitable for use in a diagram. With `box=` set, it is put in a hbox with its natural size, suitable for use in text. The class



, made by `\umlClass[box=]{Class in text}{}{}`, can serve as an example.

6.3 Named options concerning visual appearance

6.3.1 grayness

`grayness-` The grayness in the background in the box, from `0` (black) to `1` (white). Default value without named option is 1 (white), with named option .85 (still quite bright).

6.3.2 border

`border-` The width of the border. Legal values are lengths. Default value without named option is 0 (no border), default value with named option without value is 0.4 pt.

6.3.3 borderLine

`borderLine-` The style of the border. Legal values are line styles (section 1.6). Default is solid.

6.3.4 innerBorder

`innerBorder-` The inner margin of the box. Legal values are lengths. Default value is 0.

6.4 Named options concerning size

Some boxes (not all) allow the user to specify the size of the interior of the box.

`sizeX-` **`sizeX`** The horizontal size of the box. Must be a valid length.

`sizeY-` **`sizeY`** The vertical size of the box. Must be a valid length.

The default size of the box is not specified here, but it normally is the natural size of ⟨*stuff*⟩.

7 Diagram

`\umlDiagram` A diagram is an area, with a coordinate system. Stuff like classes can be placed at the diagram. The syntax of the `\umlDiagram` command is `\umlDiagram[`⟨*named options*⟩`]{`⟨*stuff*⟩`}`.
⟨*stuff*⟩ is placed inside the diagram. It can be positioned in various ways, using mechanisms from TEX or PSTricks or using named options in `\umlClass` or `\umlSchema`.
Some positioning mechanisms are very sensitive to "spurious spaces", pushing some of the ⟨*stuff*⟩ to the right. Line breaks (without a comment sign before) can create such mystical spurious spaces. Using named options like `posX` and `posY` should eliminate this problem.
`\umlDiagram` is an `\umlBox`, and inherits all its named options.
Once, I discovered that `\umlDiagram` also works without the `sizeX` and `sizeY` options. Using it that way, you typically want to use `innerBorder` on the diagram and `box` on the contents. You may still use `sizeX` and `sizeY` as minimum size.
The named option `box=` is always set to true in `\umlDiagram`.

7.1 Example

The diagram in figure 6 is made by the code

```
\umlDiagram[box=,border,sizeX=7cm,sizeY=4cm]{%
  \umlClass[refpoint=bl, pos={3,3}]{ClassName}{}{}}
```

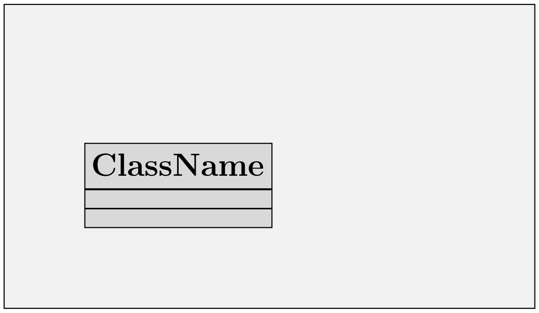


Figure 6: Example of a diagram

This creates a diagram which is 7 cm wide and 4 cm high. It contains a class, with its bottom left corner positioned at position (3,3).
Note that the class, as a stretchbox, determines its size on the basis of its contents, as opposed to a diagram, which has its size given.

8 Stretchbox

A stretchbox is a box which determines its size on the basis of its contents. It also has a name.
`\umlStretchbox` The syntaxis `\umlStretchbox[`⟨*named options*⟩`]`⟨*name*⟩⟨*contents*⟩.

8.1 Name, graphicalName and reference

`\umlStretchbox` takes a name as its second argument. Legal values is strings. The name is a default reference. If the string contains spaces or something making it a non-valid node name, remember to supply a reference with the `reference=` named option.

`graphicalName` Subclasses of `\umlStretchBox` may make a graphicalName based on the name. The graphical name is typically the string in a large font.

9 Classifier

`\umlClassifier` A classifier is a stretchbox which can be instantiated. Classifier is direct superclass to Class. It has the same syntax as `\umlStretchbox`. However, there is another named option:

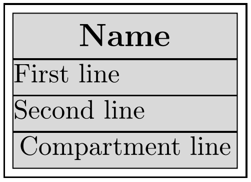
`object-` **`object`** Makes the graphical name underlined.

Classifiers are typically divided in *compartments*, stacked on top of each other. Standard UML classes are divided in three compartments.

10 Compartment

`\umlCompartment` A compartment is a part (of a classifier). The compartments are stacked on top of each other. The syntax is `\umlCompartment[`⟨*Named options*⟩`]`⟨*Contents*⟩. Compartments are to be placed inside classifiers. In the implementation, `\umlClassifier` is implemented as a table and `\umlCompartment` as table lines.

`\umlCompartment` requires its contents to be ended by a linebreak (`\\`). This is typically given by `\umlCompartmentline`.

Example: 

was made by

```
\umlClassifier[box=]{Name}{%
  \umlCompartment{First line\\}
  \umlCompartment{Second line\\}}
  \umlCompartment{%
    \umlCompartmentline{Compartment line}}
```

10.1 Suppression

By default, empty compartments are drawn, indicating that the compartment is empty. The UML specification [UML1.4, 3.22.3] states that “a separator line is not drawn for a missing compartment.”

Suppression is *styrt* by the boolean variable `umlCompartmentSuppress`. You can set this to true (`\umlCompartmentSuppresstrue`) or false (`\umlCompartmentSuppressfalse`) whenever you wish.

`suppress-` You can also use the named option `suppress=` on one compartment or something bigger (such as a classifier). Legal values are `true` and `false` when used without value, default is `true`.

10.2 Name

`name-` You can set the name with the named option `name=`.
If the boolean value `\umlCompartmentNameShow` are true, the name is shown centered bold in the top of the compartment. You may say `\umlCompartmentNameShowtrue` or `\umlCompartmentNameShowfalse` when you wish.
`showname` You can also use the named option `showname=` on a construct. Legal values are true and false, default is true.

11 Compartmentline

`\umlCompartmentline` This is a line in a compartment. It ends with a line feed. Subcommands of `\umlCompartmentline` is `\umlAttribute` and `\umlMethod`.
The syntax is as usual `\umlCompartmentline[`⟨*named options*⟩`]`⟨*Contents*⟩. For example, see above. Note the space in front of the contents when using `\umlCompartmentline`, as opposed to raw text.
All lines in compartments are required to end by a line break. `\umlCompartmentline` supplies this.

12 Feature

`\umlFeature` A feature is a line in a compartment in a classifier, such as a method, attribute or argument.

12.0.1 visibility

`visibility-` Legal values for the visibility is Visibility is typically one of + (public), # (protected), `-` (private) or `~` (package). Default is +.
`\umlTilde` The command `\umlTilde` writes a ∼.

12.0.2 type

`type-` This is the type of an attribute or an argument, or the return type of a method.

12.0.3 propertyString

`propertyString-` An UML property string.

13 Method

`\umlMethod` `\umlMethod` is of the form `\umlMethod[`⟨*options*⟩`]{`⟨*name*⟩`}{`⟨*arguments*⟩`}`.
`returntype-` `\umlMethod` provides `returntype` as an alias for `type`.
`\umlMethod[visibility=\#, returntype=real]{sqrt}{int arg}` makes #sqrt(int arg) : real .

14 Attribute

`\umlAttribute` `\umlAttribute` is of the form `\umlAttribute[`⟨*options*⟩`]{`⟨*name*⟩`}`.
`\umlAttribute[visibility=\#, default=186, type=int]{height}` makes $\boxed{\#\text{height : int} = 186}$.

15 Argument

Arguments to classifiers are not a standard UML construct. However, we have included it here.

`\umlArgument` `\umlArgument` is of the form `\umlArgument[`⟨*options*⟩`]{`⟨*name*⟩`}`.
`\umlArgument[type=Class]{nodetype}` would be rendered as $\boxed{\textit{nodetype} : \text{Class}}$.

16 Class

`\umlClass` The macro `\umlClass` draws an UML class. Its syntax is `\umlClass[`⟨*named options*⟩`]`⟨*name*⟩⟨*variables*⟩⟨*methods*⟩. The ⟨*variables*⟩ and ⟨*methods*⟩ parts are typically drawn using `\umlVariable` and `\umlMethod`. Graphically, the class is divided in three compartments above each other. The upper compartment contains the class name. The middle one contains the variables, and the lower the methods.
Example: Figure 7 on the following page is coded as:

```
\umlClass[box=,
          reference=AmericanMan,
          stereotype=Man,
          import=,
          importedFrom=America,
          comment=A man from America
         ]{American Man}{%
           \umlAttribute[visibility=\#, type=State]{State}
           \umlAttribute[visibility=\#,
             default=Mc Donalds]{Favourite burger}}{%
           \umlMethod[visibility=]{Watch TV}{}
           \umlMethod[visibility=-, returntype=int]{Vote}{Party party}
}
```

Also, see figure 12 on page 34 for a larger example.

17 Schema

Schema is not a standard UML construct. It is a generalization of class I find very useful. A schema is a class which also can

- take arguments
- be instantiated freely into other schemas
- have an internal structure, typically using its arguments

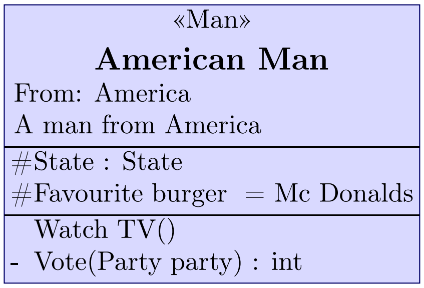


Figure 7: A class

Graphically, schemas look pretty much like classes, with attribute and method compartments. In addition, it have more compartments. Here are they all:

Name compartment contains the name and possibly stereotype and other symbols, just like in the Class symbol.

Attributes compartment contains the attributes, like in Class.

Methods compartment contains the methods, like in Class.

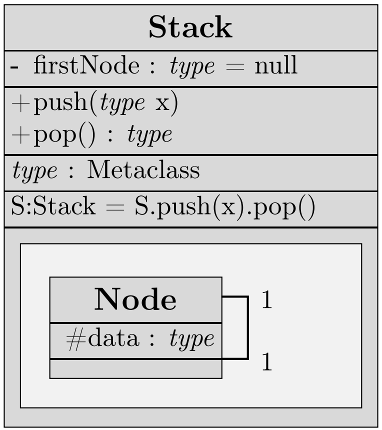
Arguments compartment contains the schema arguments. Each argument is another schema (possibly an object).

Constraints compartment contains constraints.

Structure compartment typically contains a class diagram.

`\umlSchema` This corresponds to more arguments to `\umlSchema`. The spec is `\umlSchema[`⟨*named options*⟩`]{`⟨*Name*⟩`}{`⟨*Attributes*⟩ `}{`⟨*Methods*⟩`}\`
`{`⟨*Arguments*⟩`}{`⟨*Constraints*⟩`}{`⟨*Structure*⟩`}`. Almost everything said about `\umlClass` is also true about `\umlSchema`. The two commands inherit the same named options.
`\umlSchema` is included for two reasons: Because I use them myself, and to show how you can make your own classifiers yourself.

17.1 Example (Stack)



The figure to the right is a simple stack. To start with the fourth compartment, the schema Stack takes an other metaclass as argument (named *type*). If *type* is set to, say, Procedure, we get a Stack of procedures. Stack has one attribute, a private pointer to the first Node. This first node is of type *type*. There are two procedures, the well-known push and pop, both public; push takes an instance of *type* as argument, pop returns a *type*. There is one constraint, that a Stack should be equal to itself pushed once and popped once.

The seventh compartment describes the internal structure of Stack. Here, we do not bother distinguishing between the interface and the internal implementation of Stack.
The LaTeX source is here:

```
\umlSchema[box=]{Stack}{%Attributes
  \umlAttribute[visibility=-, type=\emph{type}, default=null, ]{firstNode}
  }{% Methods
  \umlMethod[visibility]{push}{\emph{type} x}
  \umlMethod[visibility, type=\emph{type}]{pop}{}
  }{% Arguments
  \umlArgument[type=Metaclass]{type}
  }{% Constraints
  \umlCompartmentline{S:Stack = S.push(x).pop()}
  }{% Structure
  \umlDiagram[box=,innerBorder=2mm, sizeX=11em, sizeY=5em,
              ref=StackDiagram, outerBorder=2mm]{%
    \umlClass[pos={.5,.5}, ref=stackNode]{Node}{
      \umlAttribute[visibility=\#, type=\emph{type}]{data}}{}%
    \umlRelation[angleA=20, angleB=-20, armA=1em, armB=1em
                 ]{stackNode}{stackNode}{%
      \umlLabelA[height=-1ex, fraction=1.5]{stackNodestackNode}{1}%
      \umlLabelB[height=-1ex, fraction=1.5]{stackNodestackNode}{1}%
      }
    }% End of diagram
\cr}% End of Stack
```

18 Relation

Relations are all kinds of connections between classifiers (and other nodes). Examples of relations include common associations, the subclass (specialization) relation and the aggregation relation. While these constructs are semantically very different, they can all be drawn as connections between schemas.

`\umlRelation`

`\umlRelation` itself, like most of its subclasses, is of the form `\umlRelation[`⟨*Named options*⟩`]`⟨*From*⟩⟨*To*⟩⟨*Other contents*⟩, where ⟨*From*⟩ and ⟨*To*⟩ are references to nodes (e.g. classes and nodes). ⟨*Other contents*⟩ typically contains labels etc which should have the same colors etc as the relation. The argument ⟨*Other contents*⟩ are new since version 0.09. This may cause errors if one tries to use `\umlRelation` with the old arguments.

Figure 8 shows some of the capabilities of `uml.sty` relations.

The source of figure 8 is:

```
\umlDiagram[box=,sizeX=7cm, sizeY=7cm, ref=relation]{%
  \umlClass[pos=\umlBottomLeft{relation}, posDelta={1,1}, refpoint=bl,
            reference=A]{Class A}{}{}%
  \umlClass[pos=\umlTopRight{relation}, posDelta={-1,-1}, refpoint=tr,
            reference=B]{Class B}{}{}%
  \umlRelation{A}{B}{
    \umlLabelA{AB}{*}
    \umlLabelB{AB}{1}}
  \umlLabel[fraction=.5,offset=0]{AB}{center}
  \umlAssociationEnd[fraction=A, angle=U]{AB}{A}
  \umlAssociationEnd[fraction=B, angle=R]{AB}{B}
  \umlSubclass[ref=ABsub, angleA=0, armA=5, armAngleA=0,
    angleB=300, linecolor=blue,nodesep=1ex]{A}{B}
  \umlComposition[reference=ABComp,
    angleA=120, arm=4, armAngleA=80,
    angleB=180, armAngleB=190]{A}{B}
  \umlNavigabilityA{ABComp}
  }
```

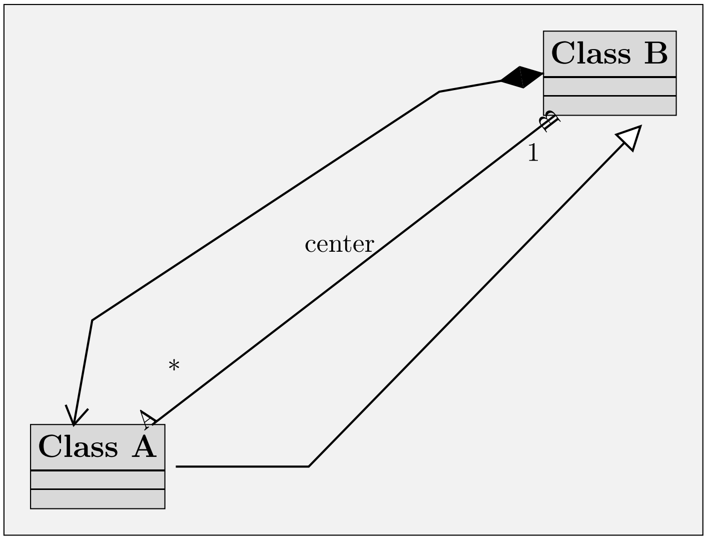


Figure 8: Some relations (silly relations, but good example)

18.1 Appearance of the connector

The visual appearance of the connector is influenced by the following named option:

umllinestyle- **`umllinestyle`** Aumllinestyle to draw the connector. Default is `solid`.

linecolor- **`linecolor`** The color of the line.

18.2 Geometry of the connector

The connector consists of three line segments: A main part in the middle and two arms. By default, the arms have zero length. The arms are sometimes referred to as `A` and `B`.
In the example, figure 8, the both arms of the middle relation and one of the blue one have lengths zero.

Where the connector hits the node By default, the connector tries to be as short as possible. With no named options, the connector will be a straight line between the nodes. This can be overridden by the named option

angle- **`angle`, `angleA` and `angleB`** Angle at which the connector hits the node. `angleA` affects ⟨*From*⟩, `angleB` ⟨*To*⟩ and `angle` both of them.

The arms By default, there are no arms (they have lengths zero). This can be overridden by the named options

arm- **`arm`, `armA` and `armB`** The length of the arm. Default is 0pt, which means there is no arm at all.

armAngle- **`armAngle`, `armAngleA` and `armAngleB`** The direction of the arm. Default is 0 (meaning right).

Nodesep It is possible to make a separation between the nodes and the connector by the named options

`nodesep`, `nodesepA` and `nodesepB` Legal values are lengths, default is 0.

18.3 Reference and placement of nodes

As `\umlElement`s, relations have references. Default is the concatenation of the references of ⟨*From*⟩ and ⟨*To*⟩. This can as usual be overridden by the named option `reference`.
With the reference set as ⟨*reference*⟩, the following nodes are set:

- `Aa`⟨*reference*⟩ Where the connector hits node A (⟨*From*⟩)
- `Ba`⟨*reference*⟩ Where the connector hits node B (⟨*To*⟩)
- `Ab`⟨*reference*⟩ One line segment from `Aa`⟨*reference*⟩.
- `Bb`⟨*reference*⟩ One line segment from `Ba`⟨*reference*⟩.
- `Ac`⟨*reference*⟩ Where the main line segment hits arm or node A

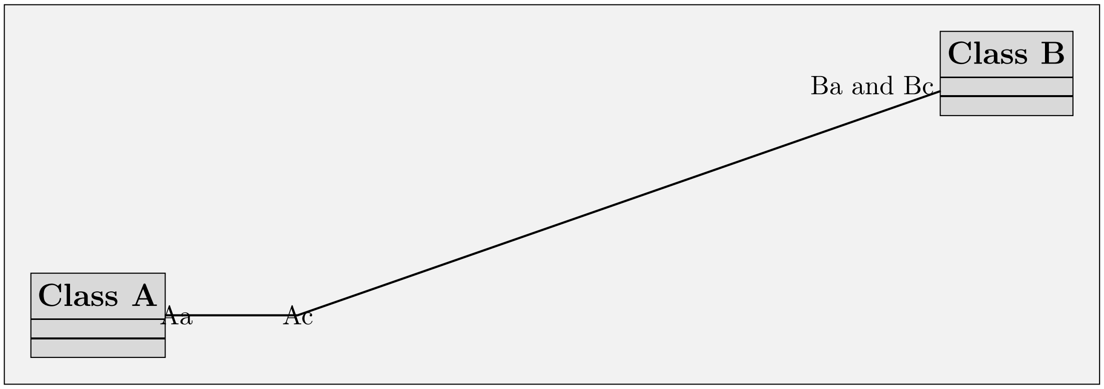


Figure 9: Placement of relation nodes. In this case, Both SymbolAb and SymbolBb is where Ac is (one line segment from resp. nodes).

- `Bc`⟨*reference*⟩ Where the main line segment hits arm or node B

The clever reader will understand that `Ab`⟨*reference*⟩ will be the same location as `Ac`⟨*reference*⟩ (if there is an arm A) or as `Bc`⟨*reference*⟩ (if there is not). In the latter case, `Ac`⟨*reference*⟩ will be the same location as `Aa`⟨*reference*⟩. Similarly in the opposite direction.
Tip: The connector is drawn with `\ncdiag`. Just after making the relation, you can make your own `\pnode` with something like `\lput{:R}(`⟨*a number 0–3*⟩`){\pnode{`⟨*your reference*⟩`}}`.

19 AssociationEnd

An relation may have different attachments attached to it. It may have labels specifying multiplicities and a end-symbol (such as a triangle in the subclass case). All this can be made with `\umlAssociationEnd`.

`\umlAssociationEnd` Its syntax is `\umlAssociationEnd[`⟨*named options*⟩`]`⟨*relation*⟩⟨*Symbol*⟩, where ⟨*relation*⟩ is the reference of a relation and ⟨*symbol*⟩ is the symbol to be placed (e.g., an asterisk or a triangle).

19.1 Placing of the symbol

The location of the symbol is determined this way:

- You pick the direction along the relation
- `posAngle` Relativeto this direction, you pick the direction determined by `posAngle`. But, since this direction is `:U`, up, by default, you probably still move in the direction along the relation. You probably don't want to modify posAngle.
- `pos-` In this direction, move the fraction specified by `pos`of the relation. This is default 0. But if you specify `fraction=0.2`, you move 20 % along the line. Legal values, in addition to

real numbers, are `A` and `From`, meaning near node A, and `B` and `To`, meaning you move from node B to A.

- `offset-` Still in this direction, move the length specified by `offset`. If you say `offset=2ex`, you are 20 % of the line + 2 ex away from the node now.
- `height-` To the right of this direction, move the length specified by `height`.
- `refpoint-` Here, place the point on the symbol specified by `refpoint`. This is default `B` (baseline).
- `angle-` Rotate the symbol in the direction specified by `angle`. This is default `U` (up).

Much of the point in `\umlLabel` and `\umlSymbol` is to provide other defaults for different uses. Common relations have their specialized commands (like `\umlSubclass` and `\umlAggregation`), to be explained later on. These commands call `\umlRelation` and then `\umlSymbol` with the appropriate symbol.

20 Label

`\umlLabel` This is an association end which new defaults: offset is 4 ex, height is 4 ex and direction is N (north, absolute up). This is good defaults for label.
In labels, if pos is `B` or `To`, the height is inverted.
`\umlLabelA` The command`\umlLabelA` is an `\umlLabel` with fraction=A. `\umlLabelB` is an `\umlLabel` with fraction=B.
In figure 8, there are three labels: 1, * and “center”. “A” and “B” is implemented directly by means of `\umlAssociationEnd`.

21 Symbol

`\umlSymbol` Thisis an association end with new defaults. Refpoint is `t` (top).
`\umlSymbolA` The command`\umlSymbolA` is an `\umlSymbol` with fraction=A. `\umlSymbolB` is an `\umlSymbol` with fraction=B.
In figure 8, there are two symbols (the diamond in the left one and the triangle in the right)

22 Navigability

Navigability indicates which direction the relation can be followed. Graphically, it looks like an open arrow.
The syntax is `\umlNavigability[`⟨*named options*⟩`]`⟨*Relation*⟩. The symbol is predefined.
`\umlNavigabilityA` Navigability can also be drawn with the commands `\umlNavigabilityA` and `\umlNavigabilityB`.
In figure 8, there is one navigability symbol (towards Class A).

23 The different relations

There are several different relations; being semantically very different, they all are connectors between nodes, and they all are implemented as subcommands of `\umlRelation`, with various end symbols.

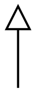
23.0.1 Association

`\umlAssociation`

`\umlAssociation` is only a wrapper to `\umlReference`. Graphically, an association is a reference without any fuzz. An example of an association (between nothing, though) is shown in the left margin of this text.

23.0.2 Subclass (generalization)

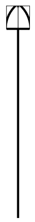
`\umlSubclass`
`\umlGeneralization`



This relation is in UML named *generalization*. We prefer to name it the *subclass relation* (really, more relations are about generalization). However, for the sake of compatibility, the commands `\umlSubclass` and `\umlGeneralization` are offered as equals.
The relation goes from the most special node to the most general one. Graphically, it is a solid line with a triangle at the general end.

23.0.3 Inner class

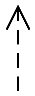
`\umlInner`



The graphical notation for this is not part of standard UML, and not very well worked through either. However, I find it very useful to have a notation for this (without drawing the classes inside each other).
Bug: I have not been able to get rid of the gray thing borders made by `\psClip`.
I have not investigated the differences and similarities in the semantics of what I call Inner Class, Java Inner Classes, Schemas which contain other classes and traditional UML composition.
The command is `\umlInner`.

23.0.4 Instance

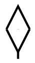
`\umlInstance`



This is the relation between a class and its metaclass. This is the real "is-a" relation. It is the strongest kind of a generalization relationship, even if it is not called generalization in standard UML. Its symbol is an open arrow on a dashed line.
The command is `\umlInstance`.

23.0.5 Aggregation

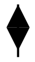
`\umlAggregation`



The target class (the upper class in the left margin) is an aggregate; therefore, the source class is a part. This is the weak form of aggregation, indicated by a hollow diamond. The part may be contained in other aggregates. The aggregation is optional, but not suppressible.

23.0.6 Composition

`\umlComposition`



The strong form of aggregation, which requires that a part instance be included in at most one composite at a time and that the composite object has sole responsibility for the disposition of its parts. In other words, a part can be part of at most one composite.

23.0.7 Application

`\umlApplication`

This is the relation between an abstraction (a schema taking arguments) and the application

(the schema with the arguments). Application, and the entire argument idea, is not part of standard UML. For example, see the section about Argument on page 18.
`\umlApplication` places a node `argument`⟨*reference*⟩ in addition to the usual ones.

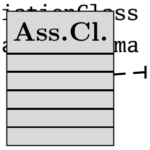
23.1 Relations to Relations

`\umlToRelation`
`posMeetLine-`

Some constructs can be viewed as relations between a classifier and a relation. They are implemented as subcommands of `\umlToRelation`.
`\umlToRelation`takes the named option `posMeetLine`, a real number between 0.0 and 1.0, which determines where at the target relation the relation shall hit. Default is 0.5, in the middle of the relation.
The node `ToRelation`⟨*reference of this relation*⟩ is placed where the lines meet.

23.1.1 AssociationClass

`\umlAssociationClass`
`\umlAssocia ma`



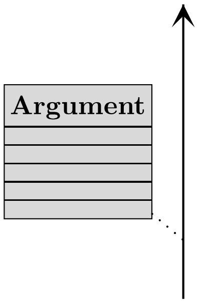
An association class is an association which also has attributes and methods. Similarly, we can speak about an association schema. Graphically, the schema is drawn as an usual schema with a dashed connector to the association.

23.1.2 ArgumentRelation

`\umlArgument`

In this construct, we speak about three differenc classifiers:

- The *abstraction* (Stack in figure 10) is the schema which can take argument.
- The *application* (Stack of books) is the abstraction after applying the argument. The application is more special than the abstraction.
- The *argument* (Book) is what is filled in.



Graphically, this is shown as two connectors: One solid with an arrow from the application to the abstraction. It is made by `\umlApplication`. Then, a dotted from this line to the argument. It is made by `\umlArgument`.
By default, the dotted line hits the solid one quite near the application, at about 20% of the main line segment (position 1.2).
An example of this construct is shown in figure 10 on the following page. The LaTeX source is here:

```
\umlDiagram[sizeX=10cm, sizeY=12cm,
            ref=argumentDiagram,box=,border]{
  \umlSchema[pos=\umlTopLeft{argumentDiagram}, posDelta={2ex, -2ex},
            refpoint=tl]{Stack}{%Attributes
    \umlAttribute[visibility=-, type=\emph{type}, default=null
                  ]{firstNode}%
    }{% Methods
    \umlMethod[visibility=+]{push}{\emph{type} x}
    \umlMethod[visibility=+, type=\emph{type}]{pop}{}
    }{% Arguments
    \umlArgument[type=Metaclass]{type}%
    }{% Constraints
    \umlCompartmentline{S:Stack = S.push(x).pop()}
    }{% Structure
    \umlDiagram[ref=StackDiagramA, outerBorder,innerBorder=2mm,box=]{%
       \umlClass[pos=\umlBottomLeft{StackDiagramA},
```

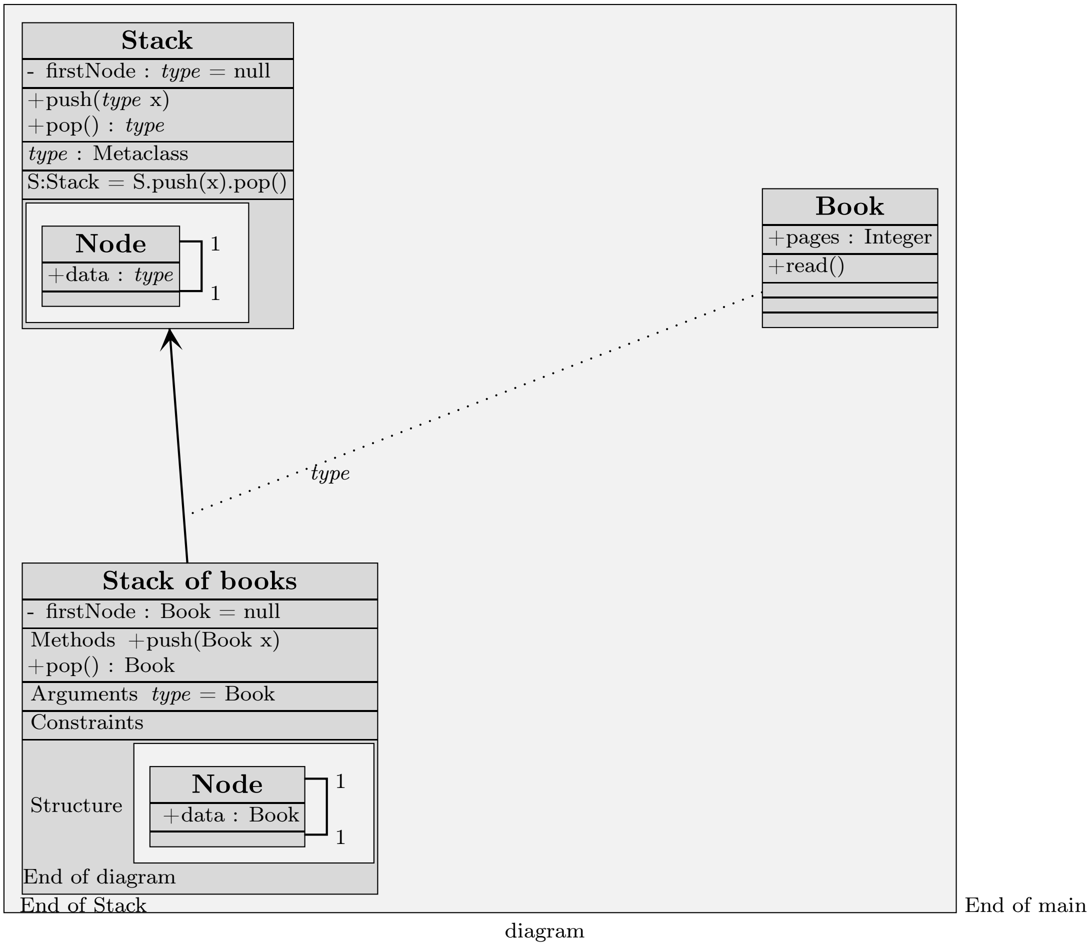


Figure 10: Example of abstraction and application

```
               posDelta={1ex, 1ex}, box=,
               ref=stackNode]{Node}{%
        \umlAttribute[visibility, type=\emph{type}]{data}}{}%
    \umlRelation[angleA=20, angleB=-20, armA=1em, armB=1em]{stackNode}{stackNode}{%
      \umlLabelA[height=-1ex, fraction=1.5]{stackNodestackNode}{1}%
      \umlLabelB[height=-1ex, fraction=1.5]{stackNodestackNode}{1}}%
    }\cr% End of diagram
  }% End of Stack
\umlSchema[refpoint=br, posY=\umlBottom{Stack},
           posX=\umlRight{argumentDiagram},
           posDelta={-2ex,0}]{%
  Book}{% Attributes
  \umlAttribute[visibility, type=Integer]{pages}
  }{% Methods
  \umlMethod[visibility]{read}{}
  }{% Arguments
  }{% Constraints
  }{% Structure
  }% End of Book
\umlSchema[pos=\umlBottomLeft{argumentDiagram}, posDelta={2ex, 2ex}, refpoint=bl,
           reference=StackofBooks]{Stack of books}{%Attributes
  \umlAttribute[visibility=-, type=Book, default=null, ]{firstNode}
  }{% Methods
  \umlMethod[visibility=+]{push}{Book x}
  \umlMethod[visibility=+, type=Book]{pop}{}
  }{% Arguments
  \umlCompartmentline{\emph{type} = Book}
  }{% Constraints
  }{% Structure
  \umlDiagram[innerBorder=2mm,box=,innerBorder=2mm,outerBorder]{%
    \umlClass[posDelta={1ex, 1ex}, ref=stackNode,box=]{Node}{
      \umlAttribute[visibility, type=Book]{data}}{}%
    \umlRelation[angleA=20, angleB=-20, armA=1em, armB=1em]{stackNode}{stackNode}{%
      \umlLabelA[height=-1ex, fraction=1.5]{stackNodestackNode}{1}%
      \umlLabelB[height=-1ex, fraction=1.5]{stackNodestackNode}{1}}%
    }\cr% End of diagram
  }% End of Stack
\umlApplication[reference=ssb]{StackofBooks}{Stack}
\umlArgumentRelation{Book}{ssb}
\umlLabel[fraction=0.7]{Bookssb}{\emph{type}}
}% End of main diagram
```

24 Package

A package is a collection of classes, relations and other elements. It is simply a grouping of different elements.
The graphic symbol consists of two rectangles above each other: The upper one is small and contains the name. The lower one is typically large and contains the elements. The lower rectangle typically contains an `\umlDiagram`.

In uml.sty, packages are drawn by the command `\umlPackage[`⟨*named options*⟩`]{`⟨*name*⟩`}{`⟨*stuff*⟩`}`. `\umlPackage` is an `\umlBox`, and inherits all its named options.

24.1 Example

In figure 11, the package "Package" contains four classes, four usual associations and one subclass relation.

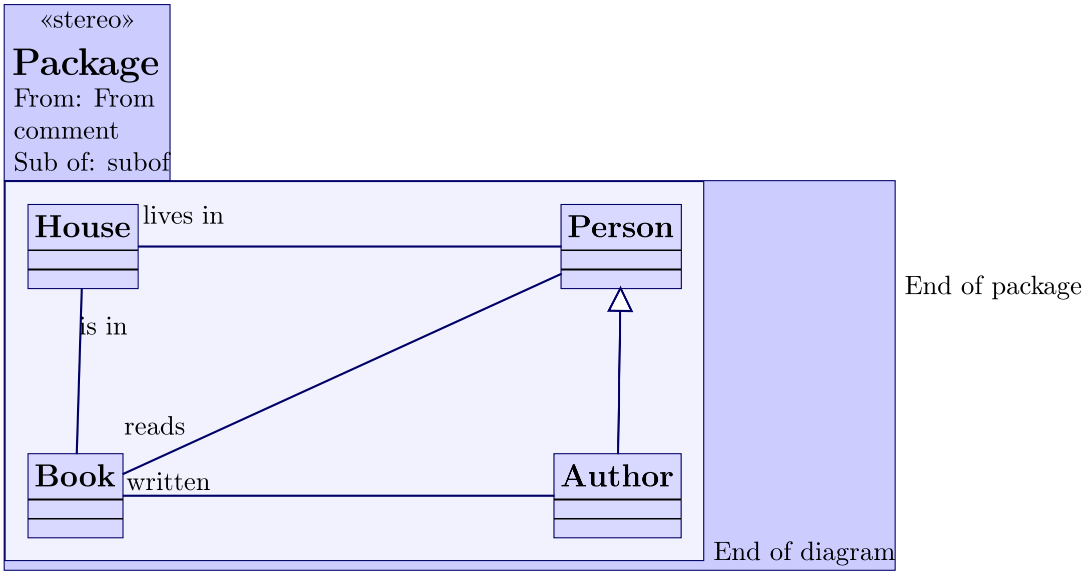


Figure 11: Example of package

The source:

```
\umlPackage[border=,box=,
      subof=subof, stereotype=stereo, importedFrom=From,
      comment=comment]{Package}{%
   \umlDiagram[sizeX=7cm, sizeY=5cm,box=,ref=pack]{
     \umlClass[pos=\umlBottomLeft{pack}, posDelta={2ex, 2ex},
               refpoint=bl]{Book}{}{}
     \umlClass[pos=\umlTopLeft{pack}, posDelta={2ex, -2ex},
               refpoint=tl]{House}{}{}
     \umlClass[pos=\umlTopRight{pack}, posDelta={-2ex,-2ex},
               refpoint=tr]{Person}{}{}
     \umlClass[pos=\umlBottomRight{pack}, posDelta={-2ex, 2ex},
               refpoint=br]{Author}{}{}
     \umlAssociation[]{Book}{House}
     \umlLabelB{BookHouse}{is in}
     \umlAssociation[]{Book}{Person}
     \umlLabelA{BookPerson}{reads}
     \umlSubclass{Author}{Person}
     \umlAssociation{Book}{Author}
```

```
    \umlLabelA[height=.5ex]{BookAuthor}{written}
    \umlAssociation{House}{Person}
    \umlLabelA{HousePerson}{lives in}
    }% End of diagram
}% End of package
```

24.2 Connecting packages

Two packages can be connected in the same way, and using the same function, as two classes. Symbols, labels etc. work the same way, and you can even connect a package and a class. Not all these possibilities make sense semantically.
However, due to the geometrical shape of packages, there is some problems with the connections. I have found no good solution to this. Now, default, relations is connected to an imaginary rectangle covering the entire package. The problem occur if the node hits the package in the upper right corner, where the package does not fill out the rectangle.
To solve this problem preliminarily, two more nodes are placed: The upper rectangle is given the name `small`⟨*reference*⟩, and the lower `big`⟨*reference*⟩. You can make the connector connect to one of these if desired.

25 Colors

The metamodel for this is in figure 4 on page 10.

25.1 Colorset

In uml.sty, there is always an *color set* in action. There are currently six defined color sets:

ColorsDefault uses normal black and white colors.

ColorsGray uses more gray colors. Typically used for constructs which not is to be emphazised.

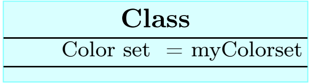
ColorsImport uses blue colors. Typically used for constructs which are defined in another document, and only are imported here.

ColorsArgument uses green colors. Typically used for constructs given as arguments.

ColorsRed uses red colors. Use it as you like.

ColorsSub is intended to be used on material inherited from other material.

Demonstrations of this color sets is in figure 4.
If you want to make your own color set, you may say something like



```
\newcommand\myColorset{%
  \umlColorsSet{%
    \newrgbcolor{umlColor}{0 .4 .4}%
    \newrgbcolor{umlLinecolor}{0.5 1 1}%
    \newrgbcolor{umlFillcolor}{.8 1 1}%
    \newrgbcolor{umlClassifierFillcolor}{.85 1 1}%
    \newrgbcolor{umlDiagramFillcolor}{.95 1 1}%
    \newrgbcolor{umlRelationColor}{0 1 1}%
    }%
}
```

You may want to use the commands `\newrgbcolor{`⟨*name*⟩`}{`⟨*red*⟩ ⟨*green*⟩ ⟨*blue*⟩`}`, `\newgray{`⟨*name*⟩`}{`⟨*grayness*⟩`}`, `\newhsbcolor{`⟨*name*⟩`}{`⟨*hue*⟩ ⟨*saturation*⟩ ⟨*brightness*⟩`}` or `\newcmykcolor{`⟨*name*⟩`}{`⟨*cyan*⟩ ⟨*magenta*⟩ ⟨*yellow*⟩ ⟨*black*⟩`}`, where all the parameters except ⟨*name*⟩ are real numbers between 0 and 1.

25.2 Using the color sets

You can use the color sets in different ways:

`\umlColors...`

- You can use the command directly (e.g., say `\umlColorsDefault`), and the color set will take effect on the following constructs.

`umlColorset`

- You can use the environment , which takes the color set as argument (e.g. `\begin{umlColors}{\umlColorsDefault}`).

`colorSet-`

- Indrawables, (i.e., most constructs) you can use the named option `colorSet` (e.g., `\umlClass[colorSet=\umlColorsDefa`
- Some color sets are also implied in other named options (`import=`) and otherwise.

Colors are primarily handled by Drawable. However, due to some technical obscurities in TEX, the implementation is done in the known subclasses (`\umlElement`, `\umlCompartment` and `\umlCompartmentline`).

Technically, the commands only define another command (`\umlColorsAdjust`), which is called in every construct. The value of `\umlColorsAdjust` *til enhver tid* is the color set in action. See 45.1 for more details on this.

26 Positions

The metamodel for this is in figure 5 on page 11.

In PSTricks, a Node can serve as a Coordinate. There exist other types of coordinates too. Box.pos uses `\rput`, and is a coordinate. Box.posDelta should be a relative coordinate (a cartesian number pair), but could really be any coordinate.

Relation.NodeA and Relation.NodeB, however, uses `\ncdiag`, and must be Nodes. This is confusing. It is also confusing that coordinates in PSTricks are written in parenthesis, while nodes are not. In uml.sty, no parenthesis are used.

This imply that relation only can be between Nodes, and not between other coordinates. But sometimes, we want to use the power of other coordinate kinds while placing relations.

26.1 PlaceNode

To do that, use the command `\umlPlaceNode`, which has a lot of power, and places a node. Note that `\umlPlaceNode` only places a new node (named after its third argument); it is no node itself.

One may want to do something like

```
\umlPlaceNode[top=-2em,rightside]{A}{Aright}
\umlPlaceNode[top=-2em,leftside]{B}{Bleft}
\umlSubclass[ref=AB]{Aright}{Bleft}
```

Why should it not be legal to say something like

```
\umlSubclass[top=-2em,rightsideA, leftsideB]{A}{B}
```

?

It is possible to implement the latter syntax, but that would require a great amount of new nodes. It is already a problem that uml.sty relations uses so many nodes, so I have chosen to keep the first syntax. After all, I do not expect `\umlPlaceNode` to be used often.

`\umlPlaceNode`

`\umlPlacenode`takes three arguments: Its named options, a node and a name to be used as a name of the new node.

It takes several named options:

`leftside-`
`rightside-`
`top-`
`bottom-`
`left-`
`right-`
`up-`
`down-`
`angle-`
`offset-`
`nodesep-`

`leftside`, `rightside`, `top` and `bottom` makes `\umlPlaceNode` start at one side of the node. They all can take one length as argument. One horizontal and one vertical of these can be combined.

`left`, `right`, `up` and `down` These makes `\umlPlaceNode` go in one of the four directions. Several (even of the same type) can be combined. Legal values are lengths.

`angle`, `angleX` and `angleY` angle from the node to start with.

`offset`, `offsetX` and `offsetY`

`nodesep`, `nodesepX` and `nodesepY` separation from the node.

26.2 Coordinates

Legal *coordinates* are:

⟨*x*⟩, ⟨*y*⟩ The usual Cartesian coordinate. E.g., `3,4`.

⟨*r*⟩;⟨*a*⟩ Polar coordinate, with radius ⟨*r*⟩ and angle ⟨*a*⟩. The default unit for ⟨*r*⟩ is the PSTricks `unit`. E.g., `3;110`.

⟨*node*⟩ The center of ⟨*node*⟩. ⟨*node*⟩ can be any valid PSTricks node. Various commands in `uml.sty` places a number of nodes, which can be used as reference points.

- Most constructs can be given a node name using the `reference=` named option.
- For classes and schemas, default node name (when `reference=` is not used) is the Class name or Schema name itself.
- Relations place nodes, see documentation in 18 on page 20.

[⟨*par*⟩]⟨*node*⟩ The position relative to ⟨*node*⟩ determined using the `angle`, `nodesep` and `offset` parameters. E.g., `([angle=90]Car)`.

⟨*coor1*⟩|⟨*coor2*⟩ The *x* coordinate from ⟨*coor1*⟩ and the *y* coordinate from ⟨*coor2*⟩. ⟨*coor1*⟩ and ⟨*coor2*⟩ can be any other coordinates. For example, `(Car|1in;30)`. However, you may instead want to use named options such as `posX` and `deltaPosY` to achieve this.

Position commands take a node as parameter and return an coordinate. The different position commands are shown below. E.g., `\umlClass[pos=\umlBottomRight{Car}]{Wheel}{}{}`.

Note that you should usually omit the parentheses in `uml.sty`. This is due to the fact that the `(`⟨*coor1*⟩`|`⟨*coor2*⟩`)` construct requires two parts without parentheses.

Also note that coordinate pairs containing parentheses as values to named options my cause problems. You have to enclose them in braces. E.g., `\umlClass[pos={3,3}]{Car}{}{}`.

26.2.1 Coordinate commands

This is kept for backward compatibility. However, only the first group of Coordinate commands should be useful now. The others are deprecated.

A coordinate command takes an node (and possibly other things) as argument, and return a coordinate. Several groups of coordinate commands exist:

`\umlTop`
`\umlRight`
`\umlBottom`
`\umlLeft`
`\umlTopRight`
`\umlBottomRight`
`\umlBottomLeft`
`\umlTopLeft`

Simple commands The commands `\umlTop`, `\umlRight`, `\umlBottom`, `\umlLeft`, `\umlTopRight`, `\umlBottomRight`, `\umlBottomLeft` and `\umlTopLeft` all take one argument: the node

`\umlTopSep`
`\umlRightSep`
`\umlBottomSep`
`\umlLeftSep`
`\umlTopRightSep`
`\umlBottomRightSep`
`\umlBottomLeftSep`
`\umlTopLeftSep`
`\umlNodeSep`

Commands with separation The commands `\umlTopSep`, `\umlRightSep`, `\umlBottomSep`, `\umlLeftSep`, `\umlTopRightSep`, `\umlBottomRightSep`, `\umlBottomLeftSep` and `\umlTopLeftSep` returns a position `\umlNodeSep` from the position of the argument. `\umlNodeSep` is default 1em, but may be changed by the user.

`\umlTopOpt`
`\umlRightOpt`
`\umlBottomOpt`
`\umlLeftOpt`
`\umlTopRightOpt`
`\umlBottomRightOpt`
`\umlBottomLeftOpt`
`\umlTopLeftOpt`

Commands taking options The commands `\umlTopOpt`, `\umlRightOpt`, `\umlBottomOpt`, `\umlLeftOpt`, `\umlTopRightOpt`, `\umlBottomRightOpt`, `\umlBottomLeftOpt` and `\umlTopLeftOpt` takes two arguments, both mandatory: a comma-separated list of ⟨*key*⟩=⟨*value*⟩ pairs, where ⟨*key*⟩ can be `angle` (value: number), `nodesep` (value: length) or `offset` (value: length). The second argument is the node

`\umlTopSepOpt`
`\umlRightSepOpt`
`\umlBottomSepOpt`
`\umlLeftSepOpt`
`\umlTopRightSepOpt`
`\umlBottomRightSepOpt`
`\umlBottomLeftSepOpt`
`\umlTopLeftSepOpt`
`\umlNodeSepOpt`

Commands with separation taking options The commands `\umlTopSepOpt`, `\umlRightSepOpt`, `\umlBottomSepOpt`, `\umlLeftSepOpt`, `\umlTopRightSepOpt`, `\umlBottomRightSepOpt`, `\umlBottomLeftSepOpt` and `\umlTopLeftSepOpt` takes two arguments and do just what you expect them to.

27 A large example (Abstract Data Structures)

Figure 12 on the next page is a more large model of some standard ADTs. It is an example of both `\umlSchema` and some relations. It also is an example of the Schema construct itself, but is primarily intended to be a `uml.sty` example, and is not intended to be a good ADT model.

The command Technically, the following source defines a command `\umlADTExample`, which is used in figure 12 on the following page.

```
\providecommand\umlADTExample{
```

The diagram The diagram is a box which uses room. It is 15 cm wide and 16 cm high, and has the reference `ADTdiagram`. It has a grayness of 92 %. If we did not supply a grayness, it would get the `umlDiagramFillcolor` of the current color set.
Note that, technically, it is not necessary to have the semantical contents of the diagram inside its LaTeX argument.

```
   \umlDiagram[box=,sizeX=15cm, sizeY=16cm,ref=ADTdiagram,
               grayness=0.92]{}% End of diagram
```

ADT The schema (or class) ADT is located in the top right corner of the diagram (which had reference `ADTdiagram`. More accurately, its top right (`tr`) corner is placed half an unit left and half an unit down relative the the top right corner.

```
     \umlSchema[pos=\umlTopRight{ADTdiagram}, posDelta={-.5,-.5},
       refpoint=tr]{ADT}{% Attributes
       \umlAttribute[visibility,type=String]{name}}{}{}{}{}
```

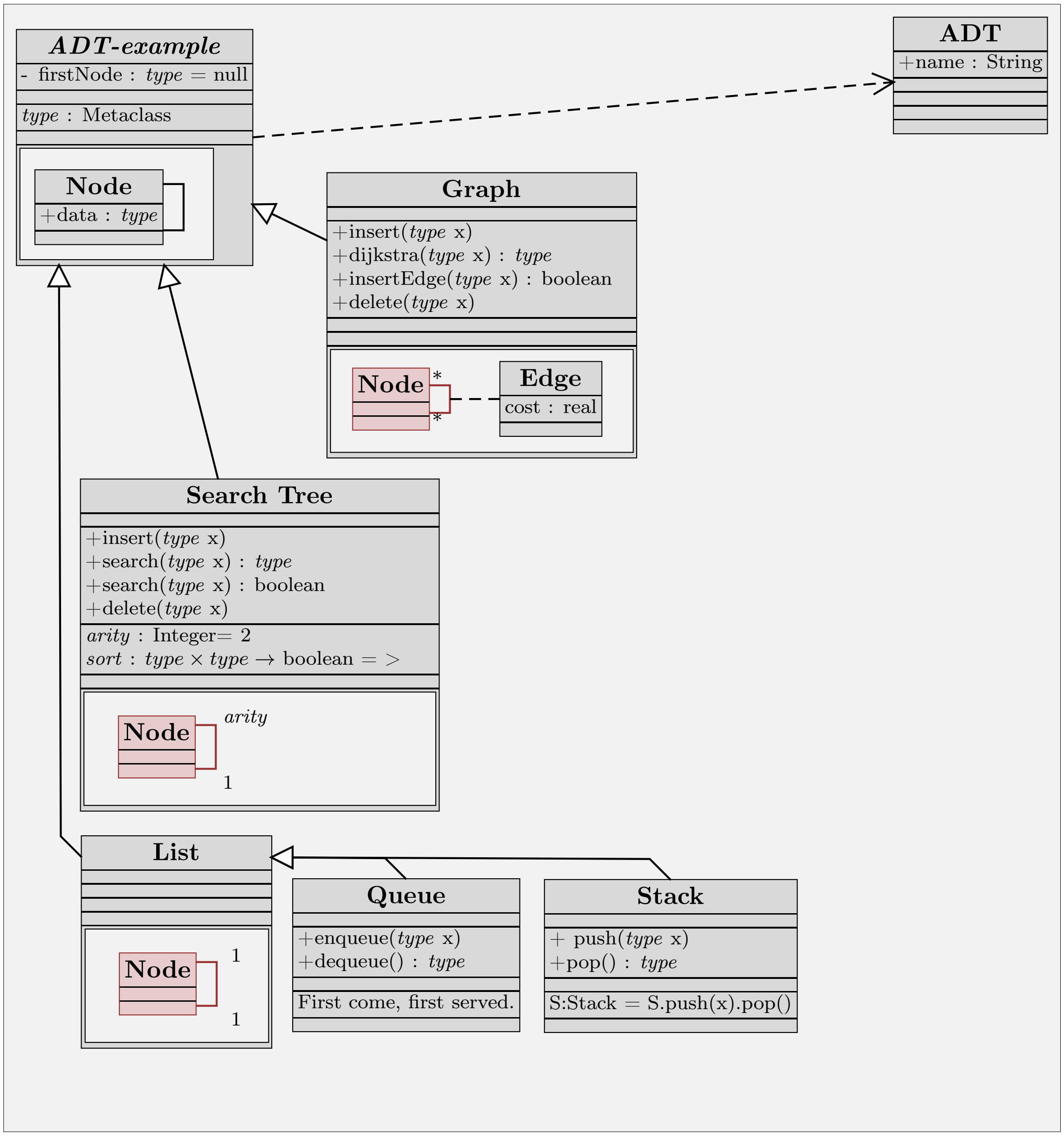


Figure 12: ADT model. Bad model, good uml.sty example.

ADT-example The schema ADT-example is located in the upper left corner of the diagram. It is an abstract schema, giving italics name in the diagram. Because the class name contains a dash, a reference (`ADTexample`) must be supplied.

This schema also takes an *argument* (type). This is not the place to fully explain the semantics of arguments, but *type* can be given any Metaclass as value.

Everywhere in the schemas an argument is used, this color is used. Here, the code is getting a bit *clogged up* with color code. If you read this the first time, please ignore the commands starting with `\umlColor`.

```
\umlSchema[pos=\umlTopLeft{ADTdiagram}, posDelta={.5,-1},
           refpoint=lt, abstract,
           ref=ADTexample]{ADT-example}{%
  \umlAttribute[visibility=-,
    type=\emph{\umlColorsArgument\umlColorsAdjust type},default=null]{%
    firstNode}
  }{ %Methods
  }{ %Arguments
  \umlArgument[type=Metaclass]{type}
  }{ %Constraints
  }{ %Structure
    \umlDiagram[box=,innerBorder=2mm,outerBorder]{%
      \umlClass[pos={.5,.5}, ref=adtNode,box=]{Node}{%
        \umlAttribute[visibility,
          type=\emph{\umlColorsArgument\umlColorsAdjust type}]{%
          data}}{}%
      \umlAssociation[angleA=20, angleB=-20,
                      arm=1em, arm=1em]{adtNode}{adtNode}%
   }\cr% End of Diagram
 }% End of ADT-example
```

Then the isInstance relation between ADT-example and ADT. This is shown as a dashed arrow with an arrowhead towards ADT.

```
\umlInstance{ADTexample}{ADT}%
```

Graph Graph is a subclass (subschema) of ADT-example. Thus, it is implicitly inheriting the attribute firstNode, the argument *type*, the structure Node (with the relation), and the isInstance-relationwhip to ADT.

What is added, is four well-known procedures, the *–*-specification of the Node–Node-relation, and the association class Edge.

What is inherited from ADT-example (Node and the Node–Node-relation) is drawn with the `\umlColorsSub` color set. Note that the relation is brown, while the relation multiplicities is black.

All the nodes are given references, in order to separate them from each other (they have equal class names). However, it might work to let `Node` mean the last defined Node all the time.

```
\umlSchema[pos=\umlRight{ADTexample}, posDelta={3,-1},
           refpoint=tl, ]{Graph}{% Attributes
  }{% Methods
  \umlMethod[visibility,]{%
    insert}{\emph{\umlColorsArgument\umlColorsAdjust type} x}
  \umlMethod[visibility,
      type=\emph{\umlColorsArgument\umlColorsAdjust type}]{%
    dijkstra}{\emph{\umlColorsArgument\umlColorsAdjust type} x}
  \umlMethod[visibility, type=boolean]{%
    insertEdge}{\emph{\umlColorsArgument\umlColorsAdjust type} x}
  \umlMethod[visibility, ]{%
    delete}{\emph{\umlColorsArgument\umlColorsAdjust type} x}
  }{% Arguments
  }{% Constraints
  }{% Structure
    \umlDiagram[box=,innerBorder=2mm, outerBorder,
                sizeX=11em,sizeY=3.5em,ref=GraphDiagram]{%
         \begin{umlColors}{\umlColorsSub}
      \umlClass[pos=\umlBottomLeft{GraphDiagram},
                posDelta={1,1}, ref=graphNode]{Node}{}{}%
      \umlAssociation[angleA=20, angleB=-20, armA=1em, armB=1em
                   ]{graphNode}{graphNode}%
         \end{umlColors}
        \umlLabelA[height=0mm,offset=1ex]{graphNodegraphNode}{*}%
        \umlLabelB[height=0mm,offset=1ex,refpoint=t
                   ]{graphNodegraphNode}{*}%
      \umlSymbol[fraction=.5]{graphNodegraphNode}{\pnode{gngn}}
      \umlClass[pos=gngn, posDelta={2,0},
                         ref=graphEdge, refpoint=l]{Edge}{%
      \umlAttribute[type=real]{cost}}{}%
    \umlAssociationClass[]{graphEdge}{gngn}%
    }\cr% End of diagram
  }% End of Graph
```

Graph is an subschema of ADT-example.

```
      \umlSubclass{Graph}{ADTexample}
```

Search Tree This schema is vertically positioned relative to ADT-example, and horizontally below Graph.

It has four well-known methods. It has two arguments (in addition to *type*); *arity* is an integer which default to 2, giving a binary tree. *sort* is a function itself taking as arguments two instances of *type* and returning true or false. It defaults to “greater than”.

```
      \umlSchema[posX=\umlLeft{ADTexample}, posDelta={3em,-1em},
                 posY=\umlBottom{Graph},
                 refpoint=tl, ref=searchTree]{Search Tree}{% Attributes
        }{% Methods
        \umlMethod[visibility,]{%
          insert}{\emph{\umlColorsArgument\umlColorsAdjust type} x}
        \umlMethod[visibility,
             type=\emph{\umlColorsArgument\umlColorsAdjust type}]{%
          search}{\emph{\umlColorsArgument\umlColorsAdjust type} x}
        \umlMethod[visibility, type=boolean]{%
          search}{\emph{\umlColorsArgument\umlColorsAdjust type} x}
        \umlMethod[visibility, ]{%
```

```
    delete}{\emph{\umlColorsArgument\umlColorsAdjust type} x}
  }{% Arguments
  \umlArgument[type=Integer, initialValue=2]{arity}
  \umlArgument[type={${\umlColorsArgument\umlColorsAdjust type}
    \times {\umlColorsArgument\umlColorsAdjust type}\rightarrow$
      boolean}, default=>]{sort}
  }{% Constraints
  }{% Structure
    \umlDiagram[box=, sizeX=14em, sizeY=4em,
                innerBorder=2mm, outerBorder]{%
      \begin{umlColors}{\umlColorsSub}
      \umlClass[pos={.5, .5}, ref=treeNode]{Node}{}{}%
      \umlAssociation[angleA=30, angleB=-30, armA=1em, armB=1em
                   ]{treeNode}{treeNode}%
       \end{umlColors}
        \umlLabelA[height=1mm, offset=4mm,refpoint=l
          ]{treeNodetreeNode}{%
          \emph{\umlColorsArgument\umlColorsAdjust arity}}%
        \umlLabelB[refpoint=tl,height=-1mm, offset=4mm,
                   ]{treeNodetreeNode}{1}%
      }\cr% End of diagram
  }%
\umlSubclass{searchTree}{ADTexample}%
```

List There is not much new here.

```
\umlSchema[pos=\umlBottomLeft{searchTree},posDelta={0,-1},
           refpoint=lt]{List}{%Attributes
  }{% Methods
  }{% Arguments
  }{% Constraints
  }{% Structure
    \umlDiagram[box=, sizeX=6em, sizeY=4em,
                innerBorder=2mm, outerBorder]{%
      \begin{umlColors}{\umlColorsSub}
        \umlClass[pos={.5, .5}, ref=listNode]{Node}{}{}%
        \umlAssociation[angleA=30, angleB=-30, armA=1em, armB=1em
                     ]{listNode}{listNode}%
      \end{umlColors}
        \umlLabelA[height=1mm, offset=5mm, refpoint=l
                   ]{listNodelistNode}{1}%
        \umlLabelB[refpoint=tl, height=-1mm,offset=5mm
                   ]{listNodelistNode}{1}%
      }\cr% End of diagram
  }% End of Schema List
```

Before making the actual subclass relation from List to ADT-example, we place node to point to and from. This is in order to better control the placement of the end points. (Really, it is to demonstrate `\umlPlaceNode` :-)

The first one is called `Listtl`, and is placed 1 em below the top left corner of List. The second is named `ADTexamplebl`, and placed 2 em to the right of the bottom left corner of ADT-example.
The subclass relation itself has an arm A of 1 em sticking out from List.

```
      \umlPlaceNode[leftside, top, down=1em]{List}{Listtl}
      \umlPlaceNode[leftside,right=2em,bottom]{ADTexample}{ADTexamplebl}
      \umlSubclass[armA=1.4142em, armAngleA=135]{Listtl}{ADTexamplebl}%
```

Queue A Queue inherits pretty much from List, ADT-example and ADT. Only two methods and one constraint (in natural language) is added explicitly.

```
      \umlSchema[pos=\umlTopRight{List},posDelta={1em,-2em},
                 refpoint=tl]{Queue}{% Attributes
        }{\umlMethod[visibility]{%
          enqueue}{\emph{\umlColorsArgument\umlColorsAdjust type} x}
        \umlMethod[visibility, type=\emph{\umlColorsArgument\umlColorsAdjust type}]{%
          dequeue}{}}{%Arguments
        }{%Constraints
        \umlCompartmentline{First come, first served.}
        }{% Structure
        }% End of Queue
      \umlPlaceNode[rightside, top, down=1em]{List}{Listtr}
      \umlSubclass[angleA=90, armAngleA=135, armA=1.4142em]{Queue}{Listtr}%
```

Stack Remember, Stack is still a subclass of List and of ADT-example, and an instance of ADT. Thus, Stack really resembles figure 10 on page 27 closely.

```
      \umlSchema[pos=\umlTopRight{Queue},posDelta={\umlNodeSep,0em},
                 refpoint=tl]{Stack}{%Attributes
        }{% Methods
        \umlMethod[visibility]{
          push}{\emph{\umlColorsArgument\umlColorsAdjust type} x}
        \umlMethod[visibility, type=\emph{\umlColorsArgument\umlColorsAdjust type}]{%
          pop}{}
        }{% Arguments
        }{% Constraints
        \umlCompartmentline{S:Stack = S.push(x).pop()}
        }{% Structure
        }% End of Stack
      \umlSubclass[angleA=90, armAngleA=135, armA=1.4142em]{Stack}{Listtr}%
}
```

28 Typesetting the implementation

I want the line numbers and the macro names to be right-aligned together, close to the text. It is important to make a virtual vertical line in order to make some order in the pages.
However, `doc.sty` contains a spurious space (in `\m@cro@`, where `\PrintMacroName` and where `\PrintEnvName` are used). This space shows up after the macro name. To overcome this, I make

a length `\minusSpace` of length one negative space in the right font, and make a `\hspace` of this length.

28.1 The documentation driver file

The next bit of code contains the documentation driver file for TeX, i.e., the file that will produce the documentation you are currently reading. It will be extracted from this file by the `docstrip` program. Since it is the first code in the file one can alternatively process this file directly with LaTeX 2$_\varepsilon$ to obtain the documentation.
<*package>

29 Introductory code

29.1 Identification

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{uml}
```

29.2 Processing of options

29.2.1 debug

`debug` The option `debug` is used if you want some extra lines shown. Intended for debugging of this package.

```
\DeclareOption{debug}{
  \def\umlDebugLineStyle{dashed}
  \def\umlDebugLength{1pt}
  }
\DeclareOption{nodebug}{
  \def\umlDebugLineStyle{none}
  \def\umlDebugLength{0pt}
}
```

`\umlDebugLinestyle` A linestyle normally none. The linestyle of invisible leading lines.

```
\def\umlDebugLinestyle{none}
```

`\umlDebugLength` A length normally 0pt. E.g., the breadth of invisible lines.

```
\def\umlDebugLength{0pt}
```

29.2.2 index

`index` The option `index` makes all Stretchboxes and Features (all Drawables with names make an index entry of the form ⟨*name*⟩!⟨*type*⟩.

```
\newcommand\umlIndexOn{\renewcommand\umlIndex[2]{\index{##1!##2}}}
\newcommand\umlIndexOff{\renewcommand\umlIndex[2]{}}
\DeclareOption{index}{\umlIndexOn}
\DeclareOption{noindex}{\umlIndexOff}
```

`\umlIndex` Takes two arguments: Type and Name.

```
\newcommand\umlIndex[2]{\index{#1!#2}}
```

29.2.3 Processing the options

Default is the `index` option.

```
\ExecuteOptions{nodebug}
\ProcessOptions
```

29.3 Using other packages

`uml.sty` relies heavily on the LaTeX packages `pstricks` and `pst-node` [PSTricks]. Most of the graphics is displayed using `pstricks`.

```
\RequirePackage{pstricks}
\RequirePackage{pst-node}
```

`pst-xkey` [xKeyval] is the package which handles Keys can be set as usual for PStricks related package with `\psset[uml]{...}`, where the family name `uml` is optional. named options (like `[name=Name, reference=Ref]`).

```
\RequirePackage{pst-xkey}
\pst@addfams{uml,umlAE,umlPlaceNode}
```

`relsize` [Relsize] handles relative sizes, with macros like `\Larger` and `\relsize{-1}`.

```
\RequirePackage{relsize}
```

We also need the `color` package, which is already loaded by the `pstricks` package.

30 General syntax

Users used to LaTeX and PSTricks will recognize a lot of syntax.
An important implementation point is that any contents (which can affect variables) is typeset after all the variables is used. If it is not, the different boxes inside each other get intertwined with strange results. `\umlDiagram` uses `\expandafter` to assert this.

```
\def\umlTrueVariable{true}
```

30.1 Lengths

```
\psset{unit=1em}
```

30.2 Angles

31 Drawable

Each main `uml.sty` command (those drawn as schemas at pages 7–11) is implemented in the same pattern.

`\umlDrawableNull` First, all the variables (in the `\umlDrawable` case, just one) are set to their default values.

```
\def\umlDrawableNull{%
   \def\umlImport{}%
   \def\umlKind{Drawable}%
   \gdef\umlName{DrawableNameDefault}%
   \def\umlNameDefault{DrawableNameDefault}%
   %\ifx\umlName\umlNameDefault\else umlName is changed
}
```

31.1 Named options

Of course, it would be more elegant to treat one variable at a time, setting it to its default value and handling its named options at the same place. However, I have not found any way to do so.
Then, the named options are handled. Most of them modify the variables. In this case, there is only one named option (`import`) which modify one variable (`\umlColor`). In most cases, the named option and the variable has corresponding names.

`kind-` The kind (type, metaclass) of the Drawable, e.g., `Class`.

```
\define@key[psset]{uml}{kind}{\def\umlKind{#1}}
```

`name-` The name of the Drawable, e.g., `Car`.

```
\define@key[psset]{uml}{name}{\gdef\umlName{#1}}
```

31.2 Colors

As said in section **??**, colors are primarily handled by Drawable. However, due to some technical obscurities in TEX, the implementation is done in the known subclasses (`\umlElement`, `\umlCompartment` and `\umlCompartmentline`).
The technical obscurities in TEX is this: The brackets (`{}`) needed to limit the scope of colors, cannot contain table line breaks. `\umlCompartmentline`, which uses to be placed inside a table (`\umlClassifier`), must put the line break outside the brackets.

`import-`

```
\define@key[psset]{uml}{import}{\umlColorsImport}
```

`noimport-`

```
\define@key[psset]{uml}{noimport}{\umlColorsDefault}
```

`argument-`

```
\define@key[psset]{uml}{argument}{\umlColorsArgument}
```

`argument-`

```
\define@key[psset]{uml}{colorset}[\umlColorsDefault]{#1}
```

31.3 The command

And then the command itself.

`\umlDrawable` This command does not do much work. It just process the nmed options.

```
\newcommand\umlDrawable[2][]{%
```

It sets its variables to default values,

```
  \umlDrawableNull%
```

process the named options

```
  \psset[uml]{kind=Drawable,#1}%
```

and typesets it second argument. This argument, the contents, typically uses some of the variables.

```
  #2%
}
```

32 Element

`\umlElementNull` `\umlElement` follows the same pattern as `\umlDrawable` and the other main commands; first, it sets its variables to the default values, then handles the named options, and least define the command itself to call its “supercommand”.

```
\def\umlElementNull{%
  \def\umlReference{refdef}%
  \def\umlStereotype{}%
  \def\umlSubof{}%
  \def\umlImportedFrom{}%
  \def\umlComment{}%
  \def\umlAbstract{}%
}
```

`reference-` `ref` is provided as an short-form for `reference`.

```
\define@key[psset]{uml}{reference}{\def\umlReference{#1}}
\define@key[psset]{uml}{ref}{\def\umlReference{#1}}
```

`stereotype-` The LaTeX variable itself sometimes contains not only the value itself, but some grahical stuff around.

```
\define@key[psset]{uml}{stereotype}{%
  \def\umlStereotype{{\hfil\phantom{x}<<#1>>\phantom{x}\hfil}\\}}
```

`subof-`

```
\define@key[psset]{uml}{subof}{\def\umlSubof{{~Sub of: #1}\\}}
```

`abstract-` The `abstract` named option also affects the graphical name in `\umlStretchbox`.

```
\define@key[psset]{uml}{abstract}[]{\def\umlAbstract{\emph}}%
```

`importedFrom-`

```
\define@key[psset]{uml}{importedFrom}{%
  \def\umlImportedFrom{{~From: #1}\\}%
  \umlColorsImport%
}
```

`comment-`

```
\define@key[psset]{uml}{comment}{\def\umlComment{{~#1}\\}}
```

`\umlElement` The command itself just calls `\umlDrawable`.

```
\newcommand\umlElement[2][]{%
  \umlElementNull%
  {\umlDrawable[kind=Element,#1]{%
      \umlColorsAdjust%
      #2}}}
```

33 Box

33.1 Positioning

One of the main responsibilities of `\umlBox` is to place the box in the right position. In order to achieve this, `\umlBox` uses two macros, `\umlBoxPosCommand` and `\umlBoxPosDeltaCommand`. Each of these, in turn, uses other macros, which ultimately are set to default values. How this happends is indicated in figure 13.

The user can modify this tree by the named options `pos`, `posX`, `posY`, `posDelta`, `posDeltaX`, `posDeltaY` and `refpoint`.

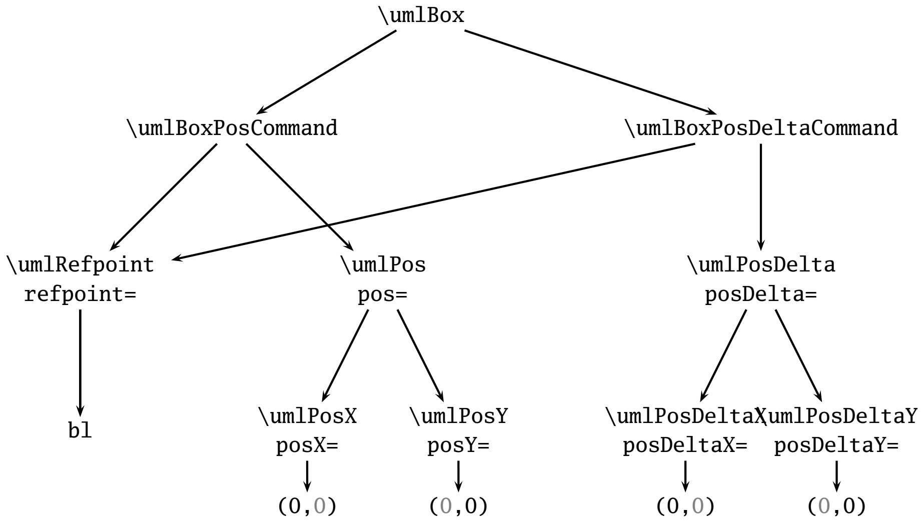


Figure 13: Positioning of boxes. The arrows here means "calls". The user can affects this tree by several named options.

`\umlBoxNullPositions` First, the variables are set to their null values. This command is called from `\umlBox` via `\umlBoxNull`.

```
\def\umlBoxNullPositions{%
  \def\umlPosCommand{%
    \rput[\umlRefpoint](\umlPos)}%
  \def\umlPos{\umlPosX|\umlPosY}%
  \def\umlPosX{0,0}%
  \def\umlPosY{0,0}%
  \def\umlPosDeltaCommand{%
    \rput[\umlRefpoint](\umlPosDelta)}%
  \def\umlPosDelta{\umlPosDeltaX|\umlPosDeltaY}%
  \def\umlPosDeltaX{0,0}%
  \def\umlPosDeltaY{0,0}%
  \def\umlRefpoint{bl}%
  }%
```

`pos-` Note that all the named options starting with `pos` takes legal values as arguments. You must write `posX={1em,0}` even if he zero is ignored.

```
\define@key[psset]{uml}{pos}[0,0]{\def\umlPos{#1}}
\define@key[psset]{uml}{posX}[0,0]{\def\umlPosX{#1}}
\define@key[psset]{uml}{posY}[0,0]{\def\umlPosY{#1}}
```

`posDelta-` The reference point of the box is placed at $pos + posDelta$.

```
\define@key[psset]{uml}{posDelta}[0,0]{\def\umlPosDelta{#1}}
\define@key[psset]{uml}{posDeltaX}[0,0]{\def\umlPosDeltaX{#1}}
\define@key[psset]{uml}{posDeltaY}[0,0]{\def\umlPosDeltaY{#1}}
```

`refpoint-` Legal values are *reference points* (sec. 1.4)

```
\define@key[psset]{uml}{refpoint}{\def\umlRefpoint{#1}}
```

33.2 Boxes in text

`\umlBoxNullBoxes` Normally, a box is an empty hbox. This can be changed using the named option `box=`. It takes no values (i.e. possible values are ignored). It makes the box taking the natural size of its contents.

```
\def\umlBoxNullBoxes{%
```

This is how the box is made a zero size hbox then the `box=` named option are not in effect.

```
  \def\umlBoxH{\hbox to 0cm}%
  \def\umlBoxV{\smash}%
  }%
```

`box-` This gives the box its natural size.

```
\define@key[psset]{uml}{box}{%
  \def\umlBoxH{}% no \hbox to 0cm
  \def\umlBoxV{}% no \smash anymore
  \def\umlPosCommand{}% no \rput... anymore
  \def\umlPosDeltaCommand{}}% Ditto
```

33.3 The visual appeareance

`\umlBoxNullVisual`

```
\def\umlBoxNullVisual{%
  \def\umlGrayness{1}%
  \def\umlBorder{0mm}%
  \def\umlInnerBorder{0mm}%
  \def\umlOuterBorder{0mm}%
  \def\umlFillcolorCommand{umlFillcolor}%
  }%
```

`grayness-` The grayness of the background in the box. Legal values are real numbers between 0 (black) and 1 (white). Default without named option is 1. Default with named option is 0.85.

```
\define@key[psset]{uml}{grayness}[.85]{\definecolor{umlFillcolor}{gray}{#1}}
```

`fillcolorCommand-` `\umlFillcolorCommand` returns the name of the current fill color.

```
\define@key[psset]{uml}{fillcolorCommand}[umlFillcolor]{%
  \def\umlFillcolorCommand{#1}}
```

`border-` The thickness of the outer border. Default without named option is 0 mm, with 0.4 pt. Legal values are lengths.

```
\define@key[psset]{uml}{border}[0.4pt]{\gdef\umlBorder{#1}}
```

`outerBorder-` The margin around the border. Default is `\umlhsep`.

```
\define@key[psset]{uml}{outerBorder}[1pt]{\def\umlOuterBorder{#1}}
```

`innerBorder-` The space left between the edge of the box and its contents. Default is `\umlhsep`.

```
\define@key[psset]{uml}{innerBorder}[\umlhsep]{\def\umlInnerBorder{#1}}
```

33.4 Size

`\umlBoxNullSize` The minimum size of the box (or rather, the space left for the contents). Different boxes (i.e., `\umlStretchBox` and `\umlDiagram` use different algorithms for sizeing the box.

```
\def\umlBoxNullSize{%
  \def\umlSizeX{5mm}%
  \def\umlSizeY{7mm}%
}
```

`size-` Minimum values, used mostly by `\umlDiagram`. Legal values are lengths.

```
\define@key[psset]{uml}{sizeX}{\def\umlSizeX{#1}}
\define@key[psset]{uml}{sizeY}{\def\umlSizeY{#1}}
```

33.5 Holding together all the named options

`\umlBoxNull` Just to invoke the other macros.

```
\def\umlBoxNull{%
  \umlBoxNullPositions%
  \umlBoxNullBoxes%
  \umlBoxNullVisual%
  \umlBoxNullSize%
  }%
```

33.6 The command

`\umlBox` A box is a rectangle with position and size. `\umlBox` takes two arguments: The named options and the contents.

```
\newcommand\umlBox[2][]{\leavevmode%
```

The variables are set to default

```
  \umlBoxNull%
```

and `\umlElement` is invoked, with new contents.

```
  \umlElement[kind=Box,#1]{%
```

Determines how large hbox LaTeX should think this is

```
    \umlBoxH{% \hbox to 0cm or nothing
      \umlBoxV{% \smash or nothing
```

rputs the stuff the right place

```
        \umlPosCommand{%
          \umlPosDeltaCommand{%
```

the box is a node

```
          \rnode{\umlReference}{%
```

with outer margin

```
              \setlength{\fboxrule}{0mm}%
              \setlength{\fboxsep}{\umlOuterBorder}%
              \fbox{%
```

border

```
%                 \setlength{\fboxrule}{\umlBorder}%
%                 \setlength{\fboxsep}{0mm}%
%                 \fbox{%
```

and some color and inner margin

```
                \psframebox[framesep=\umlInnerBorder,
                    linewidth=\umlBorder,
                    fillcolor=\umlFillcolorCommand, fillstyle=solid]{%
```

around the contents.

```
                    #2}%
%                  }%
                }%
              }%
            }%
          }%
        }%
      }%
    }%
  }
```

34 Diagram

A diagram determines its size before typesetting the contents. This in order to be able to place the contents relative to the borders of the diagram.

`\umlDiagramNull` The macro `\umlDiagramNull` sets the variables to their default variables.

```
\newcommand\umlDiagramNull{%
  \def\umlGrid{}%
  }
```

`grid-`

```
\define@key[psset]{uml}{grid}[1]{\message{named option grid is deprecated and of no use}}
```

34.1 The command

`\umlDiagram`

```
\newcommand\umlDiagram[2][]{%
```

First, the variables are set to their default values.

```
  \umlDiagramNull%
```

For some reason, `\umlDiagram` without the `box=` named option gives an error. I do not understand why, but it appears to be an illegal paragraph break at `\umlBoxV` in `\umlBox`.

```
  \umlBox[kind=Diagram, fillcolorCommand=umlDiagramFillcolor,
          box=,#1]{%
```

#2 is the contents. The rules are to make sure the diagram is big enough. The rules are normally invisible, because `\umlDebugLength` normally is 0 pt.
However, the rules must be evaluated before the contents, so possible `\umlBox`es does not inflict on `\umlSizeX` and `\umlSizeY`. Thus, the `\expandafter`, which assert that `#2` is typeset before, but expanded after, the rules.

```
      \expandafter{#2}{%
        \rule{\umlDebugLength}{\umlSizeY}%
        \rule{\umlSizeX}{\umlDebugLength}}%
    }%
  }%
```

35 Stretchbox

`\umlStretchbox` is the first command to take three arguments: The optional named options, a name and some contents.

`\umlStretchboxNull`

```
\newcommand\umlStretchboxNull{%
  \def\umlGraphicalName{StrechboxDefault{\umlAbstract\umlName}\\}%
}
```

`\umlGraphicalName` `\umlGraphicalName` is a name, possibly with some graphical fuzz, to be printed in the box.

`\umlStretchbox` The macro itself just handles the name. The reference is by default set to the name.
A stretchbox first typesets its contents, and then take size depending on the contents. This as opposed to a diagram, which determine the size first.

```
\newcommand\umlStretchbox[3][]{%
  \umlStretchboxNull%
  \umlBox[kind=Stretchbox, name={#2}, ref={#2}, #1]{%
    \umlIndex{\umlName}{\umlKind}%
    \rnode{\umlReference}{#3}%
    }%
  }%
```

36 Package

`\umlPackage`

```
\newcommand\umlPackage[3][]{%
  % Null unneccessary
  \def\umlName{#1}%
  \begin{tabular}{@{}l@{}}%
```

The upper rectangle. It may seem natural to make this a `\umlStretchbox`, but then we risk variable interference between the two instances of `\umlStretchbox`. It does not work in practice either.

```
    \umlStretchbox[kind=Package, border=0.4pt,#1]{#2}{%
      \def\umlGraphicalName{%
        {\umlhspace\hfill\relsize{2}\textbf{\umlName}%
          \rule{0mm}{2.3ex}\hfill\umlhspace}\\%
        }%
      \rnode{small\umlReference}{%
        \begin{tabular}{@{}l@{}}%
          \umlStereotype%
          \umlGraphicalName%
          \umlImportedFrom%
          \umlComment%
          \umlSubof%
        \end{tabular}}}%
```

There is no need for double width border between the parts of the package, so we step back a bit.

```
    \cr\noalign{\vskip -\umlBorder}%
```

The lower rectangle

```
    \umlStretchbox[kind=Package,border=0.4pt,#1]{#2}{%
      \rnode{big\umlReference}{%
        \begin{tabular}{@{}l@{}}%
```

Start the boxes in the lower rectangle

Insert the contents

```
          #3%
          \cr%
```

Here (in the `\cr`?), there is some vertical space. I still have some work to do to understand the vertical lengths in tabular.

```
        \end{tabular}}}% End of lower rectangle
  \end{tabular}%
  }%
```

37 Classifier

A classifier is a stretchbox which contains compartments. A classifier can be instanciated, or be an instance.

`\umlClassifierNull`

```
\newcommand\umlClassifierNull{%
  \def\umlObject{}%
  \umlClassifierEmptytrue%
}
\newif\ifumlClassifierEmpty%
```

`object-` This makes the classifier be an instance (object). Graphically, this is shown by an line under the classifier name. Note that instances cannot be abstract.

```
\define@key[psset]{uml}{object}{\def\umlObject{\underbar}}
```

`suppress-` Makes `\umlSuppressionOn` active in the classifier (empty compartments are not shown). For implementation: see on page 50.

`instance-` `instance` is provided as an equivalent to `object`.

```
\define@key[psset]{uml}{instance}{\def\umlObject{\underbar}}
```

`\umlClassifier` `\umlClassifier` is implemented as a table inside a `\umlStretchbox`. The contents must be able to be inside a table. The contents is typically `\umlCompartment`s.

```
\newcommand\umlClassifier[3][]{%
```

Variables set to default values

```
  \umlClassifierNull%
```

Names fixed. Uses `\umlAbstract`.

```
  \def\umlName{#2}%
```

Grayness and border are given default values, possible to override.

```
  \umlStretchbox[kind=Classifier,border=.4pt,
    fillcolorCommand=umlClassifierFillcolor,#1]{#2}{%
```

`\umlGraphicalName` must be defined here not to be overridden by `\umlStretchboxNull`. Note the invisible rule to make some space.

```
      \def\umlGraphicalName{%
        \rule{0mm}{2.8ex}%
        \umlhspace\larger\hfill\textbf{%
          \umlAbstract{\umlObject{\umlName}}}\hfill\umlhspace\\}%
      \begin{tabular}[tl]{@{}l@{}}%
        \umlStereotype%
        \umlGraphicalName%
        \umlImportedFrom%
        \umlComment%
        \umlSubof%
        #3%
%        \ifhmode hmode\else not\fi
%        \\\noalign{\vskip -15pt}%
        \ifumlClassifierEmpty\\\noalign{\vskip -2.5ex}\fi
      \end{tabular}%
    }%
  }%
```

38 Class

`\umlClass` A class is a classifier with the three usual compartments.

There is not much work left for `\umlClass`:

```
\newcommand\umlClass[4][]{%
```

A `\umlClassNull` is unneccessary, as there is no variables here.

```
  \umlClassifier[kind=Class,#1]{#2}{%
    \umlCompartment[name=attributes]{#3}%
    \umlCompartment[name=operations]{#4}%
    }%
  }%
```

39 Schema

`\umlSchema` A schema is a classifier with some more compartments.

```
\newcommand\umlSchema[7][]{%
```

`Null` unneccessary here too.

```
 \umlClassifier[kind=Schema,#1]{#2}{%
   \umlCompartment[name=attributes]{#3}% Attributes
   \umlCompartment[name=operations]{#4}% Methods
   \umlCompartment[name=arguments]{#5}% Arguments
   \umlCompartment[name=constraints]{#6}% Constraints
   \umlCompartment[name=structure]{#7}% Structure
   }%
 }%
```

40 Compartment

Classifiers (e.g., classes) are drawn as compartments (boxes) below each other.
`\umlCompartment` takes two arguments: a optional list of named options, and the contents.

`\umlCompartmentNull`

```
\newcommand\umlCompartmentNull{%
}
```

`\ifisnull` A handy command stolen from [AdvancedTeX, p.126]. If first argument is empty, execute the second; else execute the third.

```
\def\ifisnull#1#2#3{\def\inner{#1}%
  \ifx\inner\empty%
  #2\else{}#3%
  \fi}
```

`last-`

```
\define@key[psset]{uml}{last}[]{%
  \message{The named option last= is deprecated and of no use.}%
}
```

40.1 Suppression

`ifumlCompartmentSuppress` The boolean variable `umlCompartmentSuppress` affects whether empty compartments should be suppressed or not.
You can set the variable (saying `\umlCompartmentSuppresstrue` or `\umlCompartmentSuppressfalse`) whenever you like.

```
\newif\ifumlCompartmentSuppress
```

`suppress-` You can also set it for a construct (e.g., one compartment or an entire classifier) with the named option suppress. When used, it is default true.

```
\define@key[psset]{uml}{suppress}[true]{%
  \def\arg{#1}%
  \ifx\arg\umlTrueVariable\umlCompartmentSuppresstrue%
  \else\umlCompartmentSuppressfalse%
  \fi%
}
```

40.2 Compartment names

`fumlCompartmentNamesShow` The boolean variable `umlCompartmentNamesShow` affects whether compartment names should be shown or not.

Compartment names are shown top centered in a distinct font in the compartment.

You can set the variable (saying `\umlCompartmentNamesShowtrue` or `\umlCompartmentNamesShowfalse`) when you like.

```
\newif\ifumlCompartmentNamesShow
```

`showname-` You may also use the named option `showname` for one compartment or another construct.

```
\define@key[psset]{uml}{showname}[true]{%
  \def\arg{#1}%
  \ifx\arg\umlTrueVariable\umlCompartmentNamesShowtrue%
  \else\umlCompartmentNamesShowfalse%
}
```

40.3 The implementation

The implementation of `\umlCompartment` caused me some trouble. This is partly due to the many different possibilities, partly due to the funny scope rules in LaTeX tables (in `\halign`).

In tables, it seems like every line is a scope. A variable modified with `\def` in one line gets its old value when passing a line break.

Also, we cannot place the line break inside the body of an if sentence.

This is very, very odd, took me some time to detect, and destroys much beauty.

In short: TeX is a *scanalous* bad programming language, but you can make absolutely everyting in it (including object oriented programs :-)

A compartment composes classifiers, and is itself composed of compartment lines. Every compartment line ends with a line break. Every compartment starts with a `\hline` and ends with a line break.

`\umlCompartmentCommon`

```
\newcommand\umlCompartmentCommon[2][]{%
```

Even if every compartment should be preceded by a line break, we assert this is really true.

Of couse, the following line is an ad hoc hack, but I have no better solution right now.

```
  \ifhmode \vspace*{-2.5ex}\\\fi%
```

The actual line between this and the following compartment.

```
  \hline%
```

The compartment name (if it should be shown). I miss an and operator in TeX.

```
  \ifumlCompartmentNamesShow%
    \ifx\umlName\umlNameDefault\else%
      \omit\hfil\textbf{\umlName}\hfil\\%
    \fi%
  \fi%
```

This is really not neccesary, as it is defined in `\umlCompartment`.

```
  \def\umlCompartmentContents{#2}%
```

If the compartment is empty (but not suppressed), It looks better to make it shorter. (But why isn't this like `\hline\hline` in the first place?

```
  \ifx\umlCompartmentContents\empty%
    \vspace*{-1.5ex}%
  \else% There is contents
    \umlClassifierEmptyfalse%
    #2%
  \fi%
```

Assuring we end with a line break.

```
  \ifhmode\\\fi%
  }
```

`\umlCompartment` `\umlCompartment` itself mainly calls `\umlCompartmentCommon`.

```
\newcommand\umlCompartment[2][]{%
  \umlCompartmentNull%
  \def\umlCompartmentContents{#2}%
  \umlDrawable[kind=Compartment,#1]{%
    \umlColorsAdjust%
    \ifumlCompartmentSuppress%
      \ifx\umlCompartmentContents\empty\else%
        \umlCompartmentCommon[#1]{#2}%
      \fi%
    \else%
      \umlClassifierEmptyfalse%
      \umlCompartmentCommon[#1]{#2}%
    \fi}}%
```

41 Compartmentline

A compartmentline is a line of text designed to be in a compartment. Such a line should have some room before and after it, in order not to touch the compartment border.

`\umlhspace` This make some horizontal space.

```
\def\umlhsep{.5ex}
\newcommand\umlhspace{\hspace*{\umlhsep}}
```

`\umlCompartmentline` This should be straight-forward…

```
\newcommand\umlCompartmentline[2][]{%
  \umlDrawable[kind=Compartmentline,#1]{%
    {\umlColorsAdjust\umlhspace{}#2{}\umlhspace}\\}}
```

`\umlCompartmentText` Provided for backward compatibility.

```
\newcommand\umlCompartmentText[1]{%
  \umlhspace#1\umlhspace}
```

42 Feature

A feature is something attached to a classifier.

`\umlVisibilityLength` This is the hspace in front of the attribute name, where the visibility is placed.

```
\def\umlVisibilityLength{2ex}
```

`\umlFeatureNull`

```
\newcommand\umlFeatureNull{%
  \def\umlVisibility{}%
  \def\umlType{}%
  \def\umlPropertyString{}%
  \def\umlInitialValue{}%
  \def\umlName{FeatureNameDefault}%
}
```

`visibility-`

```
\define@key[psset]{uml}{visibility}[+]{%
  \def\umlVisibility{\hbox to \umlVisibilityLength{#1\hfil}}}
```

`\umlTilde` Prints a tilde.

```
\newcommand\umlTilde{\ensuremath{\sim}}
```

`propertyString-`

```
\define@key[psset]{uml}{propertyString}{%
  \def\umlPropertyString{#1}}
```

`type-` The data type returned from the feature.

```
\define@key[psset]{uml}{type}{%
  \def\umlType{: #1}}
```

`initialValue-`

```
\define@key[psset]{uml}{initialValue}{%
  \def\umlInitialValue{= #1}}
```

`\umlFeature` `\umlFeature` is implemented as a table inside a `\umlStretchbox`. The contents must be able to be inside a table. The contents is typically `\umlCompartment`s.

```
\newcommand\umlFeature[2][]{%
  \umlFeatureNull%
  \umlCompartmentline[kind=Feature, #1]{%
    \umlIndex{\umlName}{\umlKind}%
    #2}%
  }%
```

42.1 Attribute

`\umlAttributeNull`

```
\def\umlAttributeNull{%
  \def\umlMultiplicity{}%
  \def\umlOrdering{}%
  \def\umlMultiplicityOrdering{}%
}
```

`default-` This is provided as an alias to `initialValue` for the sake of backward compatibility. Use is deprecated.

```
\define@key[psset]{uml}{default}{%
  \def\umlInitialValue{ = #1}}
```

`multiplicity-` Examples of legal values are `{[1]}`, `{[1..*]}` and `{[1..3,5..]}`.

```
\define@key[psset]{uml}{multiplicity}{%
  \def\umlMultiplicity{#1}%
  \def\umlMultiplicityOrdering{[\umlMultiplicity{} \umlOrdering]}}
```

`ordering-` Legal values are `ordered` and `unordered`. Absent value imply unordered. Default value with named option is ordered.

```
\define@key[psset]{uml}{ordering}[ordered]{%
  \def\umlOrdering{#1}%
  \def\umlMultiplicityOrdering{[\umlMultiplicity{} \umlOrdering]}}
```

`\umlAttribute`

```
\newcommand\umlAttribute[2][]{%
  \umlAttributeNull%
  \umlFeature[kind=Attribute, name={#2}, #1]{%
    \umlVisibility #2 \umlType \umlMultiplicityOrdering
    \umlInitialValue \umlPropertyString}}
```

42.2 Method

`\umlMethodNull`

```
\newcommand\umlMethodNull{%
}
```

`returntype-` Alias to `type=`.

```
\define@key[psset]{uml}{returntype}{%
  \def\umlType{: #1}}
```

`\umlMethod`

```
\newcommand\umlMethod[3][]{%
  \umlMethodNull%
  \def\umlName{#2}%
  \umlFeature[kind=Method, name={#2}, #1]{%
    \umlVisibility #2(#3) \umlType \umlPropertyString}}
```

42.3 Argument

`\umlArgumentNull`

```
\newcommand\umlArgumentNull{%
}
```

`\umlArgument`

```
\newcommand\umlArgument[2][]{%
  \umlArgumentNull%
  \def\umlName{#2}%
  \umlFeature[kind=Argument, name={#2}, #1]{%
    \emph{#2} \umlType \umlInitialValue}}
```

43 Relation

43.1 Node connection points

mlRelationNullConnection

```
\newcommand\umlRelationNullConnection{%
  \def\umlNodesepA{0pt}%
  \def\umlNodesepB{0pt}%
  \def\umlOffsetA{0pt}%
  \def\umlOffsetB{0pt}%
  \def\umlAngleA{}%
  \def\umlAngleB{}%
}%
```

`angle-` This is the angle at which the connector hits the node

```
\define@key[psset]{uml}{angleA}{\def\umlAngleA{#1}}
\define@key[psset]{uml}{angleB}{\def\umlAngleB{#1}}
\define@key[psset]{uml}{angle}{\def\umlAngleA{#1}\def\umlAngleB{#1}}
```

`nodesep-` The distance from the node to the connector end. Legal values are lengths.

```
\define@key[psset]{uml}{nodesepA}{\def\umlNodesepA{#1}}
\define@key[psset]{uml}{nodesepB}{\def\umlNodesepB{#1}}
\define@key[psset]{uml}{nodesep}{\def\umlNodesepA{#1}\def\umlNodesepB{#1}}
```

`offset-` After the connection point is calculated, it is shift right (assumed direction onto node) by this value. Legal values are lengths.

```
\define@key[psset]{uml}{offsetA}{\def\umlOffsetA{#1}}
\define@key[psset]{uml}{offsetB}{\def\umlOffsetB{#1}}
\define@key[psset]{uml}{offset}{\def\umlOffsetA{#1}\def\umlOffsetB{#1}}
```

43.2 Arm geometry

lRelationNullArmGeometry

```
\newcommand\umlRelationNullArmGeometry{%
  \pssetlength\umlArmA{0pt}%
  \pssetlength\umlArmB{0pt}%
  \def\umlArmAngleA{0}%
  \def\umlArmAngleB{0}%
}
```

`armA-` This is the lengths of the arms.

```
\newlength\umlArmA
\newlength\umlArmB
\define@key[psset]{uml}{armA}{\pssetlength\umlArmA{#1}}
\define@key[psset]{uml}{armB}{\pssetlength\umlArmB{#1}}
\define@key[psset]{uml}{arm}{%
  \pssetlength\umlArmA{#1}%
  \pssetlength\umlArmB{#1}}
```

armAngle= This is the angle of the arm.

```
\define@key[psset]{uml}{armAngleA}{\def\umlArmAngleA{#1}}
\define@key[psset]{uml}{armAngleB}{\def\umlArmAngleB{#1}}
\define@key[psset]{uml}{armAngle}{%
  \def\umlArmAngleA{#1}%
  \def\umlArmAngleB{#1}%
}
```

43.3 Visual appeareance

\umlRelationNullVisual

```
\newcommand\umlRelationNullVisual{%
  \def\umlLinestyle{solid}%
}
```

umllinestyle= Legal values are `none`, `solid`, `hashed` and `dotted`.

```
\define@key[psset]{uml}{umllinestyle}{\def\umlLinestyle{#1}}
```

relationColor= The color of the line.

```
\define@key[psset]{uml}{relationColor}[black]{\pst@getcolor{#1}\pslinecolor}
```

lineColor= Alias for `relationColor=`.

```
\define@key[psset]{uml}{linecolor}{\pst@getcolor{#1}\pslinecolor}
```

\umlRelationNull

```
\newcommand\umlRelationNull{%
  \umlRelationNullConnection%
  \umlRelationNullArmGeometry%
  \umlRelationNullVisual%
}
```

43.4 The macro

\umlRelation The command itself:

```
\newcommand\umlRelation[4][]{%
  \umlRelationNull%
```

The default reference is the concatenation of the two references

```
  \umlElement[kind=Relation,ref={#2#3}, #1]{%
```

Putting the `Aa` and `Ba` nodes, default (without `angle=`) first

```
  \ncline[linecolor=green, linestyle=\umlDebugLinestyle,
          offsetA=\umlOffsetA, nodesepA=\umlNodesepA,
          offsetB=\umlOffsetB, nodesepB=\umlNodesepB]{#2}{#3}%
  \lput{:R}(0){\pnode{Aa\umlReference}}%
  \lput{:R}(1){\pnode{Ba\umlReference}}%
```

Then modifying `Aa`, if `angleA=` or `angle=` is used

```
  \ifx\umlAngleA\empty \else
     \ncdiag[linestyle=\umlDebugLinestyle, linecolor=magenta, %
             angleA=\umlAngleA,
             offsetA=\umlOffsetA, nodesepA=\umlNodesepA,
             offsetB=\umlOffsetB, nodesepB=\umlNodesepB
            ]{#2}{#2}%
     \lput{:R}(0){\pnode{Aa\umlReference}}\fi%
```

And `Ba`.

```
  \ifx\umlAngleB\empty \else
     \ncdiag[linestyle=\umlDebugLinestyle, linecolor=magenta, %
             angleA=\umlAngleB,
             offsetA=\umlOffsetA, nodesepA=\umlNodesepA,
             offsetB=\umlOffsetB, nodesepB=\umlNodesepB
            ]{#3}{#3}%
     \lput{:R}(0){\pnode{Ba\umlReference}}\fi%
```

Now, we can draw the line, from the `Aa` to the `Ba` node.

```
    \ncdiag[linestyle=\umlLinestyle,  linecolor=umlLinecolor,
            angleA=\umlArmAngleA, angleB=\umlArmAngleB,
            armA=\umlArmA, armB=\umlArmB
            ]{%
              Aa\umlReference}{%
              Ba\umlReference}%
```

Placing nodes `Ab` and `Bb`. If there is no arm A,

```
  \ifdim \umlArmA=0pt \lput{:R}(2){\pnode{Ab\umlReference}}%
```

Else, if there is

```
  \else \lput{:R}(1){\pnode{Ab\umlReference}} \fi%
```

If there is no arm B,

```
  \ifdim \umlArmB=0pt \lput{:R}(1){\pnode{Bb\umlReference}}%
```

Else, if there is

```
  \else \lput{:R}(2){\pnode{Bb\umlReference}} \fi%
```

Final nodes

```
   \lput{:R}(1){\pnode{Ac\umlReference}}%
   \lput{:R}(2){\pnode{Bc\umlReference}}%
```

Other contents

```
   #4}% of \umlElement
}
```

43.5 About the spesific relations

The different relations are specialized relations; they typically call `\umlRelation` and `\umlSymbol` with an appropriate symbol.

`\umlSymbolHeightDefault` All the symbols are drawn with `0` as the upper border, `-\umlSymbolHeightDefault` as the lower, `-\umlSymbolWidthDefault` as the left and `\umlSymbolWidthDefault` as the right one. These lengths can be changed by the user. See, however, section 43.8 on the next page.

```
\newlength\umlSymbolHeightDefault
\setlength\umlSymbolHeightDefault{1em}
\newlength\umlSymbolWidthDefault
\setlength\umlSymbolWidthDefault{.5em}
```

43.6 Association

`\umlAssociation` `\umlAssociation` is a relation without any other contents.

```
\newcommand\umlAssociation[3][]{%
  \umlRelation[kind=Association, #1]{#2}{#3}{}%
}
```

43.7 Subclass (generalization)

`\umlSubclass` A simple relation with a triangle as an endsymbol.

```
\newcommand\umlSubclass[3][]{%
  \def\umlEndSymbol{%
     \pspolygon*[linecolor=white](0,0)%
       (-\umlSymbolWidthDefault,-\umlSymbolHeightDefault)%
       (\umlSymbolWidthDefault,-\umlSymbolHeightDefault)%
     \pspolygon[](0,0)% Why does not dimen=inner work?
       (-\umlSymbolWidthDefault,-\umlSymbolHeightDefault)%
       (\umlSymbolWidthDefault,-\umlSymbolHeightDefault)%
      }%
%   \def\umlEndSymbol{% Alternative \umlEndSymbol
%     \pstriangle[dimen=inner](0,-\umlSymbolHeightDefault)%
%     (\umlSymbolWidthDefault,\umlSymbolHeightDefault)}
  \umlRelation[kind=Subclass, #1]{#2}{#3}{%
    \umlSymbol[fraction=B]{\umlReference}{\umlEndSymbol}}%
  }
\def\umlGeneralization{\umlSubclass}
```

43.8 Inner class

The making of this symbol gave some problems. After experimenting with different interpolations and Bézier curves, I defined it to consist of two clipped wedges. Each should of course have width $w =$ `\umlWidthDefault` and height $h =$`\umlHeightDefault`. The radius of the circle is r, and $r = v + w$. This gives figure 14.

This figure is drawn with an angle of 60 degrees.

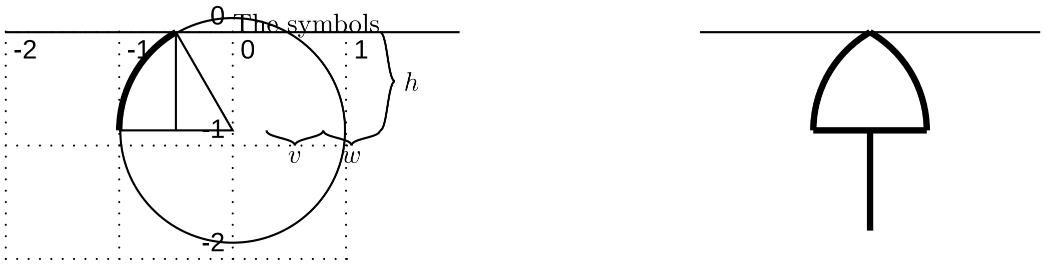


Figure 14: The inner class symbol

We have

$$r = v + w$$

and, from Pythagoras,

$$r^2 = h^2 + w^2$$

This gives

$$r = \frac{h^2}{2v} + \frac{v}{2}$$

and

$$w = \frac{h^2}{2v} - \frac{v}{2}$$

and we know where to locate the wedge.

However, while addition and subtraction of lengths is easy in PSTricks (only stepping back and forth), multiplication and division of lengths is difficult, if not impossible. I really haven't found a good solution to this problem.

The not-so-good problem is to define w and r as TEX lengths (`\umlInnerWidthDefault` and `\umlInnerRadiusDefault`) and then assign them values manually. We have to remember, then, that Pythagoras still should work.

Page 57 assign the default values $h = 1$ em, $v = .5$ em. This gives

`\umlInnerWidthDefault` $w = 0.75$ em

```
\newlength\umlInnerWidthDefault
\setlength\umlInnerWidthDefault{0.75em}
```

`\umlInnerRadiusDefault` and $r = 1.25$ em.

```
\newlength\umlInnerRadiusDefault
\setlength\umlInnerRadiusDefault{1.25em}
```

If we should implement the symbol by `\umlArc`, we had to know some angles. They would be easy to compute using trigonometry, but that is difficult within TEX. Then, we use `\umlCircle` and `\psclip` instead.

Maybe this could be done easily using raw postscript?

`\umlInner` On some systems, the clipping makes some borders.

```
\newcommand\umlInner[3][]{%
  \def\umlEndSymbol{%
     \pspolygon*[linecolor=white](0,0)%
       (-\umlSymbolWidthDefault,-\umlSymbolHeightDefault)%
       (\umlSymbolWidthDefault,-\umlSymbolHeightDefault)%
    \psclip{%
      \psframe[linewidth=0pt]%
        (-\umlSymbolWidthDefault, -\umlSymbolHeightDefault)(0,0)}
      \pscircle(\umlInnerWidthDefault,-\umlSymbolHeightDefault)%
        {\umlInnerRadiusDefault}
    \endpsclip
    \psclip{%
      \psframe[linewidth=0pt]%
        (\umlSymbolWidthDefault, -\umlSymbolHeightDefault)(0,0)}
      \pscircle(-\umlInnerWidthDefault,-\umlSymbolHeightDefault)%
        {\umlInnerRadiusDefault}
    \endpsclip
    \psline(-\umlSymbolWidthDefault, -\umlSymbolHeightDefault)%
      (\umlSymbolWidthDefault, -\umlSymbolHeightDefault)}
  \umlRelation[kind=Inner, #1]{#2}{#3}{%
    \umlSymbol[fraction=B]{\umlReference}{\umlEndSymbol}
  }}
```

43.9 Instance

`\umlInstance` The only new thing about `\umlInstance` is the addition of one named option, the linestyle.

```
\newcommand\umlInstance[3][]{%
  \def\umlEndSymbol{%
    \psline(-\umlSymbolWidthDefault, -\umlSymbolHeightDefault)%
      (0,0)%
      (\umlSymbolWidthDefault, -\umlSymbolHeightDefault)}%
  \umlRelation[kind={Instance-of}, umllinestyle=dashed, #1]{#2}{#3}{%
    \umlSymbol[fraction=B]{\umlReference}{\umlEndSymbol}}
  }
```

43.10 Aggregation

`\umlAggregation` Endpoint is a diamond.

```
\newcommand\umlAggregation[3][]{%
  \def\umlEndSymbol{%
```

Anyone said addition of lengths in PSTricks was difficult?

```
    \rput(0,-\umlSymbolHeightDefault){%
      \psline*[linecolor=white](-\umlSymbolWidthDefault, 0)%
        (0,-\umlSymbolHeightDefault)%
        (\umlSymbolWidthDefault, 0)}
    \psline*[linecolor=white]%
      (-\umlSymbolWidthDefault, -\umlSymbolHeightDefault)%
      (0,0)%
      (\umlSymbolWidthDefault, -\umlSymbolHeightDefault)
    \rput(0,-\umlSymbolHeightDefault){%
      \psline(-\umlSymbolWidthDefault, 0)%
        (0,-\umlSymbolHeightDefault)%
        (\umlSymbolWidthDefault, 0)}
    \psline(-\umlSymbolWidthDefault, -\umlSymbolHeightDefault)%
      (0,0)%
      (\umlSymbolWidthDefault,
      -\umlSymbolHeightDefault)}
  \umlRelation[kind=Aggregation, #1]{#2}{#3}{%
    \umlSymbol[fraction=B]{\umlReference}{\umlEndSymbol}}
}
```

43.11 Composition

`\umlComposition` End symbol is a filled diamond.

```
\newcommand\umlComposition[3][]{%
  \def\umlEndSymbol{%
    \rput(0,-\umlSymbolHeightDefault){%
      \psline*(-\umlSymbolWidthDefault, 0)%
```

```
        (0,-\umlSymbolHeightDefault)%
        (\umlSymbolWidthDefault, 0)}
    \psline*(-\umlSymbolWidthDefault, -\umlSymbolHeightDefault)%
      (0,0)%
      (\umlSymbolWidthDefault,
      -\umlSymbolHeightDefault)}
  \umlRelation[kind=Composition, #1]{#2}{#3}{%
    \umlSymbol[fraction=B]{\umlReference}{\umlEndSymbol}}
}
```

43.12 Application

`\umlApplication` End symbol is a filled arrow.

```
\newcommand\umlApplication[3][]{%
  \def\umlEndSymbol{%
    \pspolygon*(-\umlSymbolWidthDefault, -\umlSymbolHeightDefault)%
      (0,0)%
      (\umlSymbolWidthDefault, -\umlSymbolHeightDefault)%
      (0,-\umlSymbolWidthDefault)}%
  \umlRelation[kind=Application, #1]{#2}{#3}{%
    \lput(1.2){\pnode{argument\umlReference}}%
    \umlSymbol[fraction=B]{\umlReference}{\umlEndSymbol}}
  }
```

43.13 ToRelation

`\umlToRelationNull`

```
\newcommand\umlToRelationNull{%
  \def\umlPosMeetLine{.5}%
}
```

`posMeetLine-` Where this relation shall meet the relation.

```
\define@key[psset]{uml}{posMeetLine}[.5]{\def\umlPosMeetLine{#1}}
```

`\umlAssociationClass` Relation from a relation to a classifier

```
\newcommand\umlToRelation[3][]{%
  \umlToRelationNull%
```

We have to process the `posMeetLine` option before the `\lput`. This introduces some overhead, as `\psset` is run twice. However, I expect this to be used relatively few times.

```
  \psset[uml]{kind=ToRelation, #1}%
  \ncline[linecolor=red, linestyle=\umlDebugLinestyle]{Ac#3}{Bc#3}%
  \lput{:R}(\umlPosMeetLine){\pnode{ToRelation#3}}%
  \umlRelation[ref={#2#3}, #1]{#2}{ToRelation#3}{}%
}
```

43.14 AssociationClass and AssociationSchema

`\umlAssociationClass` Relation from a relation to a schema symbol.

```
\newcommand\umlAssociationSchema[3][]{%
```

```
  \umlToRelation[kind=AssociationSchema,
    posMeetLine=.5, umllinestyle=dashed,#1]{#2}{#3}%
}
\newcommand\umlAssociationClass[3][]{%
  \umlAssociationSchema[kind=AssociationClass,#1]{#2}{#3}}
```

43.15 ArgumentRelation

`\umlArgumentRelation` Relation from an application to an argument.

```
\newcommand\umlArgumentRelation[3][]{%
  \umlToRelation[kind=ArgumentRelation,
    posMeetLine=.2,umllinestyle=dotted,#1]{#2}{#3}%
  }
```

44 AssociationEnd

`\umlAssociationEndNull`

```
\newcommand\umlAssociationEndNull{%
  \def\umlAEOffset{\umlAEOffsetDefault}%
  \def\umlAEOffsetDefault{0pt}%
  \def\umlAEFraction{0}%
  \def\umlAEFractionAngle{\umlAEFractionAngleDefault}%
  \def\umlAEFractionAngleDefault{:U}%
  \def\umlAEAngle{\umlAEAngleDefault}%
  \def\umlAEAngleDefault{U}%
  \def\umlAERefpoint{B}%
  \def\umlAEHeight{\umlAEHeightDefault}%
  \def\umlAEHeightDefault{0pt}%
  \def\umlAENoderefClose{Ac}%
  \def\umlAENoderefFar{Bc}%
  \def\umlAEType{AssociationEnd}%
  \def\umlAEKind{AssociationEnd}
}
```

`import-`

```
\define@key[psset]{umlAE}{import}{\def\umlAEColor{red}}
```

`type-`

```
\define@key[psset]{umlAE}{type}{\def\umlType{#1}}
```

`kind-` E.g., AssociationEnd.

```
\define@key[psset]{umlAE}{kind}{\def\umlKind{#1}}
```

`offset-`

```
\define@key[psset]{umlAE}{offset}{\def\umlAEOffset{#1}}
```

`angle-` Angle used to rotate the symbol itself.

```
\define@key[psset]{umlAE}{angle}{\def\umlAEAngle{#1}}
```

fractionAngle- The angle used when positioning the symbol. Legal values includes angles preceded by a colon (:), indicting positioning relative to the relation. This is expected to be used by subcommands, but seldom by users.

```
\define@key[psset]{umlAE}{fractionAngle}{\def\umlAEFractionAngle{#1}}
```

height-

```
\define@key[psset]{umlAE}{height}{\def\umlAEHeight{#1}}
```

refpoint-

```
\define@key[psset]{umlAE}{refpoint}{\def\umlAERefpoint{#1}}
```

refpoint-

```
%% \define@key[psset]{umlAE}{type}{\def\umlAEType{#1}}%    %%%%%%%%%%%%
```

44.1 Fraction

fraction- This value is used by `\umlAEFixFractionLabel` and `\umlAEFixFractionSymbol`.

```
\define@key[psset]{umlAE}{fraction}{\def\umlAEFraction{#1}}
```

\umlFromTo A handy little procedure. If its first argument is `A` or `From`, it executes the second argument. If it is `B` or `To`, it executes the third. Used in the procedures like `\umlAssociationEndMakeA`, `\umlLabelMakeA` and `\umlSymbolMakeA`.

```
\newcommand\umlFromTo[3]{%
  \edef\umlAEFractionArgument{#1}%
  \def\umlAEFractionTmp{From}\ifx\umlAEFractionArgument\umlAEFractionTmp{}#2\fi%
  \def\umlAEFractionTmp{A}\ifx\umlAEFractionArgument\umlAEFractionTmp{}#2\fi%
  \def\umlAEFractionTmp{To}\ifx\umlAEFractionArgument\umlAEFractionTmp{}#3\fi%
  \def\umlAEFractionTmp{B}\ifx\umlAEFractionArgument\umlAEFractionTmp{}#3\fi%
}
%
```

ssociationEndUseFraction If `\umlFraction` is `A` or something (i.e., if `fraction=A` or sth), adjust some other parameters.

```
\newcommand\umlAssociationEndUseFraction{%
%    \begin{macrocode}
  \umlFromTo{\umlAEFraction}{% If A or From
    \def\umlAENoderefClose{Aa}%
    \def\umlAENoderefFar{Ab}%
    \def\umlAEFraction{0}%
    }{%
    \def\umlAENoderefClose{Ba}% If B or To
    \def\umlAENoderefFar{Bb}%
    \def\umlAEFraction{0}%
```

If this is a “`B`” type association end, and this is an Label type association end, invert the height.

```
    \def\umlTmp{Label}%
    \ifx\umlTmp\umlAEType%
      \edef\umlAEHeight{-\umlAEHeight}\fi%
    }%
  }
```

44.2 The command

`\umlAssociationEnd` This places a symbol (third argument) on the ⟨*From*⟩ end of the relation (indicated by the second argument).

```
\newcommand\umlAssociationEnd[3][]{%
  \umlAssociationEndNull%
  \psset[umlAE]{kind=AssociationEnd,#1}%
  \umlAssociationEndUseFraction%
%  AE:#2:\umlAENoderefClose:\umlAENoderefFar:
  \ncline[linecolor=red, linestyle=\umlDebugLinestyle]{%
    \umlAENoderefClose#2}{\umlAENoderefFar#2}%
  {\umlColorsAdjust%
    \lput[\umlAERefpoint]{\umlAEFractionAngle}(\umlAEFraction){%
      \rput[\umlAERefpoint]{\umlAEAngle}(\umlAEOffset, \umlAEHeight){%
        #3}%
      }%
    }%
  }%
```

44.3 Label

`\umlLabel` A label is a symbol with default height and offset.

```
\newcommand\umlLabel[3][]{% Null unneccesary
  \umlAssociationEnd[kind=Label,offset=4ex,height=2ex,angle=N, #1]{#2}{#3}%
}
```

`\umlLabelA` `\umlLabelA` and `\umlLabelB` are provided for convenience and backward compatibility.

```
\newcommand\umlLabelA[2][]{\umlLabel[#1,fraction=A]{#2}}
\newcommand\umlLabelB[2][]{\umlLabel[#1,fraction=B]{#2}}
```

44.4 Symbol

`\umlSymbol` A symbol is a symbol with default height and offset.

```
\newcommand\umlSymbol[3][]{%  Null unneccesary
  \umlAssociationEnd[kind=Symbol,offset=0ex,height=0ex,
                     fractionAngle=:L,refpoint=t,#1]{%
    #2}{\umlSymbolUseFraction%
    #3}}
```

`ssociationEndUseFraction`

```
\newcommand\umlSymbolUseFraction{%
  \umlFromTo{\umlAEFraction}{%
    }{%
    }%
  }
```

`\umlSymbolA` `\umlSymbolA` and `\umlSymbolB` are provided for convenience and backward compatibility.

```
\newcommand\umlSymbolA[2][]{\umlSymbol[#1,fraction=A]{#2}}
\newcommand\umlSymbolB[2][]{\umlSymbol[#1,fraction=B]{#2}}
```

44.5 Navigability

`\umlNavigability` A specialized version of `\umlAssociationEnd`. Takes two arguments: a list of named options, and the relation reference.

```
\newcommand\umlNavigability[2][]{
  \def\umlEndSymbol{\psline%
    (-1ex, -1.618ex)%
    (0,0)%
    (1ex, -1.618ex)}%
  \umlSymbol[kind=Navigability, #1]{#2}{\umlEndSymbol}%
  }
```

`\umlNavigabilityA` `\umlNavigabilityA` and `\umlNavigabilityB` are provided for convenience and backward compatibility.

```
\newcommand\umlNavigabilityA[2][]{\umlNavigability[#1,fraction=A]{#2}}
\newcommand\umlNavigabilityB[2][]{\umlNavigability[#1,fraction=B]{#2}}
```

45 Colors

45.1 Colorset

`\umlColorset` Every `\umlDrawable` (really, every instance of a subcommand) calls `\umlColorsAdjust`. Then, the colors is set anew for the Drawable. The effect then depends on the value of `\umlColorsAdjust`. This value is set by `\umlColorsDefault`, `\umlColorsImport` etc.

```
\newcommand\umlColorset[1]{%
  \def\umlColorsAdjust{#1%
  \psset{linecolor=umlLinecolor, fillcolor=umlFillcolor}}}
```

`\umlColorsDefault`

```
\newcommand\umlColorsDefault{%
  \umlColorset{%
    \definecolor{umlColor}{gray}{0}%
    \definecolor{umlLinecolor}{gray}{0}%
    \definecolor{umlFillcolor}{gray}{1}%
    \definecolor{umlClassifierFillcolor}{gray}{0.85}%
    \definecolor{umlDiagramFillcolor}{gray}{0.95}%
    \definecolor{umlRelationColor}{gray}{0}%
}}
```

`\umlColorsGray`

```
\newcommand\umlColorsGray{%
  \umlColorset{%
    \definecolor{umlColor}{gray}{0.4}%
    \definecolor{umlLinecolor}{gray}{0.4}%
    \definecolor{umlFillcolor}{gray}{1}%
    \definecolor{umlClassifierFillcolor}{gray}{0.90}%
    \definecolor{umlDiagramFillcolor}{gray}{0.98}%
    \definecolor{umlRelationColor}{gray}{0.4}%
}}
```

`\umlColorsImport` The import color set makes the boxes blue.

```
\newcommand\umlColorsImport{%
  \umlColorset{%
    \definecolor{umlColor}{rgb}{0, 0, 0.4}%
    \definecolor{umlLinecolor}{rgb}{0, 0, 0.4}%
    \definecolor{umlFillcolor}{rgb}{.8, .8, 1}%
    \definecolor{umlClassifierFillcolor}{rgb}{.85, .85, 1}%
    \definecolor{umlDiagramFillcolor}{rgb}{.95, .95, 1}%
    \definecolor{umlRelationColor}{rgb}{0, 0, 0.4}%
}}
```

`\umlColorsArgument` This color set makes the boxes green.

```
\newcommand\umlColorsArgument{%
  \umlColorset{%
    \definecolor{umlColor}{rgb}{0, 0.4, 0}%
    \definecolor{umlLinecolor}{rgb}{0, 0.4, 0}%
    \definecolor{umlFillcolor}{rgb}{.8, 1, .8}%
    \definecolor{umlClassifierFillcolor}{rgb}{.85, 1, .85}%
    \definecolor{umlDiagramFillcolor}{rgb}{.95, 1, .95}%
   \definecolor{umlRelationColor}{rgb}{0, 0.7, 0}%
}}
```

`\umlColorsRed`

```
\newcommand\umlColorsRed{%
  \umlColorset{%
    \definecolor{umlColor}{rgb}{0.4, 0, 0}%
    \definecolor{umlLinecolor}{rgb}{0.4, 0, 0}%
    \definecolor{umlFillcolor}{rgb}{1, .8, .8}%
    \definecolor{umlClassifierFillcolor}{rgb}{1, .85, .85}%
    \definecolor{umlDiagramFillcolor}{rgb}{1, .95, .95}%
    \definecolor{umlRelationColor}{rgb}{0.4, 0, 0}%
}}
```

`\umlColorsSub`

```
\newcommand\umlColorsSub{%
  \umlColorset{%
    \definecolor{umlColor}{rgb}{.6, .2, .2}%
    \definecolor{umlLinecolor}{rgb}{.6, .2, .2}%
    \definecolor{umlFillcolor}{rgb}{.9, .8, .8}%
    \definecolor{umlClassifierFillcolor}{rgb}{.9, .8, .8}%
    \definecolor{umlDiagramFillcolor}{rgb}{.97, .95, .95}%
    \definecolor{umlRelationColor}{rgb}{.6, .2, .2}%
}}
```

```
\umlColorsDefault
\umlColorsAdjust
```

45.2 Using color sets

`\umlColors`

```
\newenvironment{umlColors}[1]{\bgroup#1}{\egroup}
```

46 Positions

```
\SpecialCoor
\newlength{\umlNodeSep}
\setlength{\umlNodeSep}{1em}
```

A historical note here: First, `\umlBox` used to throw lots of pnodes around. However, this used huge memory space. This way works much better. However, I have not found any way to do the corresponding thing in the relations.

46.1 PlaceNode

```
\newlength\umlPlaceNodeX
\newlength\umlPlaceNodeY
```

`\umlPlaceNodeNull`

```
\newcommand\umlPlaceNodeNull{%
  \def\umlPlaceNodeNodesepX{0pt}%
  \def\umlPlaceNodeNodesepY{0pt}%
  \def\umlPlaceNodeAngleX{}%
  \def\umlPlaceNodeAngleY{}%
  \def\umlPlaceNodeOffsetX{}%
  \def\umlPlaceNodeOffsetY{}%
  \setlength\umlPlaceNodeX{0pt}%
  \setlength\umlPlaceNodeY{0pt}%
}
```

`leftside-`

```
\define@key[psset]{umlPlaceNode}{leftside}[0pt]{%
  \def\umlPlaceNodeAngleX{,angle=180}%
  \def\umlPlaceNodeNodesepX{#1}}%
```

`rightside-`

```
\define@key[psset]{umlPlaceNode}{rightside}[0pt]{%
  \def\umlPlaceNodeAngleX{,angle=0}%
  \def\umlPlaceNodeNodesepX{#1}}%
```

`up-`

```
\define@key[psset]{umlPlaceNode}{top}[0pt]{%
  \def\umlPlaceNodeAngleY{,angle=90}%
  \def\umlPlaceNodeNodesepY{#1}}%
```

`bottom-`

```
\define@key[psset]{umlPlaceNode}{bottom}[0pt]{%
  \def\umlPlaceNodeAngleY{,angle=270}%
  \def\umlPlaceNodeNodesepY{#1}}%
```

`left-`

```
\define@key[psset]{umlPlaceNode}{left}[0pt]{%
  \addtolength\umlPlaceNodeX{-#1}}%
```

`right-`

```
\define@key[psset]{umlPlaceNode}{right}[0pt]{\addtolength\umlPlaceNodeX{#1}}
```

up-

```
\define@key[psset]{umlPlaceNode}{up}[0pt]{\addtolength\umlPlaceNodeY{#1}}
```

down-

```
\define@key[psset]{umlPlaceNode}{down}[0pt]{\addtolength\umlPlaceNodeY{-#1}}
```

angle-

```
\define@key[psset]{umlPlaceNode}{angle}{%
  \def\umlPlaceNodeAngleX{,angle=#1}%
  \def\umlPlaceNodeAngleY{,angle=#1}}%
\define@key[psset]{umlPlaceNode}{angleX}{\def\umlPlaceNodeAngleX{,angle=#1}}%
\define@key[psset]{umlPlaceNode}{angleY}{\def\umlPlaceNodeAngleY{,angle=#1}}%
```

offset-

```
\define@key[psset]{umlPlaceNode}{offset}{
  \def\umlPlaceNodeOffsetX{,offset=#1}%
  \def\umlPlaceNodeOffsetY{,offset=#1}}%
\define@key[psset]{umlPlaceNode}{offsetX}{\def\umlPlaceNodeOffsetX{,offset=#1}}%
\define@key[psset]{umlPlaceNode}{offsetY}{\def\umlPlaceNodeOffsetY{,offset=#1}}%
```

nodesep-

```
\define@key[psset]{umlPlaceNode}{nodesep}{%
  \def\umlPlaceNodeNodesepX{#1}%
  \def\umlPlaceNodeNodesepY{#1}}%
\define@key[psset]{umlPlaceNode}{nodesepX}{\def\umlPlaceNodeNodesepX{#1}}%
\define@key[psset]{umlPlaceNode}{nodesepY}{\def\umlPlaceNodeNodesepY{#1}}%
```

\umlPlaceNode

```
\newcommand\umlPlaceNode[3][]{%
  \umlPlaceNodeNull%
  \psset[umlPlaceNode]{#1}%
```

Placement relative to the node

```
  \rput(%
    [nodesep=\umlPlaceNodeNodesepX\umlPlaceNodeOffsetX\umlPlaceNodeAngleX]#2|%
    [nodesep=\umlPlaceNodeNodesepY\umlPlaceNodeOffsetY\umlPlaceNodeAngleY]#2){%
```

Placement relative to that

```
%    \rput(\umlPlaceNodeX, \umlPlaceNodeY){%
```

The new node is placed

```
      \pnode(\umlPlaceNodeX, \umlPlaceNodeY){#3}}%
}
```

\umlRight The first coordinate commands are very simple. They takes as argument node.

```
\newcommand\umlRight[1]{[angle=0]#1}
\newcommand\umlTop[1]{[angle=90]#1}
\newcommand\umlLeft[1]{[angle=180]#1}
\newcommand\umlBottom[1]{[angle=270]#1}
\newcommand\umlTopRight[1]{[angle=0]#1|[angle=90]#1}
\newcommand\umlBottomRight[1]{[angle=0]#1|[angle=270]#1}
\newcommand\umlTopLeft[1]{[angle=180]#1|[angle=90]#1}
\newcommand\umlBottomLeft[1]{[angle=180]#1|[angle=270]#1}
```

`\umlRightSep` The `Sep` coordinate commands use `\umlNodeSep` to make some space between the nodes.

```
\newcommand\umlRightSep[1]{[angle=0, nodesep=\umlNodeSep]#1}
\newcommand\umlTopSep[1]{[angle=90, nodesep=\umlNodeSep]#1}
\newcommand\umlLeftSep[1]{[angle=180, nodesep=\umlNodeSep]#1}
\newcommand\umlBottomSep[1]{[angle=270, nodesep=\umlNodeSep]#1}
\newcommand\umlTopRightSep[1]{%
    [angle=0, nodesep=\umlNodeSep]#1|[angle=90, nodesep=\umlNodeSep]#1}
\newcommand\umlBottomRightSep[1]{%
  [angle=0, nodesep=\umlNodeSep]#1|[angle=270, nodesep=\umlNodeSep]#1}
\newcommand\umlTopLeftSep[1]{%
  [angle=180, nodesep=\umlNodeSep]#1|[angle=90, nodesep=\umlNodeSep]#1}
\newcommand\umlBottomLeftSep[1]{%
  [angle=180, nodesep=\umlNodeSep]#1|[angle=270, nodesep=\umlNodeSep]#1}
```

`\umlRightOpt` This takes two mandatory arguments: Named options and the usual node.

Of course, it would be nice to make the first argument optional, thus combining `\umlRight` and `\umlRightOpt`. However, this does not work together with mandatory argument in `\umlBox`. I have found no elegant solution to this (despite some nights…)

```
\newcommand\umlRightOpt[2]{[angle=0, #1]#2}
\newcommand\umlTopOpt[2]{[angle=90, #1]#2}
\newcommand\umlLeftOpt[2]{[angle=180, #1]#2}
\newcommand\umlBottomOpt[2]{[angle=270, #1]#2}
\newcommand\umlTopRightOpt[2]{[angle=0, #1]#2|[angle=90, #1]#2}
\newcommand\umlBottomRightOpt[2]{[angle=0, #1]#2|[angle=270, #1]#2}
\newcommand\umlTopLeftOpt[2]{[angle=180, #1]#2|[angle=90, #1]#2}
\newcommand\umlBottomLeftOpt[2]{[angle=180, #1]#2|[angle=270, #1]#2}
```

`\umlRightSep`

```
\newcommand\umlRightSepOpt[2]{[angle=0, nodesep=\umlNodeSep, #1]#2}
\newcommand\umlTopSepOpt[2]{[angle=90, nodesep=\umlNodeSep, #1]#2}
\newcommand\umlLeftSepOpt[2]{[angle=180, nodesep=\umlNodeSep, #1]#2}
\newcommand\umlBottomSepOpt[2]{[angle=270, nodesep=\umlNodeSep, #1]#2}
\newcommand\umlTopRightSepOpt[2]{[angle=0, nodesep=\umlNodeSep, #1]#2|[angle=90, nodesep=\umlNodeSep, #1]#2
\newcommand\umlBottomRightSepOpt[2]{[angle=0, nodesep=\umlNodeSep, #1]#2|[angle=270, nodesep=\umlNodeSep, #
\newcommand\umlTopLeftSepOpt[2]{%
  [angle=180, nodesep=\umlNodeSep, #1]#2|[angle=90, nodesep=\umlNodeSep, #1]#2}
\newcommand\umlBottomLeftSepOpt[2]{%
  [angle=180, nodesep=\umlNodeSep, #1]#2|[angle=270, nodesep=\umlNodeSep, #1]#2}
```