The `umldoc` UML Documentation Package

Doug Palmer*

December 14, 1999

Abstract

The `umldoc` package allows the layout of UML documentation and diagrams. With the addition of appropriate scripts, models built using UML modelling tools can be automatically documented. The current package concentrates on the logical aspects of UML: use cases, classes, associations and their associated diagrams.

1 Introduction

Most UML modelling tools, such as Rational Rose, provide plenty of space for documentation within the model being created. Unfortunately, the documentation usually has to be plain text with no room for mathematics, emphasis, citations or any more sophisticated layout. In addtion, the documentation is scattered about the model, attached to the various model parts; there is no human-accesible documentation.

The `umldoc` package allows the layout of UML documentation and class, use case, etc. diagrams. The LaTeX input for this package is generated from a UML model built in one of the UML modelling tools available for object modelling. The package allows this input to be converted into professional-looking documentation, along with diagrams, tables of contents, indexes, etc. Any LaTeX constructs within the documentation will appear in the final documentation.

This package provides a set of nested environments for documentation. In most cases, the nesting structure is obvious and an error will occur if there is an attempt to use an environment outside it's correct position in the nesting structure.

1.1 Modelling Style

There are several ways of building UML models. The methodology implicitly assumed by the `umldoc` package is that the model will be built in terms of composable packages. These packages are then merged together to form a complete model. Within a package is a major category, which contains the bulk of the new classes, operations, associations, etc. The major category is supported by

*`doug@tarms.com`

extensions to existing packages which add additional operations and associations to other packages.

All documentation for a package is grouped together into a single chapter or article, with extensions documented within that unit. It is possible to ignore the support of extensions with no ill-effects.

1.2 Documentation Generation

This package can be used in conjunction with the `latex.ebs` Rose script to allow Rational Rose model to be documented. This script takes a model, asks for the major category and then generates a documentation file and a wrapper file. The documentation file contains the model's documentation. The wrapper file provides an article-style wrapper, so that the package documentation can be produced as a self-contained unit. As an alternative, multiple documentation files can be strung together to form a documentation book.

1.3 Package Use

To use the `umldoc` package, add the following line to the preamble:

```
\usepackage[⟨options⟩]umldoc
```

The following options are available:

noindex By default, index entries for classes, operations, defined terms, etc. are created, using a standardised schema. If you do not want index entries, use the noindex option.

nosectionmarks Cross references to classes and interfaces are usually marked with a section mark (§). This option prevents section marks.

nomarginnotes By default, operation definitions create a marginal note giving the operation name. This option prevents the creation of the marginal notes.

2 Document Structure

The `umldoc` package uses a series of nested environments, described below. The basic document consists of a series of categories and category extensions (section 2.2). Within each category are various class and use case diagrams (section 2.7), class groups (section 2.3), classes (section 2.5) and use cases (section 2.4).

2.1 Basic Documentation

Basic documentation for various parts of the model is identified by enclosing it within a `umldocumentation` environment. The type of documentation can be specified, so that context-specific layout can occur.

`umldocumentation`

`\begin{umldocumentation}[⟨type⟩]`

The `umldocumentation` environment provides a general envrionment for documentation. The optional type argument can be one of `category`, `class`, `operation`, `association` or `generic`. If no type is specified, then `generic` is assumed.

`umlexternalreferences`

`\begin{umlexternalreferences}`

The `umlexternalrefereces` is used within the `umldocumentation` environment to group references to other documents or URLs together. Since external references can be handled more elegantly by BibTEX's functions, suitably extended to handle URLs, it is generally more advisable to embed citations into the documentation.

`umlexternalurl`

`\umlexternalurl[⟨title⟩]{⟨url⟩}`

Within the `umlexternalreferences` environment, the `umlexternalurl` allows the specification of a URL. The title is printed, if available and a link to the URL provided. If no title is provided, then the URL is used as a title. If the `\href` command is available, through the use of the `hyperref` package, then a hyperlink is included, otherwise, the full URL is included as a footnote.

`umlexternalfile`

`\umlexternalfile[⟨title⟩]{⟨path⟩}`

Within the `umlexternalreferences` environment, the `umlexternalfile` allows the specification of an additional document. The title is printed, if available and a path to the file provided. If no title is provided, then the file name is used as a title. If there is a sensible title, the full path is included as a footnote.

`defterm`

`\defterm[⟨index⟩]{⟨term⟩}`

The `defterm` command indicates the definition of a term in the documentation. The term is emphasised and an index entry is made for the term. If the optional index entry is not specified, then the index entry will be the same as the defined term.

`compliance`

`\compliance[⟨index⟩]{⟨standard⟩}`

The `compliance` command indicates a section devoted to the documentation of compliance to some standard. The standard is bold-faced and an index entry is made. If the optional index entry is not specified, then the index entry will be the same as the standard.

`umlconstraints`

`\begin{umlconstraints}`

Within the `umldocumentation` environment, explicitly documented constraints can be included within a `umlconstraints` environment. In most cases, this environment simply appears as some additional paragraphs, with a suitable header.

`ocl`

`\begin{ocl}`

To add Object Constraint Language (OCL) documentation, enclose the OCL in the `ocl` environment. The `ocl` environment is a verbatim environment.

2.2 Categories

`umlcategory`

`\begin{umlcategory}[`⟨*options*⟩`]{`⟨*name*⟩`}`

The `umlcategory` environment groups all the classes, associations, etc. within a category. If the `extension` optional argument is given, then this category is an extension to some major category.

If the document class allows chapters, and this category is not an extension, then the category begins as a new chapter. In article-type document classes the category. Extensions begin a new section.

2.3 Class Groups

`umlclassgroup`

`\begin{umlclassgroup}{`⟨*name*⟩`}`

Classes within a category can be grouped in the `umlclassgroup` environment. This environment is usually used to group classes with common stereotypes together: interfaces, implementation classes, etc. The name supplied is usually the sterotype name for the class.

Class groups within category extensions cause no special behaviour.

2.4 Use Cases

Use cases are generally treated as classes within their own class group.

`umlusecases`

`\begin{umlusecases}`

This environment is a class group environment for listing use-cases.

`umlusecase`

`\begin{umlusecase}{`⟨*name*⟩`}`

Use cases are described in the `umlusecase` environment, where the name argument gives the name of the use case. Use cases can, theoretically, be treated as classes (section 2.5) but generally contain little more than documentation.

2.5 Classes

umlclass

\begin{umlclass}[⟨type⟩]{⟨name⟩}

Classes are described in the umlclass environment, where the name argument is usually the class name. The optional type argument can be set to interface to indicate an interface, rather than an ordinary class.

umlpreamble

\begin{umlpreamble}

Operations and classes can have a preamble. The preamble to an class consists of the set of parameters that the class uses.

umlparameter

\umlparameter{⟨name⟩}{⟨type⟩}{⟨description⟩}

Within the umlpreamble environment, the umlparameter macro specifies a parameter, either a class parameter or an operation argument. The name argument is the formal name of the parameter. The type argument is the type of the parameter; if this argument is empty, the type declaration is suppressed. The description argument is a description of the parameter.

umlclassref

\umlclassref{⟨name⟩}

umlclassnamed

\umlclassnamed{⟨name⟩}

These two commands allow error-free references to classes that may or may not be defined in the current document. The umlclassnamed macro gives the class name and a section cross-reference if the class is defined in the current document. The umlclassref provides a section cross-reference if the class is defined in the current document. Unless the nosectionmarks option is used, the section references are preceeded by a section mark (§).

2.5.1 Attributes

umlattributes

\begin{umlattributes}

The umlattributes environment is used to group attributes together into an atribute list. This environment may only be used within the umlclass environment.

umlattribute

\umlattribute[⟨options⟩]{⟨attribute⟩}{⟨type⟩}{⟨default⟩}{⟨description⟩}

The `\umlattribute` macro allows the specification of an attribute within the `umlattributes` environment. Possible options are `static`, `derived` or both, seperated by commas. If there is no default, then the default argument should be left empty.

2.5.2 Operations

Classes usually contain a set of operations that the class implements. The operations environments allow these operations to be grouped together and documented. Since several operations may share the same name (eg., `doSomething()` and `doSomething(Type argument)`), it is possible to optionally number operations with an index number to disambiguate names.

`umloperations`

`\begin{umloperations}`

The `umloperations` environment is used to group all the operations within a class.

`umloperation`

`\begin{umloperation}[`⟨*index*⟩`]{`⟨*name*⟩`}`

The `umloperation` environment encloses the documentation for an operation. The optional index argument, if present is an index number for the operation; if absent an index of 1 is assumed. The operation name is the name of the operation, without parentheses, return types, arguments, etc.

Operations have a preamble. The preamble to an operation — the signature, arguments, exceptions, etc. — is grouped together into a `umlpreamble` environment, see section 2.5.

`umlsignature`

`\umlsignature{`⟨*signature*⟩`}`

The `umlsignature` command can be used within the `umloperation` environment. The signature argument is the signature of the operation: the name, return type, arguments, etc. For example `Time addTime(Integer hours, Integer minutes, Integer seconds)`

`umlexceptions`

`\umlexceptions{`⟨*exceptions*⟩`}`

The `umlexceptions` macro, for use within the `umloperation` environment allows the exceptions that an operation may raise to be listed. The exceptions argument is a comma-seperated list of exceptions.

2.5.3 Relationships

Relationships covers such ideas as class specialisation, realization,[1] instantiation, association and dependency.

`umlrelationships`

[1] Note the US spelling.

`\begin{umlrelationships}`

An environment grouping the various statements of relationship. The relationships are summarised in a table. The `\umlinherits`, `\umlrealizes`, `\umlinstantiates`, `\umlinheritedby`, `\umlrealisedby`, `\umlinstantiatedby` and `\umlassocrelationship` commands all occur within this environment.

`umlinherits`

`\umlinherits`

`{`⟨*class*⟩`}` Indicates that this class inherits from/specialises the class argument.

`umlinheritedby`

`\umlinheritedby`

`{`⟨*class*⟩`}` Indicates that the argument class inherits from this class.

`umlrealizes`

`\umlrealizes`

`{`⟨*interface*⟩`}` Indicates that this class realizes the interface argument.

`umlrealizedby`

`\umlrealizedby`

`{`⟨*class*⟩`}` Indicates that the argument class realizes this class.

`umlinstantiates`

`\umlinstantiates`

`{`⟨*class*⟩`}` Indicates that this object instantiates the class argument.

`umlinstantiatedby`

`\umlinstantiatedby`

`{`⟨*class*⟩`}` Indicates that the argument class instantiates this class.

`umlassocrelationship`

`\umlassocrelationship`

`[`⟨*options*⟩`]{`⟨*class*⟩`}{`⟨*named*⟩`}{`⟨*multiplicity*⟩`}` Indicates that this class has an association named `name` with multiplicity `multiplicity` with the argument `class`. The optional `options` argument can specify the same options as a `\umlrole` command (see section 2.6.

2.6 Associations

Associations within a category are allocated a seperate section. Extensions to other categories are grouped with the extensions.

`umlassociations`

`\begin{umlassociations}`

Group all the associations for a category together. The environments and macros within this section must all be grouped within this environment.

`umlassocsummary`

`\begin{umlassocsummary}`

The `umlassocsummary` environment provides an environment for summarising the roles in this category.

`umlassociation`

`\umlassociation{`⟨*name*⟩`}`

Define an association with the supplied argument as a name. If used within the `umlassocsummary` environment, then a summary entry is generated, otherwise a full association definition is assumed.

`umlrole`

`\umlrole[`⟨*options*⟩`]{`⟨*role*⟩`}{`⟨*class*⟩`}{`⟨*cardinality*⟩`}`

Define an association role in the `umlassocsummary` or `umlassociations` environments. Options are: `navigable` indicating that the association is navigable to this role, `aggregate` indicating that the role represents an aggregate, `composite` indicating that the role represents a composition.

2.7 Diagrams

Class, interaction, package diagrams, etc. can be added to the documentation by importing graphics files EPS files are the best format, since UML diagrams are likely to be quite large and EPS files can be expanded and contracted without losing information. Many UML diagrams are wider than they are tall. The `wide` option rotates a diagram (although not the caption or supporting text) 90 degrees to allow a better fit on the page. Diagrams are implemented as floating figures, with the diagram filling the page. Any additional notes can be typed into the diagram's environment body, to be carried with the diagram.[2]

`umlclassdiagram`

`\begin{umlclassdiagram}[`⟨*options*⟩`]{`⟨*name*⟩`}{`⟨*file*⟩`}`

The `umlclassdiagram` inserts the specified graphics file into a floating figure with appropriate captions, etc. If the `wide` option is included in the optional argument, the diagram will be turned on its side.

[2] Note that there is only limited space for additional notes

3 Languages

Certain "name" macros insert text into the document. These macros are renewable, so that languages other than English can be supported. The name macros are summarised in table 1.

Similarly, certain "icon" macros insert symbols into the document where compact notes are needed. The icon macros are summarised in tableřefTab:Icons.

4 Limitations

The current `umldoc` package concentrates on the declarative, logical side of UML: use cases, classes and class relationships. Deployment information, such as components, component diagrams and deployment diagrams are not, currently supported.

On the class side, the more esoteric diagrams, such as state diagrams and interaction diagrams are not supported.

5 The Code

5.1 Identification

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{umldoc}[\filedate\ UML Documentation Environments \fileversion]
\typeout{Package 'umldoc' \fileversion}
```

5.2 Preamble

Packages used by this package. I should probably try and cut down on the number of required packages, since I'll have to keep up with the changes in all of them.

```
\RequirePackage{ifthen}
\RequirePackage{graphicx}
\RequirePackage{verbatim}
\RequirePackage{amssymb}
\RequirePackage{longtable}
\RequirePackage{xspace}
```

Utilities. The main one handles any errors.

```
\newcommand{\UML@Error}[1]{\@latex@error{umldoc: #1}}
```

Option processing

```
\newboolean{UML@index@option}
\setboolean{UML@index@option}{true}
\newboolean{UML@sectionmark@option}
\setboolean{UML@sectionmark@option}{true}
\newboolean{UML@margin@option}
\setboolean{UML@margin@option}{true}
\DeclareOption{noindex}{\setboolean{UML@index@option}{false}}
\DeclareOption{nosectionmarks}{\setboolean{UML@sectionmark@option}{false}}
```

Macro	Default
`\umlaggregatename`	Aggregate
`\umlassociationname`	Association
`\umlassociationsname`	Associations
`\umlattributesname`	Attributes
`\umlcategorydocumentationheader`	Package Description
`\umlcategoryextensionpostfix`	Package
`\umlcategoryextensionprefix`	Extensions to the
`\umlcardinalityname`	Card.
`\umlclassdiagramname`	Class Diagram
`\umlclassname`	Class
`\umlcompliancename`	Compliance
`\umlcompositename`	Composite
`\umlconstraintsname`	**Constraints**
`\umlcontinuedname`	. . . continued
`\umlderivedattributename`	*derived*
`\umldescriptionname`	Description
`\umlinheritsname`	Inherits
`\umlinheritedbyname`	Inherited by
`\umlinterfacename`	Interface
`\umlinstantiatesname`	Instantiates
`\umlinstantiatedbyname`	Instantiated by
`\umlnavigablename`	Navigable
`\umlnotesname`	Notes
`\umloclname`	**OCL**
`\umloperationsname`	Operations
`\umlraisesname`	Raises
`\umlrealizesname`	Realizes
`\umlrealizedbyname`	Realized by
`\umlrelationshipname`	Relationship
`\umlrelationshipsname`	Relationships
`\umlrolename`	Role
`\umlrolesname`	Roles
`\umlstaticattributename`	*static*
`\umltermsname`	Terms
`\umlusecasename`	Use Case
`\umlusecasesname`	Use Cases

Table 1: Renewable Language-Specific Name Commands

Macro	Default
\umlaggregateicon	◊
\umlassociationicon	↔
\umlcompositeicon	♦
\umlfileicon	△
\umlinheritsicon	⇑
\umlinheritedbyicon	⇓
\umlinstantiatesicon	≻
\umlinstantiatedbyicon	≺
\umlnavigableicon	→
\umlrealizesicon	↑
\umlrealizedbyicon	↓
\umlurlicon	▷

Table 2: Renewable Language-Specific Icon Commands

```
\DeclareOption{nomarginnotes}{\setboolean{UML@margin@option}{false}}
\DeclareOption*{\UML@Error{Unknown option \CurrentOption}}
\ProcessOptions
```

Sizes for diagrams. Diagrams are scaled to fit onto a single page, with a little bit of space for the caption.

```
\newlength{\UML@diagramwidth}
\newlength{\UML@diagramheight}
\setlength{\UML@diagramwidth}{\textwidth}
\addtolength{\UML@diagramwidth}{-4em}
\setlength{\UML@diagramheight}{\textheight}
\addtolength{\UML@diagramheight}{-4.5cm}
```

5.3 Basic Documentation

`umldocumentation` Provide a grouping environment for documentation. For the most part, this doesn't do anything very exciting. However, if this is category documentation for an article-style environment, then we provide a suitable header.

```
\newenvironment{umldocumentation}[1][generic]{%
    \ifthenelse{\equal{#1}{category}}{%
        \ifthenelse{\isundefined{\chapter} \and \not \boolean{UML@category@extension}}{%
            \section*{\umlcategorydocumentationheader}
        }{}
    }{}
    \par%
}{%
    \hfill%
}
```

`umlexternalreferences` Group external references together into a single list. Ideally, we should use little

icons for the list entries, but so it goes.

```
\newenvironment{umlexternalreferences}{%
    \begin{list}{\ensuremath{\circ}}{%
        \setlength{\itemsep}{0pt}
        \setlength{\parsep}{0pt}
    }
}{%
    \end{list}
}
```

`umlexternalurl` Include a URL reference, along with a hyperlink if `\href` is present.

```
\newcommand{\umlexternalurl}[2][]{%
    \item[\umlurlicon]
    \ifthenelse{\equal{#1}{}}{%
        \def\UML@ref@title{\texttt{#2}}
        \def\UML@ref@foot{}
    }{%
        \def\UML@ref@title{#1}
        \def\UML@ref@foot{\footnote{\texttt{#2}}}
    }
    \ifthenelse{\isundefined{\href}}{%
        \UML@ref@title\UML@ref@foot
    }{%
        \href{#2}{\UML@ref@title}
    }
}
```

`umlexternalfile` Include a file reference.

```
\newcommand{\umlexternalfile}[2][]{%
    \item[\umlfileicon]
    \ifthenelse{\equal{#1}{}}{%
        \texttt{#2}
    }{%
        #1\footnote{\texttt{#2}}
    }
}
```

`defterm` Define a term.

```
\newcommand{\defterm}[2][]{%
    \ifthenelse{\boolean{UML@index@option}}{%
        \ifthenelse{\equal{#1}{}}{%
            \index{\umltermsname!#2}%
            \index{#2}%
        }{%
            \index{\umltermsname!#1}%
            \index{#1}%
        }%
    }{}%
    \emph{#2}\xspace}
```

`compliance` Define a compliance entry.

```
\newcommand{\compliance}[2][]{%
    \ifthenelse{\boolean{UML@index@option}}{%
        \ifthenelse{\equal{#1}{}}{%
            \index{\umlcompliancename!#2}%
            \index{#2}%
        }{%
            \index{\umlcompliancename!#1}%
            \index{#1}%
        }%
    }{}%
    \textbf{#2}\xspace}
```

`umlconstraints` Print the header for each set of constraints. This may need to be changed to a single header at some point?

```
\newenvironment{umlconstraints}{%
    \par
    \umlconstraintsname
    \par
}{%
}
```

`ocl` Print out OCL constraints in a suitable verbatim environment.

```
\newenvironment{ocl}{%
    \par
    \umloclname
    \par
    \verbatim
}{%
    \endverbatim
}
```

`umlurlicon` The symbol to use as a URL icon. This can be redefined if another symbol is needed.

```
\newcommand{\umlurlicon}{\ensuremath{\triangleright}}
```

`umlfileicon` The symbol to use as a file icon. This can be redefined if another symbol is needed.

```
\newcommand{\umlfileicon}{\ensuremath{\triangle}}
```

`umltermsname` How to title term definition index entries. This can be redefined if another name is needed.

```
\newcommand{\umltermsname}{Terms}
```

`umlcompliancename` How to title compliance discussion index entries. This can be redefined if another name is needed.

```
\newcommand{\umlcompliancename}{Compliance}
```

`umlconstraintsname` How to title constraints sections. This can be redefined if another name is needed.

```
\newcommand{\umlconstraintsname}{\textbf{Constraints}}
```

umloclname How to title OCL sections. This can be redefined if another name is needed.

```
\newcommand{\umloclname}{\textbf{OCL}}
```

umlnotesname How to say notes in a table (redefinable).

```
\newcommand{\umlnotesname}{Notes}
```

umldescriptionname How to say description in a table (redefinable).

```
\newcommand{\umldescriptionname}{Description}
```

5.4 Categories

umlcategory Use these variables to handle whether we are inside a category or not.

```
\newboolean{UML@inside@category}
\setboolean{UML@inside@category}{false}
```

Whether this category is an extension or not.

```
\newboolean{UML@category@extension}
\setboolean{UML@category@extension}{false}
```

How to generate index entries for the category

```
\gdef\UML@category@index@start{\relax}
\gdef\UML@category@index@end{\relax}
```

The category labels

```
\def\UML@category@name{}
\def\UML@category@major{}
```

Define based on whether it looks like we are in a book or not if we are, then use chapters. Defer printing stuff about package documentation until we have some documentation to refer to. Don't worry about an index entry for the major package if it looks like that this is the whole article.

```
\ifthenelse{\isundefined{\chapter}}{%
    \def\UML@category@header{%
        \gdef\UML@category@index@start{\relax}
        \gdef\UML@category@index@end{\relax}
    }%
    \def\UML@category@extension@header{%
        \section{\umlcategoryextensionprefix\ \UML@category@name\ \umlcategoryextensionpostf
        \gdef\UML@category@index@start{\index{\UML@category@name!extensions}}
        \gdef\UML@category@index@end{}
    }
}{%
    \def\UML@category@header{%
        \chapter{\UML@category@name}
        \gdef\UML@category@index@start{\index{\UML@category@name}}
        \gdef\UML@category@index@end{}
    }%
    \def\UML@category@extension@header{%
        \section{\umlcategoryextensionprefix\ \UML@category@name\ \umlcategoryextensionpostf
        \gdef\UML@category@index@start{\index{\UML@category@name!extensions}}
```

```
        \gdef\UML@category@index@end{}
    }%
}
\newenvironment{umlcategory}[2][major]{%
    \ifthenelse{\boolean{UML@inside@category}}{%
        UML@error{Can't nest categories}
    }{}%
    \gdef\UML@category@name{#2}
    \ifthenelse{\equal{#1}{extension}}{%
        \setboolean{UML@category@extension}{true}
        \UML@category@extension@header
        \label{UML:Category:\UML@category@major:\UML@category@name}
    }{%
        \setboolean{UML@category@extension}{false}
        \UML@category@header
        \let\UML@category@major=\UML@category@name
        \label{UML:Category:\UML@category@name}
    }
    \UML@category@index@start
    \setboolean{UML@inside@category}{true}
}{%
    \setboolean{UML@category@extension}{false}
    \setboolean{UML@inside@category}{false}
    \UML@category@index@end
}
```

`umlcategoryref` A reference to the named category.

```
\newcommand{\umlcategoryref}[1]{\ref{UML:Category:#1}}
```

`umlcategorydocumentationheader` The header for the basic package description. This can be redefined if another language is needed.

```
\newcommand{\umlcategorydocumentationheader}{Package Description}
```

`umlcategoryextensionprefix` The prefix for package extensions. This can be redefined if another language is needed.

```
\newcommand{\umlcategoryextensionprefix}{Extensions to the}
```

`umlcategoryextensionpostfix` The postfix for package extensions. This can be redefined if another language is needed.

```
\newcommand{\umlcategoryextensionpostfix}{Package}
```

5.5 Class Groups

Class groups are simple grouping environments.

Labels for the class group

```
\gdef\UML@classgroup@name{}
```

umlclassgroup

```
\newenvironment{umlclassgroup}[1]{%
    \ifthenelse{\boolean{UML@inside@category}}
    {}{
        \UML@Error{Class groups can only occur inside categories.}
    }
    \gdef\UML@classgroup@name{#1}
    \ifthenelse{\boolean{UML@category@extension}}{}{%
        \section{\UML@classgroup@name}
        \label{UML:ClassGroup:\UML@category@name:\UML@classgroup@name}
    }
    \ifthenelse{\boolean{UML@index@option}}{%
        \index{\UML@category@name!\UML@classgroup@name}
        \index{\UML@classgroup@name}
    }{}
}{%
}
```

umlclassgroupref Make a reference to a class group.

```
\newcommand{\umlclassgroupref}[1]{\ref{UML:ClassGroup:\UML@category@name:#1}}
```

umlusecases Use cases lists are basically a special case of class groups.

```
\newenvironment{umlusecases}{%
    \begin{umlclassgroup}{\umlusecasesname}
}{%
    \end{umlclassgroup}
}
```

umlusecasesname The redefinable name of the use case group.

```
\newcommand{\umlusecasesname}{Use Cases}
```

5.6 Classes

Classes are simple grouping environments.

Labels for the class. And an indicator to see whether we are inside a class or not.

```
\gdef\UML@class@name{}
\newboolean{UML@inside@class}
\setboolean{UML@inside@class}{false}
```

UML@class A generalised class-like object to handle classes, use cases, interfaces and anything else that is essentally a class under a different name.

```
\newenvironment{UML@class}[2]{%
    \ifthenelse{\boolean{UML@inside@category}}
    {}{
        \UML@Error{Class-like things can only occur inside categories.}
    }
    \ifthenelse{\boolean{UML@inside@class}}{
```

```
        \UML@Error{Class-like things cannot be nested.}
    }{}
    \setboolean{UML@inside@class}{true}
    \gdef\UML@class@name{#2}
    \subsection{\UML@class@name}
    \ifthenelse{\boolean{UML@category@extension}}{%
        \label{UML:Class:\UML@category@major:\UML@class@name}
    }{%
        \label{UML:Class:\UML@class@name}
    }
    \ifthenelse{\boolean{UML@index@option}}{%
        \ifthenelse{\boolean{UML@category@extension}}{%
            \index{\UML@class@name}
        }{%
            \index{\UML@class@name}
        }
    }{}
}{%
    \setboolean{UML@inside@class}{false}
}
```

`umlclass` Stuff for classes and interfaces.

```
\newenvironment{umlclass}[2][class]{%
    \ifthenelse{\equal{#1}{interface}}{%
        \def\UML@class@type{\umlinterfacename}
    }{%
        \def\UML@class@type{\umlclassname}
    }%
    \begin{UML@class}{\UML@class@type}{#2}
}{%
    \end{UML@class}
}
```

`umlusecase` Stuff for use cases.

```
\newenvironment{umlusecase}[1]{%
    \begin{UML@class}{\umlusecasename}{#1}
}{%
    \end{UML@class}
}
```

`umlpreamble` Group all the preamble for an operation or class into a suitable environment so that the layout looks reasonably sensible.

```
\newboolean{UML@inside@preamble}
\setboolean{UML@inside@preamble}{false}
```

The parameter list environment.

```
\newenvironment{umlpreamble}{%
    \setboolean{UML@inside@preamble}{true}%
    \begin{list}{}{
        \setlength{\topsep}{0cm}
```

```
        \setlength{\parsep}{0cm}
        \setlength{\itemsep}{0cm}
        \setlength{\leftmargin}{0cm}
        \setlength{\rightmargin}{1cm}
    }
}{%
    \end{list}
    \setboolean{UML@inside@preamble}{false}%
}
```

`umlparameter` Add a parameter to the preamble list. The name and type are bold, any description in normal font.

```
\newcommand{\umlparameter}[3]{%
    \ifthenelse{\boolean{UML@inside@preamble}}{}{%
        \UML@Error{Parameters must be within a umlpreamble environment.}%
    }%
    \item
    \textbf{#1\ifthenelse{\equal{#2}{}}{}{: #2}} #3
}
```

`umlclassref` Make a reference to a class (or use case or whatever). If this reference is absent, then print a warning and ignore the reference.

```
\newcommand{\umlclassref}[1]{%
    \@ifundefined{r@UML:Class:#1}{%
        \typeout{Class #1 not in document - ignoring cross-reference}%
    }{%
        \ifthenelse{\boolean{UML@sectionmark@option}}{\S}{}\ref{UML:Class:#1}%
    }%
}
```

`umlclassnamed` Name a class, including a reference if there is one. If this reference is absent, then print a warning and ignore the reference.

```
\newcommand{\umlclassnamed}[1]{%
    \@ifundefined{r@UML:Class:#1}{%
        #1\typeout{Class #1 not in document - ignoring cross-reference}%
    }{%
        #1~\ifthenelse{\boolean{UML@sectionmark@option}}{\S}{}\ref{UML:Class:#1}%
    }%
}
```

`umlclassname` The (redefinable) name of a class.

```
\newcommand{\umlclassname}{Class}
```

`umlinterfacename` The (redefinable) name of an interface.

```
\newcommand{\umlinterfacename}{Interface}
```

`umlusecasename` The (redefinable) name of a use case.

```
\newcommand{\umlusecasename}{Use Case}
```

5.6.1 Attributes

Handle attributes as a list.[3]

umlattributes

```
\newenvironment{umlattributes}{%
    \ifthenelse{\boolean{UML@inside@class}}{}{
        \UML@Error{Attributes must be inside a umlclass.}
    }
    \subsubsection{\umlattributesname}
    \ifthenelse{\boolean{UML@category@extension}}{%
        \label{UML:\UML@category@major:\UML@class@name:Attributes}
    }{
        \label{UML:\UML@class@name:Attributes}
    }
    \ifthenelse{\boolean{UML@index@option}}{%
        \index{\UML@class@name!\umlattributesname}
    }{}
    \begin{description}
}{%
    \end{description}
}
```

umlattribute

```
\newcommand{\umlattribute}[5][]{%
```

Process options.

```
    \def\UML@staticoption{}
    \def\UML@derivedoption{}
    \@for\UML@option:=#1\do{%
        \ifthenelse{\equal{\UML@option}{static}}{%
            \def\UML@staticoption{\umlstaticattributename~}
        }{}
        \ifthenelse{\equal{\UML@option}{derived}}{%
            \def\UML@derivedoption{\umlderivedattributename~}
        }{}
    }
```

Produce the list item

```
    \item[#2: #3 \ifthenelse{\equal{#4}{}}{}{ = #4}]
    \UML@staticoption \UML@derivedoption #5
}
```

Changable names for things in the attributes list. Re-define these for other languages.

umlattributesname

```
\newcommand{\umlattributesname}{Attributes}
```

[3] Should this be a table? Maybe, in the future, this should get handled in a different way, with entries in the .aux file?

umlstaticattributename

```
\newcommand{\umlstaticattributename}{\emph{static}}
```

umlderivedattributename

```
\newcommand{\umlderivedattributename}{\emph{derived}}
```

5.6.2 Operations

umloperations Surrounding wrapper for operations. The pagebreak is to stop the operations from being de-widowed, which causes difficulties with the margined operation name.

```
\newenvironment{umloperations}{%
    \ifthenelse{\boolean{UML@inside@class}}{}{
        \UML@Error{Operations must be inside a umlclass.}
    }
    \subsubsection{\umloperationsname}
\pagebreak[0]
    \ifthenelse{\boolean{UML@category@extension}}{%
        \label{UML:\UML@category@major:\UML@class@name:Operations}
    }{
        \label{UML:\UML@class@name:Operations}
    }
    \ifthenelse{\boolean{UML@index@option}}{%
        \index{\UML@class@name!\umloperationsname}
    }{}
}{%
}
```

Indicate whether we are processing an operation at the moment, and keep this operation's name.

```
\newboolean{UML@inside@operation}
\setboolean{UML@inside@operation}{false}
\gdef\UML@operation@name{}
```

umloperation Full operation documentation is enclosed in this environment. The optional argument can be used to identify similarly named operations.

```
\newenvironment{umloperation}[2][1]{%
    \ifthenelse{\boolean{UML@inside@class}}{}{%
        \UML@Error{Operations must be inside a class.}%
    }%
    \ifthenelse{\boolean{UML@inside@operation}}{%
        \UML@Error{Operations cannot be nested.}%
    }{}%
    \setboolean{UML@inside@operation}{true}%
    \gdef\UML@operation@name{#2}%
    \gdef\UML@operation@id{#2-#1}%
    \ifthenelse{\boolean{UML@index@option}}{%
        \index{\UML@operation@name @\texttt{\UML@operation@name}}
    }{}%
```

```
}{%
    \setboolean{UML@inside@operation}{false}%
    \noindent \hspace{0cm} \\[0.1ex]%
}
```

`umlsignature` Put the signature on a single line, filled to stop TEX attempting to make odd padding arrangements.

```
\newcommand{\umlsignature}[1]{%
    \ifthenelse{\boolean{UML@category@extension}}{%
        \label{UML:Operation:\UML@category@major:\UML@class@name:\UML@operation@id}%
    }{%
        \label{UML:Operation:\UML@class@name:\UML@operation@id}%
    }%
    \ifthenelse{\boolean{UML@inside@preamble}}{%
        \item
        \textbf{#1}
        \ifthenelse{\boolean{UML@margin@option}}{%
            \marginpar{\hspace{0cm}\footnotesize\UML@operation@name}%
        }{}
    }{%
        \textbf{#1}
    }%
}
```

`umlexceptions` Provide a list of exceptions that this operation raises.

```
\newcommand{\umlexceptions}[1]{%
    \ifthenelse{\boolean{UML@inside@preamble}}{}{%
        \UML@Error{Exceptions must be within a umlpreamble environment.}%
    }%
    \item
    \textbf{\umlraisesname:} #1
}
```

`umloperationref` Make a reference to an operation. This only works within the local class that is being defined.

```
\newcommand{\umloperationref}[2][1]{%
    \ifthenelse{\boolean{UML@inside@class}}{}{
        \UML@Error{Operation references must be inside a class.}
    }
    \ifthenelse{\boolean{UML@category@extension}}{%
        \ref{UML:Operation:\UML@category@major:\UML@class@name:#2-#1}
    }{%
        \ref{UML:Operation:\UML@class@name:#2-#1}
    }
}
```

`umloperationpageref` Make a page reference to an operation. This only works within the local class that is being defined.

```
\newcommand{\umloperationpageref}[2][1]{%
```

```
    \ifthenelse{\boolean{UML@inside@class}}{}{
        \UML@Error{Operation references must be inside a class.}
    }
    \ifthenelse{\boolean{UML@category@extension}}{%
        \pageref{UML:Operation:\UML@category@major:\UML@class@name:#2-#1}
    }{%
        \pageref{UML:Operation:\UML@class@name:#2-#1}
    }
}
```

umlraisesname A redefinable macro for whatever word means "raises."

```
\newcommand{\umlraisesname}{Raises}
```

umloperationsname A redefinable macro for whatever word means "operations."

```
\newcommand{\umloperationsname}{Operations}
```

5.7 Relationships

All inheritance styles work off the `UML@inheritance` and `UML@inherits` environment and macro.

Make sure that we are sensibly inside the environment for the macro calls. Also, keep track of what elements we should put into the key line.

```
\newboolean{UML@inside@relationships}
\setboolean{UML@inside@relationships}{false}
```

Sizes for the summary table

```
\newlength{\UML@summary@class@width}
\setlength{\UML@summary@class@width}{6cm}
\newlength{\UML@summary@doc@width}
\setlength{\UML@summary@doc@width}{\textwidth}
\addtolength{\UML@summary@doc@width}{-\UML@summary@class@width}
\addtolength{\UML@summary@doc@width}{-4cm}
```

umlrelationships A general grouping of relationship information, defined in terms of the utility environment.

```
\newenvironment{umlrelationships}{%
    \ifthenelse{\boolean{UML@inside@class}}%
    {}{%
        \UML@Error{Relationships can only occur inside classes.}%
    }%
    \ifthenelse{\boolean{UML@inside@relationships}}{%
        \UML@Error{Relationships cannot be nested.}%
    }{}%
    \setboolean{UML@inside@relationships}{true}%
    \ifthenelse{\boolean{UML@index@option}}{%
        \index{\UML@class@name!\umlrelationshipsname}%
    }{}%
    \subsubsection{\umlrelationshipsname}
\gdef\UML@inherits@tablenote{}
```

```
    \gdef\UML@inheritedby@tablenote{}
\gdef\UML@realizes@tablenote{}
\gdef\UML@realizedby@tablenote{}
\gdef\UML@instantiates@tablenote{}
\gdef\UML@instantiatedby@tablenote{}
\gdef\UML@association@tablenote{}
\begin{center}
    \begin{tabular}{lp{\UML@summary@class@width}p{\UML@summary@doc@width}r}
    \hline
    & \umlclassname & \umldescriptionname  & \umlnotesname \\
    \hline
}{%
    \hline
\multicolumn{4}{l}{
     \footnotesize
\UML@inherits@tablenote\
     \UML@inheritedby@tablenote\
\UML@realizes@tablenote\
\UML@realizedby@tablenote\
\UML@instantiates@tablenote\
\UML@instantiatedby@tablenote\
\UML@association@tablenote\
} \\
    \end{tabular}
\end{center}
    \setboolean{UML@inside@relationships}{false}
}
```

`UML@relationship` Generic class inheritance/specialisation/realisation, etc. Refined versions of this can be defined for the various inheritance types.

```
\newcommand{\UML@relationship}[4]{%
    \ifthenelse{\boolean{UML@inside@relationships}}{}{%
        \UML@Error{Relationships can only occur inside the umlrelationships environment.}%
    }%
    \ifthenelse{\boolean{UML@index@option}}{%
        \index{#2!\umlrelationshipsname}%
    }{}%
    #1 & {\hspace{0cm}\umlclassnamed{#2}} & {\hspace{0cm}#3} & #4 \\
}
```

`umlinherits` The inheritance from/specialisation relationship.

```
\newcommand{\umlinherits}[2]{
\gdef\UML@inherits@tablenote{\umlinheritsicon:\umlinheritsname}%
    \UML@relationship{\umlinheritsicon}{#1}{#2}{}
}
```

`umlrealizes` The realization from an interface relationship.

```
\newcommand{\umlrealizes}[2]{
    \gdef\UML@realizes@tablenote{\umlrealizesicon:\umlrealizesname}%
```

```
    \UML@relationship{\umlrealizesicon}{#1}{#2}{}
}
```

`umlinstantiates` This class instantiates another class (factory).

```
\newcommand{\umlinstantiates}[2]{
    \gdef\UML@instantiates@tablenote{\umlinstantiatesicon:\umlinstantiatesname}%
    \UML@relationship{\umlinstantiatesicon}{#1}{#2}{}
}
```

`umlinheritedby` The inheritance from/specialisation relationship.

```
\newcommand{\umlinheritedby}[2]{
    \gdef\UML@inheritedby@tablenote{\umlinheritedbyicon:\umlinheritedbyname}%
    \UML@relationship{\umlinheritedbyicon}{#1}{#2}{}
}
```

`umlrealizedby` The realization from an interface relationship.

```
\newcommand{\umlrealizedby}[2]{
    \gdef\UML@realizedby@tablenote{\umlrealizedbyicon:\umlrealizedbyname}%
    \UML@relationship{\umlrealizedbyicon}{#1}{#2}{}
}
```

`umlinstantiatedby` This class instantiates another class (factory).

```
\newcommand{\umlinstantiatedby}[2]{
    \gdef\UML@instantiatedby@tablenote{\umlinstantiatedbyicon:\umlinstantiatedbyname}
    \UML@relationship{\umlinstantiatedbyicon}{#1}{#2}{}
}
```

`umlassocrelationship` This is inside a relationships table, so just generate a summary line.

```
\newcommand{\umlassocrelationship}[4][]{%
    \gdef\UML@association@tablenote{
\umlassociationicon:\umlassociationname\
\hfill
        \umlnavigableicon:\umlnavigablename\
       \umlaggregateicon:\umlaggregatename\
        \umlcompositeicon:\umlcompositename\
}%
    \umlassociationicon &
    {\hspace{0cm}\umlclassnamed{#2}} &
    {\hspace{0cm}#3 #4} &
    \@for\UML@option:=#1\do{%
                \ifthenelse{\equal{\UML@option}{navigable}}{%
                \umlnavigableicon%
            }{}%
                \ifthenelse{\equal{\UML@option}{aggregate}}{%
                \umlaggregateicon%
            }{}%
                \ifthenelse{\equal{\UML@option}{composite}}{%
                \umlcompositeicon%
            }{}%
```

```
        }%
    \\
}
```

umlrelationshipname A redefinable macro for whatever word means “relationship.”

```
\newcommand{\umlrelationshipname}{Relationship}
```

umlrelationshipsname A redefinable macro for whatever word means “relationships.”

```
\newcommand{\umlrelationshipsname}{Relationships}
```

umlinheritsname A redefinable macro for whatever word means “inherits.”

```
\newcommand{\umlinheritsname}{Inherits}
```

umlinheritedbyname A redefinable macro for whatever phrase means “inherited by.”

```
\newcommand{\umlinheritedbyname}{Inherited by}
```

umlrealizesname A redefinable macro for whatever word means “realizes.”

```
\newcommand{\umlrealizesname}{Realizes}
```

umlrealizedbyname A redefinable macro for whatever phrase means “realized by.”

```
\newcommand{\umlrealizedbyname}{Realized by}
```

umlinstantiatesname A redefinable macro for whatever phrase means “instantiates.”

```
\newcommand{\umlinstantiatesname}{Instantiates}
```

umlinstantiatedbyname A redefinable macro for whatever phrase means “instantiated by.”

```
\newcommand{\umlinstantiatedbyname}{Instantiated by}
```

umlinheritsicon

```
\newcommand{\umlinheritsicon}{\ensuremath{\Uparrow}}
```

umlinheritedbyicon

```
\newcommand{\umlinheritedbyicon}{\ensuremath{\Downarrow}}
```

umlrealizesicon

```
\newcommand{\umlrealizesicon}{\ensuremath{\uparrow}}
```

umlrealizedbyicon

```
\newcommand{\umlrealizedbyicon}{\ensuremath{\downarrow}}
```

umlinstantiatesicon

```
\newcommand{\umlinstantiatesicon}{\ensuremath{\succ}}
```

umlinstantiatedbyicon

```
\newcommand{\umlinstantiatedbyicon}{\ensuremath{\prec}}
```

5.8 Associations

The `umlassociation` and `umlrole` commands must be able to work in both environments. Provide some stuff for handling this side of things.

```
\newboolean{UML@inside@associations}
\setboolean{UML@inside@associations}{false}
\newboolean{UML@inside@assocsummary}
\setboolean{UML@inside@assocsummary}{false}
\newboolean{UML@inside@assocfull}
\setboolean{UML@inside@assocfull}{false}
```

Keep track of what association that we are operating within.

```
\def\UML@association@name{}
```

`umlassociations` A grouping of all associations. This is handled differently for packages and package extensions.

```
\newenvironment{umlassociations}{%
    \ifthenelse{\boolean{UML@inside@category}}{%
    }{%
        UML@Error{umlassociations must be inside a umlcategory.}
    }
    \ifthenelse{\boolean{UML@inside@associations}}{%
        UML@Error{umlassociations must not be nested.}
    }{}
    \ifthenelse{\boolean{UML@category@extension}}{%
        \subsection{\umlassociationsname}
        \label{UML:\UML@category@major:\UML@category@name:Associations}
        \ifthenelse{\boolean{UML@index@option}}{
            \index{\umlassociationsname!\UML@category@name}
            \index{\UML@category@name!\umlassociationsname}
        }{}
    }{
        \section{\umlassociationsname}
        \label{UML:\UML@category@name:Associations}
        \ifthenelse{\boolean{UML@index@option}}{
            \index{\umlassociationsname!\UML@category@name}
            \index{\UML@category@name!\umlassociationsname}
        }{}
    }
    \setboolean{UML@inside@associations}{true}
    \setboolean{UML@inside@assocfull}{true}
}{%
    \setboolean{UML@inside@associations}{false}
    \setboolean{UML@inside@assocfull}{false}
}
```

`umlassocsummary` Do the summary as a table.

```
\newenvironment{umlassocsummary}{%
    \ifthenelse{\boolean{UML@inside@associations}}{%
    }{%
```

```
        UML@Error{umlassocsummary must be used inside umlassociations.}
    }
    \ifthenelse{\boolean{UML@inside@assocsummary}}{%
        UML@Error{umlassocsummary must not be nested.}
    }{}
    \setboolean{UML@inside@assocsummary}{true}
    \setboolean{UML@inside@assocfull}{false}
    \ifthenelse{\boolean{UML@category@extension}}{%
        \gdef\UML@tab@label{UML:\UML@category@major:\UML@category@name:Associations:Summary}
    }{%
        \gdef\UML@tab@label{UML:\UML@category@name:Associations:Summary}
    }
    \begin{longtable}{llp{\UML@summary@class@width}ll}
    \hline
    \caption{\UML@category@name --- \umlassociationsname \label{\UML@tab@label}}\\
    \hline
    \multicolumn{5}{l}{\umlassociationname} \\
    \hspace{1em} & \umlrolename & \umlclassname & \umlcardinalityname & \umlnotesname \\
    \hline
    \endfirsthead
    \hline
    \caption{\umlcontinuedname \label{\UML@tab@label}} \\
    \hline
    \multicolumn{5}{l}{\umlassociationname} \\
    \hspace{1em} & \umlrolename & \umlclassname & \umlcardinalityname & \umlnotesname \\
    \hline
    \endhead
    \hline
    \endfoot
    \hline
    \multicolumn{5}{r}{\footnotesize
        \umlnavigableicon:\umlnavigablename\
        \umlaggregateicon:\umlaggregatename\
        \umlcompositeicon:\umlcompositename\
    } \\
    \endlastfoot
}{%
    \hline
    \end{longtable}
    \setboolean{UML@inside@assocsummary}{false}
    \setboolean{UML@inside@assocfull}{true}
}
```

`umlassociation` Start an association. This is handled differently, depending upon whether this is the full thing or just a summary. The peculiar construction, with `\UML@assoc@output` is because `\ifthenelse` causes funny errors in table construction.

```
\newcommand{\umlassociation}[1]{%
    \ifUML@inside@assocsummary%
        \multicolumn{5}{l}{#1} \\
```

```
    \else%
        \ifthenelse{\boolean{UML@category@extension}}{%
               \subsubsection{#1}
               \label{UML:\UML@category@major:\UML@category@name:Association:#1}
        }{%
               \subsection{#1}
               \label{UML:\UML@category@name:Association:#1}
        }
        \ifthenelse{\boolean{UML@index@option}}{%
               \index{\umlassociationsname!#1}%
               \index{#1}%
        }{}%
    \fi%
    \gdef\UML@association@name{#1}%
}
```

`umlrole` Define a role. This is handled differently, depending upon whether this is the full thing or just a summary.

```
\newcommand{\umlrole}[4][]{%
    \ifthenelse{\boolean{UML@inside@associations}}{%
    }{%
        UML@Error{umlrole must be used inside umlassociations.}
    }

```

This is inside a summary table, so just generate the summary line.

```
    \ifUML@inside@assocsummary%
        \hspace{1em} & #2 & {\hspace{0cm}\umlclassnamed{#3}} & #4 &
        \@for\UML@option:=#1\do{%
                \ifthenelse{\equal{\UML@option}{navigable}}{%
                \umlnavigableicon%
            }{}%
                \ifthenelse{\equal{\UML@option}{aggregate}}{%
                \umlaggregateicon%
            }{}%
                \ifthenelse{\equal{\UML@option}{composite}}{%
                \umlcompositeicon%
            }{}%
        }%
        \\
    \else\fi%
```

This is part of the main definition, so do a full role spec.

```
    \ifUML@inside@assocfull%
        \par \noindent
           \textbf{\umlrolename: #2}
        \@for\UML@option:=#1\do{%
                \ifthenelse{\equal{\UML@option}{navigable}}{%
                \emph{ \umlnavigablename }%
            }{}%
```

```
                \ifthenelse{\equal{\UML@option}{aggregate}}{%
                \emph{ \umlaggregatename }%
             }{}%
                \ifthenelse{\equal{\UML@option}{composite}}{%
                \emph{ \umlcompositeicon }%
             }{}%
        }%
        \ifthenelse{\equal{#4}{}}{ #3.}{ #3, #4.}
        \ifthenelse{\boolean{UML@index@option}}{%
               \index{\umlrolesname!#2}
               \index{#2}
        }{}%
    \else\fi%
}
```

Various redefinable names for things.

`umlassociationsname` The (redefinable) name of the associations section.

```
\newcommand{\umlassociationsname}{Associations}
```

`umlassociationname` The (redefinable) name of an association.

```
\newcommand{\umlassociationname}{Association}
```

`umlassociationicon` The (redefinable) icon for an association.

```
\newcommand{\umlassociationicon}{\ensuremath{\leftrightarrow}}
```

`umlrolesname` The (redefinable) name of some roles.

```
\newcommand{\umlrolesname}{Roles}
```

`umlrolename` The (redefinable) name of a role.

```
\newcommand{\umlrolename}{Role}
```

`umlcontinuedname` How to say continued in a table (redefinable).

```
\newcommand{\umlcontinuedname}{\ldots continued}
```

`umlcardinalityname` The (redefinable) name of a cardinality.

```
\newcommand{\umlcardinalityname}{Card.}
```

`umlnavigablename` The (redefinable) name of a navigable option.

```
\newcommand{\umlnavigablename}{Navigable}
```

`umlaggregatename` The (redefinable) name of an aggregate option.

```
\newcommand{\umlaggregatename}{Aggregate}
```

`umlcompositename` The (redefinable) name of a composite option.

```
\newcommand{\umlcompositename}{Composite}
```

`umlnavigableicon` The (redefinable) icon for a navigable option.

```
\newcommand{\umlnavigableicon}{\ensuremath{\rightarrow}}
```

umlaggregateicon The (redefinable) icon for an aggregate option.

```
\newcommand{\umlaggregateicon}{\ensuremath{\lozenge}}
```

umlcompositeicon The (redefinable) icon for a composite option.

```
\newcommand{\umlcompositeicon}{\ensuremath{\blacklozenge}}
```

5.9 Diagrams

Diagrams essentially work by importing EPS graphics.

Labels to carry across the diagram

```
\gdef\UML@diagram@caption{}
\gdef\UML@diagram@type{}
```

UML@diagram A generic diagram environment.

```
\newenvironment{UML@diagram}[4]{%
    \ifthenelse{\boolean{UML@inside@category}}
    {}{
        \UML@Error{Diagrams can only occur inside categories.}
    }
    \begin{figure}[p]
    \begin{center}
```

Rotate wide diagrams to get a better fit.

```
    \ifthenelse{\equal{#1}{wide}}{%
        \includegraphics*[
            angle=90,
            width=\UML@diagramwidth,
            height=\UML@diagramheight,
             keepaspectratio=true
        ]{#4}
    }{%
        \includegraphics*[
            width=\UML@diagramwidth,
            height=\UML@diagramheight,
             keepaspectratio=true
        ]{#4}
    }
    \gdef\UML@diagram@caption{#3}
    \gdef\UML@diagram@type{#2}
```

Put all the captions, indexing at the end of the diagram. This will make things look right if you have text, etc. inside the diagram environment. Diagrams that are part of extensions get labelled with the major package name to ensure unique labels.

```
}{%
    \caption{\UML@diagram@type --- \UML@diagram@caption}
    \ifthenelse{\boolean{UML@category@extension}}{%
        \label{UML:Diagram:\UML@category@major:\UML@category@name:\UML@diagram@caption}
    }{
```

```
        \label{UML:Diagram:\UML@category@name:\UML@diagram@caption}
    }
    \ifthenelse{\boolean{UML@index@option}}{
        \index{\UML@diagram@type!\UML@diagram@caption}
        \index{\UML@diagram@caption!\UML@diagram@type}
        \index{\UML@category@major!\UML@diagram@type}
    }{}
    \end{center}
    \end{figure}
}
```

`umlclassdiagram` An environment for class diagrams, defined in terms of the general diagram environment.

```
\newenvironment{umlclassdiagram}[3][tall]{%
    \begin{UML@diagram}{#1}{\umlclassdiagramname}{#2}{#3}
}{%
    \end{UML@diagram}
}
```

`umlclassdiagramname` The (redefinable) name of the class diagram.

```
\newcommand{\umlclassdiagramname}{Class Diagram}
```

`umldiagramref` Make a reference to a diagram. See if this diagram is part of a local extension, first.

```
\newcommand{\umldiagramref}[1]{%
    \@ifundefined{r@UML:Diagram:\UML@category@major:\UML@category@name:#1}{%
        \ref{UML:Diagram:\UML@category@name:#1}
    }{%
        \ref{UML:Diagram:\UML@category@major:\UML@category@name:#1}
    }
}
```