



# UML2MMBase User's Guide

A code generator for the MMBase CMS

March 25, 2005

Author: Rudie Ekkelenkamp



## Table of Contents

1. What is UML2MMBase? .....	4
2. Requirements .....	5
3. Installation .....	6
4. Quick start .....	7
5. Using UML2MMBase .....	8
5.1. Creating a MMBase cloud model .....	8
5.2. MMBase Objects .....	8
5.3. MMBase Relations .....	17
5.4. Option Lists .....	23
5.5. Documentation .....	24
5.6. Inheritance .....	24
5.7. Interfaces .....	25
5.8. MMBase Packages .....	25
6. Generating the MMBase artifacts .....	26
6.1. Configuration .....	26
6.2. MMBase builder definitions .....	26
6.3. Application configuration file .....	26
6.4. Edit Wizards .....	26
6.5. JSP menu and search pages to access the edit wizards .....	26
6.6. HTML and XML docbook report of the object cloud .....	27
A. Frequently Asked Questions .....	28
B. References .....	29

# 1. What is UML2MMBase?

UML2MMbase is a code generation tool that can be used to convert a UML model of the MMBase object cloud into a basic MMBase application. The following artifacts can be created automatically from a UML model. UML2MMBase can greatly speed up development of MMBase applications while enforcing a well documented cloud model. The current version supports MMBase 1.7, but can still be configured to support mmbase 1.6.

- MMBase builder definitions
- Application configuration file
- Edit Wizards
- JSP menu and search pages to access the edit wizards
- HTML and XML docbook report of the object cloud

UML2MMBase works with UML Class diagrams based on the UML 1.4 definition.

## 2. Requirements

In order to use UML2MMBase tool some installation is required. The following tools are necessary to use UML2MMBase:

- Java JDK1.4: UML2MMBase has been tested with the Sun JDK 1.4.2\_04.
- MMBase distribution: UML2MMBase has been tested with the MMBase 1.7.1.
- UML tool that can export an UML 1.4 model in XMI format: Tested with the community edition of PoseidonUML 3.0 and with the community edition of MagicDraw 8.0 SP1.

## 3. Installation

UML2MMBase is distributed as a zip file: `uml2mmbase-1-4.zip`. After unzipping the file the following file structure is created.

`conf/`

Contains configuration files needed to run `uml2mmbase`.

`docs/`

Contains the documentation you are reading now.

`lib/`

Contains the necessary libraries to run `uml2mmbase`.

`modules/`

Contains the Velocity templates per module that can be customized.

`web/`

Contains some basic web pages and edit wizards for core objects like images, attachments and urls

`model.zuml`

Sample Poseidon 3.0 UML model that can be used to create your own models.

`model.xml.zip`

Sample MagicDraw 8 UML model that can be used to create your own models.

`uml2mmbase.bat`

The Windows script that has to be run to start the UML2MMBase codegeneration for a PoseidonUML 3.0 model. This script will read the `model.zuml` and generate all MMBase artifacts in the generated directory. If you want to use `magicdraw`, the `uml2mmbase.bat` script has to be changed. Change the value: `mmbase.poseidon2` into `mmbase.magicdraw8`.

`mmbase2uml.bat`

The Windows script that has to be run to start the MMBase2UML reverse engineering for one or all applications in a MMBase configuration. See the MMBase2UML documentation for more information.

`build.xml`

XML project file needed by the startup script to run the codegeneration.

`mmbase.properties`

Property file where the `mmbase` version and output directory can be configured.

## 4. Quick start

After installation of UML2MMBase has been completed, the following steps can be followed for a quick start:

1. Install java 1.4. Tested with JDK1.4.2\_04.
2. Open the model.zuml from PoseidonUML 3.0.
3. Create an object cloud by using the sample model as an example.
4. Save the model as model.zuml to the uml2mmbase directory.
5. Start a command prompt and change the prompt to the installation directory of uml2mmbase.
6. Run the codegeneration by starting the uml2mmbase.bat script.
7. Install Tomcat 5.0.16.
8. Unzip the mmbase 1.7.1 distribution in a temp directory and copy the mmbase-webapp directory to the tomcat webapps directory.
9. Copy all files in the generated directory of uml2mmbase to the webapps/mmbase-webapp directory of tomcat and start tomcat in the bin directory.
10. Start the generated edit wizards from a browser with: <http://localhost:8080/mmbase-webapp/editors/>.
11. Login with admin/admin2k and test the generated edit wizards.

## 5. Using UML2MMBase

### 5.1. Creating a MMBase cloud model

A MMBase cloud can be modeled using a UML Class diagram. For example UML classes can be used to specify an object in the MMBase cloud and UML dependencies or associations can be used to define the relations between the cloud objects.

To allow code generation from a class diagram an UML profile has been defined which is a standard UML extension that allows the usage of stereotypes and tagged values to add semantics to a UML Model. For UML2MMBase a MMBase UML Profile is defined. An overview of this model is shown in the following table.

Model Element	Name	StereoTypes	Tagged Values	Visibility
Class	builder name	<<MMBase>> <<Relation>> <<OptionList>> <<RelDef>>	title, subtitle, version	public, protected, private
Attribute	field name	<<required>>	minsize, maxsize, prompt, system	public, protected, private
Dependency	relation name	<<required>>	-	-
Association	relation name	-	-	-
Package	maintainer name for builder or dependent mmbase module.	<<MMBasePackage>>	-	-

## 5.2. MMBase Objects

### 5.2.1. Introduction

Objects in the MMBase cloud are modeled with class files and attributes. In the following figure a sample MMBase object cloud is shown that represents a simple web site with menus, pages, articles, images etc. This model will be used to demonstrate the different possibilities of UML2MMBase. The following two figures show the UML model from which an MMBase application can be generated and a figure of the resulting edit wizards.

**Figure 1. The sample UML model of an MMBase object cloud**





MyIE2 - [http://localhost:8080/uml2mmbase/editors/index.jsp]

File Edit View Favorites Groups Options Tools Window Help

Address http://localhost:8080/uml2mmbase/editors/index.jsp

http://localh...

menu

menu\_item

page

teaser

interview

report

article

paragraph

Zoek in title

Zoek in starttime

Zoek in endtime

Zoek in status

Typical fields for an article.

**title**

**author**

**starttime** 7 | october | 2003 at 13 : 04

**endtime** 7 | october | 2003 at 13 : 04

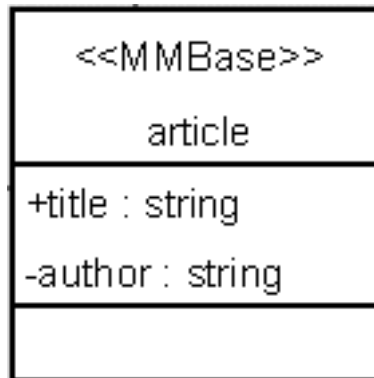
**status**

[cancel](#) - [save](#)

### 5.2.2. Class

The class name specifies the name of a MMBase builder. The name should be specified in lowercase and no spaces can be used. The class has to be marked with the <<MMBase>> stereotype (that is case sensitive!). It is also possible to define classes without the <<MMBase>> stereotype if no builder has to be generated. This is the case for the core builders like images and attachments and can be used to implement search functionality to the basic builders. In the next figure the "article" object is shown.

**Figure 3. The class definition of an article MMBase object**



So in this sample class, 2 string attributes have been defined: title and author. The title field has been specified as public (this is displayed as a + sign in front of the attribute name). The author field has been specified as private (this is displayed as a - sign in front of the attribute name). Public and private is the visibility of an attribute and will be used to define which fields of an object should be used for searching and displaying information in lists in the edit wizards. Since the article class has been given the MMBase stereotype, which is displayed as <<MMBase>>, a builder will be generated for this class.

The following visibilities have been defined for attributes: **public**: visible in edit wizard, wizard list and search field. **protected**: visible in edit wizard and wizard list. **private**: visible in edit wizard. In some case you don't want to see a field in the wizard, but should be available in the builder file. In that case on a field, the **system** tagged value can be set to true.

The visibility of the class itself can be used to determine if the wizards should be accessible from the menu list. The class should be set to **public** to be in the list. Both protected and private classes will delete the wizard from the menu list.

### 5.2.3. Field sizes and types of the class attributes

The fields of a cloud object are specified with attributes. Both a field type and field name should be specified for the object fields. The fieldname is used to create a database table, so no special characters should be used. For the field types a list of all possible values is defined in the following table. Also the default sizes are specified that are used in the GUI (this may deviate from the database size).

Type	Descriptions	Default GUI size	GUI display
html	WYSIWYG HTML	1024	WYSIWYG text area
string	Text	40	text area
date	Date and time	-	dropdown list
float	Floating point (32 bits)	20	text line
double	Double precision (64 bits)	20	text line
int or integer	Integer (32 bits signed)	11	text line
long	Long (64 bits)	20	text line
boolean	Boolean: 0 or 1	1	text line
byte	Not used in GUI. Needed to store images and attachments	-	-

The UML field type mappings to MMBase database types, gui types and sizes can be changed by editing a property file in the conf directory. This file is called: **uml2mmbase-fieldtypes.properties**. For example the key "html.gui.size" means that for the uml type "html" will be mapped in the edit wizards on the gui size of 1024. An example can be found in the following code fragment:

```

# Define the mappings of the fieldtypes to the gui sizes.
html.gui.size      =1024
float.gui.size     =20
double.gui.size    =20
string.gui.size    =40
  
```

For the article class two string attributes have been defined which will show up in the builder file as a gui name and type, editor positions and db name and type.

#### Gui name and type for the title field.

```
<gui>
  <guiname xml:lang="nl">title</guiname>
  <guiname xml:lang="en">title</guiname>
  <guitype>string</guitype>
</gui>
```

#### Editor positions for the title field.

Because the title field was set to public, the list and search positions are set to 4 which means they will be shown as a search or list fields in the 4th position.

```
<!-- editor related -->
<editor>
  <positions>
    <!-- position in the input area of the editor -->
    <input>4</input>
    <!-- position in the list area of the editor -->
    <list>4</list>
    <!-- position in the search area of the editor -->
    <search>4</search>
  </positions>
</editor>
```

#### Editor positions for the author field.

Because the author field was set to private, the list and search positions are -1. This means that this field won't show up as a search or list field.

```
<editor>
  <positions>
    <!-- position in the input area of the editor -->
    <input>5</input>
    <!-- position in the list area of the editor -->
    <list>-1</list>
    <!-- position in the search area of the editor -->
    <search>-1</search>
  </positions>
</editor>
```

Since the author of an article was set to private, in the edit wizards list the author won't show up.

### Figure 4. A list of articles in the edit wizard for an article

**Typical fields for an article.**

---

**title**

**author**

**starttime**    at  :

**endtime**    at  :

**status**

---

cancel - save

---

#### Database name and type for the title field.

The database column name will be the attribute name converted to lowercase.

```
<db>
  <!-- name of the field in the database -->
  <name xml:lang="nl">title</name>
  <!-- MMBase datatype and demands on it -->
  <type key="false" notnull="true" size="50"
        state="persistent">STRING</type>
</db>
```

### 5.2.4. Stereotypes and tagged value

If a field in the object is mandatory, the <<required>> stereotype can be added to the field. This will result in a notnull="true" in the builder definition. In the edit wizard the input field will be mandatory and the object can't be save before the field is set.

In the sample model in the article object a <<required>> stereotype has been set on the title:

**Figure 5. The stereotypes on the article title attribute**

Properties | Style | To Do Items | Documentation | Java Source | Constraints | Tagged Values

Attribute

Name: title

Multiplicity: 1

Stereotype: required

Owner: article

Visibility:  public  protected  package  private

Modifiers:  static  final  transient  volatile

Type: string

Initial Value:

Accessors methods: none

So in the builder for article for the title field the notnull attribute of the db type element is set to "true":

```
<type key="false" notnull="true" size="50"
state="persistent">STRING</type>
```

The size of a field can be specified using the tagged values minsize and maxsize. If these sizes are not defined the default sizes in the previous table will be used. For the title field of article the minsize has been set to 3 and the maxsize has been set to 50. As shown in the previous code snippet, the size attribute has been set to 50.

**Figure 6. The tagged values on the article title attribute**

Tag	Value
minsize	3
maxsize	50
documentation	<p>Title of the article.</p>

The minsize and maxsize settings will also show up in the edit wizard xml files. If a new article is created with the edit wizards, the title field will be red because it is a required field and the save button will be inactive until all required fields have been filled.

**Figure 7. An empty edit wizard form for an article**

**Typical fields for an article.**

---

**title**

**author**

**starttime**    at  :

**endtime**    at  :

**status**

---

cancel - save

---

If a maxsize is specified for a string field the gui type in the edit wizard will be a text line if the maxsize  $\leq$  40. Otherwise a text area will be generated for the GUI. So for the title a text area is generated in the edit wizard. For the author field no maxsize has been specified so the default gui size of 40 will be used which results in a text line in the edit wizard. In the next picture a filled edit wizard form for article is shown where the title is a text area and the author is a text line. The save button has been activated.

**Figure 8. A filled edit wizard form for an article**

**Typical fields for an article.**

**title**

**author**

**starttime**    at  :

**endtime**    at  :

**status**

cancel - save

If a HTML field is specified, a required stereotype will be ignored due to a bug in the editwizards. If required is set, it is impossible to save an object since the validation on html fields will fail and the save button will never be activated.

The tagged value prompt can be used to overrule the prompt name in front of an input field in the edit wizards. By default the field name will be used as the prompt name. If a tagged value prompt is specified on a field, that value will be used as the prompt. It is also possible to specify tagged values on a class. Currently it is possible to specify the tagged values "title" and "subtitle". These values will appear in the Edit Wizards. For example for the teaser wizard in the UML model the title tagged value has been set as "Teaser" and the subtitle has been set to "A catchy phrase". This is shown in the following figure:



**Figure 9. Tagged values title and subtitle on teaser class**

Tag		Value
title		Teaser
subtitle		A catchy phrase

These tagged values on the uml class will appear in the edit\_teaser.xml file as 2 xml elements. These values will be displayed in the Edit Wizards, which can be seen in the following figure. In that figure is also shown that the prompt for the status field has been changed into: "Workflow status".

**Figure 10. Edit Wizard of teaser with set Tagged values title and subtitle**



Teaser		Teaser to make the visitor interested in a page.	
A catchy phrase			
starttime	27	november	2003 at 13 : 24
endtime	27	november	2003 at 13 : 24
Workflow status	select...		
teaserText	1st teaser's teaserText		
<b>page (*)</b>			
Zoek page	any age	title bevat	<input type="text"/> 
Nieuw			
cancel - save			

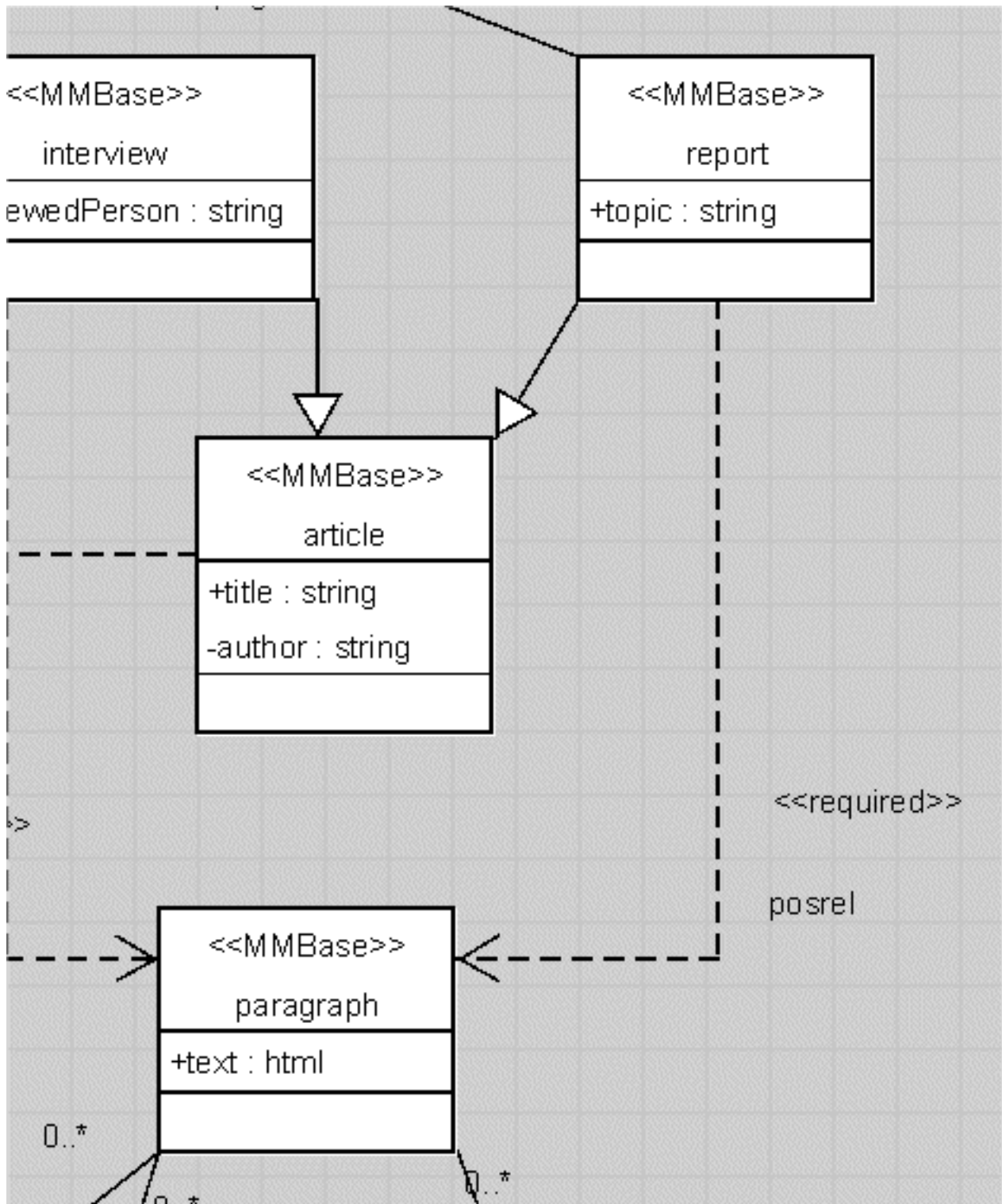
## 5.3. MMBase Relations

### 5.3.1. Dependencies

A dependency between two objects defines a 0..1 to 0..N relation. The name of a dependency is used as the type of the relation. If the name isn't specified, the name of the dependency will be changed into "related".

Relations that are specified by a dependency are normally implemented by the insrel builder. If the name of the dependency is set to "posrel", the builder that is used to implement the relation is posrel. The posrel differs from the insrel builder in a way that also the order of the related objects can be specified. In the next figure an example of a dependency relation is shown between a report and a paragraph. This dependency has the name posrel because a report has an ordered set of paragraphs. The dependency also has the stereotype <<required>> because at least one paragraph has to be associated with a report.

**Figure 11. A required posrel dependency relation between a report and a paragraph**



A dependency can be given the stereotype `<<required>>`. If this stereotype is specified, the relation will be implemented as a 0..1 to 1..N relation. In the editwizards this will result in the requirement that at least one relation to the related object will have to be created before the object itself can be saved. In the generated XML files this will show up as a `minoccurs="1"` for the list element.

```
<list role="posrel" destination="paragraph" minoccurs="1" maxoccurs="*"
  orderby="field[@name='pos']" ordertype="number">
```

In the gui of the edit wizard the relation is displayed in red to indicate that a relation is required.

**Figure 12. The required relation in the edit wizard**

A Report about a topic.

**topic**

**title**

**author**

**starttime**    at  :

**endtime**    at  :

**status**

**page**  
 Zoek page      
 Nieuw

**paragraph (\*)**  
 Zoek paragraph      
 Nieuw

cancel - save

When using custom relations, it is also possible to define an option list in the relation. In the generated edit wizard it is possible to add both the relation fields and the fields of the related object. See for example the "menu\_relation" between menu and menu\_item. In the wizard the title field of the related menu\_item can be edited, and the relation fields: "menu\_type" and "option\_list\_test" can be also edited.



**Figure 13.** The relations and related object can be edited, including option lists.

MENU WITH SEVERAL ITEMS.

**title** mijn menu


---

**menu\_item**



Wijzigen...  

**title** mijn menu item

**menu\_type** relatie 1


**option\_list\_test** selecteer... 




---


Wijzigen...  

**title** mijn menu item

**menu\_type** relatie 2

**option\_list\_test** selecteer... 

**zoek menu\_item** new  title bevat   

Nieuw 

selecteer...  
 selecteer...  
 new  
 staged  
 published  
 archived  
 hidden

### 5.3.2. Associations

An association between two objects defines a relation which can be specified in more detail than a dependency.

Relations that are specified by an association are normally implemented by the insrel builder. If the name of the association is set to "posrel", the builder that is used to implement the relation is posrel. The posrel differs from the insrel builder in a way that also the order of the related objects can be specified.

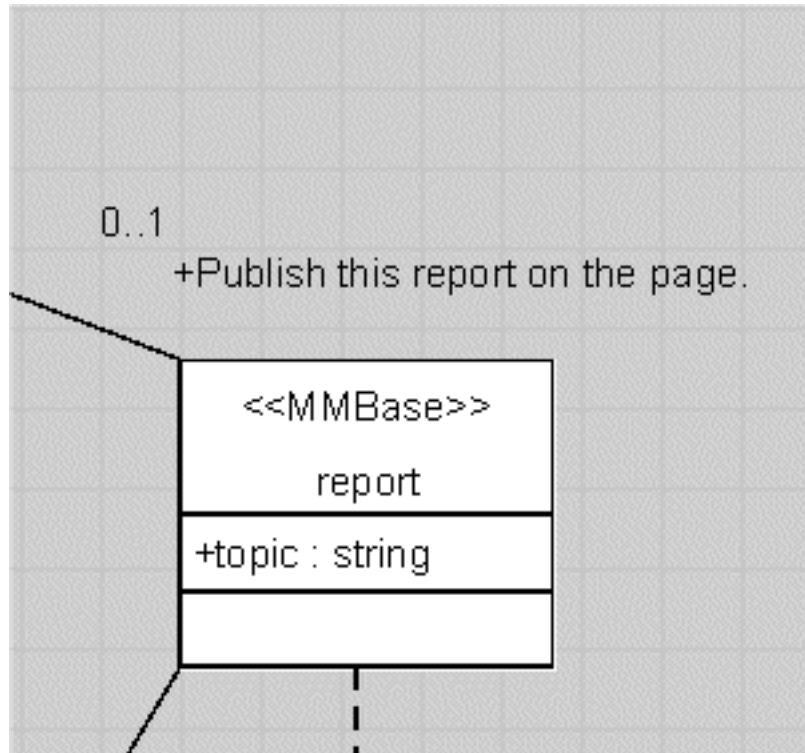
The upper and lower cardinality of the association can be specified on both the source as the target of the association. If the lower cardinality is set to a value above 0, the editwizards requires that at least one reference to an object is defined before the object itself can be saved.

The way an relation is navigable can be specified by the navigable properties of the association. If the navigable property is turned of, the relation to the object will not be displayed in the editwizard.

An association can be given two role names. When a role name is specified, this text will be displayed in the editwizard of the object. If no rolename is specified, the name of the object referred to will be displayed.

In the following example the association between a page and a report is shown. The rolename has been specified as "Publish this report on the page". The cardinality of the association is 0..1 which means that the relation is not required and at most 1 relation can be created.

**Figure 14. Association relation between a page and a report**



In the edit wizard of page the relation to the report is shown in the next figure.

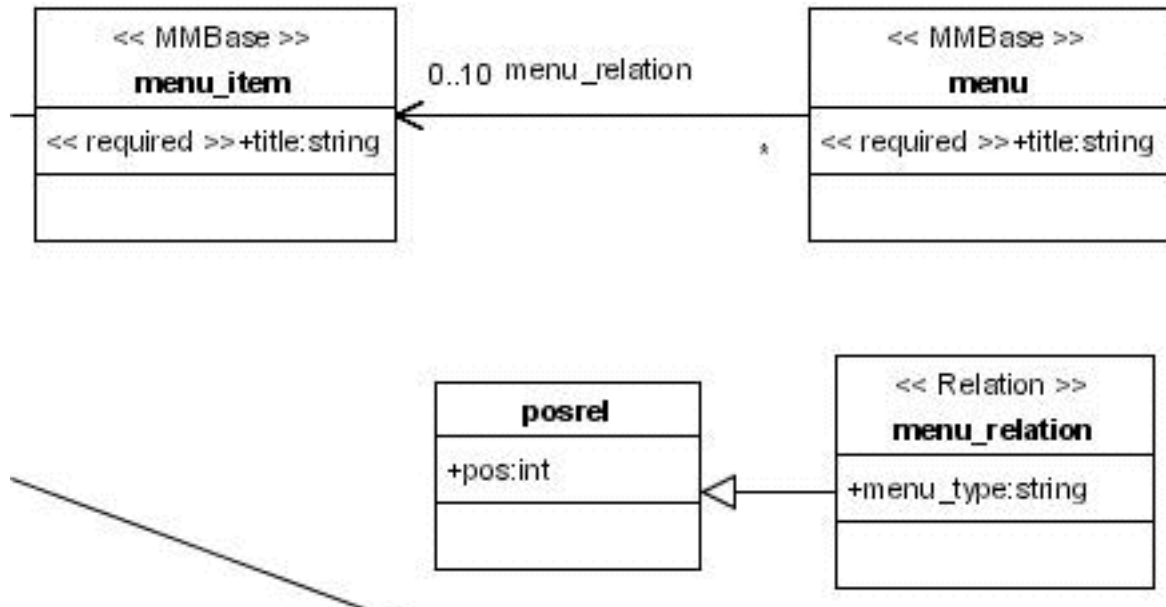
**Figure 15. Association relation between a page and a report in the edit wizard**

### 5.3.3. Custom relation types

The previous discussion about relations, assumes existing relation types are used like `insrel` and `posrel`. It is also possible to model a custom relation type in UML. To define a custom relation type the `<<Relation>>` stereotype has to be used on a class diagram. The relation is assumed to inherit from the `insrel` builder, unless another object is explicitly used. In the following example a relation type: `menu_relation` has been model with a `menu_type` field which extends from the `posrel` relation. After the custom relation type has been defined, it can be used as the name of dependencies or associations to use this relation type. In the example the `menu_relation` type has been used as the relation type on the association between `menu` and `menu_item`.

UML2MMBase will generate the builder for the custom relation type.

**Figure 16. Custom MMBase relation type: `menu_relation`**



The fields of the custom relation can be edited in the Edit Wizards. In the example it is possible to edit the pos field and menu type field in the relation between a menu and menu\_item.

**Figure 17. Custom MMBase relation fields in the edit wizard**

**menu\_item**

<b>title</b>	3rd menu_item's title
<b>pos</b>	<input type="text" value="1"/>
<b>menu_type</b>	<input type="text"/>

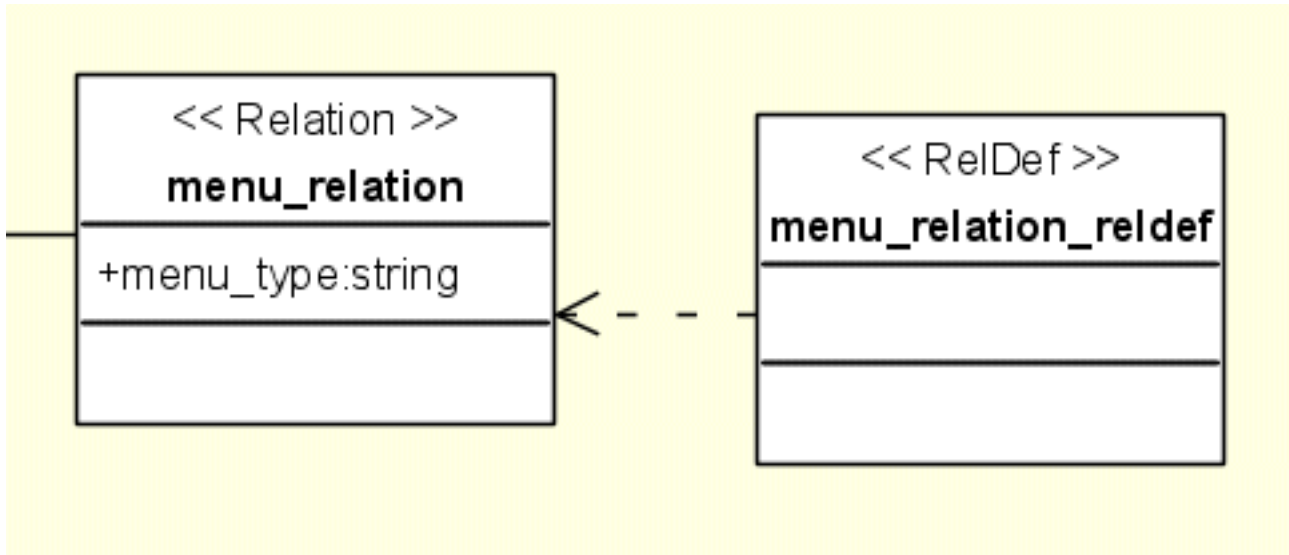
[Change...](#)

Zoek menu\_item

Nieuw

It is also possible to create a relation definition for a custom relation. This way the same builder will be used for the specified relation. The relation can be modeled using a object class with the <<RelDef>> stereotype. The class should use a dependency relation to the custom relation type. In the following figure a reldef is modeled with an object class with name "menu\_relation\_reldef". The class has a dependency relation to the menu\_relation object class. This will result in using the menu\_relation builder for the menu\_relation\_reldef relation.

**Figure 18. Reldef using a custom MMBase relation**



## 5.4. Option Lists

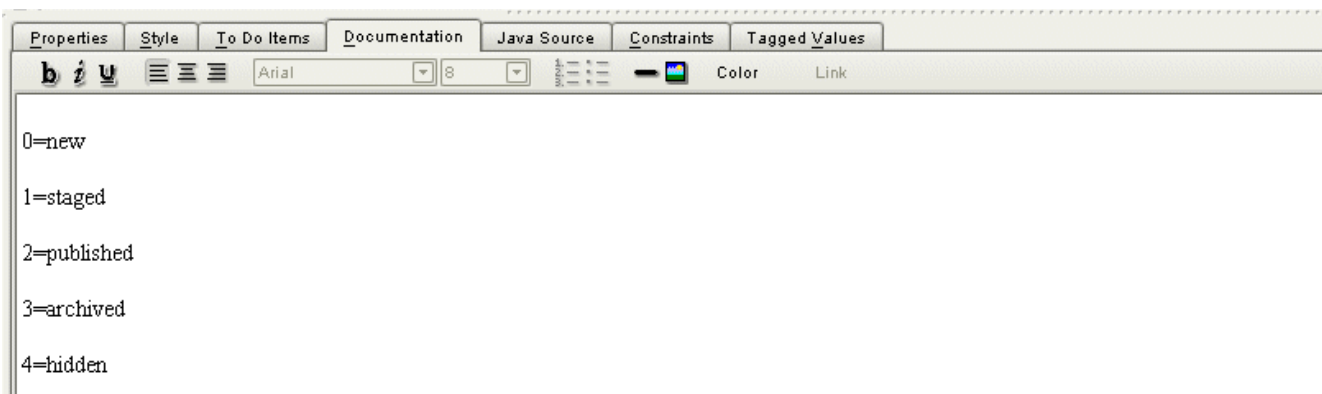
In MMBase it is possible to use an OptionList in the editors to allow for selecting values from a dropdown list. In UML2MMBase it is possible to define OptionLists as a Class in the class diagram. The class that defines the optionlist should be given the stereotype `<<OptionList>>`. The name of the class defines the type of the optionlist in MMBase.

**Figure 19. Option List definition: publication\_state**



An option list in MMBase contains key/value pairs that can be specified. The key is the string value that will be stored in the database. The value string will be displayed in the editwizard. The key/value pairs are stored in the default property notation (e.g. 0=new) in the documentation of the UML class.

**Figure 20. Option List specification in the documentation of the class**



After the optionlist is declared in the model, it can be used by a class. This is done by setting the declared optionlist as the type of an attribute. The field defining an optionlist is stored in the builder as a string field of 1024 bytes. If a different database size of the field is requested, then the size can be changed by setting a "maxsize" tagged value on the attribute.

**Figure 21. Option List referenced from the publishable interface**



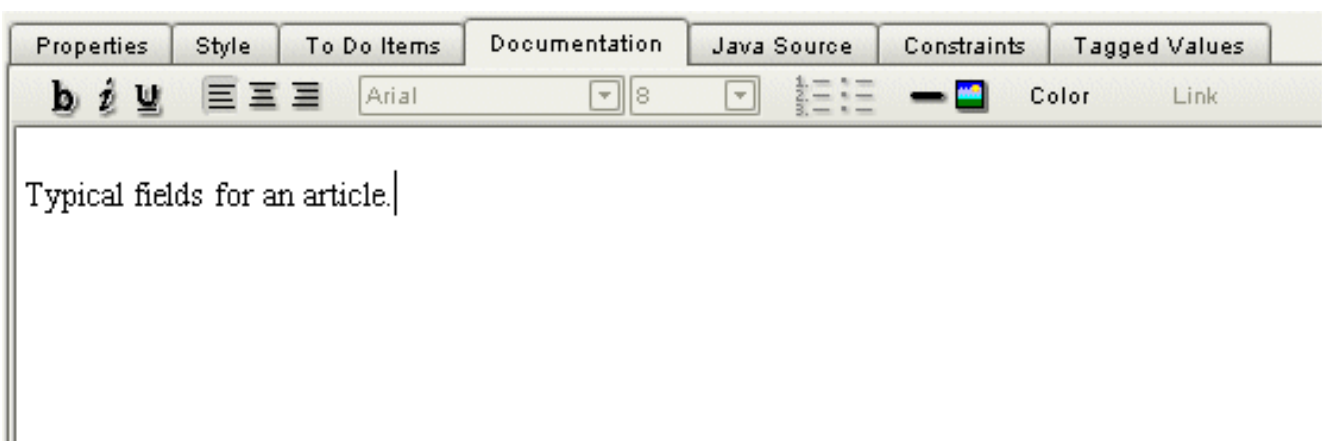
When the UML2MMBase application is executed an editwizard containing the optionlist is created. In this example the status field of publishable objects is shown as a drop down list.

**Figure 22. Option List in the edit wizard of teaser**

## 5.5. Documentation

UML has default support for specifying documentation. If documentation is added to a class diagram, or to an attribute this will be used in the codegeneration.

**Figure 23. The class documentation of an article MMBase object**



If documentation is specified for a class it will be used to set the description of the builder. This will also show up in the editwizard at the top of the edit screen.

Documentation that will be added to an attribute, will be used as a tooltip on the field names on the edit wizards.

## 5.6. Inheritance

In the UML model inheritance can be used to share common attributes between builders. For example an article object al-



ways has an author and a title. Specializations of an article might be a report or an interview. By modeling these objects using inheritance the generated builder file will use the extends attribute to specify that the builder inherits the article object. This will look as follows in the builder XML definition of report.xml:

```
<builder extends="article" maintainer="mmbase.org"
name="report" version="0">
```

## 5.7. Interfaces

To model common behaviour for different object in the cloud, UML interfaces can be used. The interfaces specify recurring functionality, for example objects that are publishable on a web site. To model this behaviour a "publishable" interface can be specified with operations that define the behaviour. A startTime, endTime and status might be specified as operations. The defined operations will be mapped to attributes in the builder definition of the implementing objects. For example the article object implements the publishable interface which implicitly means the article object has the attributes startTime, endTime and status. These fields will be added to the article builder definition.

## 5.8. MMBase Packages

To model applications or modules that are always required for a MMBase application, UML packages can be used. For example, the Resource application of MMBase with images, urls etc. are almost always needed by an MMBase application. A dependent package should have a unique name in the model and should be marked with the <<MMBasePackage>> stereotype.

In a post generation step customized operations can be performed for every dependent package in the model with a MMBasePackage stereotype. For every package a mapping is made from a property file to a Java class that implements the MMBasePackageGenerator interface. The interface contains 3 methods that will be executed in the following order: init, process, destroy. The mapping of the package name to a java class can be found in the property file:

**uml2mmbase-package-generator.properties.** In the following example the Resources package is mapped to the DefaultMMBasePackageGenerator class. This is actually the class that will be used if no mapping was specified for a package name with the <<MMBasePackage>> stereotype.

To use your custom PackageGenerator you can add a jar file with your implementations in de lib dir (this will make the class available in the classpath for uml2mmbase) and add the mapping in the property file.

```
#
# This property file can be used to define custom package generators
# for a UML Package in the model.
#
# Define the mappings of the package names with the
# MMBasePackage stereotype to a java class
# the implements the MMBasePackageGenerator interface.
#
#The MMBase resources package contains the basic resources and UML profile.
Resources=com.finalist.mmbase.modules.DefaultMMBasePackageGenerator
```

## 6. Generating the MMBase artifacts

### 6.1. Configuration

Before generating the UML2MMBase artifacts, some properties are relevant to inspect. They are set in the `mmbase.properties` file.

First the `mmbase` version can be specified for which we do the code generation. Currently `mmbase 1.6` and `1.7` are supported. Default `1.7` is selected. Also the output directory where the generated code should be stored can be specified. Default the generated directory is selected. To specify the name of the generated application, the `mmbase.application` property can be set which defaults to `APPLICATION`. In the edit wizards it is also possible to edit fields of a related object directly. This behaviour can be configured using the `edit.related.object` setting. Default this option is set to `false`.

`mmbase.version=1.7`

`generated.dir=generated`

`mmbase.application=APPLICATION`

`edit.related.objects=false`

### 6.2. MMBase builder definitions

For all objects with the `<<MMBase>>` or `<<Relation>>` stereotype a builder XML file will be generated in the `generated/config/applications/APPLICATION/builders` directory.

### 6.3. Application configuration file

A MMBase application is configured with an application configuration file. UML2MMBase will generate the `APPLICATION.XML` file in the `generated/WEB-INF/config/`

### 6.4. Edit Wizards

For all objects with the `<<MMBase>>` stereotype an edit wizard will be generated in the `generated/editors/config/` directory. For every edit wizard a subdirectory will be create for the object which will contain some XML files needed by the edit wizards. Also some basic wizards for images, attachments and urls will be copied to this directory.

For the article object the "article" directory will be created in which the following files will be generated.

`article.xml`

Main XML configuration that is used to include the other XML files.

`create_article.xml`

Creating a new article.

`delete_article.xml`

Delete an article.

`load_article.xml`

Loads the fields of the object and related objects.

`edit_article.xml`

Define the fields that are editable in the edit wizard.

`search_...xml`

If an object has relations the search functions are generated in the target object directory. For example: the `posrel` relation from `report` to `paragraph` in the `paragraph` subdirectory the `search_report_posrel_paragraph.xml` file is generated.

If option lists have been defined, they will be generated in the `generated/editors/config/` directory. In the sample model an option list has been defined for the publication state. This will result in the following file:

`publication_state.xml`

XML file that contains the key value pairs of the options.

### 6.5. JSP menu and search pages to access the edit wizards

For all objects with the `<<MMBase>>` stereotype a link in a JSP menu page is created from which the edit wizard can be ac-

cessed. The sources will be generated in the generated/editors/ directory.

index.jsp

Main jsp page from which a frame is built. This page also makes sure a user logs in for the edit wizards by setting the rank to administrator. So the default account admin/admin2k can be used.

leftmenu.jsp

jsp with all links to the edit wizards.

search.jsp

jsp that will generate a search screen for the edit wizards.

logout.jsp

Logout user

empty.html

Empty screen.

css/

Directory with the stylesheets for this edit wizard.

xsl/

Directory with extra XSL sheets for restricting the size of the fields in a display list and of readonly data.

## 6.6. HTML and XML docbook report of the object cloud

UML2MMBase will generate 2 reports with information about the object cloud. The following files will be generated in the "generated" directory:

report.html

HTML page with an overview of the object cloud with links to builders and editors

report.xml

Simple XML report in docbook format of the object cloud.

## Frequently Asked Questions

Q: Why are no builders and editors generated for an object defined in my model?

A: If no code is generated for a specific object, probably the <<MMBase>> stereotype has not been set on the object. The stereotype is needed to trigger the code generation for builders and edit wizards.

Q: I removed classes from my model but they still appear in the generated output?

A: For UML2MMBase the model that is exported to XMI is used for generating the output sources. This means all classes and attributes that have not been removed from the model but only from the view, will still be present in the XMI file. So if a model element is removed, make sure it is removed from the model and not only from the view!

Q: When starting uml2mmbase.bat I get the error: 'java' is not recognized as an internal or external command, operable program or batch file.

A: Probably java JDK1.4 has not been installed or the java executable can not be found in your path. You can download java at: <http://java.sun.com/getjava> to install the latest version. To check if java has been successfully installed type from a command prompt: "java -version". This should result in a message like: `java version "1.4.1_02"`

# References

- AndroMDA: <http://www.andromda.org>
- Jakarta Ant: <http://ant.apache.org>
- Velocity: <http://jakarta.apache.org/velocity/index.html>
- PoseidonUML: <http://www.gentleware.com/products/index.php3>
- MagicDraw: <http://www.magicdraw.com> [<http://www.magicdraw.com/>]
- Together Control Center: <http://www.togethersoft.com>
- UML1.4 specification: <http://www.omg.org/docs/formal/01-09-67.pdf>
- MOF specification: <http://www.omg.org/docs/formal/00-04-03.pdf>
- JMI Implementations: <http://java.sun.com/products/jmi/implementations.html>
- JMI specification: <http://java.sun.com/products/jmi/download.html>