

Database Generator



Esito products are copyrighted and all rights are reserved by Esito. This document is also copyrighted and all rights are reserved. This document may not, in whole or in part, be copied, photocopied, translated, or reduced to any electronic medium or machine-readable form without prior consent, in writing, from Esito. The information in this document is subject to change without notice, and Esito assumes no responsibility for any errors that may appear in this document. The references in this document to specific platforms supported are subject to change. Genova, Sysdul, Systemator are trademarks or service marks of Esito.

Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. IBM and Rational Rose are registered trademarks of IBM Corporation. GWT and GWT-based marks are trademarks or registered trademarks of Google Corporation in the U.S. and other countries. Windows, ASP and Visual Basic are registered trademarks of Microsoft Corporation. Enterprise Architect is a registered trademark of Sparx Systems. Any other trademarks or trade names contained herein are the property of their respective owners.

Copyright 2005-2009 © All rights reserved

Esito AS
Postboks 191
N-1325 Lysaker
Norway

DATABASE GENERATOR – TOC

1	Introduction	1
2	Starting generation.....	2
3	Setup for Database Generator.....	4
3.1	Data Access Generator setup	4
3.2	DBMS Schema Generator setup	8
4	Iteration structure and substitution variables.....	10
4.1	Top level node type.....	10
4.2	Package	10
4.3	Enumerator.....	11
4.4	EnumValue.....	11
4.5	Table.....	11
4.6	Column.....	13
4.7	Key.....	14
4.8	Group	15
4.9	Procedure	16
4.10	Parameter	16
4.11	Relation.....	16
4.12	Member	18
5	Generating target DBMS scripts.....	19
5.1	Target DBMS templates	19
5.1.1	DB Init script	19
5.1.2	DB Schema script	20
5.1.3	DB Drop script.....	21
6	Generating data access code	22
6.1	Generating the data access source files.....	22
6.1.1	Data access templates	22
6.1.2	Mapping source file	23
6.2	Generating Hibernate mapping files	23
6.3	Generating EJB mappings	24
6.4	Generating Genova Data Objects mapping	25
6.5	Generating Sysdul mapping	25

1 Introduction

This manual describes Genova's database generator. Database Generator uses the database mappings created with Database Designer. Database Generator is used to generate database schema scripts to help in creating databases, and it is used to generate data access code for different targets.

The following chapters contain information about the elements in and use of Database Generator:

- Chapter 1 ["Introduction" on page 1](#) is the chapter you are now reading.
- Chapter 2 ["Starting generation" on page 2](#) explains how to start generating DBMS scripts and data access code.
- Chapter 3 ["Setup for Database Generator" on page 4](#) describes the setup parameters used by Database Generator.
- Chapter 4 ["Iteration structure and substitution variables" on page 10](#) describes the iteration structure and substitution variables available when writing templates for Database Generator.
- Chapter 5 ["Generating target DBMS scripts" on page 19](#) describes how to generate the SQL scripts that will create the target DBMS and, if necessary, the data access scripts for the run-time application.
- Chapter 6 ["Generating data access code" on page 22](#) describes the access and mapping code functionality including Java and XML code generation for Hibernate, for various Enterprise Java Beans targets, as well as OO-to-DB mapping code for the access targets.

2 Starting generation

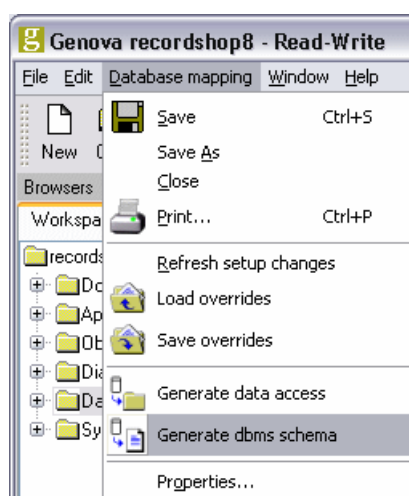
Database Generator operates on an opened database mapping created with Database Designer. See [section 4.1 on page 13](#) in Database Designer manual on how to open a database mapping. For an open database mapping two different generation tools are available from the *Database mapping* menu and from the *Database Mapping Tools* tool bar. See [section 4.2.1 on page 15](#) in the Database Designer manual about the *Database mapping* menu and [section 4.2.2 on page 16](#) about the *Database Mapping Tools* tool bar.

The two generation tools are:

Data access	Generates data access code for the database mapping and its data access target.
Schema	Generates DBMS schema files for the database. Which files will be generated is target dependent, but will normally include initialization and drop scripts in addition to a script for creating tables and indexes.

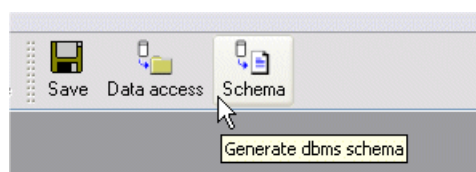
To generate a dbms schema for a database mapping:

- ◆ **Make sure a database mapping window is open and active.**
- ◆ **Select the menu option *Database mapping/Generate dbms schema*.**

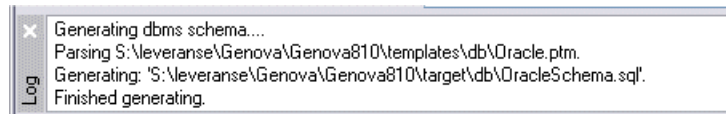


OR

- ◆ **Click the *Schema* button in the database mapping tool bar.**



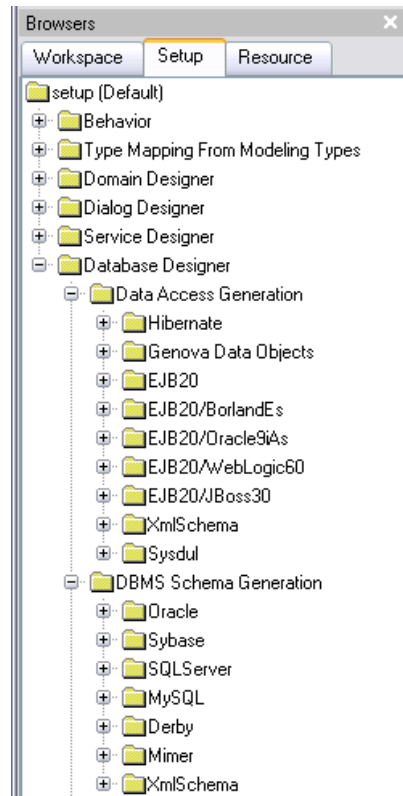
In the log window, Genova displays messages showing the progress of the schema generation.



```
Generating dbms schema...
Parsing S:\neveranse\Genova\Genova810\templates\db\Oracle.ptm.
Generating: 'S:\neveranse\Genova\Genova810\target\db\OracleSchema.sql'.
Finished generating.
```

3 Setup for Database Generator

This chapter gives a general description of the setup parameters used by Database Generator. Database Generator operates on two different targets, the *Data Access* target and the *DBMS Schema* target. Each of these target types has its own set of setup parameters:



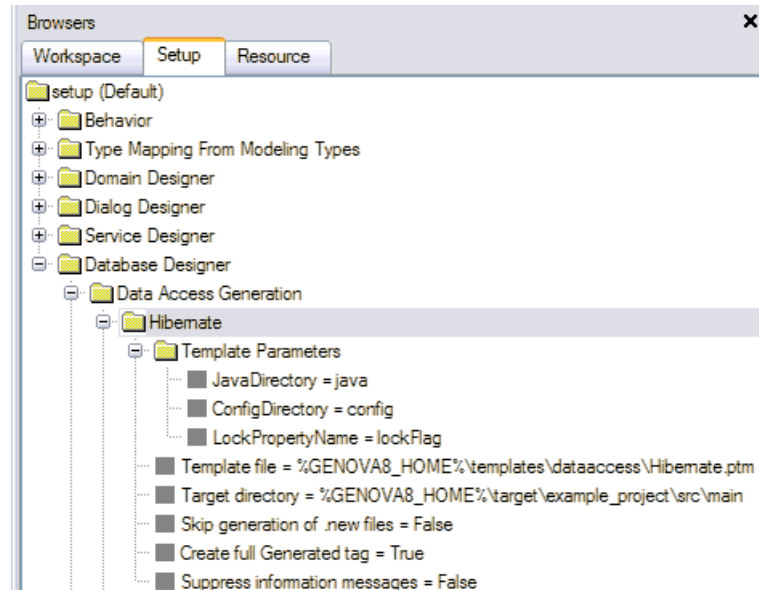
The *Data Access* targets are used to generate data access code and/or mappings between systems and the DBMS system, i.e an OO-language like Java and RDBMS system. The *Data Access* target for a database mapping can be changed and one database mapping can be used to generate access code for different targets for the same database.

The *DBMS Schema* targets are used to generate SQL scripts which can be used to generate the wanted database. The *DBMS Schema* target for a database mapping is specified when the database mapping is created and it can not be changed.

3.1 Data Access Generator setup

The setup defined for the *Data Access Generation* targets are only used by the Database Generator. None of these settings have any influence on the

creation of the database mapping itself. The next figure shows the parameters available with Hibernate as an example:



Template Parameters: This category can hold parameters used in the templates for the target. See [chapter 9 on page 13](#) in the Template Parser manual on how such parameters are used.

Template file: This parameter specifies the name of the target's main template file. The template files all have the extension ".ptm". Template files are provided for Hibernate, GDO (Genova Data Objects), EJB2 (with support for various application servers), Sysdul and XML Schema.

For more information on data access template files, see [section 6.1.1 on page 22](#).

Target directory: This parameter specifies the default directory where all generated target *Data Access* files are stored.

Skip generation of .new files: In the templates there are three different ways to specify an output file, see [chapter 15 on page 23](#) in the *Template Parser* manual. By default the *NEWFILE* directive will make the template parser generate a file with extension .new if the file already exists. Setting this setup parameter to *True* will make the template skip the creation of the .new file. No file will be created if the file already exists.

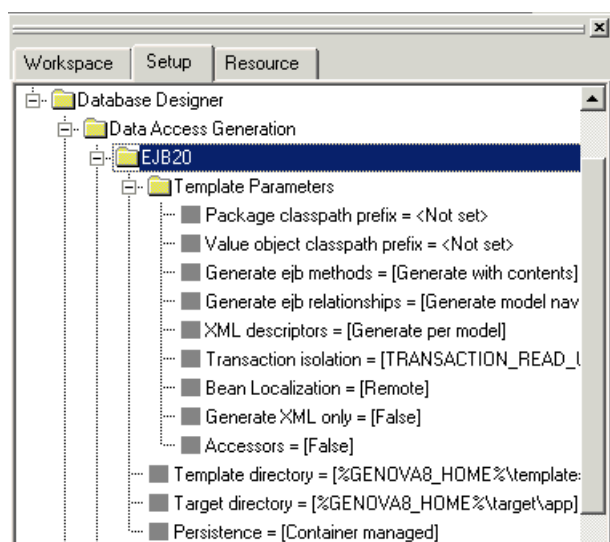
Create full Generated tag: When set to *True*, the substitution variable *GeneratedWith* (see [chapter 9 on page 13](#) in the *Template Parser* manual) will deliver an identification of program version, user doing the generation and a timestamp. This will produce differences to identical results generated by different users or at a different time. When set to *False*, this replacement string will only contain the text *Generated with 'Genova'* and two re-

sults will not show a differences because of a difference in timestamp or user doing the generation.

Suppress information messages: When set to *True*, only a minimum set of messages are written to the log window during the generation. Both information messages produced by Template Parser itself and messages included in the templates are suppressed, while warning messages are not suppressed.

Currently supported targets are Hibernate, EJB application servers (WebLogic, Oracle9iAS, BorlandES and JBoss), Genova Data Objects, Sysdul and XML Schema. By selecting the Hibernate target, mapping files for domain classes and Hibernate configuration files are generated. By selecting the EJB targets, Java files (bean class, home interface, remote interface) and deployable XML, descriptors are generated. By selecting the Sysdul target, the *SysdulMap.c* mapping file is generated, which must be compiled and linked with the application.

The Data Access Generation category of the Setup database is where all access generator parameters are defined. .



Certain parameters are available only for EJB data access generators:

Persistence: The parameter is applicable for the EJB targets only, indicating container persistence type supported. For EJB20 and EJB20/WebLogic60 both container and bean managed persistence is supported. With container managed persistence, the **cmp.ptm* templates are used, and with bean managed persistence, the **.bmp.ptm* templates are used.

Package classpath prefix: The parameter indicates the directory where the generated java files for the beans (bean class, home interface, remote interface, generated from Genova) reside. These are referred to in the generated code. Note that this holds for one (model) package at a time – i.e.

when generating for several packages at the same time, the setting will be invalid for all but one.

Value object classpath prefix: The parameter indicates the directory where the generated java files for the domain classes (generated from Genova or the UML modeling tool) reside.

Generate ejb methods: The parameter is applicable for EJB application servers only, and can be set to *Don't generate* (default), *Generate empty* or *Generate with contents*, indicating how to generate the possible ejb*-methods in the bean class. If *Generate with contents* is set, a print statement is generated stating which bean class and method is invoked.

Generate ejb relationships: The parameter indicates whether or not to generate code for container managed relationships (CMRs). If selected, model navigability is followed. Note that for the EJB2.0/WebLogic6 target, in order to use CMRs beans must reside in the same physical *.jar* file (and the same XML descriptors). Thus, this parameter is relevant only when generating on per package or per model basis.

XML descriptors: The parameter specifies the type of XML descriptors that can be generated. XML descriptors can be generated on a per bean basis, per package basis and per model basis.

Transaction isolation: The parameter is applicable for EJB application servers only, and indicates the isolation level for the generated beans. Possible settings are RDBMS independent, except for the Oracle-dependent TRANSACTION_READ_COMMITTED_FOR_UPDATE.

Bean Localization: The parameter is applicable for EJB application servers only, and indicates whether to generate remote (default) or local beans.

Generate XML only: If the parameter is set to *True* (default *False*), XML files only are generated – not Java files. May be useful when generating per model or package, and not wanting every *.java* file (time-consuming).

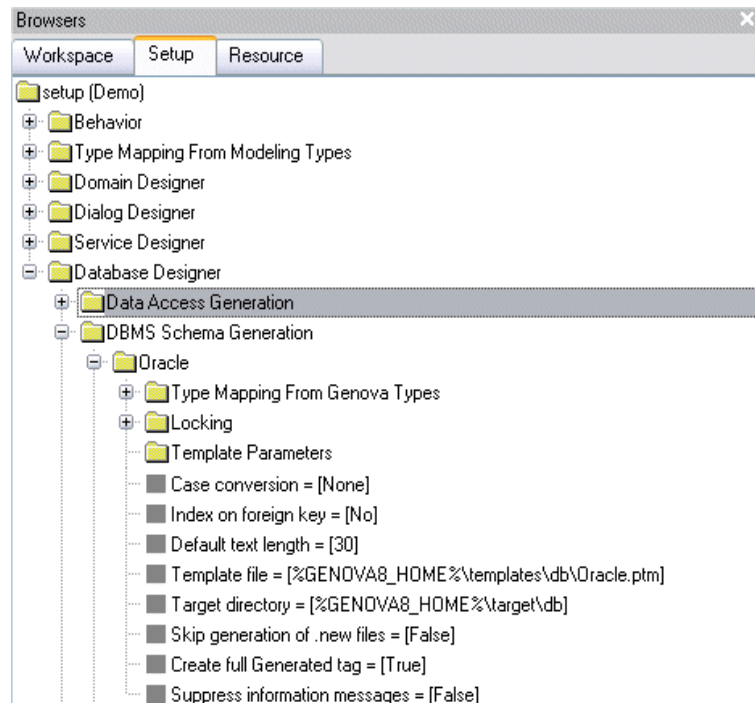
Accessors: If private attributes are used in the model, this parameter should be set to *True* (the default is *False*) in order to generate parameter access by set/get methods from the model VO classes.

If public attributes are used, this parameter should be set to *False* (default), causing direct parameter access in the generated code.

Be aware that VO class code generated from the UML modeling tool must be changed to include the necessary set/get methods.

3.2 DBMS Schema Generator setup

The *DBMS Schema* targets all have one category and four setup parameters that influence the result of the schema generation. The next figure shows the parameters available with Oracle as an example:



Note: Categories and parameters not described below are not used in the generation of the *DBMS Schema* targets, they are used when a database mapping is created. See [section 6.7 on page 34](#) in the Database Designer manual for further description of these parameters.

Template Parameters: This category can hold parameters used in the templates for the target. See [chapter 9 on page 13](#) in the Template Parser manual on how such parameters are used. None of the current *DBMS Schema* targets uses template parameters.

Template file: This parameter specifies the name of the target's main template. The template files all have the extension ".ptm". Template files are provided for MySQL, Oracle, SQLServer, Sybase, Derby and Mimer.

Target directory: This parameter specifies the default directory where all generated target DBMS files are stored. This includes the *initialization*, *schema* and *drop* files.

Skip generation of .new files: In the templates there are three different ways to specify an output file, see [chapter 15 on page 23](#) in the *Template Parser* manual. By default the *NEWFILE* directive will make the template parser generate a file with extension .new if the file already exists. Setting this setup parameter to *True* will make the template skip the creation of the .new file. No file will be created if the file already exists.

Create full Generated tag: When set to *True*, the substitution variable *GeneratedWith* (see [chapter 9 on page 13](#) in the *Template Parser* manual) will deliver an identification of program version, user doing the generation and a timestamp. This will produce differences to identical results generated by different users or at a different time. When set to *False*, this replacement string will only contain the text *Generated with 'Genova'* and two results will not show a differences because of a difference in timestamp or user doing the generation.

Suppress information messages: When set to *True*, only a minimum set of information messages are written to the log window during the generation. Both information messages produced by Template Parser itself and messages included in the templates are suppressed, while warning messages are not suppressed.

4 Iteration structure and substitution variables

The Database Generator is template based and uses the Template Parser when reading its templates, see the Template Parser manual for a general description of the parser. This chapter gives a description of the iteration structures available when writing templates for the Database Generator, and for each node type in the structure the available substitution variables are described.

4.1 Top level node type

The outermost level represents the database mapping.

Substitution variables available at the top level:

Name	Description
ModelName	The name of the database mapping
ModelDescription	The description text for the database mapping
LogicalName	The full path and name for the UML model file
DataAccessTarget	The name of the data access target
DatabaseSchema-Target	The name of the database schema target
LockFlag	Is optimistic locking on (conditional)
LockColLen	The number of characters in the lock column name
LockColName	The name of the lock column
DefaultTextLength	The default length for text columns

Iterations at the top level:

Name	Description
Package	Packages in the database mapping
Table	Tables in the database mapping
Enumerator	Enumerators in the class model

4.2 Package

This represents a Package in the database mapping.

Substitution variables available at the Package level:

Name	Description
PackageName	The full name of the package

Iterations at the Package level:

Name	Description
Table	Tables in the package

4.3 Enumerator

This represents an Enumerator in the class model.

Substitution variables available at the Enumerator level:

Name	Description
EnumName	The class name of the enumerator
PackageName	The full package name for the enumerator class

Iterations at the Enumerator level:

Name	Description
EnumValue	A value definition for an Enumerator

4.4 EnumValue

This represents a value definition for an Enumerator.

Substitution variables available at the EnumValue level:

Name	Description
EnumValueName	The name of the enum value
EnumValueOrdinalNumber	The ordinal number of the enum value
EnumValueText	The title of the enum value

No iterations are available at the EnumValue level.

4.5 Table

This represents a Table in the Package or database mapping, depending on the parent iteration level.

Substitution variables available at the Table level:

Name	Description
TableName	The table name
LogicalName	The class name of the table

Name	Description
TableDescription	The description text for the table
TableIsInDB	Is there at least one included column in the table (conditional)
HasSchema	Is the "schema name" property set (conditional)
SchemaName	The value of the "schema name" property
TableIsSubclass	Does this table inherit from a superclass (conditional)
SuperClassName	The name of the super class
SuperClassPackage	The package of the super class
SuperTableName	The name of the table containing the super class
InheritInfo	A number indicating the chosen inheritance strategy
HasRelations	Are there any relations from this table to another (conditional)
HasKey	Is the primary key defined (conditional)
HasDiscriminatorColumn	Does this table contain an anchester column (conditional)
SuperTableHasDiscriminatorColumn	Does the supertable to this table contain an anchester column (conditional)
IsSpezializedAttributes	True if table is from a class inheriting from a superclass and the generalization is marked as SpezializedAttributes
IsOneTable	True if table is from a class inheriting from a superclass and the generalization is marked as OneTable
IsInheritedAttributes	True if table is from a class inheriting from a superclass and the generalization is marked as InheritedAttributes
IsInheritedAttributesDuplicateObjects	True if table is from a class inheriting from a superclass and the generalization is marked as InheritedAttributesDuplicateObjects

Iterations at the Table level:

Name	Description
Column	A column in the table
Key	The primary key

Name	Description
Group	A group in the table
Procedure	A procedure in the table
Relation	A relation between this table and another

4.6 Column

This represents a Column in a Table.

Substitution variables available at the Column level:

Name	Description
ColumnName	The column name
LogicalName	The attribute name for the column
ColumnDescription	The description text for the column
ColumnIndex	Is the "index" property set (conditional)
ColumnNotNull	Is the "not null" property set (conditional)
IsPartOfPK	Is the column primary key or part of the primary key (conditional)
ColumnPrimaryKey	Is the column the primary key (conditional)
ColumnForeignKey	Is the column a foreign key (conditional)
ColumnPKFKOrPartOfPK	Is the column (part of) the primary key or a foreign key (conditional)
IsLockColumn	Is this the optimistic lock column (conditional)
IsDiscriminatorColumn	Is this a generated column used to distinguish records for different classes contained in the same table. (conditional)
ColumnUnique	Is the "unique" property set (conditional)
Clustered	Is the "clustered" property set (conditional)
ColumnTypeName	The UML attribute type (without package)
ModelTypeIsJavaPrimitive	Is the UML attribute type a Java primitive (conditional)
GenovaType	The Genova data type for the attribute
ModelTypePackageName	The package part of the UML attribute type
DBType	The column database type
TypeLength	The "length" property
HasLength	Is the "length" property set (conditional)

Name	Description
IsInherited	Is the column inherited (conditional)
InheritedName	The column name without a super class prefix
TypePrecision	The "precision" property
HasPrecision	Is the "precision" property set (conditional)
TypeScale	The "scale" property
HasScale	Is the "scale" property set (conditional)
TypeDefault	If the "default value" property is set, return it (also conditional)
TypeConstraints	If the "dbms constraint" property is set, return it (also conditional)
FirstElement	Is this column the first in the table (conditional)
SingleGroupColumnRecursive	Is this column the single member of a group, recursively (conditional)
FirstInIndex	Is this column the first in a group (conditional)
RelationForeignKey	If this is a foreign key, get the foreign key name
RelationForeignKeyLogical	If this is a foreign key, get the foreign key model name
GroupMember	Is this column a member of any group (conditional)
RelationForeignRole	If this is a foreign key, get the foreign table role name
RelationForeignTable	If this is a foreign key, get the foreign table name
RelationForeignTableLogical	If this is a foreign key, get the foreign table model name
RelationForeignTablePackage	If this is a foreign key, get the package name of the foreign table

No iterations are available at the Column level.

4.7 Key

This represents a Primary Key for a table.

Substitution variables available at the Key level:

Name	Description
KeyName	The name of the primary key column or group
LogicalName	The model name of the primary key attribute or group
KeyDescription	The description text for the key

Name	Description
IsInherited	Is the primary key inherited from a super class (conditional)
Nonclustered	Is the primary key "clustered" (conditional)
IsGroup	Is the primary key a group (conditional)

No iterations are available at the Key level.

4.8 Group

This represents a Group in a Table.

Substitution variables available at the Group level:

Name	Description
GroupName	The name of the group
LogicalName	The UML model name of the group
GroupPrimaryKey	Is this group the primary key (conditional)
GroupUnique	Is the "unique" property set (conditional)
Clustered	Is the "clustered" property set (conditional)
GroupIndex	Is the "index" property set (conditional)
IsInherited	Is this group inherited from a super class (conditional)
RelationForeignKey	If this is a foreign key, get the foreign key name
RelationForeignTable	If this is a foreign key, get the foreign table name
RelationForeignTableLogical	If this is a foreign key, get the foreign table model name
RelationForeignTablePackage	If this is a foreign key, get the package name of the foreign table
GroupMember	Is this group a member of another group (conditional)
HasRelation	Is this a foreign key (conditional)
RelationRoleName	If this is a foreign key, get the foreign table role name
NoGroupMembers	Number of members in this group
FirstInIndex	Is this group the first part of another group (conditional)

Iterations at the Group level:

Name	Description
Member	A member in the group. Members are either of the type Column or Group.

4.9 Procedure

This represents a Procedure in a Table, i.e. a stored procedure.

Substitution variables available at the Procedure level:

Name	Description
ProcedureName	The procedure name
LogicalName	The UML model name of the procedure
ProcedureDescription	The description text for the procedure

Iterations at the Procedure level:

Name	Description
Parameter	A procedure parameter

4.10 Parameter

This represents a Parameter in a Procedure.

Substitution variables available at the Parameter level:

Name	Description
ParameterName	The parameter name
LogicalName	The UML model name of the parameter
ParameterType	The database type of the parameter

No iterations are available at the Parameter level.

4.11 Relation

This represents a Relation between two tables.

Note: Each association between two tables in the class model is represented as two relations in the database mapping. The two relations are identical except for the relation direction, and by iterating over relations in a table each association will appear twice. The relation direction is given by the substitution variable *RelationDirection*. The two tables involved in a rela-

tion is name *Foreign* and *Other*, and which is who is switched between the two relation instances.

Substitution variables available at the Relation level:

Name	Description
RelationName	The name of the relation
RelationRoleName	The role name at this tables end of the relation. If one of the tables is a subclass, the table name is added to avoid name conflicts.
RelationRoleNamePlain	As above, but without adding table names
RelationForeignKey	The foreign key name of the relation
RelationForeignKeyLogical	The UML model name of the foreign key
RelationForeignTable	The foreign table name
RelationForeignTableLogical	The UML model name of the foreign table
RelationOtherTableLogical	The UML model name of the other table
RelationOtherTablePK	The primary key of the other table
RelationOtherTablePackage	The package name of the other table
RelationForeignRole	The role name of the foreign table
SelfRelation	Is this a relation to the same table (conditional)
RelationCardinality	The cardinality, " <i>One</i> " or " <i>Many</i> "
IsMandatory	Is this a mandatory relation (conditional)
IsManyRelation	Is the cardinality " <i>Many</i> ", (conditional)
RelationForeignCardinality	The cardinality at the foreign table end, " <i>One</i> " or " <i>Many</i> "
ManyRelation	As <i>IsManyRelation</i> , but in the opposite direction
OneOneRelation	Is the cardinality <i>one-to-one</i> (conditional)
IsForeignMandatory	As <i>IsMandatory</i> , but in the opposite direction (conditional)
ColumnName	As <i>RelationForeignKey</i> , but in the opposite direction
ColumnLogicalName	As <i>RelationForeignKeyLogical</i> , but in the opposite direction
RelationDirection	The direction, either " <i>0</i> " or " <i>1</i> ".
IsUpRelation	Is this an upwards relation, (conditional)
IsMemberInAssoc	Is this the owner side, (conditional)
IsOwnerInAssoc	Is this the member side, (conditional)

Name	Description
SamePackage	Are both tables located in the same package, (conditional)
IsNavigable	Is this relation navigable (conditional)
IsOppositeNavigable	As above, but in the opposite direction
IsPartOfPK	Is this relation part of the primary key (conditional)
IsNavigableOrPartOfPK	Is this relation navigable or part of the primary key (conditional)
HasGroupFK	Is the foreign key a group (conditional)
IsInherited	Is the relation inherited (conditional)
IsForeignInherited	Is the other end of the relation inherited (conditional)

Iterations at the Relation level:

Name	Description
Member	The foreign key of a relation

4.12 Member

This represents a Group member, either a Column or another Group.

Substitution variables available at the Member level:

Name	Description
MemberName	The member name
LogicalName	The UML model name of the member
IsGroup	Is this member a group (conditional)

Iterations at the Member level:

Name	Description
Column	A column member of a group
Group	A group member of a group

5 Generating target DBMS scripts

A database mapping has two different targets, the *DBMS Schema* target and the *Data Access* target.

The *DBMS Schema* target is set when the database mapping is created and it may not be changed for a given mapping. With one database mapping one may only generate schema scripts for one specified database system.

Once all the modeling work has been finished in the Database Designer, one may generate the script files that can be used to create the target DBMS.

5.1 Target DBMS templates

For most target DBMSs three different SQL script files are generated: *Initialization* script, *Schema* script and *Drop* script.

Note: Within the generated script files, there are comments indicating how to execute the generated scripts against the target DBMS. For example:

```
/* Run this DBA script against Sybase as follows: */  
/* isql -Usa -P <SybaseDBA.sql */.
```

5.1.1 DB Init script

The initialization scripts generate target DBMS SQL statements that create files used to store the database, e.g. devices or table spaces, and a database owner account with password.

The generated initialization script usually has the name *<target-DBMS>DBA.sql*. In the setup database's Database Mapping/DBMS Schema Generation section, the parameter *Target directory* specifies where the generated files are stored, see "[Target directory](#)" on page 8. The output script must be executed against the specific target DBMS to create the actual database.

The initialization script needs manual modification before being run. File sizes and storage locations need to be specified. Default values exist but are enclosed in "<>" that need to be manually removed.

```

SQLServerDBA.ptm - Notepad
File Edit Format Help
@template
%FILE%SQLServerDBA.sql
/* Run this DBA script against SQLServer as follows: */
/* isql -Usa -P < SQLServerDBA.sql */

/* The CREATE DATABASE and DISK INIT command will generate necessary */
/* devices with calculated sizes. You may want to tailor file names, */
/* prefix with directories or change device numbers (depending on other */
/* databases in your sybase environment). That is, the <>'s must be remov
/* The following commands create a database of 16 MB with 8 MB on a data
/* 4 MB on an index device and 4 MB on a log device. */

disk init
name = "%ID:ModelName%.data",
physname = <"../%ID:ModelName%.data">,
vdevno = <2>, size = <4096>
go

disk init
name = "%ID:ModelName%.dind",
physname = <"../%ID:ModelName%.dind">,
vdevno = <3>, size = <2048>
go

disk init
name = "%ID:ModelName%.dlog",
physname = <"../%ID:ModelName%.dlog">,
vdevno = <4>, size = <2048>
go

/* If sizes below are 0 (given in MB) then the sizes above are too small
/* (given in 2K pages). */
/* If so, adjust all sizes to a minimum of 2 MB / 1024 2K pages. */

```

The template replacement string `%ID:ModelName%` is replaced by the *DB name* property of the Database Designer's Model object. This value can be manually modified in the generated initialization script.

5.1.2 DB Schema script

The schema scripts generate target DBMS SQL CREATE TABLE, CREATE INDEX and ALTER TABLE statements according to the current UML and Database Designer models.

The generated schema script file usually has the name `<targetDBMS>Schema.sql`. Note that the default name for the generated file can be overridden in the generation process. In the setup database's Database Mapping/DBMS Schema Generation section, the parameter *Target directory* specifies where the generated files are stored see "[Target directory](#)" on page 8. The output script must be executed against the specific target DBMS to create the actual database schema.

The schema scripts are generated using the template file given in the setup database. The template file is structured into sections as follows:

Section	Description
@reserved	The reserved section specifies all the reserved words for the target DBMS. These will have to be taken into consideration when naming objects in the modeling tool and the Database Designer. If target DBMS reserved words are attributed to model objects, the user will be notified in the script generation process and prompted to assign a new object name.
@types	The type section defines all the target DBMS data types as well as the properties supported for the various types.
@template	The template section specifies how each of the database objects and their constraints, properties and indexes will be declared in the schema script. First the name of the generated SQL script file is specified. Then the CREATE TABLE statements including basic column definitions will be generated. The tables and columns will be created in the order in which they appear in the Database Designer (with the exception of foreign keys). Finally, the ALTER TABLE and CREATE INDEX statements including constraints and indexes will be generated.

5.1.3 DB Drop script

When a target database has to be re-generated as a result of model changes, it is necessary first to delete the previously created database before re-creating the target database schema. Dropping a database is automated with the drop script that generates target DBMS SQL ALTER TABLE, DROP CONSTRAINTS, DROP INDEX and DROP TABLE statements to drop a previously existing database schema.

The generated drop script usually has the name *<targetDBMS>Drop.sql*. In the setup database's Database Mapping/DBMS Schema Generation section, the parameter *Target directory* indicates where the generated files are stored, see "[Target directory](#)" on page 8. The output script must be executed against the specific target DBMS to drop an existing database schema.

6 Generating data access code

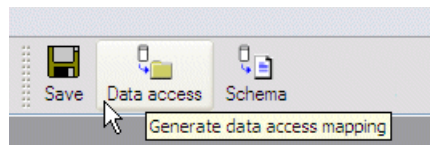
The data access target is set when the database mapping is created, but it may be changed. This target is used when Data Access is generated. Changing this target makes it possible to generate access code for different data access targets for the same database mapping.

6.1 Generating the data access source files

When accessing RDBMS data structures from object oriented programs, it is necessary to generate an OO-to-DB mapping used with the run-time application in addition to the target DBMS scripts.

To generate the target application data access source file from the Database Designer:

- ◆ **Open the desired mapping in the Database Designer and click the *Data access* button.**



The same command exists in the Database mapping menu.

6.1.1 Data access templates

Following are the application template files used to generate the OO-to-DB mappings for the various target applications:

Target application	Template files
Hibernate	Hibernate.ptm (includes the helper template files Hibernate-Group.ptm, HibernateIdElement.ptm, HibernateTypeElement.ptm, HibernateRelation.ptm and HibernateUtil.ptm)
Enterprise JavaBeans	ejb20cmp.ptm (container managed persistence) ejbbmp.ptm (bean managed persistence)
WebLogic	ejb20WebLogic60cmp.ptm (container managed persistence) ejb20WebLogic60bmp.ptm (bean managed persistence)
Oracle9iAS	ejb20Oracle9iAS.ptm
BorlandES	ejb20BorlandES.ptm
JBoss	ejb20jboss30.ptm
XMLSchema	XMLSchema.ptm
Sysdul	SysdulMap.ptm
Genova Data Objects	gdo.ptm

6.1.2 Mapping source file

Following are the files generated for the various available target applications:

Target application platform	Generated files
Hibernate	<ClassName>.hbm.xml hibernate.cfg.xml hibernate.properties
Sysdul	SysdulMap.c
Enterprise JavaBeans	<ClassName>bean\Meta-inf\ejb-jar.xml <ClassName>bean\<ClassName>Home.java <ClassName>bean\<ClassName>Local/Remote.java ^a <ClassName>bean\<ClassName>bean.java.
WebLogic	As for Enterprise JavaBeans, with the following additions: <ClassName>bean\Meta-inf\weblogic-ejb-jar.xml <ClassName>bean\Meta-inf\weblogic-cmp-rdbms-jar.xml
BorlandES	As for Enterprise JavaBeans, with the following addition: <ClassName>bean\Meta-inf\ejb-borland.xml
JBoss	As for Enterprise JavaBeans, with the following addition: <ClassName>bean\Meta-inf\jbosscomp-jdbc.xml
Oracle9iAS	As for Enterprise JavaBeans, with the following addition: <ClassName>bean\Meta-inf\orion-ejb-jar.xml
Genova Data Object	GDO.java, GDOBasis.java GDODBObject.java GDOPrintWriter.java LockFlag.java
XML Schema	XMLSchema.xsd XML-documents\<ClassName>.xml

a. File name depends on whether local beans or remote beans are generated (specified in *Bean localization* in Genova setup).

6.2 Generating Hibernate mapping files

When the data access target is *Hibernate*, a separate mapping file named <ClassName>.hbm.xml is generated for each domain class in the database mapping. The mapping files are generated to the same package directory as the domain class, using the *Target directory*/*<MappingDirectory>* as a top level directory. The *<MappingDirectory>* is replaced with the value of the *MappingDirectory* template parameter.

Two Hibernate configuration files are generated to *Target directory*/*<ConfigDirectory>*. The *<ConfigDirectory>* is replaced with the value of the *ConfigDirectory* template parameter.

- *hibernate.cfg.xml*, a XML file which contains references to all the domain class mapping files for the database mapping.
- *hibernate.properties*, a skeleton Java property file with JDBC and EJB connection properties for the database.

Setup database template parameters for the Hibernate target (see [section 3.1 on page 4](#)):

MappingDirectory	The subdirectory under the target directory for the generated Hibernate mapping files for the domain classes. Setting this parameter to the empty string will direct the output to the target directory.
ConfigDirectory	The subdirectory under the target directory for the generated Hibernate configuration files. Setting this parameter to the empty string will direct the output to the target directory.
LockPropertyName	The name of the property in the Java domain classes which holds the version value for optimistic locking in the database

6.3 Generating EJB mappings

When the data access target is either one of *EJB2.0*, *EJB2.0/Oracle9iAS*, *EJB2.0/WebLogic*, *EJB2.0/BorlandES* or *EJB2.0/JBoss30*, the *Mapping* button generates the bean skeletons consisting of EJB class (*<ClassName>Bean.java*), home interface (*<ClassName>Home.java*), local/remote interface (*<ClassName>Remote.java*¹) and Primary Key class (*<ClassName>PK.java*) according to J2EE standards. In addition, necessary XML descriptors (*ejb-jar.xml* and other application server specific XML descriptor files) are generated.

There are three different choices for the generation of XML descriptors for the generated EJB class files. The XML descriptors can be generated on a per class basis/per package basis or on a per model basis from the UML model. Besides this, in case of Oracle9iAS and Borland Enterprise Server, batch files are also generated for automatically compiling and/or deploying the generated EJBS. Clearly the error free running of these batch files would require the installation of required Java class libraries.

Also, a descriptor *application.xml* listing all involved *.jar* files is generated on the Mapping target top level directory. Depending on the setting of the

1. If local beans are generated (specified in Bean localization in Genova setup) the file is named *<ClassName>.java*.

XML Descriptors parameter, the generated directory structure differs somewhat:

- When generating on a *per bean basis*, for each class / table in the mapping a directory is generated named `<classname>bean` containing the three *.java* files and a subdirectory META-INF containing the XML descriptors (*ejb-jar.xml* for target EJB2.0 – for target EJB2.0/WebLogic6.0, two more in addition). Note that when beans need to access other beans via associations (CMR fields), they must reside locally in the same *.jar* and *.xml* file. This requires generating on a per package or model basis.
- When generating on a *per package basis* (the default), a separate directory is generated per package. Under this directory, a common META-INF directory is generated containing common XML files for the package. In addition, a subdirectory for each bean is generated as above, but without separate META-INF directories.
- When generating on a *per model basis*, separate subdirectories for each bean is generated as on a per bean basis – but a common META-INF subdirectory for the whole model is generated on the same level.

6.4 Generating Genova Data Objects mapping

When the data access target is *Genova Data Objects*, a set of Java files is generated: *GDODDBObject.java*, *Lockflag.java*, *GDOBasis.java*, *GDOPrintWriter.java*, *DomainGenerator.java* and *GDO.java*. These files contain the runtime mapping information for the OO-to-DB mapping for the GDO target.

6.5 Generating Sysdul mapping

When the data access target is *Sysdul*, the generated mapping file *Sysdul-Map.c* must be compiled and linked with the Sysdul application.

