

Domain Generator

= | e

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

DOMAIN GENERATOR – TOC

1	Introduction	1
2	Starting Domain Generator.....	2
3	Domain Generator Setup.....	5
4	Iteration structure and substitution variables.....	7
4.1	User defined tagged values	7
4.2	Top level node type.....	8
4.3	Class.....	8
4.4	Interface	9
4.5	Attribute	9
4.6	Enumerator.....	10
4.7	EnumValue.....	11
4.8	Group	11
4.9	Member	11
4.10	Method	12
4.11	Parameter	12
4.12	Association.....	13
4.13	Generalization	13
4.14	Realization	14
4.15	Converter.....	14
4.16	Domain.....	14
5	Domain implementations.....	16
5.1	Java.....	16
5.1.1	Compiling and running	16
5.1.2	Template parameters.....	17
5.1.3	Code Generation	20
5.1.4	Additional documentation	20

1 Introduction

This manual describes Genova's Domain Generator.

Domain Generator generates code for the domain model. The code generated is based on the Domain Model together with the specification given in the Domain Designer setup for the wanted target. The only target available at the moment is Java.

The Java code generated by Domain Generator is intended to be used together with code generated by Client Generator with Java/JFC as target and code generated with Service Generator with Java as target. But the generated code may also be used from manually written code.

Domain Generator is template based and uses Template Parser as parser, see the Template Parser manual for a general description of the parser.

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

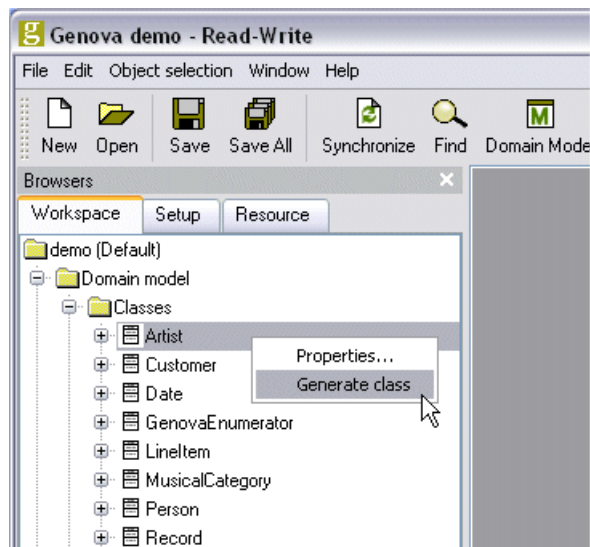
- Chapter 1 ["Introduction" on page 1](#) is the chapter you are now reading.
- Chapter 2 ["Starting Domain Generator" on page 2](#) explains how to start generating domain code.
- Chapter 3 ["Domain Generator Setup" on page 5](#) describes the setup parameters used by Domain Generator.
- Chapter 4 ["Iteration structure and substitution variables" on page 7](#) describes the iteration structure and substitution variables available when writing templates for Domain Generator.
- Chapter 5 ["Domain implementations" on page 16](#) gives a short description of the different targets available and explains the *Template parameters* for each target.

2 Starting Domain Generator

Code can be generated for a single class, a single interface or a single enumeration from the domain model or code can be generated for all classes, interfaces and enumerations contained in a selected set of packages from the UML model.

To generate code for a single class, interface or enumeration:

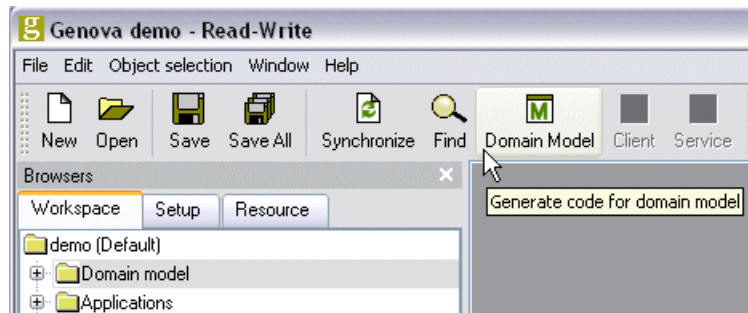
- ◆ **Right-click target class, interface or enumeration in the browser window.**
- ◆ **Select the menu item *Generate class*.**



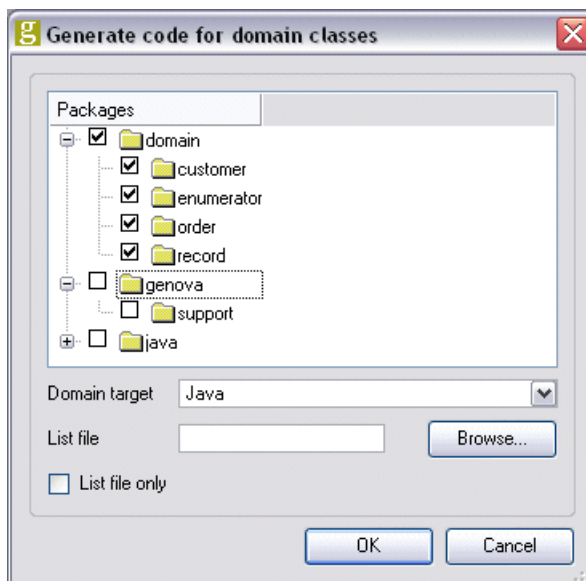
When generating for a single object Domain Generator generates for the *Default domain target* as specified in the Setup database under *Domain Designer*.

To generate code for all classes, interfaces and enumerations in a selection of packages:

- ◆ **Click the *Generate Domain Model* button in the tool bar.**



Genova will display the following dialog box showing the package hierarchy from the UML model:



- ◆ **Select the packages you wish to generate code for and click the OK button.**

Clicking the check box on one level will either check or uncheck that package and all packages below the package clicked.

Domain target: From the drop-down list, select the target for the generation of the domain model.

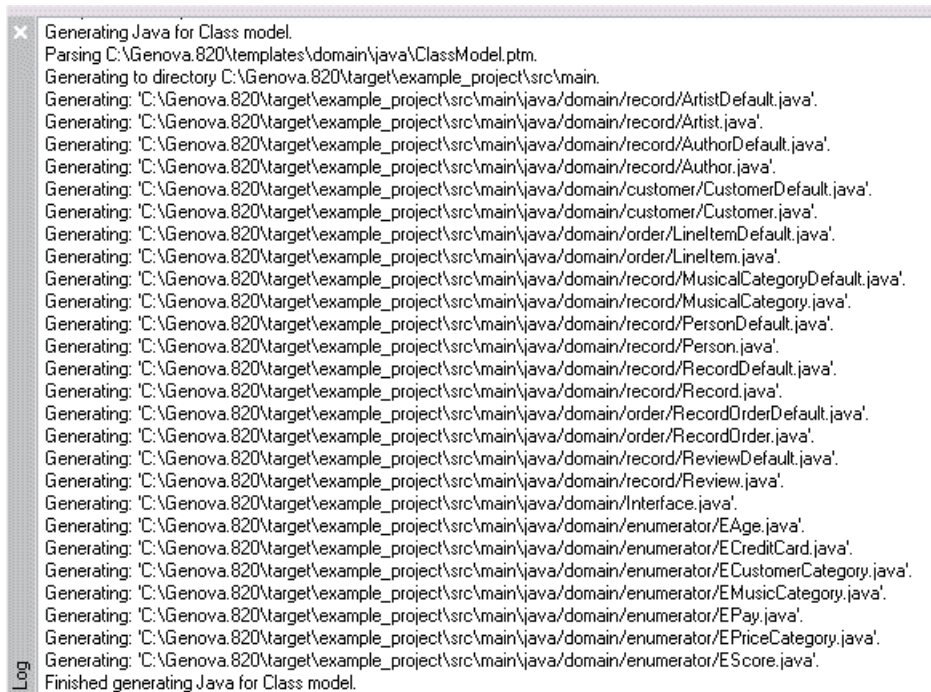
List file: To generate code for additional classes, interfaces or enumerations not in any of the selected packages, specify a list file containing names of additional classes to be included in the generation.

List file only: Check this option to ignore the selections in the package view and only generate code for the classes, interfaces and enumerations specified in the list file.

The packages shown are all packages containing a class, an interface or an enumeration. The package view used may either be the logical view or the component view of the model. By default Genova uses the logical view, but this can be changed to the component view by changing the value of a set-

ting in the setup database *Behavior* category: *Use Component View package when generating code.*

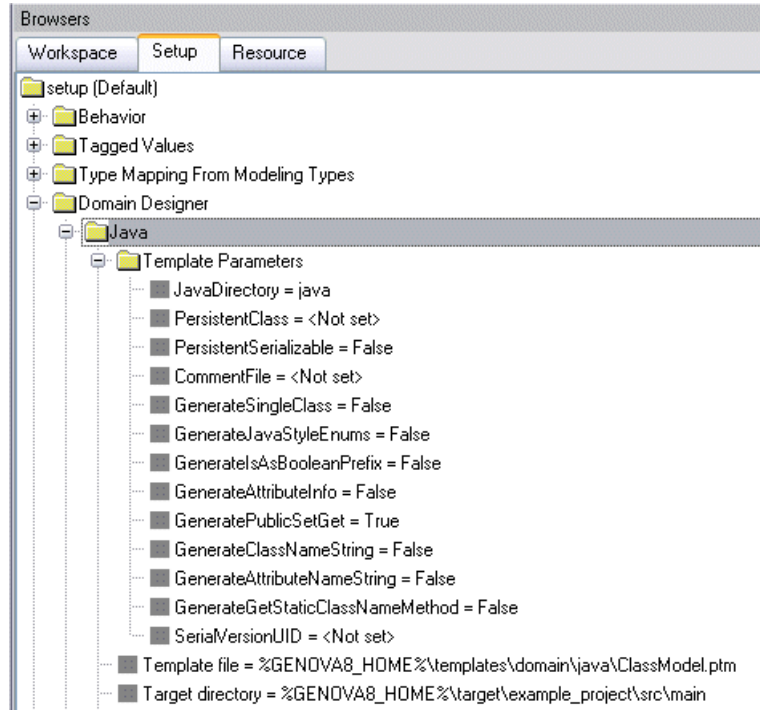
When the generation is run, Genova displays messages showing the progress of the domain model generation in the log window.



```
Generating Java for Class model.
Parsing C:\Genova.820\templates\domain\java\ClassModel.ptm.
Generating to directory C:\Genova.820\target\example_project\src\main.
Generating: 'C:\Genova.820\target\example_project\src\main\java\domain\record\ArtistDefault.java'.
Generating: 'C:\Genova.820\target\example_project\src\main\java\domain\record\Artist.java'.
Generating: 'C:\Genova.820\target\example_project\src\main\java\domain\record\AuthorDefault.java'.
Generating: 'C:\Genova.820\target\example_project\src\main\java\domain\record\Author.java'.
Generating: 'C:\Genova.820\target\example_project\src\main\java\domain\customer\CustomerDefault.java'.
Generating: 'C:\Genova.820\target\example_project\src\main\java\domain\customer\Customer.java'.
Generating: 'C:\Genova.820\target\example_project\src\main\java\domain\order\LineltemDefault.java'.
Generating: 'C:\Genova.820\target\example_project\src\main\java\domain\order\Lineltem.java'.
Generating: 'C:\Genova.820\target\example_project\src\main\java\domain\record\MusicalCategoryDefault.java'.
Generating: 'C:\Genova.820\target\example_project\src\main\java\domain\record\MusicalCategory.java'.
Generating: 'C:\Genova.820\target\example_project\src\main\java\domain\record\PersonDefault.java'.
Generating: 'C:\Genova.820\target\example_project\src\main\java\domain\record\Person.java'.
Generating: 'C:\Genova.820\target\example_project\src\main\java\domain\record\RecordDefault.java'.
Generating: 'C:\Genova.820\target\example_project\src\main\java\domain\record\Record.java'.
Generating: 'C:\Genova.820\target\example_project\src\main\java\domain\order\RecordOrderDefault.java'.
Generating: 'C:\Genova.820\target\example_project\src\main\java\domain\order\RecordOrder.java'.
Generating: 'C:\Genova.820\target\example_project\src\main\java\domain\record\ReviewDefault.java'.
Generating: 'C:\Genova.820\target\example_project\src\main\java\domain\record\Review.java'.
Generating: 'C:\Genova.820\target\example_project\src\main\java\domain\Interface.java'.
Generating: 'C:\Genova.820\target\example_project\src\main\java\domain\enumerator\EAge.java'.
Generating: 'C:\Genova.820\target\example_project\src\main\java\domain\enumerator\ECreditCard.java'.
Generating: 'C:\Genova.820\target\example_project\src\main\java\domain\enumerator\ECustomerCategory.java'.
Generating: 'C:\Genova.820\target\example_project\src\main\java\domain\enumerator\EMusicCategory.java'.
Generating: 'C:\Genova.820\target\example_project\src\main\java\domain\enumerator\EPay.java'.
Generating: 'C:\Genova.820\target\example_project\src\main\java\domain\enumerator\EPriceCategory.java'.
Generating: 'C:\Genova.820\target\example_project\src\main\java\domain\enumerator\EScore.java'.
Finished generating Java for Class model.
```

3 Domain Generator Setup

This chapter gives a general description of the setup parameters used by Domain Generator exemplified with the *Java* target:



Template Parameters: This category can hold parameters used in the templates for the target. See [chapter 9 on page 13](#) in 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". The default main template file read by Domain Generator is the file *ClassModel.ptm*.

For more information on template files, see [chapter 3 on page 5](#) in Template Parser manual.

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

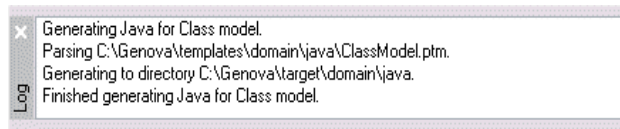
Skip generation of .new files: In the templates there are three different ways to specify an output file, see [chapter 15 on page 23](#) in the Template Parser manual. By default the *NEWFILE* directive will make the template parser generate a file with extension .new if the file already exists. Setting 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 results will not show a difference because of a difference in timestamp or user doing the generation.

Give warning for undefined parser variables: When set to *True*, Template Parser will log a warning message each time an undefined variable is used in the template files.

Suppress information messages: When set to *True*, only a minimum set of information messages are written to the log window during the generation. The next figure shows the log window for the same generation as above, when information messages are suppressed.



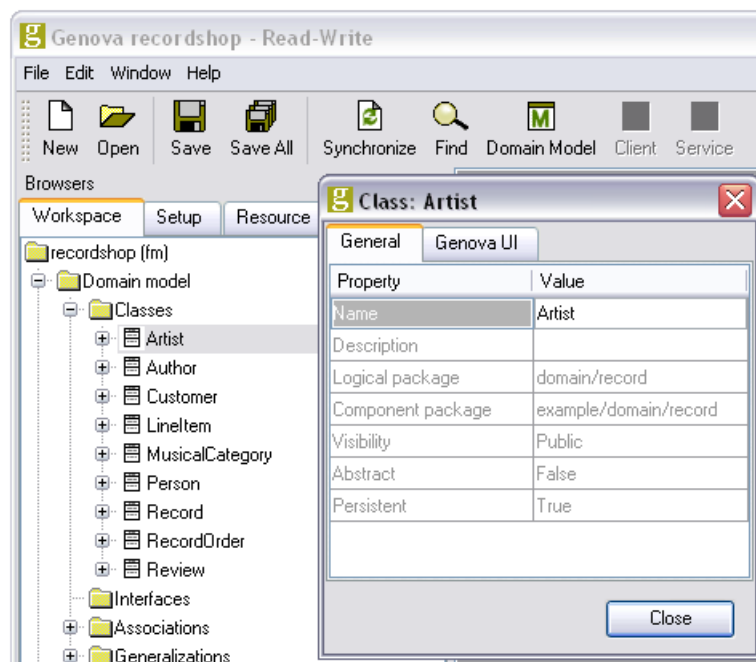
Both information messages produced by Template Parser itself and messages included in the templates are suppressed, while warning messages are not suppressed.

4 Iteration structure and substitution variables

Domain Generator is template based and uses the Template Parser to process its templates. See the [Template Parser](#) document for a general description of the parser.

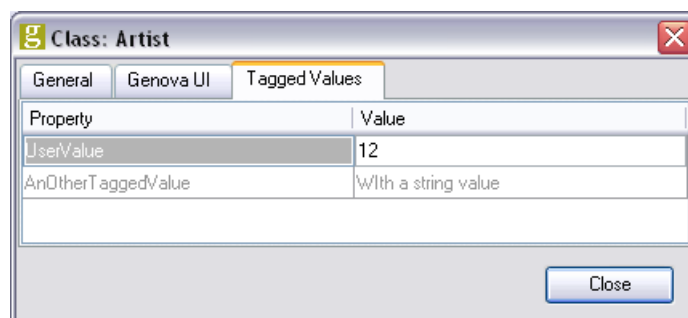
This chapter describes the iteration structures and substitution variables available at the various levels of a domain model.

In addition to the substitution variables specified below, all properties of the various domain entities are available for substitution as well. As an example, all classes offer *name*, *description*, and so forth in addition to those mentioned in [section 4.3](#) below.



4.1 User defined tagged values

When modeling with an UML 2.0 modeling tool, a user may define her own tagged values. Such values may be synchronized into the Genova workspace and they will show up as properties on the Tagged Values note page in model elements property dialog in Genova:



These properties are also available as substitution variables. The following element types in the domain model may have user defined tagged values:

- Associations
- Attributes
- Classes
- Converters
- Domains
- Enumerators
- Enumerator Values
- Generalizations
- Groups
- Interfaces
- Methods
- Realizations

See [section 8.5 on page 60](#) in the User Guide manual on how to import user defined tagged values into a Genova workspace.

4.2 Top level node type

The outermost level represents the domain model and has node type *Class-Model*.

No substitution variables are available at the top level.

Iterations provided at top level:

Name	Description
Class	All classes contained in packages included in the generation.
Interface	All interfaces contained in packages included in the generation.
Enumerator	All enumerations contained in packages included in the generation.
Domain	All domains in the model.
Converter	All convertes in the model.

4.3 Class

This node represents a class from the domain model.

Substitution variables available at Class level

Name	Description
PackageName	Name of the package the class belongs to. This variable will either use logical package or component package depending on the setting of the Behavior setup parameter <i>Use Component View package when generating code</i> .

Iterations available at Class level:

Name	Description
Attribute	All attributes defined in the class.
Enumerator	All enumerators used by attributes in the class.
Group	All groups defined in the class.
Method	All methods defined in the class.
Association	All association to or from this class.
Generalization	All generalizations to or from this class.
Realization	All interfaces implemented by this class.

4.4 Interface

This node represents an interface from the domain model.

Substitution variables available at Interface level:

Name	Description
PackageName	Name of the package the interface belongs to. This variable will either use logical package or component package depending on the setting of the Behavior setup parameter <i>Use Component View package when generating code</i> .

Iterations available at Interface level:

Name	Description
Attribute	All attributes defined in the interface.
Method	All methods defined in the interface.
Generalization	All generalizations to or from this interface.

4.5 Attribute

This node represents an attribute in a class.

In addition to the properties shown in the property dialog for an attribute, the properties of the converter are available via the prefix *Converter* I.e. *Converter.Name* gives the name of the converter connected to the attribute. See [section 4.15 on page 14](#) for further description of substitution variables for converters.

Substitution variables available at Attribute level, in addition to those seen as properties for an attribute, are:

Name	Description
StartRoleNamePath	Attributes iterated over as part of a group or an association can originate thru one or more associations from another class. This variable gives the first part of the name, while RestRoleNamePath gives the remaining part.
RestRoleNamePath	Used together with StartRoleNamePath.
MemberType	When iterated over as member of a group or an association, the variable <i>NodeType</i> returns "Member" while this variable returns "Attribute".
ModelTypeIsJavaPrimitive	True if the model data type is a Java primitive.
PackageName	Package of the attributes data type. If the data type is an enumeration, the variable is the package of the enumeration class. If the data type is the name of a class in the model, the variable is the package of that class. This variable will either use logical package or component package depending on the setting of the Behavior setup parameter <i>Use Component View package when generating code</i> .
TypePackageName	This variable is a synonym for PackageName.

Iterations provided at Attribute level:

Name	Description
EnumValue	All enumeration values. Empty except if the attribute is of type enumeration, in which case it loops through all enumeration values for that enumeration.

4.6 Enumerator

This node represents an enumeration.

Substitution variables available at Enumerator level:

Name	Description
PackageName	Name of the package the class belongs to. This variable will either use logical package or component package depending on the setting of the Behavior setup parameter <i>Use Component View package when generating code</i> .

Iterations available at Enumerator level:

Name	Description
EnumValues	All values defined for the enumeration.

4.7 EnumValue

This node represents an enumeration value.

No substitution variables or iterations are provided at this level, other than those seen as properties for an enumeration value.

4.8 Group

This node represents a group defined for a class

Substitution variables available at Group level:

Name	Description
MemberType	When iterated over as member of a group or an association, the variable <i>NodeType</i> returns "Member" while this variable returns "Attribute".

Iterations available at Group level:

Name	Description
Member	All members of the group. A group member is an Attribute, an Association or a Group.

4.9 Member

This node represents a member of a group. A group member is an attribute, an association or a group.

Substitution variables available at the Member level dependent on the member type.

Name	Description
MemberType	Type of member: Attribute, Association Group.

If the member type is an attribute, all variables defined for node type Attribute are available, if the member type is an association, all variables defined for node type Association are available and if the member type is a group, all variables defined for node type Group is available.

The iterations available are also the same as for each member type. So if the member is a group one can recursively iterate over this group's members.

4.10 Method

This node represents a method in a class or interface.

Substitution variables available at Method level:

Name	Description
ModelTypeIsJavaPrimitive	True if the methods return type is a java primitive type: <i>boolean, byte, short, int, long, char, float</i> or <i>double</i>
TypePackageName	Package of the methods return type. If the data type is an enumeration, the variable is the package of the enumeration class. If the data type is the name of a class in the model, the variable is the package of that class. This variable will either use logical package or component package depending on the setting of the Behavior setup parameter <i>Use Component View package when generating code</i> .

Iterations available at Method level:

Name	Description
Parameter	All the methods parameters.

4.11 Parameter

This node represents a method parameter.

Substitution variables available at Parameter level:

Name	Description
ModelTypeIsJavaPrimitive	True if the parameters data type is a java primitive type: <i>boolean, byte, short, int, long, char, float</i> or <i>double</i>
TypePackageName	Package of the parameters data type. If the data type is an enumeration, the variable is the package of the enumeration class. If the data type is the name of a class in the model, the variable is the package of that class. This variable will either use logical package or component package depending on the setting of the Behavior setup parameter <i>Use Component View package when generating code</i> .

No iterations are available at the Parameter level.

4.12 Association

This node represents an association to or from a class.

In addition to the properties shown in the property dialog for an association, the properties of the two classes connected thru the association are available via the two prefixes *OwnerClass* and *MemberClass*. I.e. *OwnerClass.Name* gives the name of the class connected to the association's owner role. See [section 4.3 on page 8](#) for further description of substitution variables for classes.

Substitution variables available at Association level, in addition to those seen as properties for an association, are:

Name	Description
MemberType	When iterated over as member of a group or an association, the variable <i>NodeType</i> returns "Member" while this variable returns "Attribute".
IsOwnerMany	True if the owner role of the association has many multiplicity.
IsMemberMany	True if the member role of the association has many multiplicity.
IsOwnerMandatory	True if a class instance must be connected to the owner role of the association
IsMemberMandatory	True if a class instance must be connected to the member role of the association

Iterations at the Association level:

Name	Description
Member	The attribute or the group that implements the association. By default this is the primary key for the owner class. A member of an association has the same properties as members of a group.

4.13 Generalization

This node represents either a generalization between classes or a generalization between interfaces.

In addition to the properties shown in the property dialog for a generalization, the properties of the two classes or interfaces connected thru the generalization are available via the two prefixes *SuperClass* and *SubClass*. I.e. *SuperClass.Name* gives the name of the class or interface in the generalization. See [section 4.3 on page 8](#) and [section 4.4 on page 9](#) for further description of substitution variables for classes and interfaces.

4.14 Realization

This node represents a realization between a class and an interface.

In addition to the properties shown in the property dialog for a realization, the properties of the class and the interface are available via the two prefixes *Class* and *Interface*. I.e. *Class.Name* gives the name of the class of the realization. See [section 4.3 on page 8](#) and [section 4.4 on page 9](#) for further description of substitution variables for classes and interfaces.

4.15 Converter

This node represents a class of type Converter

Substitution variables available at Converter level, in addition to those seen as properties for a converter, are

Name	Description
ModelTypeIsJavaPrimitive	True if the model data type is a Java primitive.
PackageName	Package of the converters data type. If the data type is an enumeration, the variable is the package of the enumeration class. If the data type is the name of a class in the model, the variable is the package of that class. This variable will either use logical package or component package depending on the setting of the Behavior setup parameter <i>Use Component View package when generating code</i> .
TypePackageName	This variable is a synonym for PackageName.

Iterations provided at Domain level:

Name	Description
EnumValue	All enumeration values. Empty except if the converter is of type enumeration, in which case it loops through all enumeration values for that enumeration.

4.16 Domain

This node represents a class of type Domain

In addition to the properties shown in the property dialog for a domain, the properties of the converter are available via the prefix *Converter*. I.e. *Converter.Name* gives the name of the converter connected to the domain. See [section 4.15 on page 14](#) for further description of substitution variables for converters.

Substitution variables available at Domain level, in addition to those seen as properties for a domain, are

Name	Description
ModelTypeIsJavaPrimitive	True if the model data type is a Java primitive.
PackageName	Package of the converters data type. If the data type is an enumeration, the variable is the package of the enumeration class. If the data type is the name of a class in the model, the variable is the package of that class. This variable will either use logical package or component package depending on the setting of the Behavior setup parameter <i>Use Component View package when generating code</i> .
TypePackageName	This variable is a synonym for PackageName.

Iterations provided at Domain level:

Name	Description
EnumValue	All enumeration values. Empty except if the domain is of type enumeration, in which case it loops through all enumeration values for that enumeration.

5 Domain implementations

This chapter describes the different targets available in Domain Generator. At the moment only the Java target is available.

5.1 Java

The Java code generated by Domain Generator is intended to be used alongside client code generated by Client Generator with Java/JFC as target (see [section 5.1 on page 37](#) in the Client Generator manual about the Java/JFC target) and client code code generated by Dialog Generator with ICE-faces as target (See [section 5.3 on page 21](#) in the Dialog Generator manual about the ICEfaces target). In addition the code can be used together with code generated by Service Generator with Java as target (see [section 5.1 on page 13](#) in the Service Generator manual about the Java target). The code may be used from any manually written code as well.

The Java Target manual describes the Genova java runtime, along with generated service and client code. See [chapter 4 on page 15](#) in the Java Target manual for a description of the Java target for the domain generator.

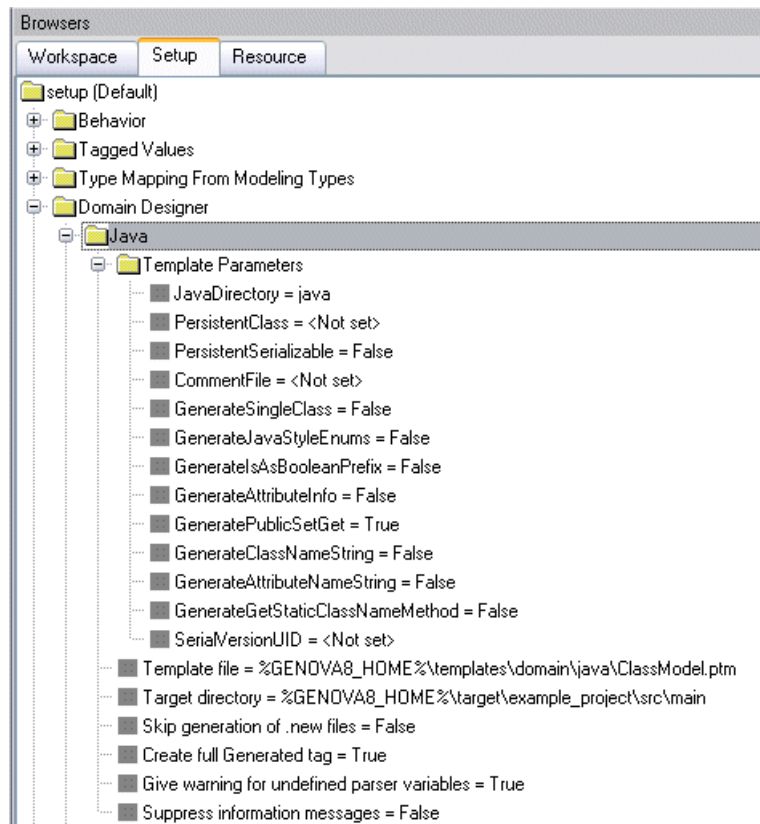
5.1.1 Compiling and running

To compile and run the generated code, you will need to install the Java runtime and compiler version. The templates are written for java version *1.5* and requires *JRE1.5* or later. Both the compiler and the run-time environment are available for free at Sun's web pages, see <http://java.sun.com/javase/downloads.html>.

It is possible to use any third party compiler or run-time, as long as it supports version 1.5 of Java.

5.1.2 Template parameters

The Java target has the following setup parameters:



In this section only the *Template Parameters* are explained. A description of the other parameters are found in [chapter 3 on page 5](#).

JavaDirectory: By default the templates are directing the output into the *java* subdirectory under the target directory. The *java* directory will contain any generated java source. By default this stucture is used by all generators when generating code for a java oriented target. If one wants to use another stucture the variable should be given a value. Setting this variable to the empty string will directed all output into the target directory without using any substructures.

PersistentClass: This parameter specifies a common super class for all persistent classes. All persistent classes not inheriting from any other class in the UML model, will subclass this class. The provided class `no.genova.domain.PersistentClass`, provides a lock flag, enabling optimistic locking by Hibernate.

PersistentSerializable: When set to *True*, this parameter makes all persistent classes implement the java `Serializable` interface.

CommentFile: This parameter specifies a user defined template file. By default the java templates contains a line that produces a comment in the

start of each java class containing the content of the *GeneratedWith* variable:

```
//Generated with Genova version 8.3.0.A by User, 2009.04.17 13:54:03.Template version 8.3.0.A.May 04, 2009
```

If a comment template file is specified, it is included instead of the line containing the default comment line, making it possible to include user defined comments.

GenerateSingleClass: When set to *True*, this parameter makes the generator create a single java class for each class in the model. In this case the templates are written so that one may choose either to replace any existing file or generate a .new file for any existing file. This behavior is managed by the setup parameter *Skip generation of .new files* for the Java target. When set to *True* the generation will overwrite any existing class file. When set to *False* the generation will create a .new file and not overwrite existing files.

When *GenerateSingleClass* is set to *False* each model class will result in two classes, class *ClassNameDefault* and class *ClassName* inheriting *ClassNameDefault*. *ClassNameDefault.java* contains the classe's attributes and property methods, while *ClassName.java* is intended for manually written code and initially only contains the empty class. When regenerating code, the generator will overwrite the default (super) class, but it will keep the subclass, to prevent overwriting of any manually written code.

GenerateJavaStyleEnums: When set to *True* the templates is set to generate Java 5 enumerations. When set to *False* the templates generates enums compatible with the Genova runtime delivered with version 8.2 of Genova.

GenerateIsAsBooleanPrefix: When set to *True* all property retrieval methods for boolean properties will be generated as *isPropertyName()*. When set to *False* such methods get the same form as any other property retrieval method, *getPropertyName()*.

GenerateAttributeInfo: When set to *True* information about display rules, blank when zero and input/output justification is generated. This is saved in objects of type *AttributeInfo*, which contains the methods

```
getDisplayRule()
isBlankWhenZero()
getInputJustification()
getOutputJustification()
```

GeneratePublicSetGet: The visibility for an attribute and its corresponding set/get methods can be generated in two fashions.

- 1) With this template variable set to *true* the attribute is given visibility as defined in the UML model, and the set and get methods are given visibility *public*.
- 2) With this template variable set to *false* the attribute is given visibility *private*, and the set and get methods are given visibility as defined for the attribute in the UML model.

If the set and/or get method are defined in the UML model, the defined method(s) get visibility as defined in the model regardless of the setting of this template parameter.

If *GenerateSingleClass* is *False* the set and/or get method will not be generated if the the methods visibility becommes *private*. In this case the java templates are producing a warinig messgae during generation.

Note: If the generated domain classes are intended to be used with code generated for the Service target Java or any of the java Client targets, both the set and the get method must have visisbility *public*.

GenerateClassNameString: When set to *True* the generated classes will contain a static String attribute containing the name of the class:

```
public static final String NAME = "ClassName";
```

GenerateAttributeNameString: When set to *True* the generated classes will contain a static String attribute for each attribute in the class, containing the name of the attribute:

```
public static final String ATTR_ATTRIBUTENAME = "attributeName";
```

GenerateGetStaticClassNameMethod: When set to *True* the generated classes will contain static String methods returning the name of the class:

```
public static String getStaticName() { return "ClassName"; }  
public static String getDomainName() { return "PackageName" + getStaticName(); }
```

SerialVersionUID: This parameter specifies a long value to be used as serialVersionUID. When not set no serialVersionUID is generated.

```
private static final long serialVersionUID = value
```

GenerateJavaStyleEnums: If true, enumerations will be proper java 1.5 enums, rather than subclasses of GenovaEnumerator.

5.1.3 Code Generation

The Java domain classes are generated with base directory set using the *Target directory* parameter concatenated with the *JavaDirectory* and the package defined for each class. The template parameter *GenerateSingle-Class* defines if all code should be generated into a single file for each class or if the each generated class should consist of two classes, a default class containing all definitions and a subclass to that one containing the user written code.

5.1.4 Additional documentation

Code generated for the Java target is further documented in chapters [Chapter 4 "Domain Generator" on page 15](#), [Chapter 5 "Service Generator" on page 17](#), [Chapter 6 "Client Generator for the Java/JFC target" on page 32](#) and [Chapter 7 "Client Generator for the Java/ICEfaces target" on page 49](#) in the Java Target manual.

The java target generates javadoc comments, based on the description given in the UML model. These comments can be turned into browsable documentation by running the javadoc tool provided by sun.

In addition, generated javadoc for the runtime libraries (Genova runtime jars) are found in the files `genova-<module>-<version>-javadoc.jar` in the jar directory of the Genova installation.