

Service 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

SERVICE GENERATOR – TOC

1	Introduction	1
2	Starting Service Generator	2
3	Server Generator Setup	5
4	Iteration structure and substitution variables	7
4.1	Top level node type	7
4.2	ObjectSelectionRef	7
4.3	ObjectSelection	7
4.4	Root	8
4.5	Role	8
4.6	Attribute	10
4.7	UniqueKey	11
4.8	KeyAttribute	11
5	Service implementations	13
5.1	Java	13
5.1.1	Compiling and running	13
5.1.2	Template parameters	13
5.1.3	Code Generation	14
5.1.4	Additional documentation	15
5.2	Java2XML Conversion	15
5.2.1	Template parameters	16
5.2.2	Code Generation	17
5.3	Sysdul2XML Conversion	17
5.3.1	Template parameters	17
5.3.2	Code Generation	17

1 Introduction

This manual describes Genova's Service Generator.

Service Generator generates code for the service part of an application. The code generated is based on the Object Selections together with the specification given in the Service Designer setup for the wanted target.

The Java code generated by Service Generator is intended to be used together with code generated by Client Generator with Java/JFC as target, but may also be used from any self written code. The service code uses an implementation of the interface `no.genova.jgrape.DataService` to access the database. The runtime library comes with an implementation based on Hibernate.

Server 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 Service Generator:

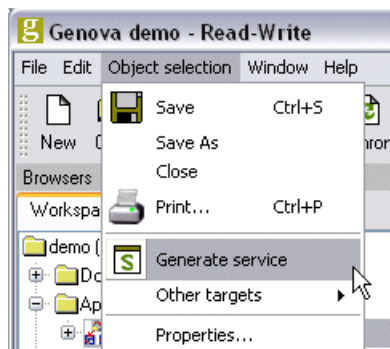
- Chapter 1 ["Introduction" on page 1](#) is the chapter you are now reading.
- Chapter 2 ["Starting Service Generator" on page 2](#) explains how to start generating service code.
- Chapter 3 ["Server Generator Setup" on page 5](#) describes the setup parameters used by Service Generator.
- Chapter 4 ["Iteration structure and substitution variables" on page 7](#) describes the iteration structure and substitution variables available when writing templates for Service Generator.
- Chapter 5 ["Service implementations" on page 13](#) gives a short description of the different targets available and explains the *Template parameters* for each target.

2 Starting Service Generator

Code can be generated for a single object selection or for a complete application.

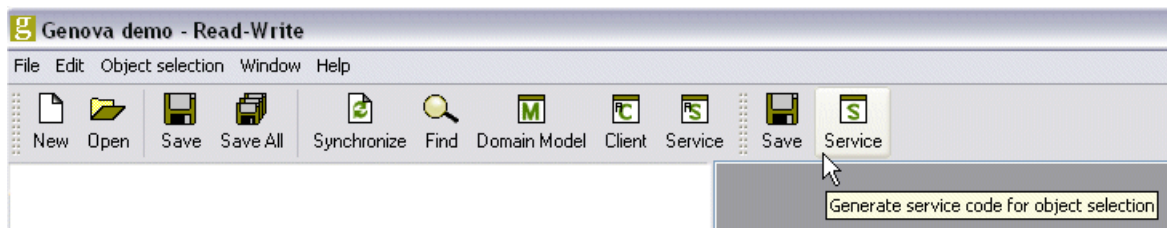
To generate service code for a single object selection at a time:

- ◆ **Make sure a object selection window is open and active.**
- ◆ **Select the menu options *Object selection/Generate service*.**

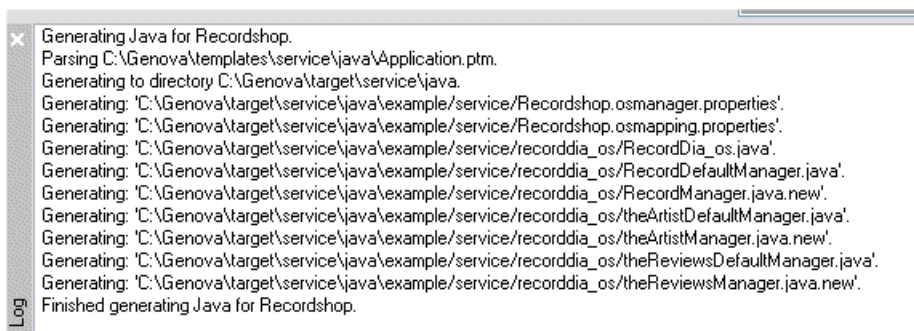


OR

- ◆ **Click the *Generate Service* button in the Object selection tool bar.**



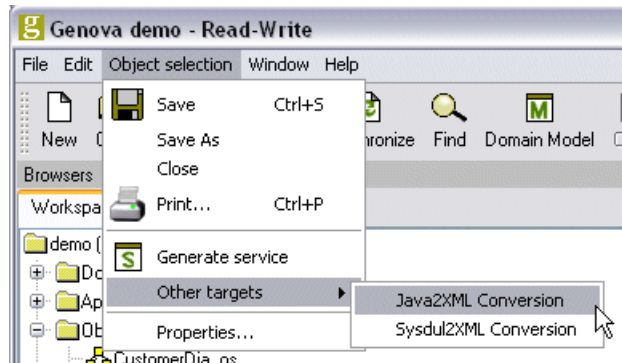
In the log window, Genova displays messages showing the progress of the service generation.



Note: The service target used when generating code for an object selection is the one specified in the object selection's *Service Target* property. See [chapter 8 on page 16](#) in the Object Selection manual for more information.

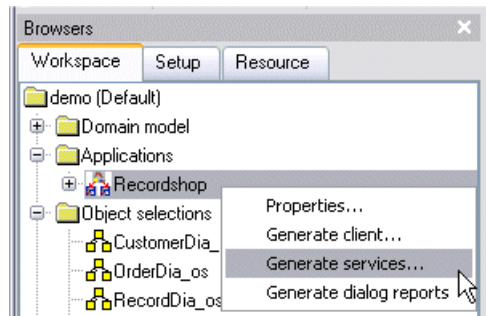
To generate service code for other targets for a single object selection:

- ◆ **Make sure a object selection window is open and active.**
- ◆ **Select a target from the menu *Object selection/Other targets*.**



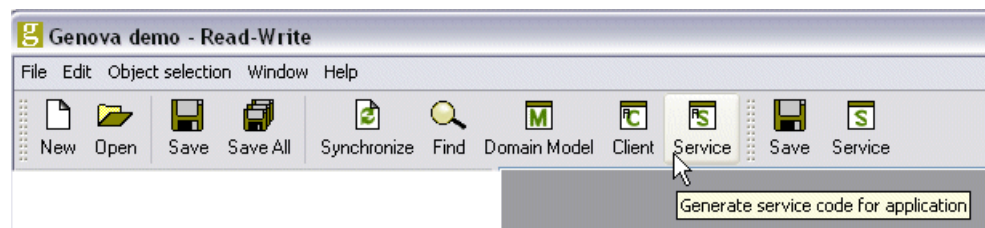
To service generate code for a whole application:

- ◆ **From the Workspace browser window, right-click the desired application and select the *Generate Service* command.**

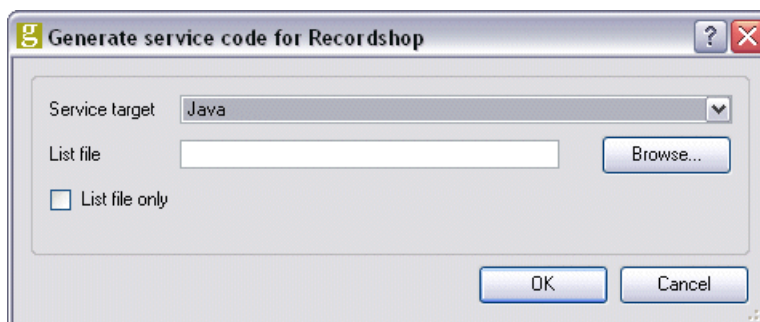


OR

- ◆ **Click the *Service* button on the toolbar.**



The following dialog is displayed:



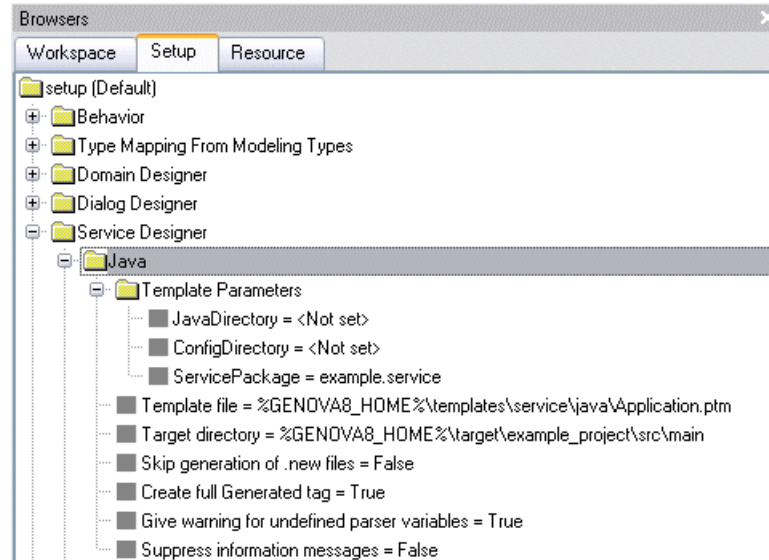
Service target: From the drop-down list, select the target for the generation of the service. By default, code will be generated only for object selections belonging too dialogs included in the application.

List file: To generate code for additional object selections not in the original application model, specify a list file containing names of additional object selections to be included in the generation.

List file only: Check this option to generate code only for the object selections in the list file and not for the object selections from the application model.

3 Server Generator Setup

This chapter gives a general description of the setup parameters used by Service 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 *Application.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 re-

sults will not show a differences 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 in [chapter 2 on page 2](#), 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

Service 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 Service Generator, and for each node type in the structure all available substitution variables are described.

4.1 Top level node type

The outermost level represents the application and has node type *Service-Application*.

Substitution variables available at the top level:

Name	Description
InApplication	True only if generating for an application. False if generating for a single object selection.

Iterations at the top level:

Name	Description
ObjectSelectionRef	All references to object selections.
ObjectSelection	All object selections with content.

4.2 ObjectSelectionRef

This only represents the reference to an object selection.

Substitution variables available at ObjectSelectionRef level:

Name	Description
Name	Name of the object selection.

None iterations is available at the ObjectSelectionRef level.

4.3 ObjectSelection

This represents an opened object selection.

Substitution variables available at ObjectSelection level are those seen as properties for the root of the object selection.

Iterations at the ObjectSelection level:

Name	Description
Root	All root roles in the object selection.

Root All root roles in the object selection.

4.4 Root

This represents a root role in the object selection. A root is also a role and has the same properties as a role, see [section 4.5](#).

4.5 Role

This represents a role in an object selection.

Substitution variables available at Role level, in addition to those seen as properties for a role, are:

Name	Description
AssociationName	Association name for the association connecting this role to its parent role.
AssociationRoleName	Association role name for this role in association from parent role.
ParentAssociationRole-Name	Association role name for parent role in association from parent role.
ClassName	Name of class.
AssociationClassName	Name of class in this roles end of association. This will be different from Class.Name if the association is inherited into this class.
ParentAssociationClass-Name	Name of class in parent roles end of association. This will be different from Parent.Class.Name if the association is inherited into parent class.
AssociationPackage-Name	Name of package of class in this roles end of the association.
ParentAssociationPack- ageName	Name of package of class in parent roles end of associa- tion.
PackageName	Name of package for class.
DefaultDomainKey	Name of default domain key.
PrimaryKey	Name of primary key.

Name	Description
MainKey	Name of the main key. The main key is the first key found when searching in the following sequence: <ol style="list-style-type: none"> 1. DefaultDomainKey 2. PrimaryKey 3. First unique attribute 4. First unique group
RoleDescription	Description of role.
ClassDescription	Description of class for role.
IsRoot	True if this is the root role.
IsMember	True if this role is the member in association from parent, i.e. the role is downward related. Returns false for a root role.
IsMany	True if this role is a many role in association from parent role. Returns true for a root role.
IsParentMany	True if parent role is a many role in association from parent role.
IsMandatory	True if this role is a mandatory role in association from parent role.
IsParentMandatory	True if parent role is a mandatory role in association from parent role.
IsNavigableToParent	True if the association is navigable from this role to parent role.
IsNavigableFromParent	True if the association is navigable from parent role to this role.
HasDefaultDomainKey	True if the role has a default domain key defined.
HasPrimaryKey	True if the role has a primary key defined.
HasMainKey	True if the role has a main key. The main key is the first key found when searching in the following sequence: <ol style="list-style-type: none"> 1. DefaultDomainKey 2. PrimaryKey 3. First unique attribute 4. First unique group
HasChildren	True if this role has at least one child role.
HasMemberChildren	True if this role has at least one downward related child role.
HasOwnerChildren	True if this role has at least one upward related child role.
HasIncludedAttributes	True if this role has at least one attribute included.

Iterations at the Role level:

Name	Description
Role	All child roles for this role.
Attribute	All attributes for the class, both own and inherited.
UniqueKey	All unique keys for the class, both own and inherited.

4.6 Attribute

This represents an attribute in a role.

Substitution variables available at Attribute level:

Name	Description
NamePath	Gives the path to the attribute. The path is a '.' separated list of the one related association role names leading to the role containing the attribute. An attribute from role where IsMany returns True, has an empty NamePath.
IsAttributePersistent	True if attribute is persistent and the role containing the attribute is persistent.
IsAttributeExcluded	True if attribute is excluded from role.
ModelTypeIsJavaPrimitive	True if the model data type is a Java primitive.
PackageName	Package name of the data type when the attribute is an enumeration.
ViewModelType	If the attribute does not have a converter this returns the same as ModelType. If the attribute has a converter this returns the converters ModelType.
ViewGenovaType	If the attribute does not have a converter this returns the same as GenovaType. If the attribute has a converter this returns the converters GenovaType.
ViewModelTypeIsJavaPrimitive	If the attribute does not have a converter this returns the same as ModelTypeIsJavaPrimitive. If the attribute has a converter this returns the converters ModelTypeIsJavaPrimitive.
ViewPackageName	If the attribute does not have a converter this returns the same as PackageName. If the attribute has a converter this returns the converters PackageName.
IsAttributeInDialog	True if attribute is in use in the dialog. This property is only valid if the object selection is parsed as part of an dialog, see section 4.5 on page 10 in Dialog Generator manual.

Iterations at the Attribute level:

Name	Description
EnumValue	All enumeration values if the attribute is an enumerator.
ViewEnumValues	If the attribute does not have a converter this is the same as iterating EnumValue. If the attribute has a converter and the converter is of enumerator type, this iterates the enumerators values.
UniqueKey	All unique keys for the class, both own and inherited.

4.7 UniqueKey

This represents a unique key for a role, both attributes and groups included.

Substitution variables available at UniqueKey level:

Name	Description
KeyName	Name of the key.
IsKeyInOs	True if all parts of the key is part of object selection.
IsDefaultDomainKey	True if this key is the default domain key for the class.
IsPrimaryKey	True if this key is the primary key for the class.
IsKeyInDialog	True if all part of the key is in use in the dialog. This property is only valid if the object selection is parsed as part of an dialog, see see section 4.5 on page 10 in Dialog Generator manual.
IsExcluded	A key is included if all attributes in the key are from classes included in the object selection and all of the attributes are included in their object selection role. If this is the case IsExcluded will return false.

Iterations at the UniqueKey level:

Name	Description
KeyAttribute	All attributes in the key.

4.8 KeyAttribute

This represents an attribute in a unique key. A key attribute may or may not come from a role that is part of the object selection. If inside such a role the key attribute is also an attribute and have the same substitution variables as Attribute. Attributes not part of the

Substitution variables available at KeyAttribute level:

Name	Description
KeyAttributeName	Name of the key attribute. For attributes in the object selection this is the same as the name of the role attribute. Attributes not part of the role are attributes reached via associations. They will be named with the their association role names as a prefix.
ClassName	Name of the class containing the attribute.
IsAttributeInOs	True if the attribute's role is part of the object selection. If False this attribute is not a role attribute and the role attributes properties are not available.
IsForeignKey	True if the attribute is part of a foreign key.

None iterations is available at the KeyAttribute level.

5 Service implementations

This chapter describes the different targets available in Service Generator.

5.1 Java

The Java code generated by Service Generator is intended to be used together with 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), but may also be used from any self written code. The service code uses an implementation of the interface `no.genova.jgrape.DataService` to access the database. The runtime library comes with an implementation based on Hibernate. The Java code generated with Service Generator and the coexisting with the runtime libraries and the code generated for the Java/JFC Client Generator target is described in the Java Target manual.

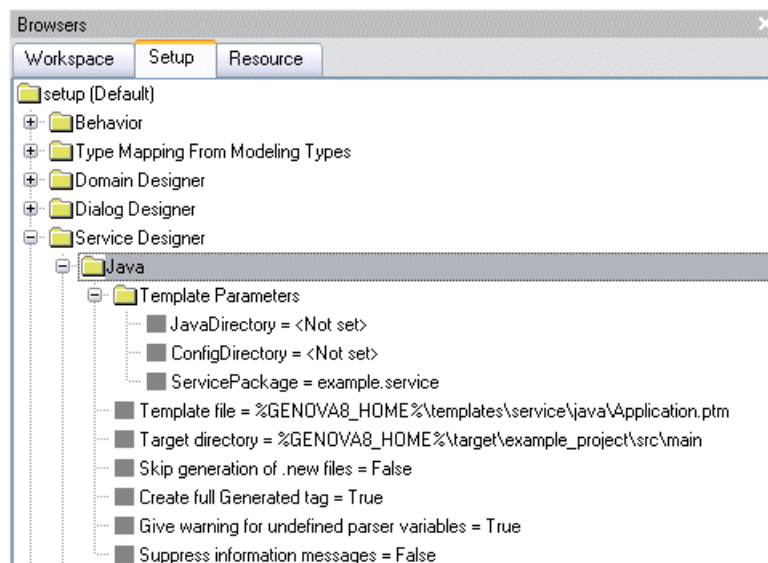
5.1.1 Compiling and running

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/j2se/downloads.html>.

The generated code is tested and run using Sun's compiler, but any third party compiler or run-time environment supporting JRE1.5 or better should work here. For instance is it possible to use Jikes from IBM to compile the generated Java files.

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 and ConfigDirectory: By default the templates are directing the output into two subdirectories under the target directory: *java* and *config*. The *java* directory will contain any generated java source, while *config* will contain any configuration files. By default this structure is used by all generators when generating code for a java oriented target. If one wants to use another structure the two variables should be given values. Setting both these variables to the empty string will directed all output into the target directory without using any substructures.

ServicePackage: This parameter specifies the top of the package hierarchy for the Java classes generated with Service Generator for the Java target. The path to the directory holding the generated source will be the concatenation of the *Target directory* parameter, the *JavaDirectory* and the directory deduced from the *ServicePackage* parameter. In the example above the path will be:

```
%GENOVA8_HOME%\target\example_project\src\main\java\example\service
```

If one wants the *ServicePackage* value to contain the name of the application one can include the *ApplicationName* as a substitution value, i.e.

```
ServicePackage=%ApplicationName%.service
```

When generating services for a whole application, the name of the application will be substituted for the *%ApplicationName%* parameter. This will however not work if one generates service code for a single object selection. In that case, the *ApplicationName* too must be defined as a template parameter.

5.1.3 Code Generation

The Java service code are generated with base directory set using the template parameters *ServicePackage* (see [section 5.1.2 on page 13](#)). Classes common to all of the application will be placed in this base directory while class for each object selection will be placed in a directory (and package) based on its object selection name.

The generated files from the object selection are as follows:

```
%ApplicationName%.osmanager.properties
```

Configuration file that can be used to relocate code for an object selection to an other package, see [section 8.2 on page 70](#) in Java Target manual. By default this file is placed in the *config* subdirectory.

<code>%ApplicationName%.osmapping.properties</code>	Configuration file mapping osmanager roles when using multiple databases, see section 8.4.2 on page 73 in Java Target manual. Target manual. By default this file is placed in the <i>config</i> subdirectory.
<code>%ObjectSelectionNameLower%\%ObjectSelectionName%.java</code>	Class containing entry points for service code for an object selection. Target manual. By default this file is placed under the <i>java</i> subdirectory.
<code>%ObjectSelectionNameLower%\%RoleName%DefaultManager.java</code>	Class containing service code for on object selection role. Service Generator creates one class for each role in the object selection. By default this file is placed under the <i>java</i> subdirectory.
<code>%ObjectSelectionNameLower%\%RoleName%Manager.java</code>	Subclasses of the DefaultManager classes. Service Generator creates one class for each role in the object selection. The classes generated are empty. These files will not be overridden when the code is regenerated. The file are intended for user overriding of the functionality in the DefaultManager classes. By default this file is placed under the <i>java</i> subdirectory.

5.1.4 Additional documentation

For code generated for the Java/ target is documented in [chapter 5 on page 17](#) in the Java Target manual.

The generated code is also documented using javadoc – The Java API Documentation Generator. After code generation, one would typically use 'javadoc' to generate browsable javadoc code for the generated files. See <http://java.sun.com/j2se/1.4/docs/tooldocs/solaris/javadoc.html> for more detailed information about Javadoc.

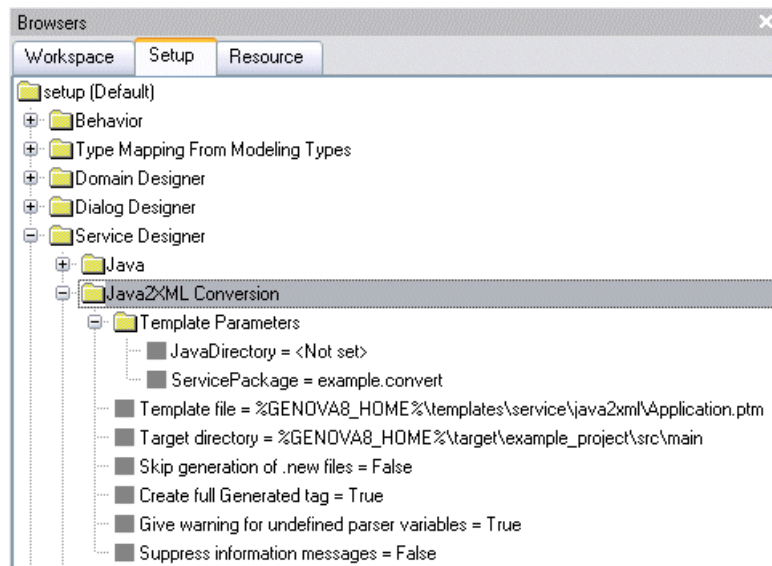
In addition a generated javadoc for the Genova runtime library is found in the file *genova-runtime.zip* in the doc directory of the Genova installation.

5.2 Java2XML Conversion

This service target generate java support classes for converting data in object selection instances to XML. The target is part of the print action support, but the generated code may also be used outside this context. See [Dialog Print](#) manual for a description of the print action support in Java generated code.

5.2.1 Template parameters

The Java2XML 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.

ServicePackage: This parameter specifies the top of the package hierarchy for the Java classes generated with Service Generator for the Java2XML. The path to the directory holding the generated source will be the concatenation of the *Target directory* parameter, the *JavaDirectory* and the directory deduced from the *ServicePackage* parameter. In the example above the path will be:

```
%GENOVA8_HOME%\templates\example_project\src\main\java\example\service
```

If one wants the *ServicePackage* value to contain the name of the application one can include the *ApplicationName* as a substitution value, i.e.

```
ServicePackage=%ApplicationName%.service
```

When generating services for a whole application, the name of the application will be substituted for the *%ApplicationName%* parameter. This will however not work if one generates service code for a single object selec-

tion. In that case, the *ApplicationName* too must be defined as a template parameter.

5.2.2 Code Generation

Two files will be generated for an object selection:

`<osName>DefaultXmlConverter.java`

`<osName>XmlConverter.java.`

For further description of the generated code see [section 5.2 on page 15](#) in Dialog Print manual.

5.3 Sysdul2XML Conversion

This service target generate sysdul code supporting converting of object selection data to XML. The target is part of the print action support, but the generated code may also be used outside this context. See [Dialog Print manual](#) for a general description of the print action support and [chapter 11 on page 27](#) in Sysdul Program Generator manual for usage of this code in generated Sysdul code.

5.3.1 Template parameters

The Sysdul2XML templates does not use any template parameters.

5.3.2 Code Generation

The Sysdul2XML target does not generate pure Sysdul code. The generated code must be converted to Sysdul using the Sysdul preprocessor, Svapp. See the [Preprocessor manual](#) for a description of Svapp.

The each object selection the following files are generated:

`<os>_drive.vsd`

`<os>.vsd`

`<os>.svp`

`<os>:default.svp`

`<os>_formats.svp`

`<os>_default_formats.svp`

In addition a application wide file, `<app>.vsd`, is generated.

A further description of the generated code is found in [section 5.3 on page 15](#) in the Dialog Print manual.

