

Dialog 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

DIALOG GENERATOR – TOC

1	Introduction	1
2	Starting Dialog Generator	2
3	Dialog Generator Setup	5
4	Iteration structure and substitution variables	7
4.1	ClientApplication	7
4.2	DialogModelRef	7
4.3	RequiredInterface	8
4.4	DialogModel	9
4.5	ObjectSelection	10
4.6	DialogObject	10
4.7	Event	13
4.8	Method	14
4.9	Action	14
4.10	ParameterBinding	16
4.11	Enumerator	17
4.12	ResourceModel	17
4.13	Template	18
4.14	Style	18
4.15	Layout	18
4.16	Color	19
4.17	Font	19
4.18	Image	19
4.19	PrintSetting	20
4.20	ComponentType	20
5	Dialog Client implementations	21
5.1	Jasper Report Templates	21
5.2	Dialog Report	21
5.3	Java ICEfaces	21
5.3.1	Template parameters	22

1 Introduction

This manual describes Genova's Dialog Generator.

Dialog Generator generates code for the client part of an application. The code generated is based on the Dialog Model and its Object Selection together with the specification given in the Dialog Designer setup for the wanted target.

Dialog 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 Dialog Generator:

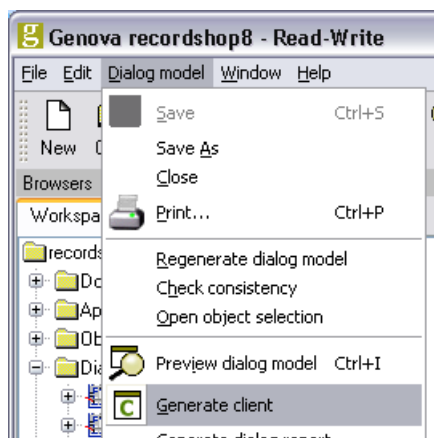
- Chapter 1 ["Introduction" on page 1](#) is the chapter you are now reading.
- Chapter 2 ["Starting Dialog Generator" on page 2](#) explains how to start generating dialog code.
- Chapter 3 ["Dialog Generator Setup" on page 5](#) describes the setup parameters used by Dialog Generator.
- Chapter 4 ["Iteration structure and substitution variables" on page 7](#) describes the iteration structure and substitution variables available when writing templates for Dialog Generator.
- Chapter 5 ["Dialog Client implementations" on page 21](#) gives a short description of the different targets available and explains the *Template parameters* for each target.

2 Starting Dialog Generator

Code can be generated for a single dialog or for a complete application.

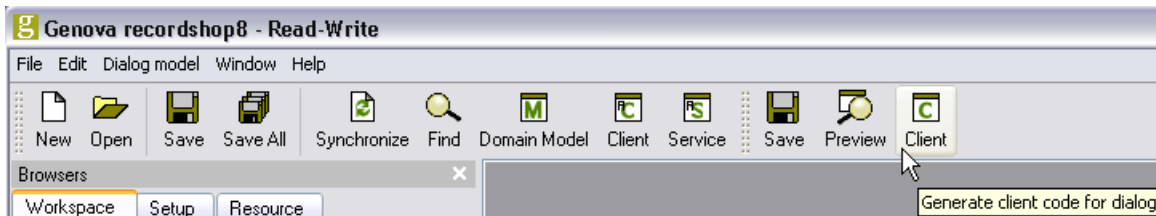
To generate code for a single dialog at a time:

- ◆ **Make sure a dialog model window is open and active.**
- ◆ **Select the menu options *Dialog model/Generate client*.**



OR

- ◆ **Click the *Generate Client* button in the Dialog model tool bar.**



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

```

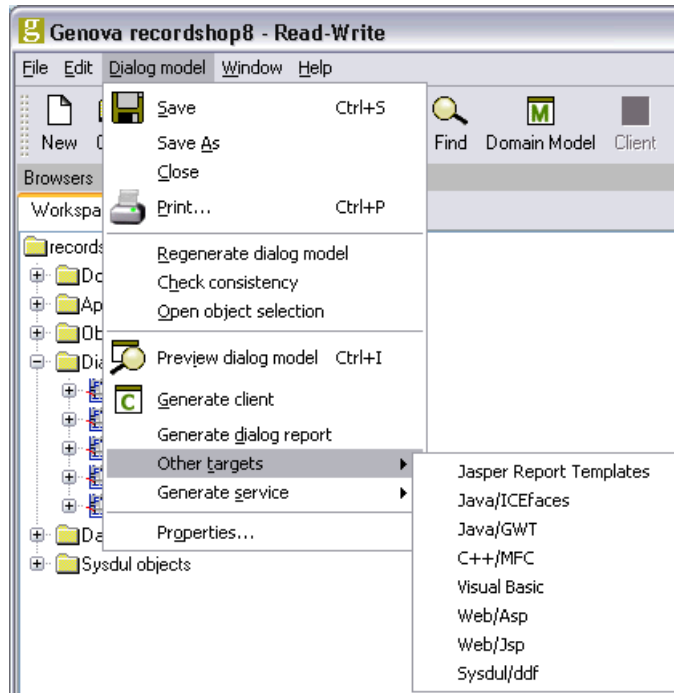
Generating client...
Parsing C:\Genova.820\templates\client\jsf\Application.ptm.
Generating to directory C:\Genova.820\target\client\jsf.
Generating: 'C:\Genova.820\target\client\jsf\java/example/faces/recorddia/generated/package-info.java'.
Generating: 'C:\Genova.820\target\client\jsf\java/example/faces/recorddia/package-info.java.new'.
Generating: 'C:\Genova.820\target\client\jsf\java/example/faces/recorddia/generated/RecordDiaDefaultController.java'.
Generating: 'C:\Genova.820\target\client\jsf\java/example/faces/recorddia/RecordDiaController.java.new'.
Generating: 'C:\Genova.820\target\client\jsf\java/example/faces/recorddia/generated/RecordDiaConst.java'.
Generating: 'C:\Genova.820\target\client\jsf\java/example/faces/recorddia/generated/ArtistNode.java'.
Generating: 'C:\Genova.820\target\client\jsf\java/example/faces/recorddia/generated/ReviewsNode.java'.
Generating: 'C:\Genova.820\target\client\jsf\webapp/RecordDia.jspx'.
Generating: 'C:\Genova.820\target\client\jsf\java/example/faces/recorddia/generated/RecordDiaDefaultView.java'.
Generating: 'C:\Genova.820\target\client\jsf\java/example/faces/recorddia/RecordDiaView.java.new'.
Finished generating Java/JSF for RecordDia.
  
```

Note: The target used when generating code for a dialog model is the one specified in the dialog model's *Client Target* property. See [section](#)

5.6 on page 23 in the Dialog Designer manual for more information.

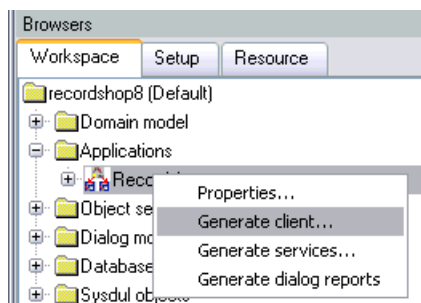
To generate dialog code for another target than the one specified in the *Client Target* property:

- ◆ **Make sure a dialog model window is open and active.**
- ◆ **Select the menu options *Dialog model/Other targets* and select target.**



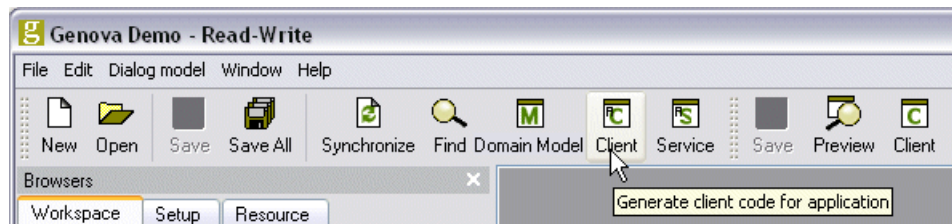
To generate client code for a whole application:

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

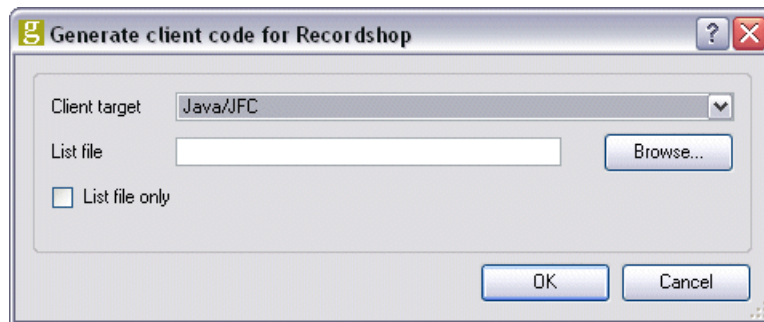


OR

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



The following dialog is displayed:



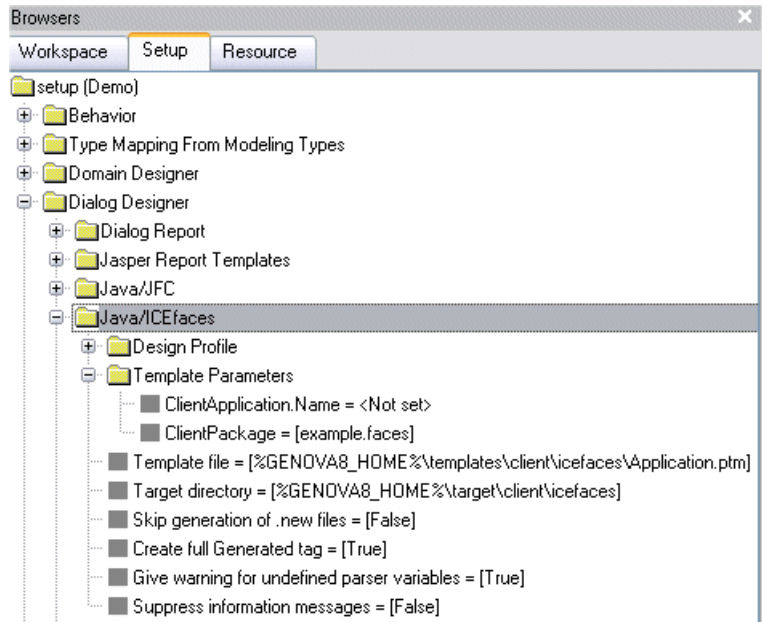
Client target: From the drop-down list, select the target for the generation of the client code. By default, code will be generated only for dialogs included in the application.

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

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

3 Dialog Generator Setup

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



Template Parameters: This category can hold parameters used in the templates for the target. See [chapter 8 on page 10](#) in the Template Parser manual on how such parameters are used.

Template file: This parameter specifies the name of the target's main template files. The template files all have the extension ".ptm". The main template file read by Dialog Generator is the file *Application.ptm*.

For more information on template files, see [chapter 3 on page 5](#) in the 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 template parameter to *True* will make the parser 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 variable: When set to *True* usage of undefined variables in the template files will give a warning message if the usage is in scope. In a construction like `%IF:"False" & VName%` , the variable `VName` will, even if undefined, never give a warning, while it will if the statement is changed to `%IF:VName & "False"%`.

Suppress information messages: When set to *True* the parser will suppress all information messages both those generated by the parser itself and those defined in the template files. The only exception is the start messages and the finish message produced during the generation.

4 Iteration structure and substitution variables

Dialog 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 Dialog Generator. For each node type in the structure the substitution variables not based directly on a model elements property are described. The substitution variables based on a model elements properties are not listed in this chapter. To see which properties a dialog model element has, see [Chapter 13 "General properties" on page 68](#) in the Dialog Designer manual. For properties of dialog resources see [Chapter 4 "Style properties" on page 24](#) and the chapters following that one in the Dialog Resources manual. For a description of how the parser transform properties to substitution variables is found in [Chapter 8 "Substitution variables" on page 10](#) in the Template Parser manual.

4.1 ClientApplication

This node type is the outermost level and represents the application and from this node's substructures the substitution variables for the application can be accessed by *ClientApplication.variable*.

Substitution variables available at the *ClientApplication* level are the properties of the application as seen in in the applications property dialog in Genova. In addition the following variables are available:

Name	Description
InApplication	<i>True</i> only if generating for an application. False if generating for a single dialog.

Iterations at the top level:

Name	Description
DialogModelRef	All references to dialog models included in this generation.
DialogModel	All dialogs with content.
ResourceModel	The resource model with content.
Enumerator	All enumerators in the class model.

4.2 DialogModelRef

This represents the reference to a dialog model and not the dialog model itself.

Substitution variables available at *DialogModelRef* level:

Name	Description
Name	Name of the dialog model.
IsTopDialogModel	<i>True</i> if the dialog is a top dialog in the applications navigation model, i.e. reachable from the application via a <i>Contains</i> dependency.
IsApplicationWindow	<i>True</i> if the dialog is an application window.
IsDialogBox	<i>True</i> if the dialog is a dialog box.
IsDocumentWindow	<i>True</i> if the dialog is a document window.
IsLinkable	<i>True</i> if the dialog is a linkable dialog.

Note: The three dialog type test variables *IsApplicationWindow*, *IsDialogBox* and *IsDocumentWindow* are only available after a dialog has been opened once, i.e. the templates have looped through an *DialogModel* iteration at least once (see [section 4.4 on page 9](#)).

Two iterations are available at *DialogModelRef* level:

Name	Description
DialogModelRef	References to all dialog models reachable from this dialog via an <i>Activate</i> dependency in the applications navigation model. This iteration possibility is only available in one level even if the dialogs reached via this iteration contain <i>Activate</i> dependencies. By default only the <i>Activates</i> dependency between dialogs are available, but when all dialog have been opened once via an <i>DialogModel</i> iteration, this iteration will also contain any dialog that is target in an <i>Open dialog</i> action from this dialog.
RequiredInterface	References to all interfaces reachable from this dialog via an <i>Required</i> dependency in the UML model.

4.3 RequiredInterface

This iteration level represents an interface connected to the dialog via an *Required* dependency.

In addition to the properties shown in the property dialog for an interface, the following Substitution variables are available at the *RequiredInterface* 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>

Name	Description
RequiredName	Name of the dependency. If no name is specified in the UML model, the name of the interface is used.

4.4 DialogModel

This iteration level represents an opened dialog model and the dialogs content can be reached from this node.

In addition to the properties shown in the property dialog node in the dialog structure, the following Substitution variables are available at the *Dialog-Model* level:

Name	Description
IsApplicationWindow	<i>True</i> if the dialog is an application window.
IsDialogBox	<i>True</i> if the dialog is a dialog box.
IsDocumentWindow	<i>True</i> if the dialog is a document window.
HasEvents	<i>True</i> if at least one component in the dialog has an event defined.
HasNondataItems	<i>True</i> if at least one component is a non-data item, i.e. it is not a data item and it is not a container.
HasDataItems	<i>True</i> if at least one component is a data item.
HasListBlocks	<i>True</i> if at least one component is a list block.
HasTableBlocks	<i>True</i> if at least one component is a table block.
HasTreeNodees	<i>True</i> if at least one component is a tree node block.

If the property *PrintSetting* is set (*HasPrintSetting* is *True*) then any properties for the print setting can be reached by the *PrintSetting* prefix, i.e. *PrintSetting.LeftHeader*. Properties for the print setting is found in [section 4.19 on page 20](#).

Iterations at the DialogModel level:

Name	Description
ObjectSelection	This iteration will just contain one element, the dialogs object selection, or be empty if the dialog does not have an object selection.
DialogObject	All window blocks in the dialog.
Method	All methods used by at least one event in the dialog.

4.5 ObjectSelection

This represents the dialogs object selection. The content for this node type and its substructures are the same as those found in the object selection structure of Service Generator. See [section 4.3 on page 7](#) in the Service Generator manual for further description of the *ObjectSelection* level.

4.6 DialogObject

This node level represents an object in a dialog and reflects the structure of dialog objects in the dialog as seen in Dialog Designer.

Substitution variables available at DialogObject level are those seen as properties for a dialog object. The kind of properties and hence substitution variables available depends on the object type. For a dialog object several substitution variables are available that can be used in the templates to test type of object or group of object:

Name	Description
ComponentType	Name of the component type. The component types are divided into three groups: Data items: <i>CheckButton, ComboBox, ListBox, RadioGroup, Scale, Stepper, TextField.</i> Non-data Items: <i>Button, Label, ImageBox, MenuItem, Separator, Spacer, Viewport, Component, ScrollBar.</i> Containers: <i>WindowBlock, SimpleBlock, TableBlock, MenuBar, Menu, Toolbar, Notebook, ListBlock, TreeView, TreeNode</i>
TabSequenceIndex	The components position in the window blocks tab sequence. Numbering starts on 1. Two special values exists: 0: The component could not be part of the tab sequence. -1: The component could be part of the tab sequence, but is excluded from it.
IsDataItem	<i>True</i> if the component is a data item.
IsNonDataItem	<i>True</i> if the component is a non-data item.
IsContainer	<i>True</i> if the component is a container.

Preview: When generating code Dialog Generator also have substitution variables deduced from Dialog Designers preview of a dialog. These variables are reached via the prefix *Preview*, i.e. *Preview.Width* gives the width in number of pixels for a dialog object. The first time Dialog Generator encounters a *Preview* substitution variable it will establish a preview of the dialog. The preview will be seen as an icon on the taskbar.

The next table describes the different *Preview* substitution variables:

Name	Description
Width	The width of the component in number of pixels.
Height	The height of the component in number of pixels.
XPos	The horizontal position of the component in number of pixels. The position is relative to the left edge of the surrounding block.
YPos	The vertical position of the component in number of pixels. The position is relative to the top edge of the surrounding block.
LineHeight	Height in number of pixels of each line in a list block. 0 for all other component types.
HeaderHeight	Height in number of pixels of header in a list block. 0 for all other component types.
ColumnWidth	Width in number of pixels of a column in a list block. The substitution variable gives the same answer whether the component is a data item inside a list block or a data items corresponding label inside a list block.
ColumnXPos	The horizontal position in number of pixels of a column in a list block. The position is relative to the left edge of the list block. The substitution variable gives the same answer whether the component is a data item inside a list block or a data items corresponding label inside a list block.

Style and layout: All dialog objects may have a style and most of the container types may have a layout. The properties of the actual style and layout are reached via the two prefixes *Style* and *Layout*, i.e. *Style.BackgroundColor* gives the name of the background color defined for dialog objects style. Properties for the style setting is found in [section 4.14 on page 18](#), while the layout setting is found in [section 4.15 on page 18](#).

Data items representation: A data items property dialog contains four different tab pages; General, Attribute, Layout-overrides and Attachment. The properties on these four pages are available as substitution variables. But in addition the properties of the representation are also directly available as variables. If a data item is represented as a *Combobox*, the six properties shown in the representation dialog are all available as substitution variables, without any prefix. For a combobox that would be *NumberOfRows*, *NumberOfCharacters*, *IsSorted*, *IsSearchable*, *CaseConversion* and *IsSelectAllOnFocus*.

Data items attribute: A data item in a dialog model are based on an attribute in the object selection. The properties for the attribute are reached via the prefix *Attribute*, i.e. *Attribute.Name*, *Attribute.ModelType* etc.

Data items iterators: The attribute connected to a data item has a data type that can be of type enumeration. If this is the case to possible iteration are available:

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.

Data items: In addition to the variables already mentioned earlier the data items have the following four substitution variables:

Name	Description
TextLength	The maximum number of characters allowed in the dialog representation of the data item. The value zero implies no restriction on the number of characters.
HasLabel	True if the data item has a label connected.
NextEquivalentName	This gives the name of the next data item connected to the same role attribute as current data item. If two or more data items are connected to the same role attribute, using this variable when iterating through a dialog structure will create a circular list of references.
HasNextEquivalentName	True if at least one more data item is connected to the same role attribute as current data item.

Images: Some dialog objects have images connected and the properties of these images are all available via special prefixes for the dialog objects. The following image prefixes are defined:

Component	Prefix
Button	<i>Image</i> and <i>DisabledImage</i> .
Imagebox	<i>Image</i> .
Simple block	<i>BackgroundImage</i> .
Windw block	<i>IIconImage</i> and <i>BackgroundImage</i> .

The substitution variables for images are described in [section 4.18 on page 19](#).

Components: A component represents an external component and can be connected to a component type defined in the resource database. The properties of this component type are available via the prefix *ComponentType*. The substitution variables for component types are described in [section 4.20 on page 20](#). A component type defines a set of parameters and a com-

ponent connected to a component type can give values to these parameters. These parameters are available as substitution variables via the prefix *Parameters*.

Object selection roles: Most of the container types have an object selection role property and the data items are connected to a role attribute belonging to a object selection role. For all components with an associated role the prefix *Role* gives access to the roles properties. The properties and hence the substitution variables for roles are described in [section 4.5 on page 8](#) in the Service Generator manual.

Iterations at the Component level:

Name	Description
DialogObject	Recursively iterate over all child components for this component. This iteration is always empty non-containers.
Event	Events defined for this component.

4.7 Event

This represents an event defined on a component.

Substitution variables available at Event level:

Name	Description
EventType	The available event types are both dialog target and component dependent. All available event types for a component type is shown in the <i>Design Profile</i> in the setup database. Each possible dialog target has a defined design profile and for each component type a set of triggers are defined. These triggers are the possible event types. See section 5.6.1 on page 24 in the Dialog Designer manual for further description of the design profile. In the dialog designer itself the possible event types for a component can be seen in the event combobox of the event dialog, see " Defining events for dialog objects " on page 145 in the Dialog Designer manual.
Accelerator	The full string of the accelerator if this is an accelerator event. See section 16.2 on page 146 in the Dialog Designer manual for a further description of the string format for an accelerator key.
AcceleratorIdentifier	The key part of the accelerator string without any of the modifiers. See section 16.2 on page 146 in the Dialog Designer manual for a further description of the string format for an accelerator key.
HasAccelerator	<i>True</i> if the event is an accelerator event and the accelerator has a value.

Name	Description
HasAltModifier	<i>True</i> if the accelerator has the Alt modifier.
HasAltGrModifier	<i>True</i> if the accelerator has the AltGr modifier.
HasShiftModifier	<i>True</i> if the accelerator has the Shift modifier.
HasMetaModifier	<i>True</i> if the accelerator has the Meta modifier.

Iterations at the Event level:

Name	Description
Method	All methods to be called from an event.

Methods may be iterated from both an event and a dialog model. The description of Method found in [section 4.8 on page 14](#) however, is only valid when iterating from the dialog model. When iterating from the event only the substitution variable *Name* returning the name of the method is available.

4.8 Method

This represents a method defined in a dialog model. Only methods used in one or more event in the dialog will be part of the dialog models method nodes.

Substitution variables available at Method level:

Name	Description
Name	Name of the method.

Iterations at the Method level:

Name	Description
Action	The actions defined for a method.

4.9 Action

This represents an action defined in the event dialog of a component, see [section 4.9 on page 14](#) in the Dialog Designer manual.

Substitution variables available at Action level:

Name	Description
ActionName	Name of the action as shown in the action list in the event dialog, see section 16.3 on page 148 in the Dialog Designer manual.

Name	Description
TargetName	Name of the target for the action, see in section 16.4 on page 149 the Dialog Designer manual.
TargetNodeName	Name of the object selection role connected to the target. The target has a connect role if the target itself is an object selection role, if the target is a container component with a connected object selection role or if the target is a data item with connected role attribute.
TargetClassName	Name of the corresponding model class for the object selection role connected to the target.
HasTarget	<i>True</i> if the target is defined for the action.
IsTargetDataItem	<i>True</i> if the target is defined and is a data item.
IsTargetRole	<i>True</i> if the target is defined and is a object selection role.
IsTargetWindowBlock	<i>True</i> if the target is defined and is a window block.
IsTargetDialog	<i>True</i> if the target is defined and is a dialog.
IsTargetObjectSelection	<i>True</i> if the target is defined and is an object selection.
IsTargetMethod	<i>True</i> if the target is defined and is a method from an interface.

If the target is a method from an interface, then the interfaces properties are available as substitution variables via the prefix *Interface*. I.e. *Interface.Name* gives the name of the interface. In addition to the properties shown in the property for an interface, the following substitution variable is available:

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> .
RequiredName	Name of the <i>Required</i> dependency connecting the interface to the dialog. If no name is specified in the UML model, the name of the interface is used.

Iterations at the Action level:

Name	Description
ParameterBinding	The return value and all parameters for an interface method.

The *ParameterBinding* is nonempty only if the action is *Invoke* and the target is the method from an interface.

4.10 ParameterBinding

This represents the parameter bindings for the target method in an *Invoke* action. When iterating the parameter bindings the first binding will represent the methods return value, while the rest represents the parameters in the sequence specified by the method.

A Parameter binding consist of three parts, a formal parameter, an actual parameter and a converter. The substitution variables for the three parts are available via the three prefixes *FormalParameter*, *ActualParameter* and *Converter*. I.e. *FormalParameter.Name* gives the name of the parameter.

Via the *FormalParameter* prefix the properties for either the method or the parameter is available. In addition the following substitution variables with the *FormalParameter* prefix are available at the *ParameterBinding* level:

Name	Description
IsReturnValue	<i>True</i> if the parameter binding represents the methods return value.
ModelTypeIsJavaPrimitive	<i>True</i> if the formal parameters model type is a java primitive type. Note: <i>void</i> is regarded as a java primitive type.
IsClass	<i>True</i> if the formal parameters model type is a class from the domain model.
IsEnumeration	<i>True</i> if the formal parameters model type is an enumeration from the domain model.

If the parameter or the return value is either a class or an enumeration the properties for the class or enumeration are available as substitution variables via the prefixes *Class* or *Enumeration*. I.e. *FormalParameter.Class.Name* gives the name of the class. In addition to the properties shown in the property dialog for a class or an enumeration, both types have the following substitution variable:

Name	Description
PackageName	Name of the package the class/enumeration 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> .

Via the *ActualParameter* prefix the properties for the element bound to the formal parameter are available:

Name	Description
IsRole	<i>True</i> if the actual parameter represent an object selection role.

Name	Description
IsAttribute	<i>True</i> if the actual parameter represent an object selection role attribute.
IsConstant	<i>True</i> if the actual parameter represent constant value.
IsUnbound	<i>True</i> if the formal parameter ain't bound to any actual parameter.
ModelTypeIsJavaPrimitive	<i>True</i> if the actual parameter is an attribute with a model type that is a java primitive type.
Value	Value of the actual parameter if it is a constant.

If the actual parameter is a model element, i.e. a role or an attribute, then all of the elements properties are available as substitution variable. I.e. `ActualParameter.Name` returns the name of the role or the role attribute.

Via the `Converter` prefix the properties for the converter is available. To test if a converter is available the `HasConverter` variable can be used. For the converter all properties shown in the property dialog for a converter are available as substitution variables. In addition the following variables may be used:

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.

4.11 Enumerator

This represents an enumeration in the class model. The enumerator node level is the same as defined in Domain Generator. See [section 4.6 on page 10](#) in the Domain Generator manual.

4.12 ResourceModel

This node level represents the resource model as defined in the resource database.

None substitution variables are available at the ResourceModel level.

Iterations at the ResourceModel level:

Name	Description
Template	All templates defined in the resource database.
Style	All styles defined in the resource database.
Layout	All layouts defined in the resource database.
Color	All colors defined in the resource database.
Font	All fonts defined in the resource database.
Image	All images defined in the resource database.
PrintSetting	All print settings defined in the resource database.
ComponentType	All component types defined in the resource database.

4.13 Template

This represents a template as defined in the resource database, see [Chapter 3 "Template properties" on page 20](#) in the Dialog Resources manual .

4.14 Style

This represents a style as defined in the resource database, see [Chapter 4 "Style properties" on page 24](#) in the Dialog Resources manual.

In addition to the properties shown in the property dialog for a style the properties of the styles background- and foreground color and the styles font is available via the three prefixes *BackgroundColor*, *ForegroundColor*, *Font*. I.e. *Font.Family* gives the family name of the styles font. See [section 4.16 on page 19](#) for further description of substitution variables for colors and [section 4.17 on page 19](#) for further description of substitution variables for fonts.

The following substitution variable is also available for styles:

Name	Description
HasFrame	<i>True</i> if the styles frame setting is either Shadow-in or Etched-in.

No iterations are available at the Style level.

4.15 Layout

This represents a layout as defined in the resource database, see [Chapter 5 "Layout properties" on page 27](#) in the Dialog Resources manual.

None substitution variables except those based on the layouts properties are available.

No iterations are available at the Layout level.

4.16 Color

This represents a color as defined in the resource database, see [Chapter 6 "Color properties" on page 35](#) in the Dialog Resources manual.

In addition to the properties shown in the property dialog for a color the following substitution variables are available:

Name	Description
Red	The colors red part of the RGB-definition given as a decimal number (0-255).
Green	The colors green part of the RGB-definition given as a decimal number (0-255).
Blue	The colors blue part of the RGB-definition given as a decimal number (0-255).
IsTransparent	<i>True</i> if the color is transparent.

No iterations are available at the Color level.

4.17 Font

This represents a font as defined in the resource database, see [Chapter 7 "Font properties" on page 38](#) in the Dialog Resources manual.

In addition to the properties shown in the property dialog for a font the following substitution variables are available:

Name	Description
Family	The family name of the font.
Size	The point size of the font.
IsBold	<i>True</i> if the font has the bold property set.
IsItalic	<i>True</i> if the font has the italic property set.

No iterations are available at the Font level.

4.18 Image

This represents an image as defined in the resource database, see [Chapter 8 "Image properties" on page 41](#) in the Dialog Resources manual.

None substitution variables except those based on the images properties are available.

No iterations are available at the Image level.

4.19 PrintSetting

This represents a print setting as defined in the resource database, see [Chapter 10 "Print Setting properties" on page 58](#) in the Dialog Resources manual.

In addition to the properties shown in the property dialog for a print setting the properties of the style is available via the prefix *Style*. Through this prefix the background- and foreground color and the font of the style is available. I.e. *Style.Font.Family* gives the family name of the styles font. See the three sections [section 4.14 on page 18](#) for further description of substitution variables for styles.

No iterations are available at the PrintSetting level.

4.20 ComponentType

This represents a component type as defined in the resource database, see [Chapter 11 "Component Type properties" on page 60](#) in the Dialog Resources manual.

None substitution variables except those based on the component type properties are available.

No iterations are available at the ComponentType level.

5 Dialog Client implementations

In this chapter, issues specific to the different client implementations supported by the Dialog Generator are covered.

5.1 Jasper Report Templates

JasperReports is an open-source Java library for creating reports from within a java environment. The Jasper Report Templates target will create .jrxml files, which are report templates based on dialog models.

See [Chapter 4 "The JasperReports target" on page 8](#) in the Dialog Print manual for a description of the Jasper Report Templates target.

5.2 Dialog Report

Dialog Report is a tool that is used to produce dialog specifications that will report selected information from dialog models. The dialog reports are generated with the Dialog Generator and the specifications are based on a set of HTML templates. The dialog reports are HTML files.

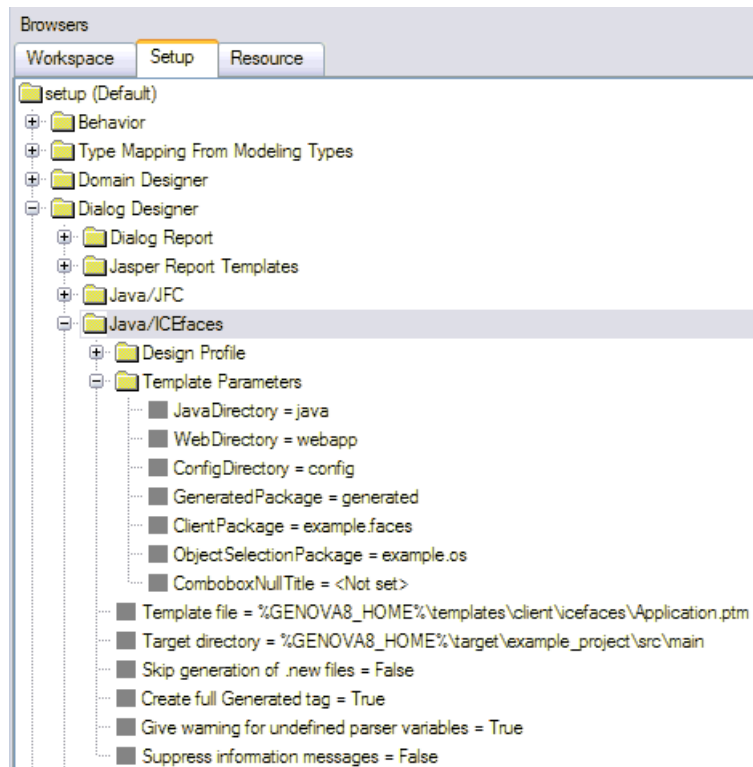
See [Chapter 18 "Dialog Reports" on page 155](#) in the Dialog Designer manual for a description of the Dialog Report target.

5.3 Java ICEfaces

Java ICEfaces is a target for the JSF based Ajax framework ICEfaces. The ICEfaces target may replace the Java/Jfc target as the client part of a generated Java application while keeping the service and dataaccess part of the application as is. See the [Chapter 7 "Client Generator for the Java/ICEfaces target" on page 49](#) in the Java Target manual for a detailed description of the ICEfaces target.

5.3.1 Template parameters

The Java ICEfaces 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, WebDirectory and ConfigDirectory: By default the templates are directing the output into three subdirectories under the target directory: *java*, *webapp* and *config*. The *java* directory will contain any generated java source, while *webapp* will contain the JSF configuration and markup files. A sample configuration file for the Jetty server is generated to the *config* directory. 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 three variables should be given values. Setting these variables to the empty string will direct output into the target directory without using any substructure.

GeneratedPackage: The package for the superclasses for the controller and view for each dialog, and the constants for the dialog contents. This package will be a sub package under the package for the rest of the generated files for a dialog. The files in the *GeneratedPackage* will be overwritten each time the dialog is generated.

ClientPackage: The top level package for generated files for dialogs.

ObjectSelectionPackage: The top level package for generated support files for the object selections used in the dialogs.

ComboboxNullTitle: The title to be used as the default selection in combo boxed, i.e. a selection representing the Java *null* value. Setting this parameter to the empty string bypasses this functionality, and the first selection in the combo box is the first value of the data item if it is of the enumeration data type.