ACTIF®
SOURCE

User Manual

	Actifsource User Manual
Notation	✍ To do ⓘ Information • **Bold**: Terms from actifsource or other technologies and tools • **Bold underlined**: actifsource Resources • Underlined: User Resources • *UnderlinedItalics*: Resource Functions • `Monospaced`: User input • *Italics*: Important terms in current situation
Disclaimer	The authors do not accept any liability arising out of the application or use of any information or equipment described herein. The information contained within this document is by its very nature incomplete. Therefore the authors accept no responsibility for the precise accuracy of the documentation contained herein. It should be used rather as a guide and starting point.
Contact	**Actifsource AG** Twärfallenstrasse 3 6313 Finstersee Switzerland www.actifsource.com
Trademark	**Actifsource** is a registered trademark of **Actifsource AG** in Switzerland, the EU, USA, and China. Other names appearing on the site may be trademarks of their respective owners.
Revision	• 2014/01/14 [rc] 5.11.0 • 2014/02/14 [rc] 5.12.0 • 2014/02/19 [rc] 5.13.0 • 2014/03/28 [rc] 5.13.1 • 2014/04/02 [rc] 5.14.0 (Template Editor) • 2014/04/15 [rc] 5.14.0 (Accessing the model from within Java function) • 2014/12/23 [sw] Code Snippet Chapter • 2015/03/23 [sw] Java (List) Functions • 2015/05/01 [ms] Java API • 2017/03/30 [gr] Generic Import Wizard • 2017/04/20 [gr] Generic Import Wizard Long Support • 2019/05/29 [gr] Context Sensitive Help

Table of Content

1 Overview

1.1 Working with models

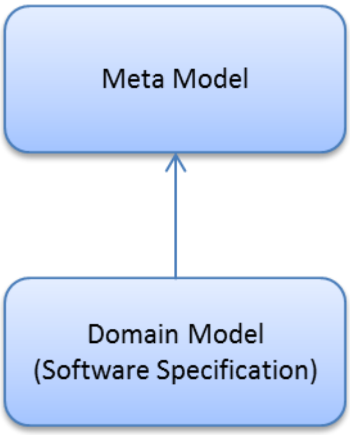
Actifsource is a comprehensive design and code generator tool, covering all aspects of domain-driven software development from domain analysis through to the design models, code generating, testing, refactoring and maintenance.

Actifsource allows you to define your domain-specific software specification.

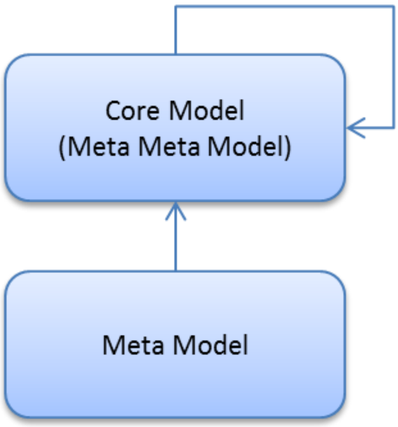
Let's Build Your **Domain Specific Modeling Tool**
That Turns Your **Software Specification** Into **Running Code**.

This software specification is also called *domain model*, or *specific model*. The domain model shall be independent from any used technology (i.e. programming language, operating system, etc.).

As the domain model is domain-specific by definition, we need to specify the structure for every domain. This task is done in the so called *meta-model*.



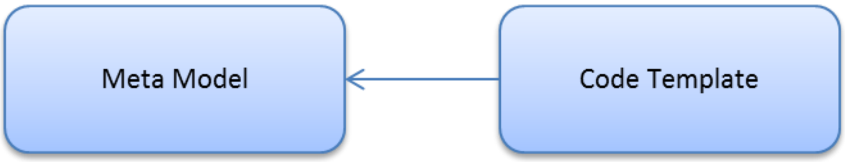
The meta-model is built upon concepts of the Actifsource *core model* (aka Meta Meta-Model). Note that the core model is self-describing, i.e., it is the meta-model of itself.



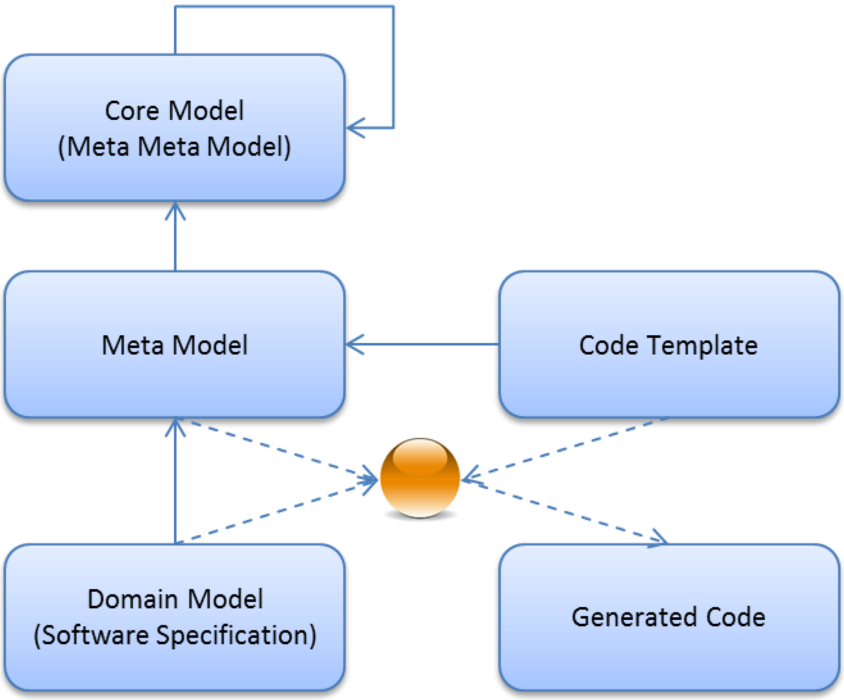
To find an adequate meta-model, you need to analyze your business domain. If you like to create a Service Oriented Architecture (SAO), your meta-model might will contain services. If you like to model state machines, your meta-model will define states, event and transitions.

Note that the Meta-Model is an abstraction of your business domain, and defines the business classes and their relationships.

As we do not know your meta-model in advance, we cannot generate any suitable code for you. This means, that you have to define your own *code templates* according to the meta-model. Defining code templates is as easy as writing normal code.



Since everything depends on the meta-model you have to start by analyzing your business domain. Once the meta-model is defined, you are able to enter you domain specific software specification accordingly. Also template code is written along the meta-model. From this three models (meta-model, domain model, code template) your code is generated by the Actifsource code generator.



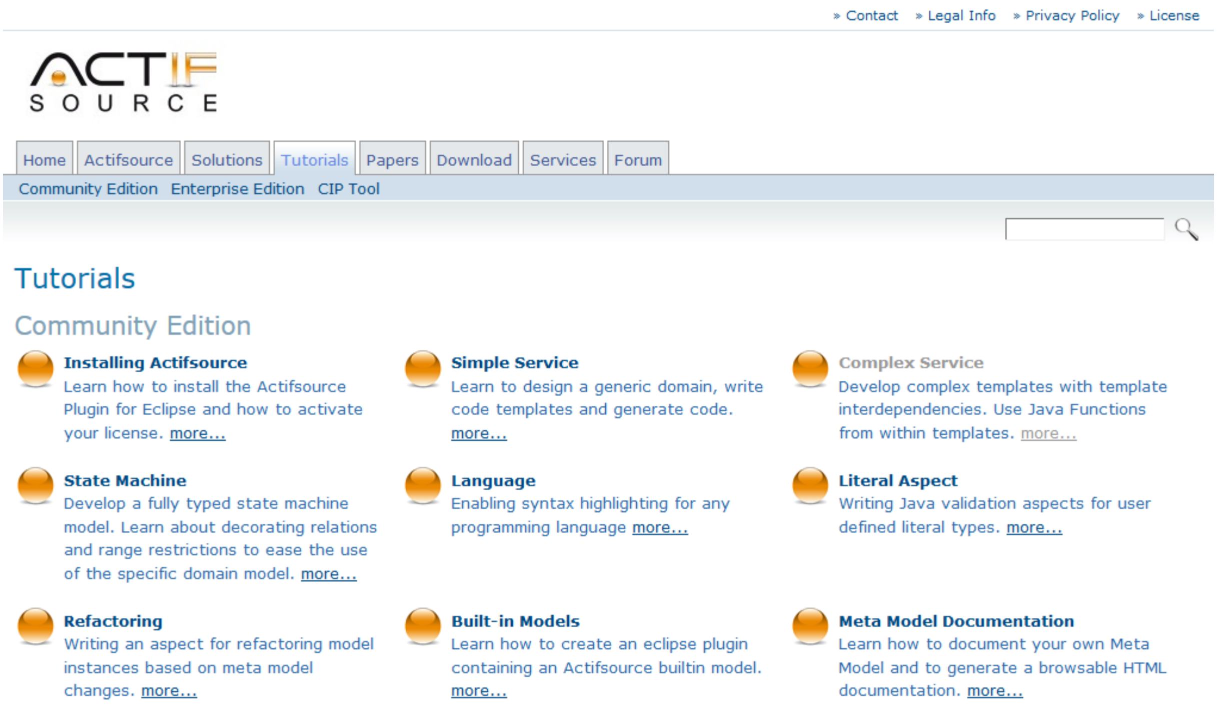
1.2 Resource

Every model consists of so called *resources*. A resource is like an object and the most abstract entity from the Actifsource core model. In fact, every model element is a resource.

Every resource is identified by a *globally unique identifier* (GUID) which is automatically assigned, if you create a new resource. Therefore changing resource names never affects any relation between resources.

1.3 Getting started

To get started with Actifsource, we suggest our tutorials on the Actifsource web site. Please visit http://www.actifsource.com/tutorials.


» Contact » Legal Info » Privacy Policy » License
ACTIF SOURCE
Home | Actifsource | Solutions | Tutorials | Papers | Download | Services | Forum
Community Edition Enterprise Edition CIP Tool

Tutorials

Community Edition

Installing Actifsource
Learn how to install the Actifsource Plugin for Eclipse and how to activate your license. more…

Simple Service
Learn to design a generic domain, write code templates and generate code. more…

Complex Service
Develop complex templates with template interdependencies. Use Java Functions from within templates. more…

State Machine
Develop a fully typed state machine model. Learn about decorating relations and range restrictions to ease the use of the specific domain model. more…

Language
Enabling syntax highlighting for any programming language more…

Literal Aspect
Writing Java validation aspects for user defined literal types. more…

Refactoring
Writing an aspect for refactoring model instances based on meta model changes. more…

Built-in Models
Learn how to create an eclipse plugin containing an Actifsource builtin model. more…

Meta Model Documentation
Learn how to document your own Meta Model and to generate a browsable HTML documentation. more…

2 Actifsource Environment

2.1 Actifsource Eclipse Plugin

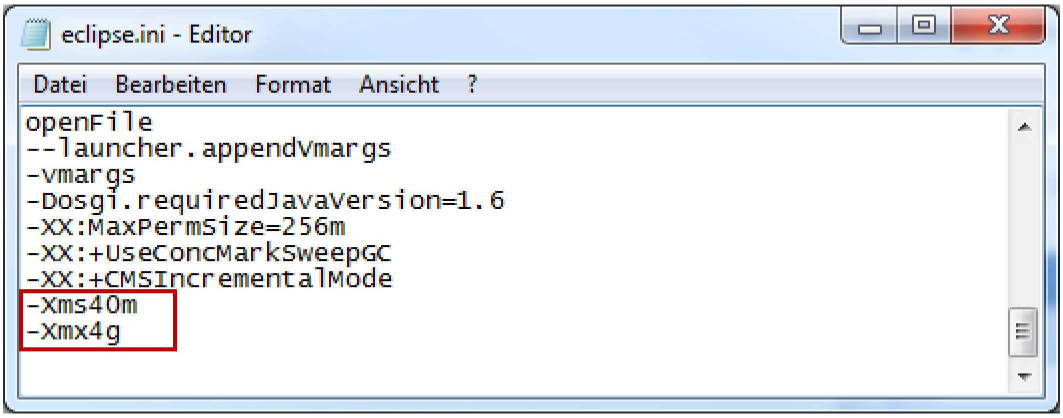
Actifsource is shipped as *Eclipse Plugin*. Please make sure to install Eclipse first. Choose *Help/Install new Software…* to install Actifsource from one of the following *Eclipse Update Sites*:

- http://www.actifsource.com/updates (Community Edition)
- http://www.actifsource.com/updates-enterprise (Enterprise Edition: password protected)

If you are not familiar with the Eclipse environment, please consult the *Actifsource Tutorial - Installing Actifsource*.

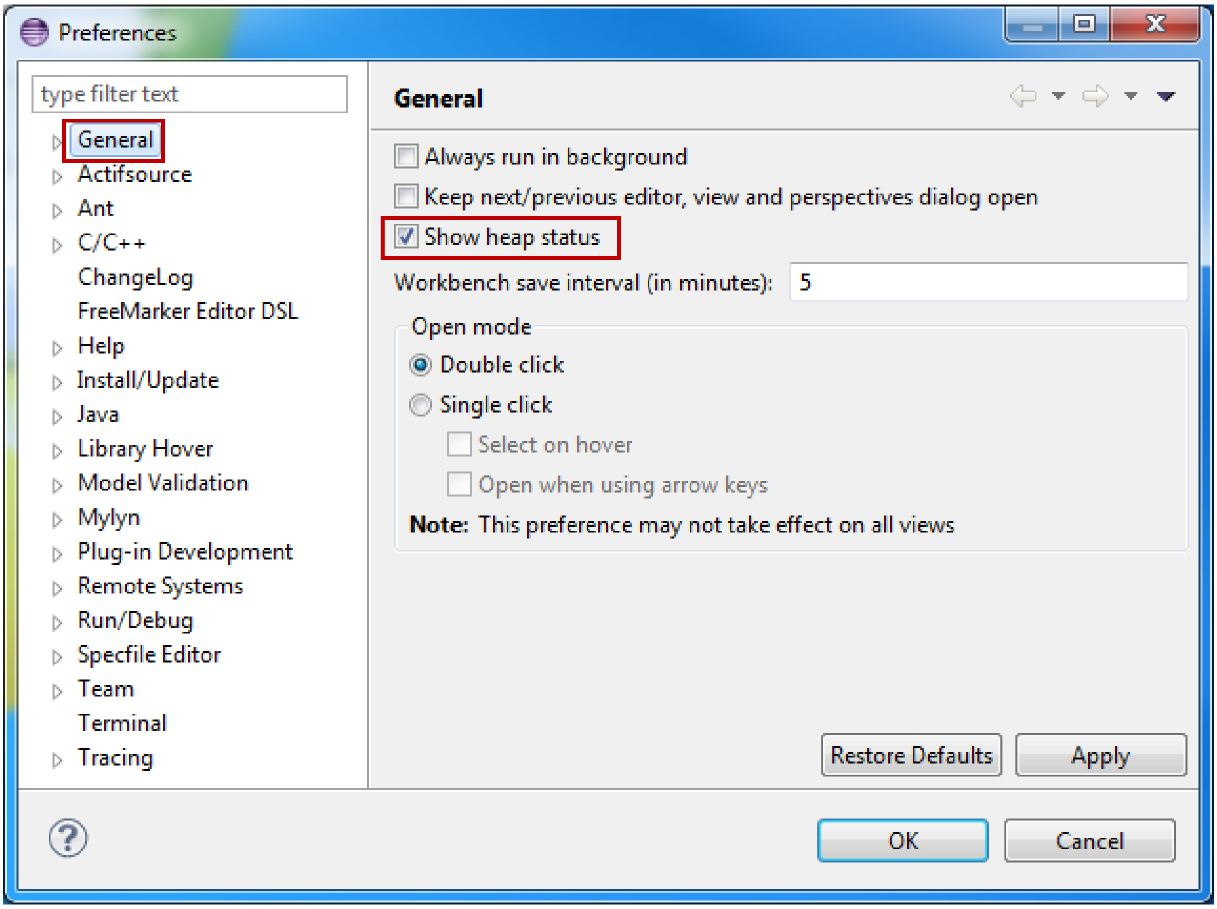
2.2 Memory Usage

The Actifsource technology allows you to track any keystroke in real-time. As a result of this feature the memory consumption might be quite high for larger models. Make sure to adopt the memory given to Eclipse if necessary in the *eclipse.ini* file in the Eclipse directory. We suggest at least 4 GB of ram.



```
openFile
--launcher.appendVmargs
-vmargs
-Dosgi.requiredJavaVersion=1.6
-XX:MaxPermSize=256m
-XX:+UseConcMarkSweepGC
-XX:+CMSIncrementalMode
-Xms40m
-Xmx4g
```

To observe the memory while working with Actifsource enable *Window/Preferences/General/Show heap status*.



Preferences

type filter text

- General
- Actifsource
- Ant
- C/C++
- ChangeLog
- FreeMarker Editor DSL
- Help
- Install/Update
- Java
- Library Hover
- Model Validation
- Mylyn
- Plug-in Development
- Remote Systems
- Run/Debug
- Specfile Editor
- Team
- Terminal
- Tracing

General

□ Always run in background

□ Keep next/previous editor, view and perspectives dialog open

☑ Show heap status

Workbench save interval (in minutes): 5

Open mode

● Double click

○ Single click

□ Select on hover

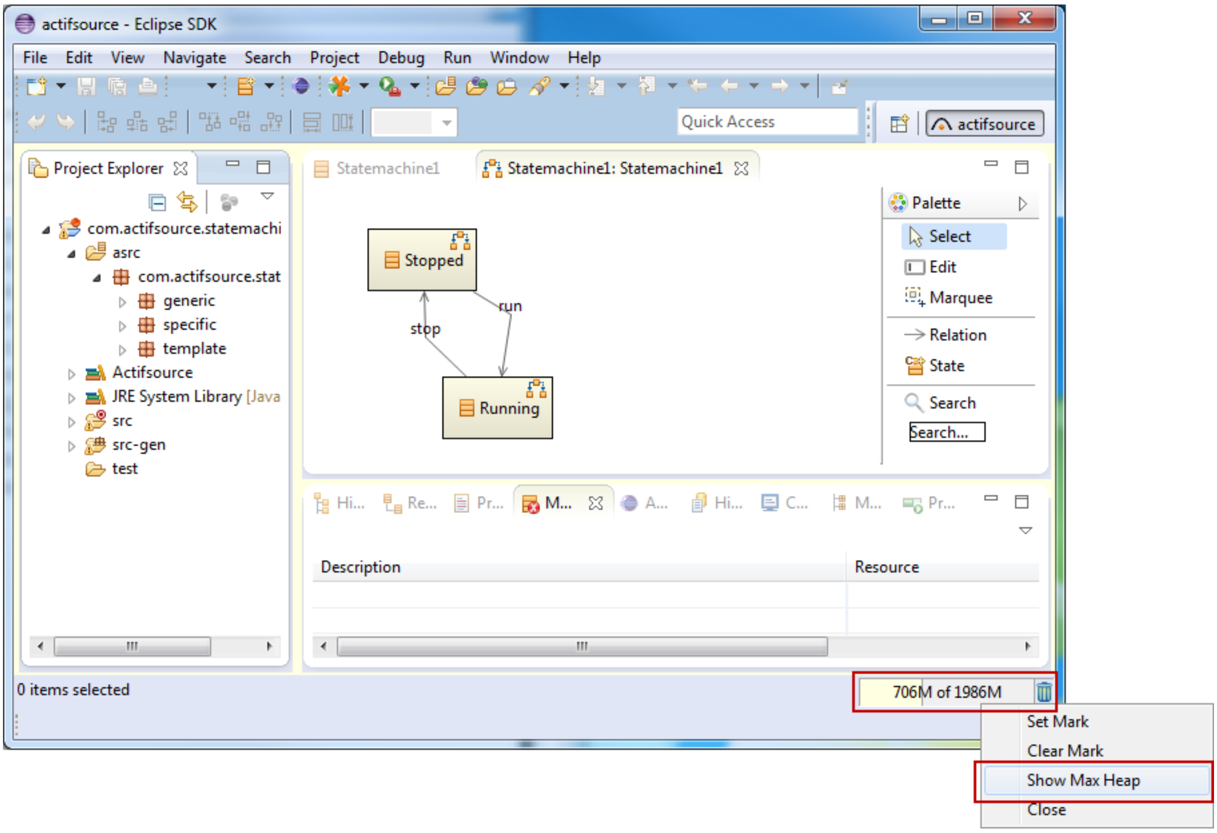
□ Open when using arrow keys

Note: This preference may not take effect on all views

Restore Defaults | Apply

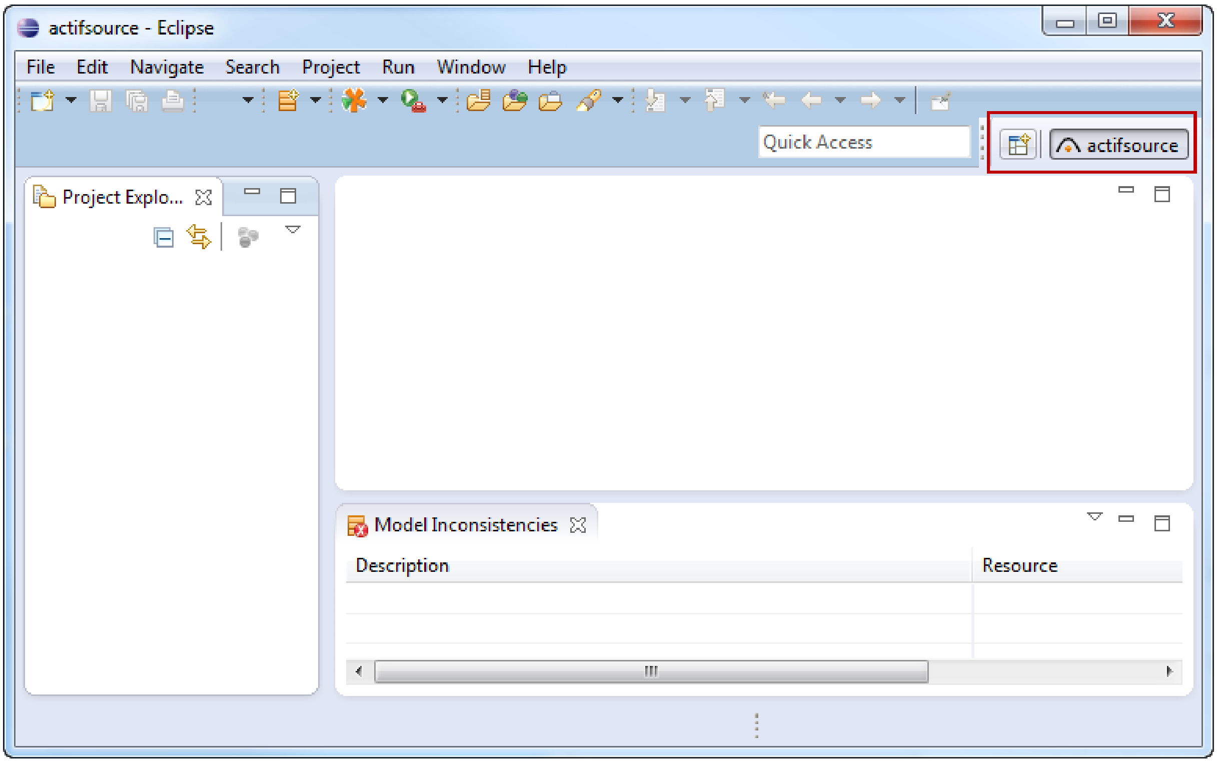
OK | Cancel

Make sure to enable *Show Max Heap* to track the maximum memory usage by using the context menu on the heap status display.



2.3 Perspectives

Since Eclipse is a general software development tool, you must be able to select different perspectives. Make sure that the *Actifsource Perspective* is selected when working with Actifsource.

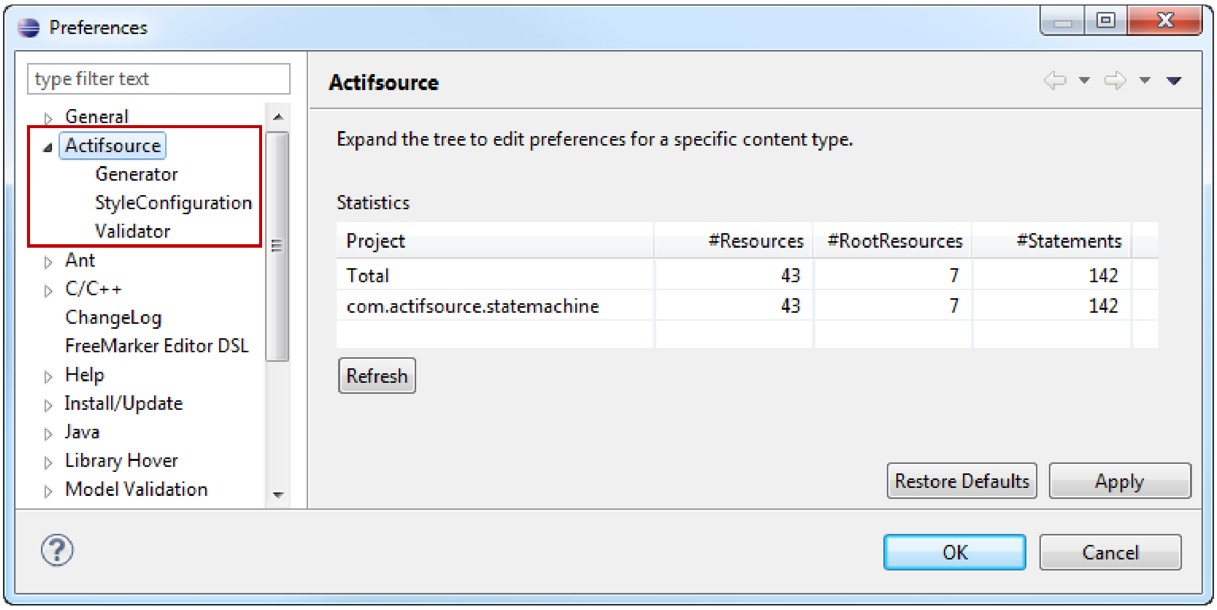


2.4 Actifsource Preferences

The preferences dialog (*Window/Preferences/Actifsource*) provides the following configuration options.

2.4.1 Statistics

Selecting the Actifsource menu shows a statistic of the resources.



Expand the tree to edit preferences for a specific content type.

Statistics

Project	#Resources	#RootResources	#Statements
Total	43	7	142
com.actifsource.statemachine	43	7	142

#Resources

The number of resources in the project.

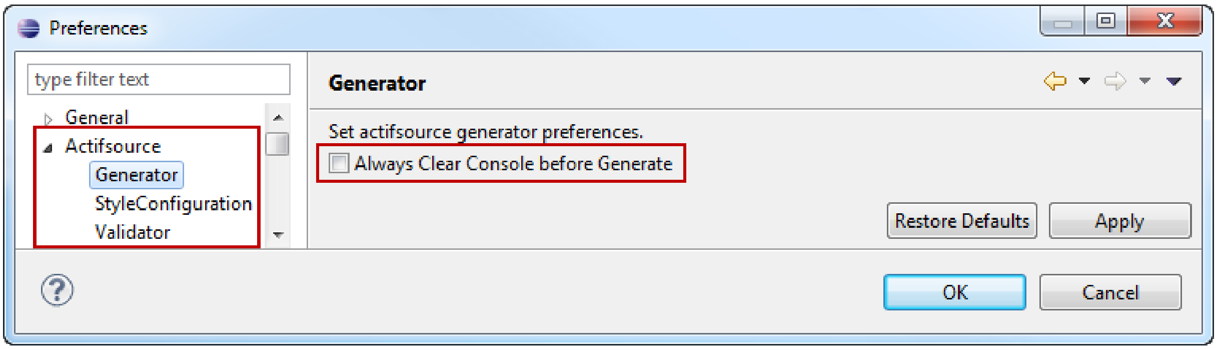
#RootResources

The number of root resources in the project. A root resource is a non-aggregated resource.

#Statements

The number of statements in the project. The statement declares three resources as follows: Subject-Predicate-Object.

2.4.2 Generator



Set actifsource generator preferences.

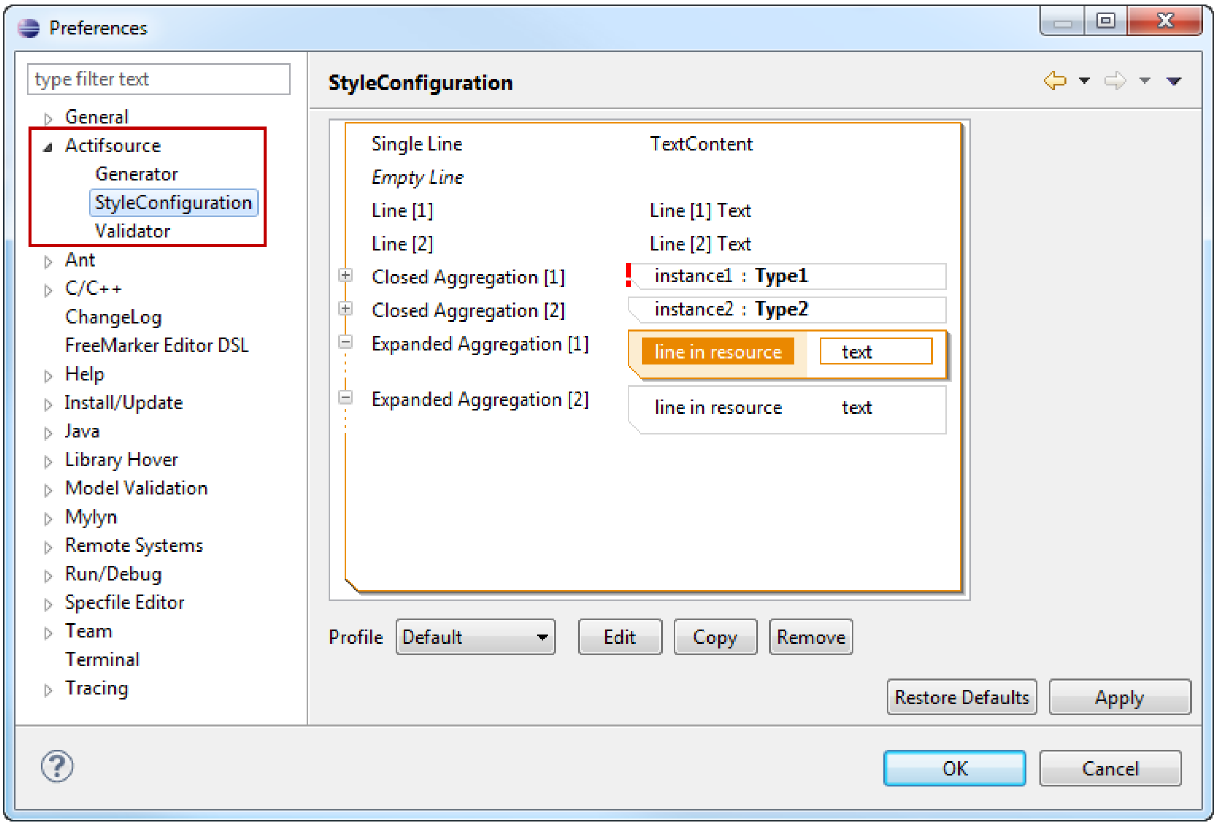
□ Always Clear Console before Generate

Always clear generator console before generate

Actifsource cleans the console output before generating code. This makes it easier to scroll to the top of the output to find error messages.

2.4.3 Style Configuration

The style configuration let you define your own colors.



Profile: Default

Profile

Shows all built-in and user-defined profiles.

Edit

Edit user-defined profiles. Note that you cannot edit built-in profiles.

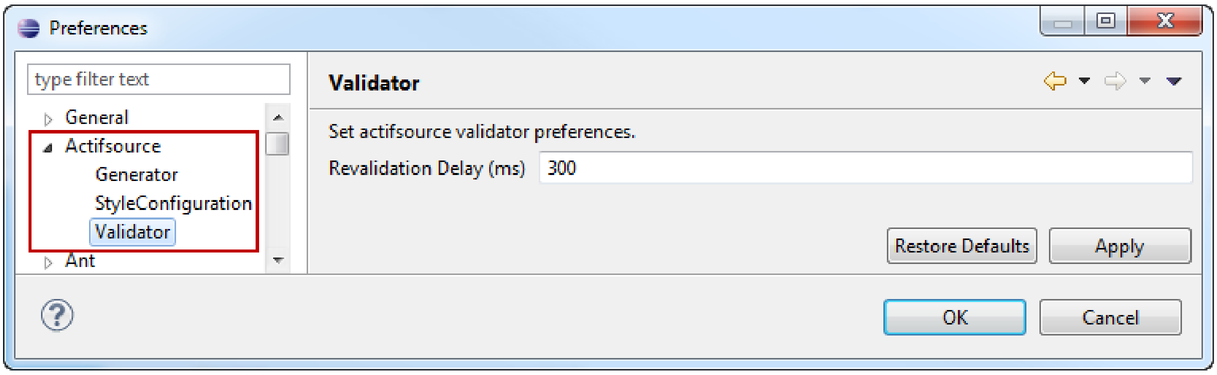
Copy

Copy built-in or user-defined profiles. Use copy on a built-in profile to create a user-defined profile.

Remove

Removes user-defined profiles. Note that you cannot remove built-in profiles.

2.4.4 Validator



Preferences

Validator

Set actifsource validator preferences.

Revalidation Delay (ms) 300

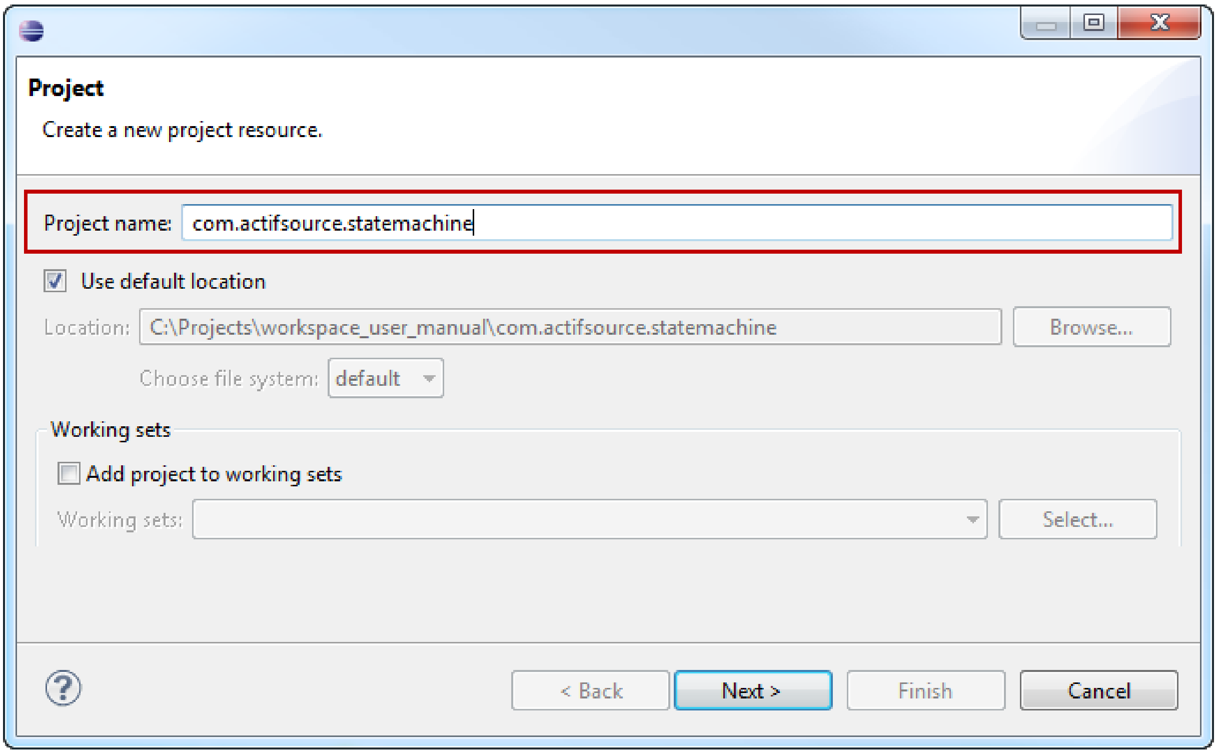
Revalidation Delay (ms)

Actifsource validates every keystroke. The validation might lead to a high CPU load for large models. For this reason you can configure the delay between validations.

2.5 Project Wizard

The project wizard allows you to create a new Actifsource project from scratch. Select *File/new/Actifsource Project*.

2.5.1 Project Name and Location



Project

Create a new project resource.

Project name: com.actifsource.statemachine

☑ Use default location

Location: C:\Projects\workspace_user_manual\com.actifsource.statemachine

Choose file system: default

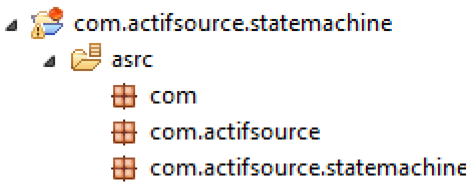
Working sets

□ Add project to working sets

Working sets:

Project name

This is the name of the new project. We suggest that the project name is given in the eclipse-like manner: *com.company.project.subproject*. Note that the dotted name automatically leads to a corresponding package structure.



```
com.actifsource.statemachine
  asrc
    com
    com.actifsource
    com.actifsource.statemachine
```

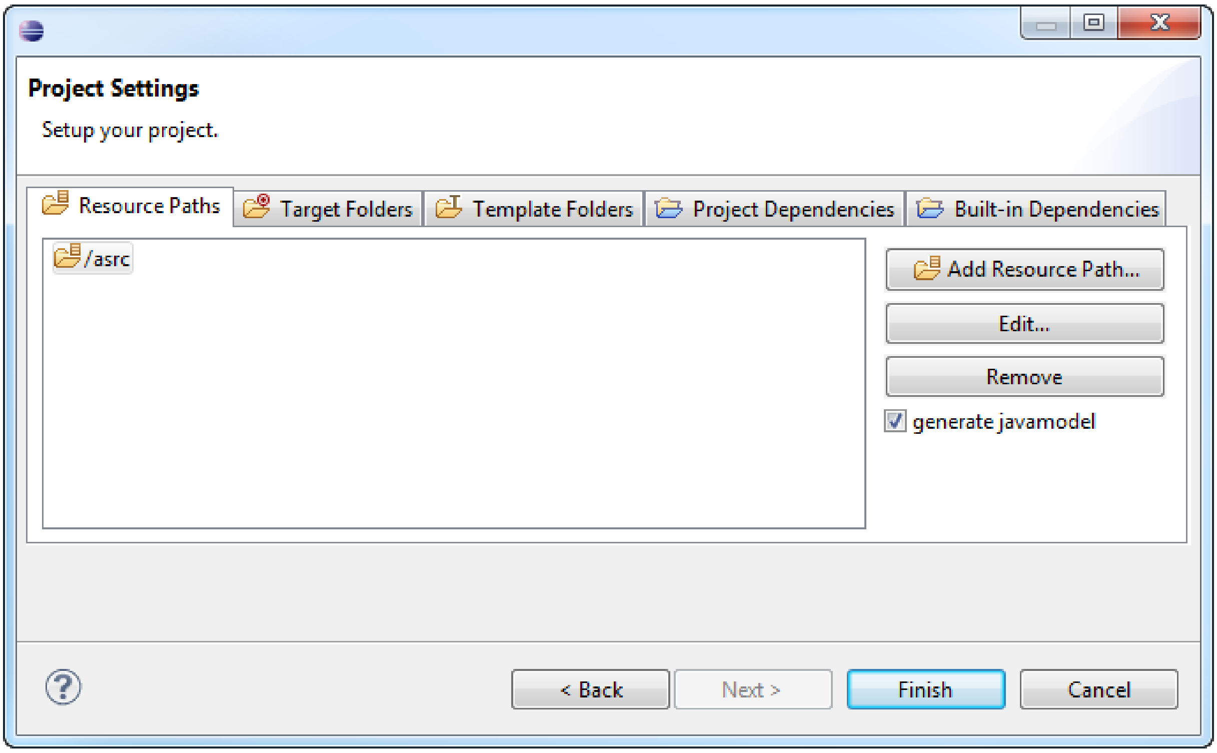
Location

This is the location of the project. The default location is in the workspace.

Working Set

You might add the project to an existing *working set*. A working set is a dedicated view to the projects of the workspace.

2.5.2 Resource Paths



Project Settings

Setup your project.

/asrc

☑ generate javamodel

All Actifsource resources are saved in *Resource Files* with the ending .asr in an xml format. The *resource path* defines where to find the model resources.

Add Resource Path

Adds a new resource path to the project.

Edit...

Edits an existing resource path.

Remove

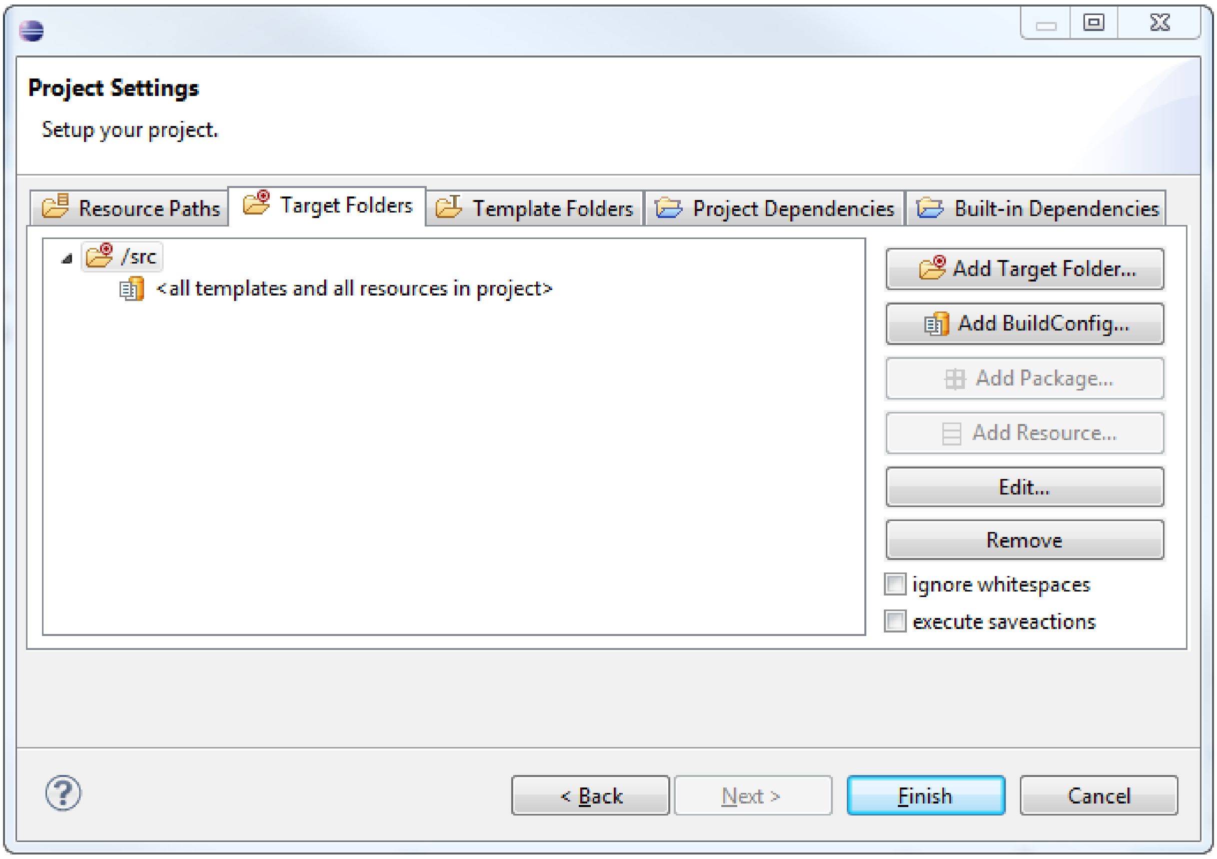
Removes an existing resource path.

Generate javamodel

Actifsource builds internal Java classes to handle your model. Do not switch off this option unless you know exactly what you do.

2.5.3 Target Folder

Generated code is written to target folders. You might specify any existing or new folder in your project as a target folder.



Project Settings

Setup your project.

Add Target Folder…

Adds a new target folder to your project.

Add Build Config

Adds a new build configuration to your target folder.

A build configuration is kind of a make file that tells actifsource which templates to build. If no build configuration is defined, Actifsource automatically generates code for all templates from the current project, combined with all matching resources from the current project (see Chapter 7 Build Config, see Chapter 8 Template Editor).

Add Package…

Adds a new package to your build configuration.

Code is only generated for matching resources found in the specified packages.

- com.actifsource.statemachine.specific.* (all resources in the package)
- com.actifsource.statemachine.specific.** (all resources in the package and its subpackages)

If no package is defined, Actifsource generates code for all matching resources found in all packages of the current project.

Note that you need to reference packages from other projects explicitly. Set the project dependencies first accordingly.

Add Resource…

Adds a single resource to your build configuration.

If no resource is defined, Actifsource generates code for all matching resources found in all packages of the current project.

Note that you need to reference resources from other project explicitly. Set the project dependencies first (see Chapter 2.5.5 Project Dependencies).

Edit…

Edits the current entry.

Remove

Removes the current entry.

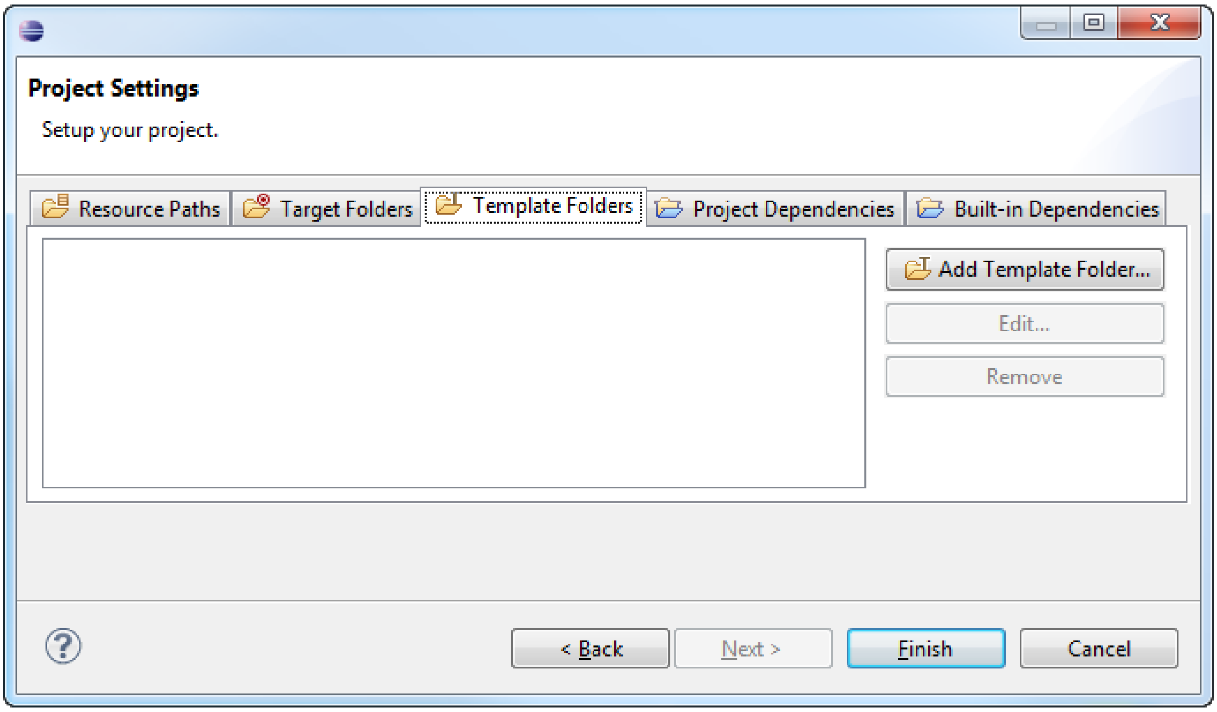
Ignore Whitespaces

Actifsource calculates a checksum (MD5 hash) for every generated file. If this option is checked, Actifsource will ignore whitespaces when calculating the checksum.

Execute Save Actions

Eclipse supports so called *Save Actions* after a file has been saved (i.e. code formatting). If this option is checked, save actions are executed after generating the files.

2.5.4 Template Folders

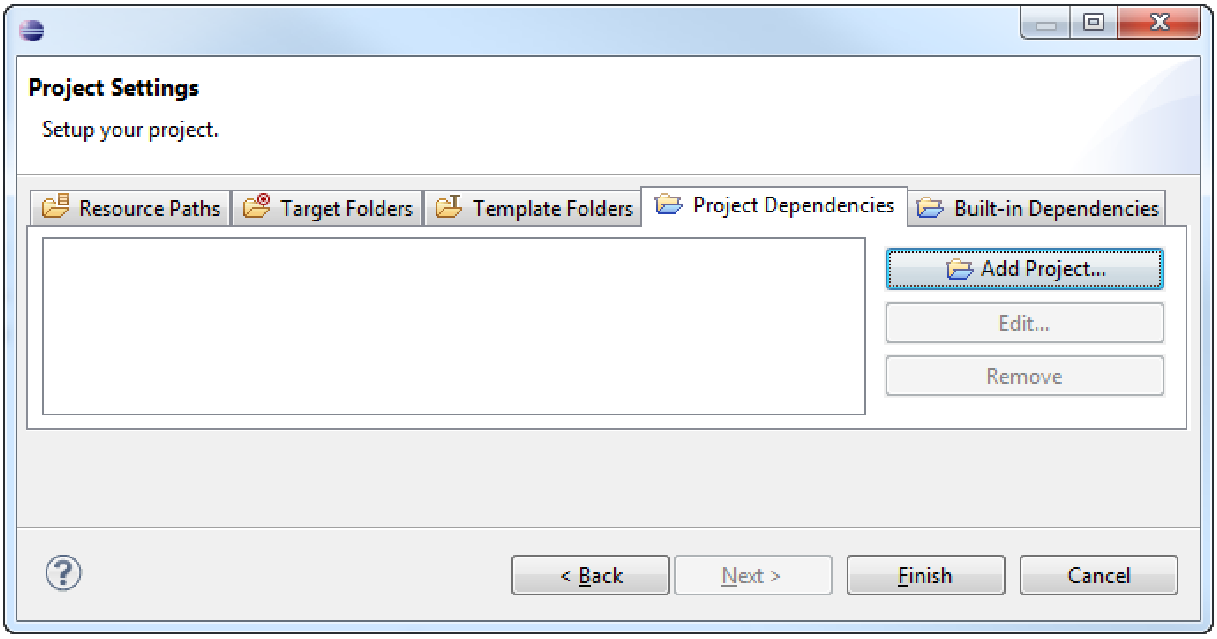


Project Settings

Setup your project.

This feature is for beta users and developers only and might be used to reference folders for templates of third party products.

2.5.5 Project Dependencies



Project Settings

Setup your project.

Use the project dependencies if you like to split your model in different Actifsource projects. Note that you have to set the project dependencies, before you might reference packages and resources from other projects in the target folder.

Add Project

Adds a new project dependency.

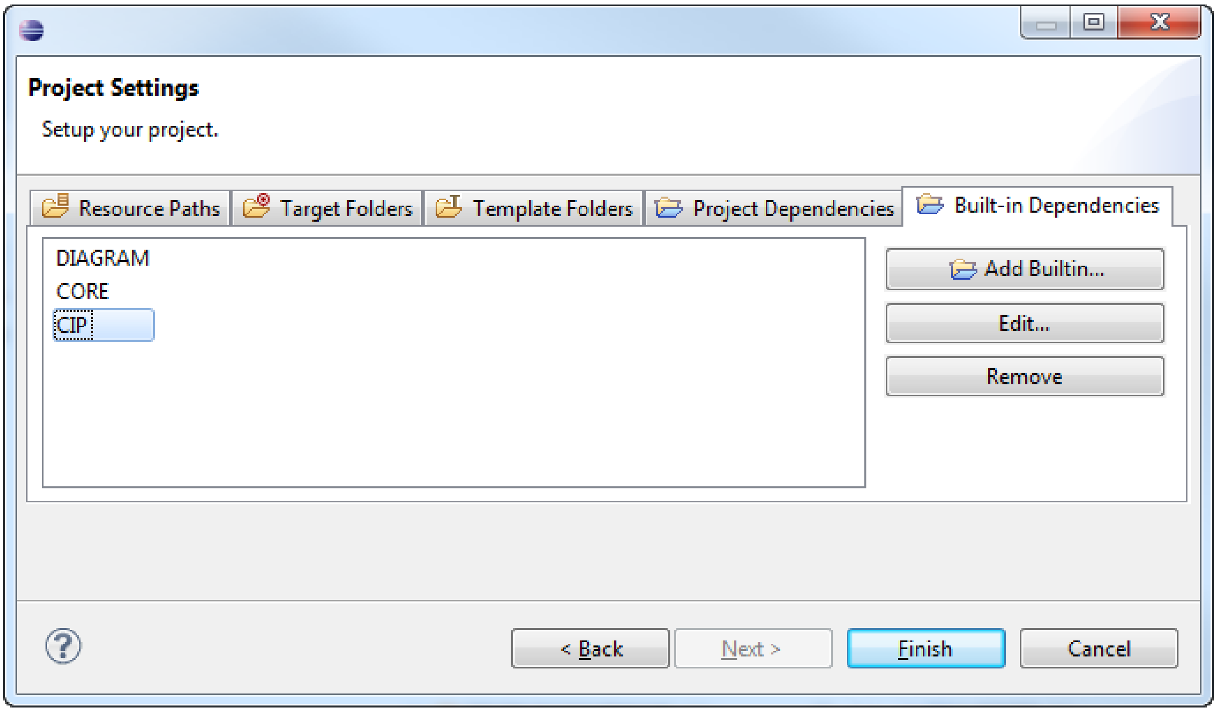
Edit…

Edits the current project dependency.

Remove

Removes the current project dependency.

2.5.6 Built-in Dependencies



Project Settings

Setup your project.

DIAGRAM
CORE
CIP

Use the built-in dependencies to reference any Actifsource built-in models.

Built-In	Description
CORE	The Actifsource core model (do not remove)
DIAGRAM	Domain Diagram
CIP	Embedded real-time state engine
DEC	Modelling I/O connection and scheduling for embedded systems
DOCUMENTATION_METAMODEL	Creates meta-model documentation from class diagrams
JAVAMODEL	Creates Java classes for model access (shipped with Core built-in)
GRAPHVIZ	Generator for graphviz
WORKSPACE	File/folder operations and generating Eclipse projects
DATATYPE	Common data type meta-model
UML	UML state engine and code generator
ECORE	ECore meta-model
MODVIS	Visualization and animation of domain diagrams in the web browser
FREEMARKER	Generator for freemarker templates
XPAND	Generator for xpand templates

Add Builtin

Adds a new built-in project dependency.

Edit…

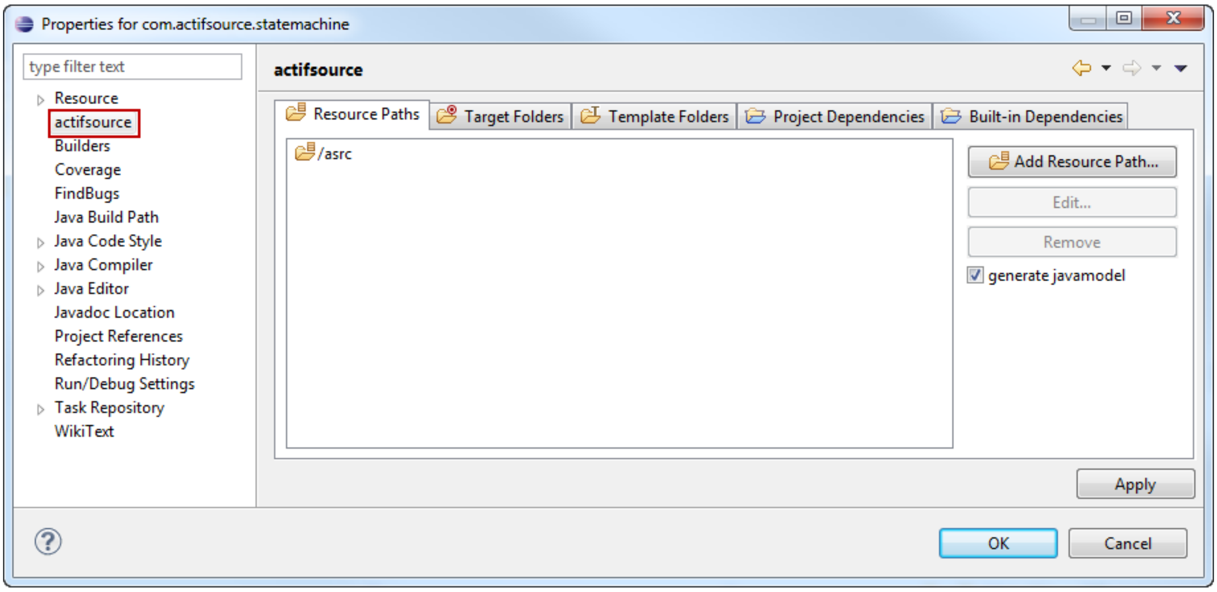
Edits the current built-in project dependency.

Remove

Removes the current built-in project dependency.

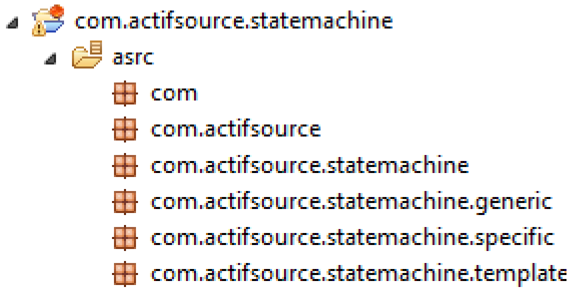
2.6 Project Properties

All settings shown in Chapter 2.5 Project Wizard can be found in *Project/Properties/Actifsource*.



2.7 Standard Package Structure

We suggest the following package structure.



```
com.actifsource.statemachine
  asrc
    com
    com.actifsource
    com.actifsource.statemachine
    com.actifsource.statemachine.generic
    com.actifsource.statemachine.specific
    com.actifsource.statemachine.template
```

Generic

The generic package contains the *meta-model*.

Specific

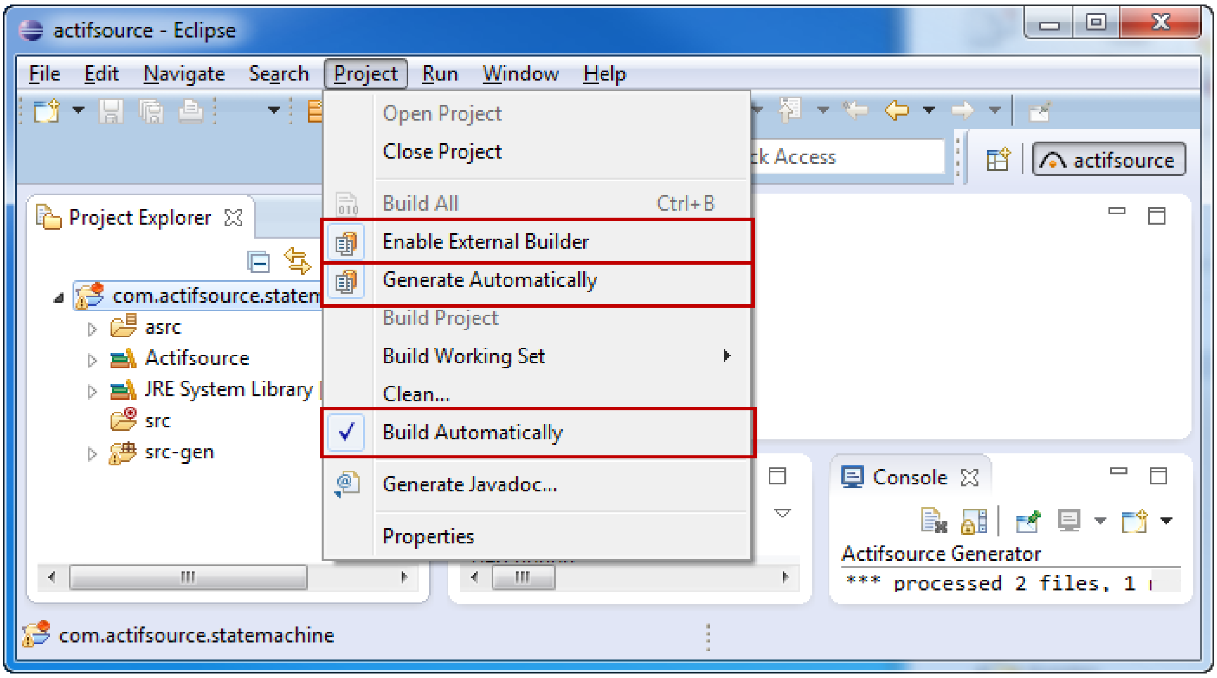
The specific package contains the *domain model*.

Template

The template package contains the *code templates*.

2.8 Project Menu

Use the project menu to control the build system of Eclipse and Actifsource.



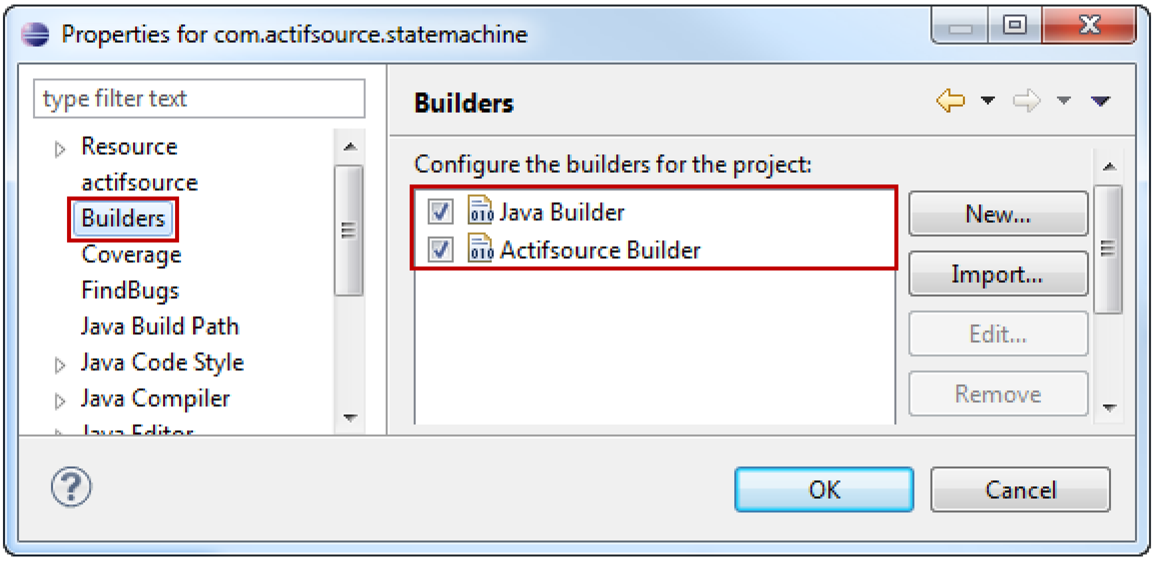
```
*** processed 2 files, 1
```

Build Automatically

If *Build Automatically* is switched on, Eclipse will build the project automatically after changed files have been saved.

Actifsource also generates Java classes for internal use. Generating these internal classes is also switched off by *Build Automatically* and you can't expect Actifsource to work correctly. Therefore, please make sure that *Build Automatically* is switched on.

Please make yourself familiar with the *Eclipse Builder* concept. See *Project/Properties/Builders* to see the active builders for your project and their execution order (see Chapter 7.7 Eclipse Builder).



Configure the builders for the project:

☑ Java Builder
☑ Actifsource Builder

Generate Automatically

If *Generate Automatically* is switched on, Actifsource will generate code after saving changed files. If Generate Automically is not switched on, you have to trigger code generation manually (Right-click on the Eclipse project and select *Generate Code*).

Actifsource also generates Java classes for internal use. Generating these internal classes is not switched off by *Generate Automatically*.

Enable External Builder

This feature is for beta users and developers only. Since Actifsource is developed by Actifsource, we must be able to build ourselves with the current version. The external builder is compiled at development time and ensures that every change in the Actifsource workspace affects the next code generation run.

2.9 Toolbar

The Actifsource toolbar provides you with two important tools.

2.9.1 New Actifsource Resource

The new resource tool lets you create a new Actifsource resource in the selected package. The new resource tool only allows creating so called root classes. [REF]

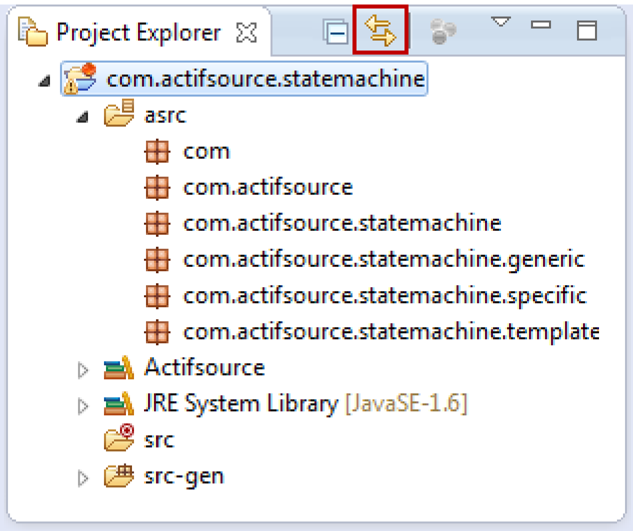
2.9.2 Open Actifsource Resource

This tool shows all resources and allows filtering by name. Please note that this operation might be slow for a large amount of resources.

2.10 Project Explorer

The project explorer let you access your resource files. Resources are stored as xml files and named by the GUID (Globally Unique Identifier) of the resource. Since this format is incomprehensible for humans, the Project Explorer shows the name for named resources.

2.10.1 Link with Editor

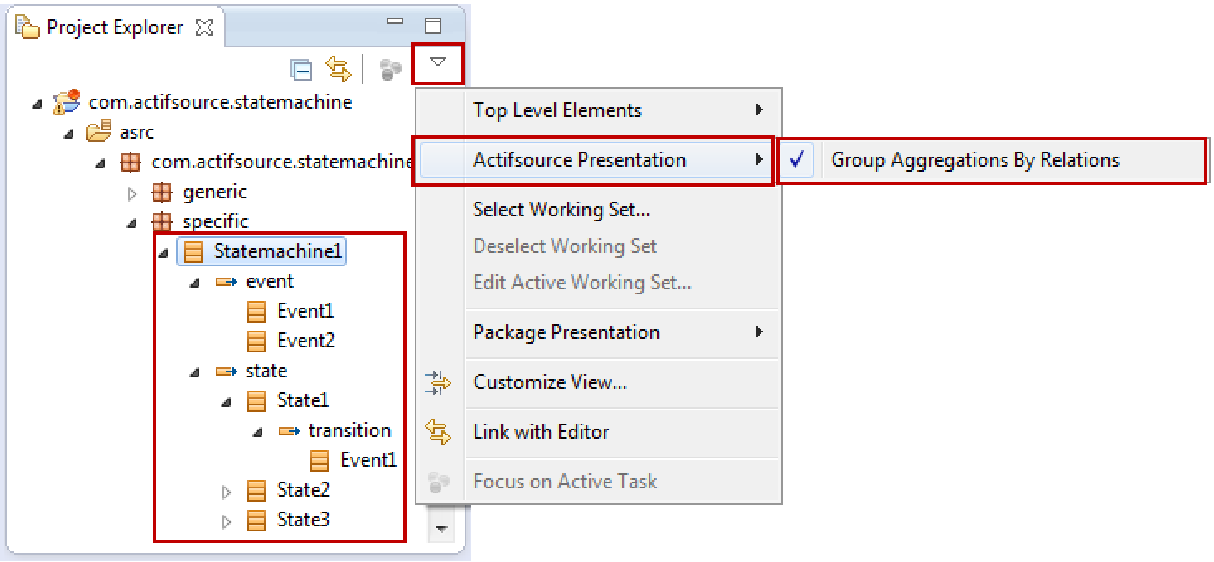

Project Explorer
com.actifsource.statemachine
asrc
com
com.actifsource
com.actifsource.statemachine
com.actifsource.statemachine.generic
com.actifsource.statemachine.specific
com.actifsource.statemachine.template
Actifsource
JRE System Library [JavaSE-1.6]
src
src-gen

Switch on the option *Link with Editor* to synchronize the project explorer with the currently active editor.

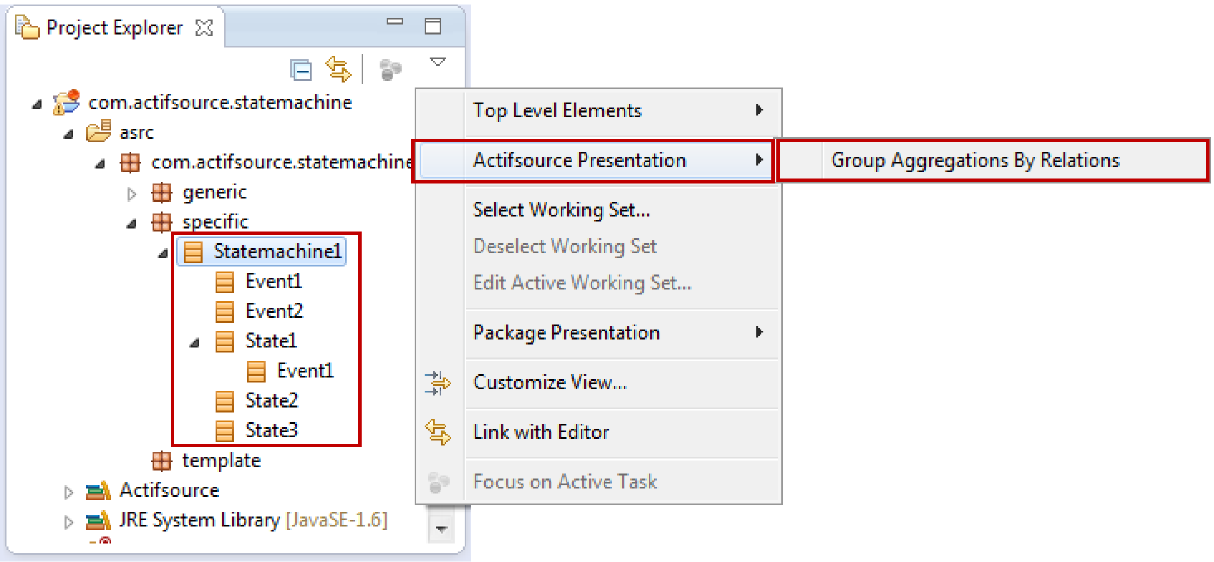
2.10.2 Actifsource Presentation

Actifsource allows you to show aggregated resources sorted by relation or just by their occurrence in the containing class.

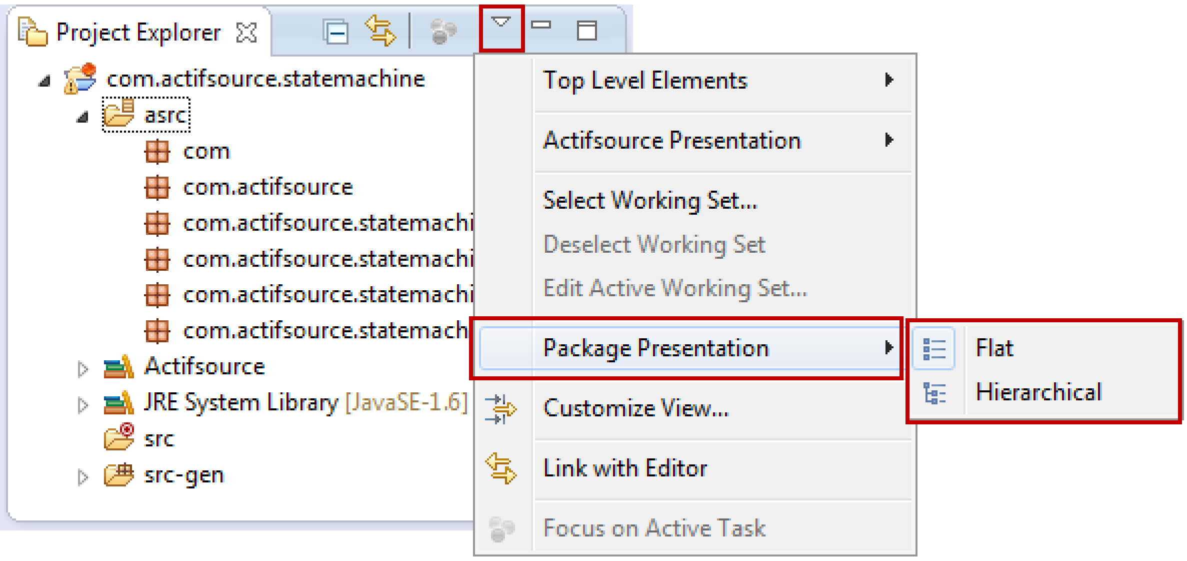
Group Aggregation By Relation switched on



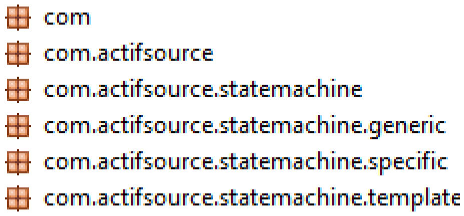
Group Aggregation By Relation switched off



2.10.3 Package Presentation



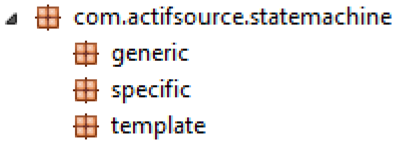
Flat Package Presentation



com
com.actifsource
com.actifsource.statemachine
com.actifsource.statemachine.generic
com.actifsource.statemachine.specific
com.actifsource.statemachine.template

Using the flat package presentation, all packages are shown as a flat list.

Hierarchical Package Presentation

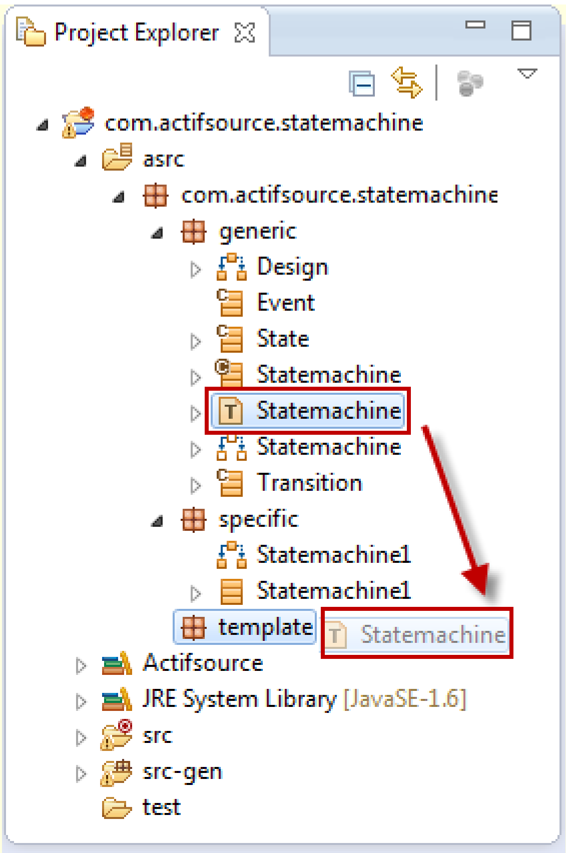


com.actifsource.statemachine
generic
specific
template

Using the hierarchical package presentation, the hierarchy is preserved. Note that folders which do not contain files are shown flat anyhow. This might lead to problems if you like to add new files or folders in a collapsed package. Just switch to the flat package presentation to solve this problem.

2.10.4 Drag and Drop

Use drag and drop to move resources between packages. All references to the resource are kept automatically.



2.11 Project Explorer Context Menu

The context menu of the project explorer supports several important operations on packages and resources.

2.11.1 New Dialog

The *new/Actifsource* dialog creates different types of actifsource files. For some types of resources, you can create new resources based on existing resources.

New Actifsource Project

Creates a new Actifsource project in the current workspace with the project wizard, as shown in Chapter 2.5 Project Wizard.

New BuildConfig

Creates a new BuildConfig which can be referenced in target folders (see Chapter 2.5.3 Target Folder and Chapter 7 Build Config).

New Class Diagram

Creates a new class diagram for UML-like Meta-Model design. The class diagram is the easiest way to create Meta-Models.

New Diagram Type

Creates a new diagram type which defines a user specific domain diagram.

Creating a new diagram type on an existing class preselects this class as the RootClass.

New Domain Diagram

Creates a new domain diagram. Domain diagrams are based on diagram types.

Creating a domain diagram on an existing resource preselects this resource as the singleRoot. The diagram type is automatically selected by the type of the singleRoot.

If no singleRoot is defined, it is created automatically with the type defined by the diagram type.

New FunctionSpace

Creates a new function space (see Chapter 9.2 Function Space).

New Package

Creates a new package.

New Resource

Creates a new resource of any type.

New Resource Folder

Creates a new resource folder (see Chapter 2.5.2 Resource Paths).

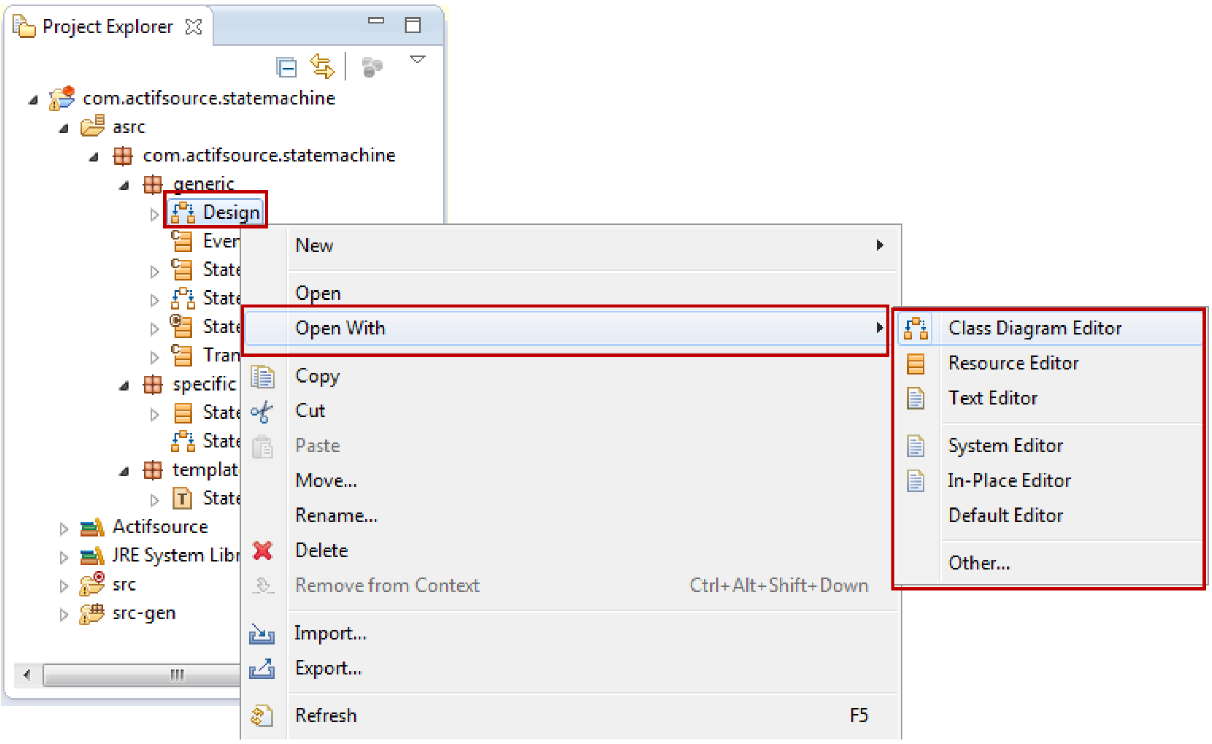
New Template

Creates a new code template (see Chapter Template Editor).

Creating a new template on an existing class preselects this class as the base type.

2.11.2 Open with

The *Open with* dialog forces eclipse to open files with a specific editor. The first element in the *Open With* list is the *default editor*. Once opened with another than the default editor Eclipse reminds this setting when double clicking the file to open. Just select *Open With/Default Editor* to restore the settings.



Actifsource supports the following editor types:

Resource Editor

The Actifsource Resource Editor is the standard editor which opens all types of Actifsource resources.

Class Diagram Editor

The Actifsource Class Diagram Editor opens resources of type ClassDiagram.

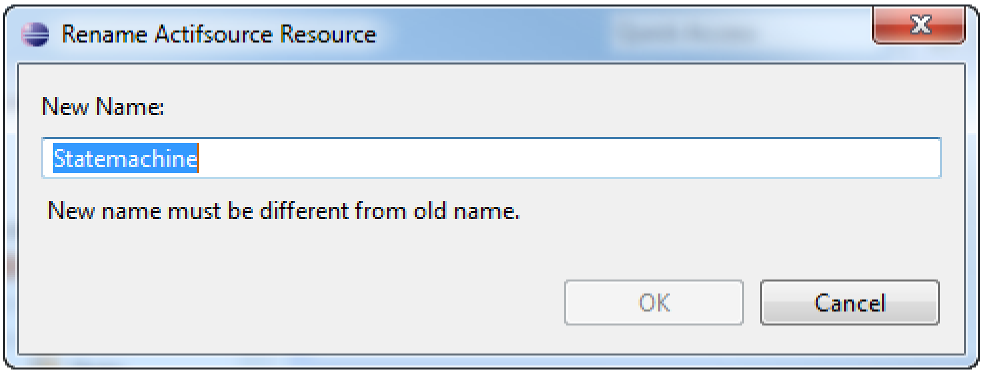
Domain Diagram Editor

The Actifsource Domain Diagram Editor opens resources of type DomainDiagram.

Template Editor

The Actifsource Template Editor opens resources of type Template.

2.11.3 Rename Resources and Packages



New Name: Statemachine

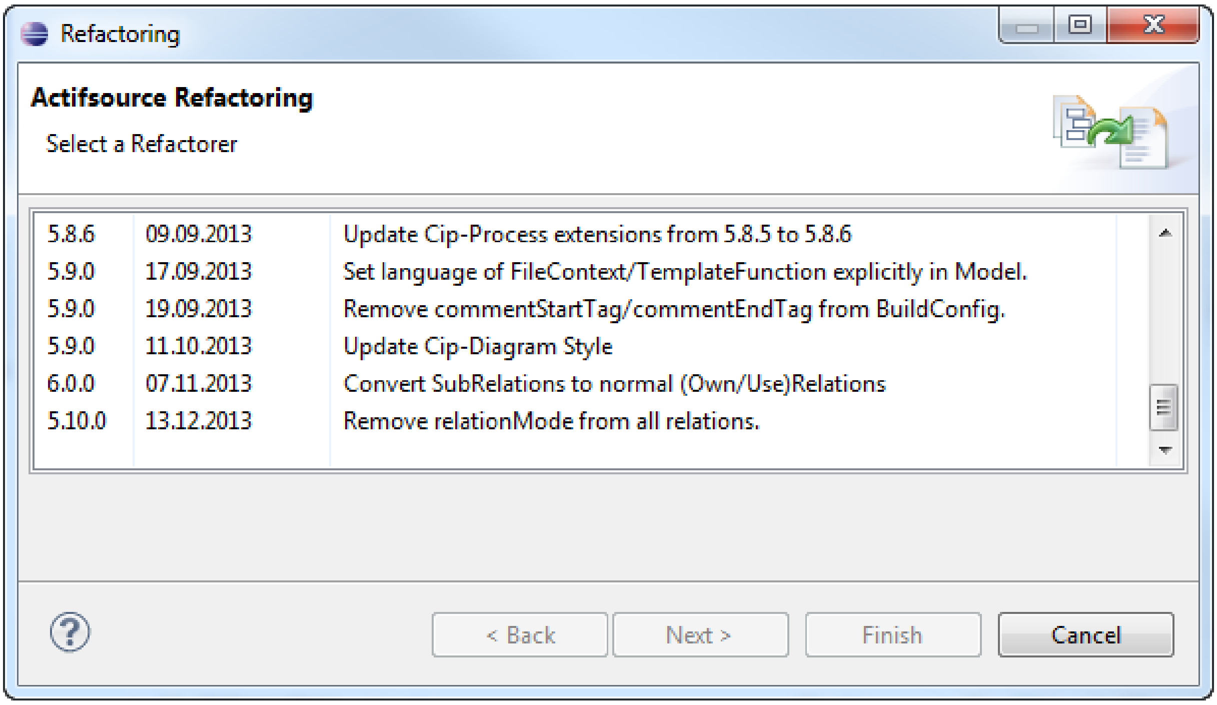
New name must be different from old name.

You may rename any resource or package via the *context menu/rename* or by pressing F2 (Windows).

2.11.4 Generic Refactoring

If you change your meta-model, any depending domain model might become invalid. Actifsource lets you register a piece of Java code, which transforms all existing domain models to fit the new meta-model. [REF]

Actifsource also uses this feature intensely if there are changes in the core model (meta meta-model). Please make sure to check the release notes to see if you need to run a *Generic Refactoring* after updating to a new Actifsource Version.

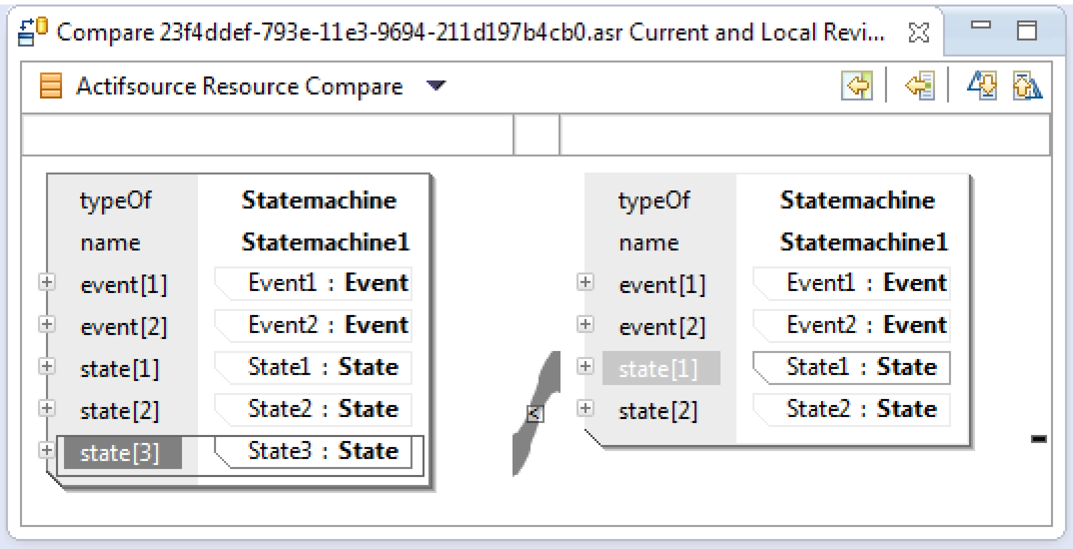


Actifsource Refactoring

Select a Refactorer

5.8.6	09.09.2013	Update Cip-Process extensions from 5.8.5 to 5.8.6
5.9.0	17.09.2013	Set language of FileContext/TemplateFunction explicitly in Model.
5.9.0	19.09.2013	Remove commentStartTag/commentEndTag from BuildConfig.
5.9.0	11.10.2013	Update Cip-Diagram Style
6.0.0	07.11.2013	Convert SubRelations to normal (Own/Use)Relations
5.10.0	13.12.2013	Remove relationMode from all relations.

2.11.5 Compare With



Compare 23f4ddef-793e-11e3-9694-211d197b4cb0.asr Current and Local Revi...
Actifsource Resource Compare

typeOf	Statemachine
name	Statemachine1
event[1]	Event1 : Event
event[2]	Event2 : Event
state[1]	State1 : State
state[2]	State2 : State
state[3]	State3 : State

typeOf	Statemachine
name	Statemachine1
event[1]	Event1 : Event
event[2]	Event2 : Event
state[1]	State1 : State
state[2]	State2 : State

Working together in a team, you might have collisions when checking Actifsource resource files (.asr) into your version control system (i.e. CVS, SVN, GIT etc.).

Actifsource lets you compare resources and resolve conflicts in the compare view of the Resource Editor.

2.11.6 Show Resource Dependencies

In the context menu of the selected resource you will find the following commands to show specific dependencies.

Show Instances

Shows all instances of the selected resource. Note that a resource must be of type Class to have instances. See also Chapter 2.12.2.

Show Types

Shows all types of the selected resource (typeof statement). See also Chapter 2.12.2.

Show Subclasses

Shows all sub classes of the selected resource (extend statement). Note that a resource must be of type Class to have sub classes. See also Chapter 2.12.2.

Show Superclasses

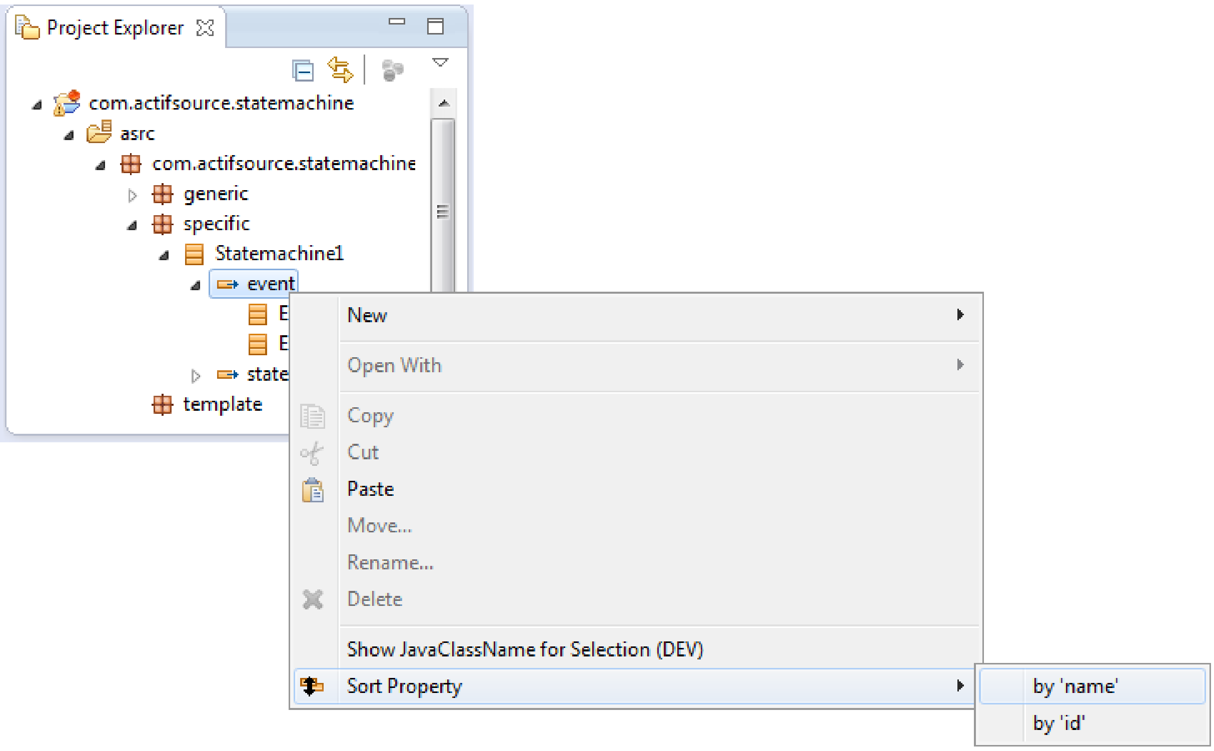
Shows all super classes of the selected resource (extend statement). Note that a resource must be of type Class to have super classes. See also Chapter 2.12.2.

Show References

Shows all resources that are referencing the selected resource. See also Chapter 2.12.6.

2.11.7 Sort Property

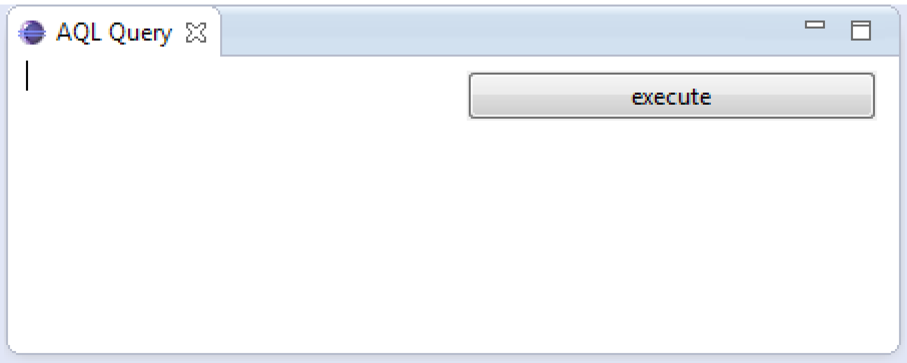
Sorts resources referenced by a relation. Note that you can sort according to any literal attribute (i.e. name).



Note that you have to switch on *Group Aggregation By Relation* (see Chapter 2.10.2 Actifsource Presentation) to see the relations.

2.12 Actifsource Views

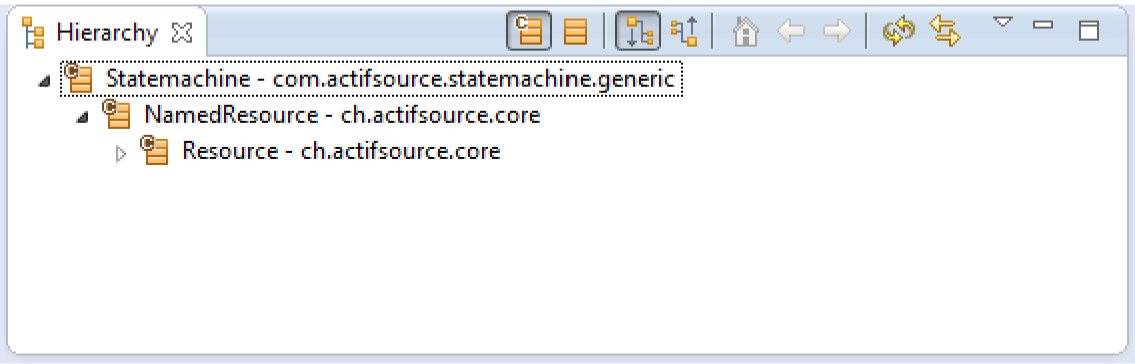
2.12.1 AQL Query



The Actifsource Query Language let you query the model.

[TODO]

2.12.2 Hierarchy

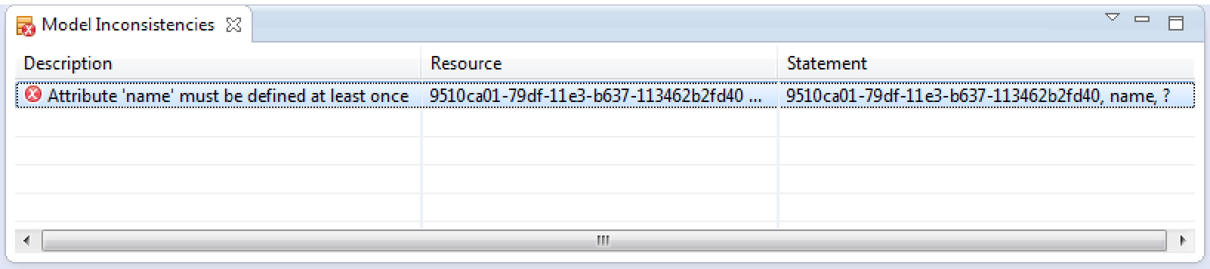


Hierarchy

- Statemachine - com.actifsource.statemachine.generic
  - NamedResource - ch.actifsource.core
    - Resource - ch.actifsource.core

Shows the hierarchy between resources.

2.12.3 Model Inconsistencies

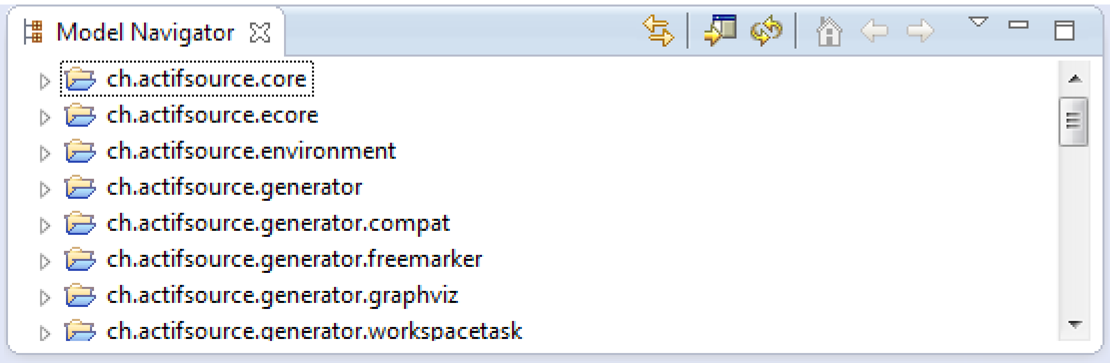


Model Inconsistencies

Description	Resource	Statement
Attribute 'name' must be defined at least once	9510ca01-79df-11e3-b637-113462b2fd40 ...	9510ca01-79df-11e3-b637-113462b2fd40, name, ?

Shows all model inconsistencies calculated by the validator. Make sure that this view is always visible to check whether your model is valid or not. Note that the code generator could throw an exception if your model is invalid.

2.12.4 Model Navigator

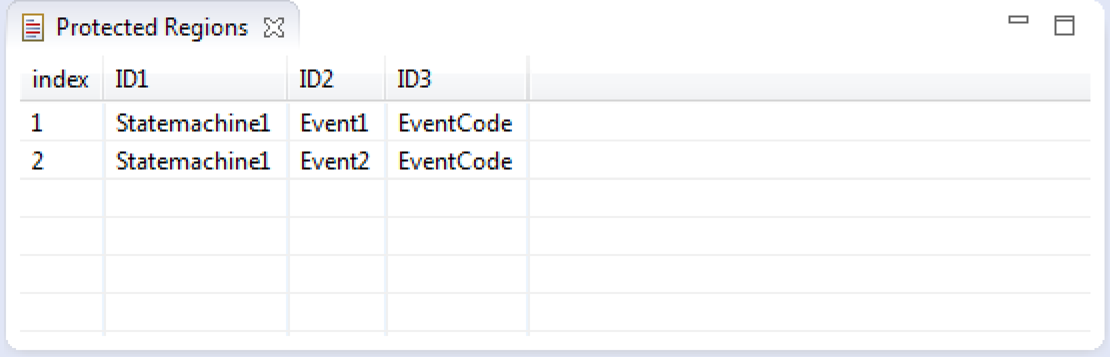


Model Navigator

- ch.actifsource.core
- ch.actifsource.ecore
- ch.actifsource.environment
- ch.actifsource.generator
- ch.actifsource.generator.compat
- ch.actifsource.generator.freemarker
- ch.actifsource.generator.graphviz
- ch.actifsource.generator.workspacetask

The model navigator shows all resources sorted by projects, packages and types. Use the model navigator to find classes outside your project.

2.12.5 Protected Regions



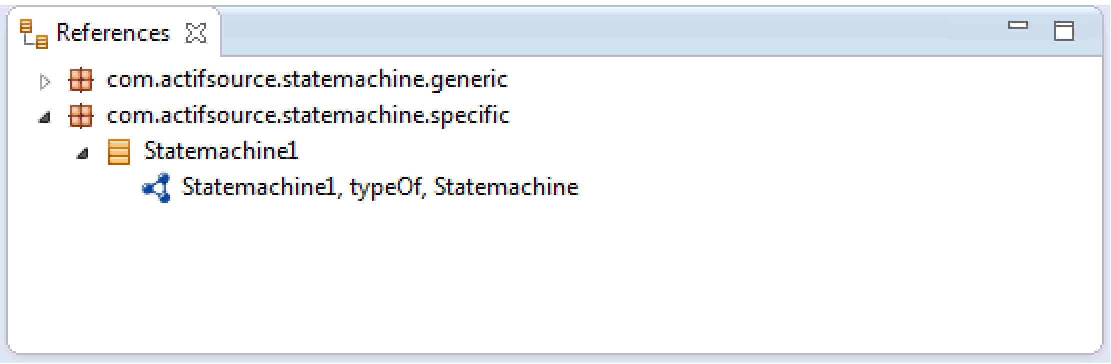
Protected Regions

index	ID1	ID2	ID3
1	Statemachine1	Event1	EventCode
2	Statemachine1	Event2	EventCode

Shows all protected regions from a generated file. Click on the entry to navigate to the protected region in the selected file.

[REF]

2.12.6 References



```
com.actifsource.statemachine.generic
com.actifsource.statemachine.specific
    Statemachine1
        Statemachine1, typeOf, Statemachine
```

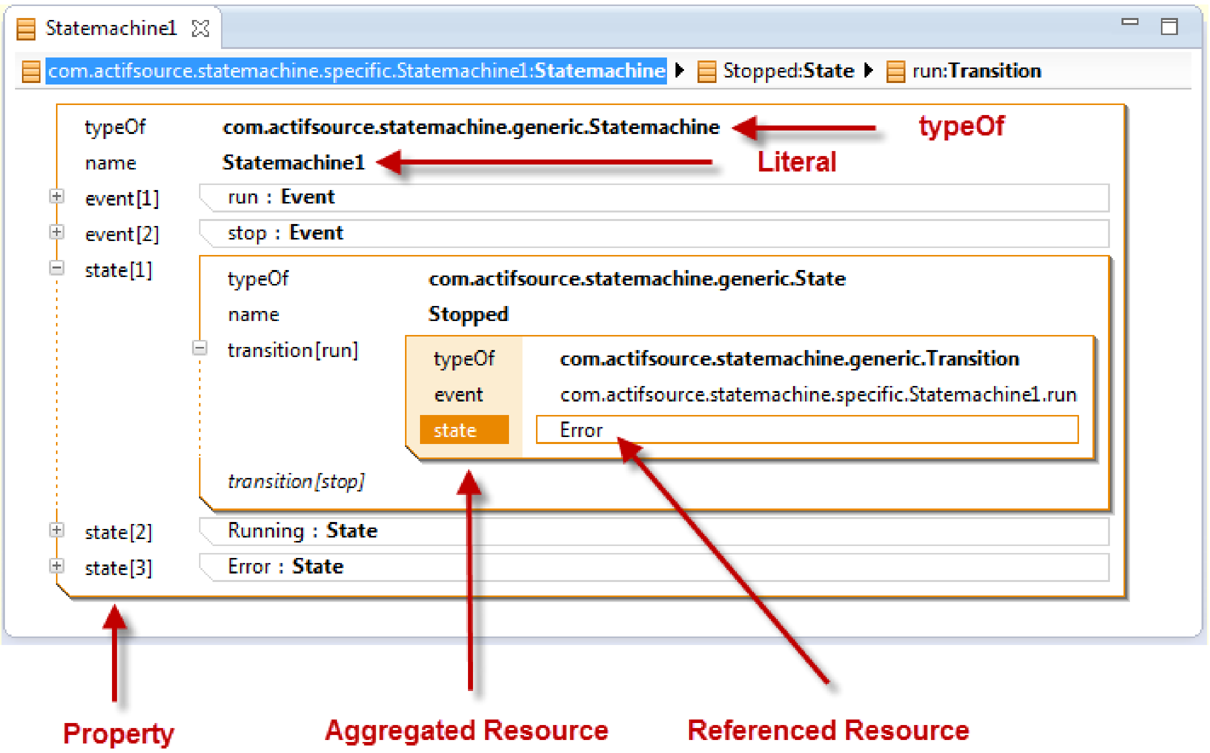
Shows all references to a specific resource in the form *Subject-Predicate-Object* while object is the referenced resource.

3 Resource Editor

3.1 Overview

The Actifsource *Resource Editor* allows you to view and/or edit any Actifsource resource. Since everything is a resource in Actifsource this is the most important editor.

The Actifsource Resource Editor shows resources as tree (similar to the Windows Explorer).



Statemachine1

com.actifsource.statemachine.specific.Statemachine1:**Statemachine** ▸ Stopped:**State** ▸ run:**Transition**

typeOf **com.actifsource.statemachine.generic.Statemachine**
name **Statemachine1**
event[1] run : **Event**
event[2] stop : **Event**
state[1]
 typeOf **com.actifsource.statemachine.generic.State**
 name **Stopped**
 transition[run]
  typeOf **com.actifsource.statemachine.generic.Transition**
  event com.actifsource.statemachine.specific.Statemachine1.run
  state Error
 transition[stop]
state[2] Running : **State**
state[3] Error : **State**

3.1.1 Aggregated vs. Referenced Resource

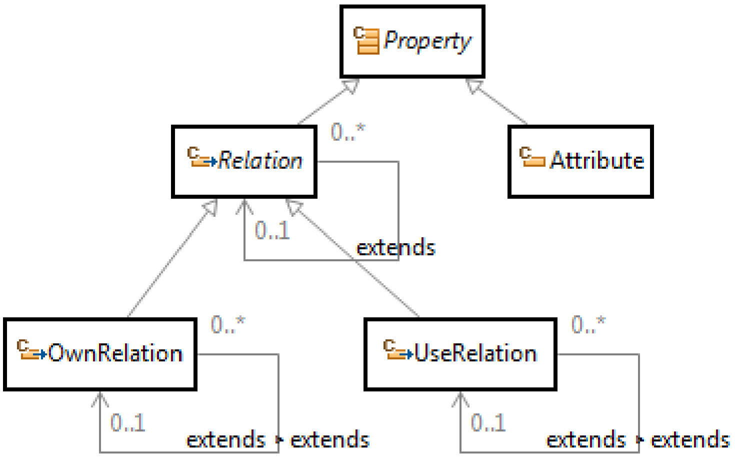
Actifsource distinguish between aggregated and referenced resources. An aggregated resource lives in the context of the parent resource. Deleting the parent resource will delete all aggregated resources.

Referenced resource may live anywhere in the model and are just referenced. Delete the referencing class will not affect the lifetime of the referenced resource.

3.1.2 Property

All information is grouped by properties (see also Chapter 0

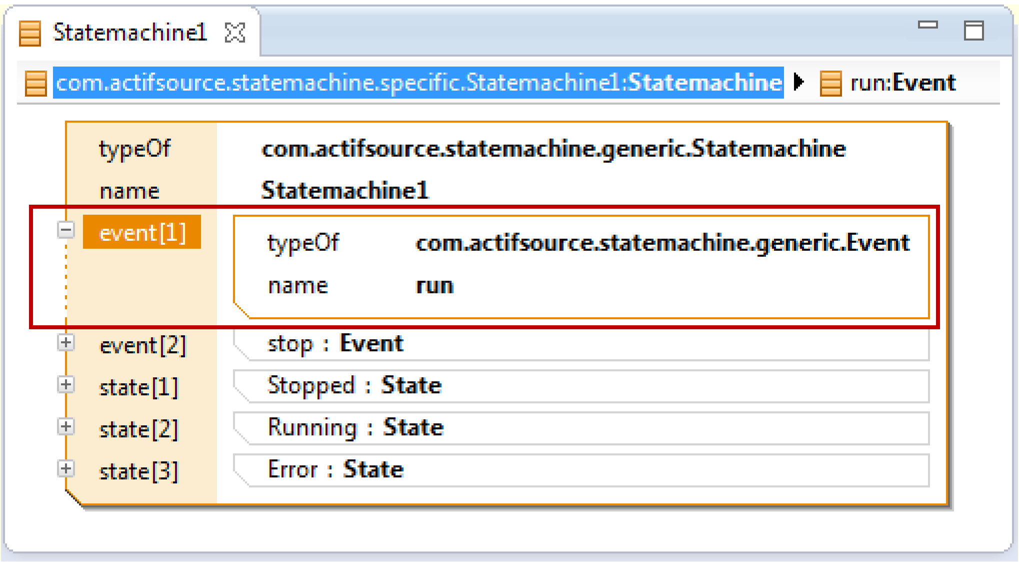
Property). Actifsource distinguish the following property types.



Class	Meaning
Property	Base class for all properties
Relation	Base class for all relations
OwnRelation	Leads to aggregated resources (UML: Aggregation, Composition)
UseRelation	Leads to referenced resources (UML: Association)
Attribute	Primitive literals (String, Boolean, Integer, etc.)

3.1.3 Open/Close Folding

To work with large resources efficiently, Actifsource can expand or collapse aggregated resources.


Statemachine1
com.actifsource.statemachine.specific.Statemachine1:Statemachine
run:Event

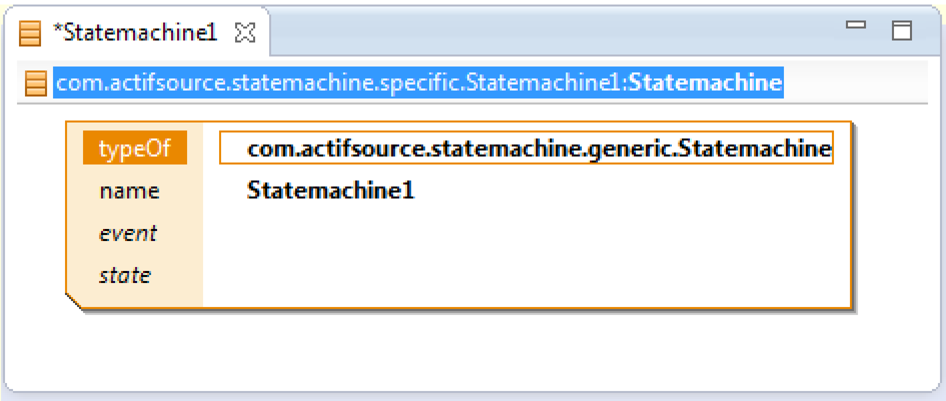
typeOf	com.actifsource.statemachine.generic.Statemachine
name	Statemachine1
event[1]	typeOf com.actifsource.statemachine.generic.Event name run
event[2]	stop : Event
state[1]	Stopped : State
state[2]	Running : State
state[3]	Error : State

Use the following possibilities to expand/collapse resources.

Device	Action
Mouse	Klick the [+] [-] sign
	DoubleClick the property
Context menu	Open/Close Folding
Keyboard	Enter (Open/Close Folding) Backspace (Close Folding)

3.1.4 The typeOf Statement

The **typeOf** statement of a resource shows the instantiation relation and declares the type of this resource.

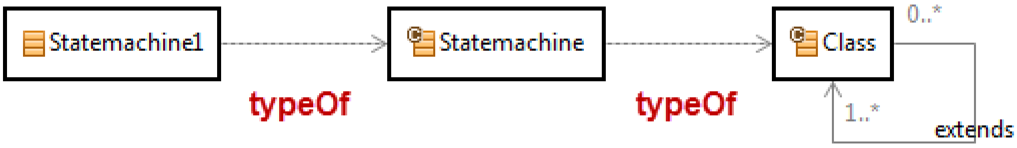


*Statemachine1

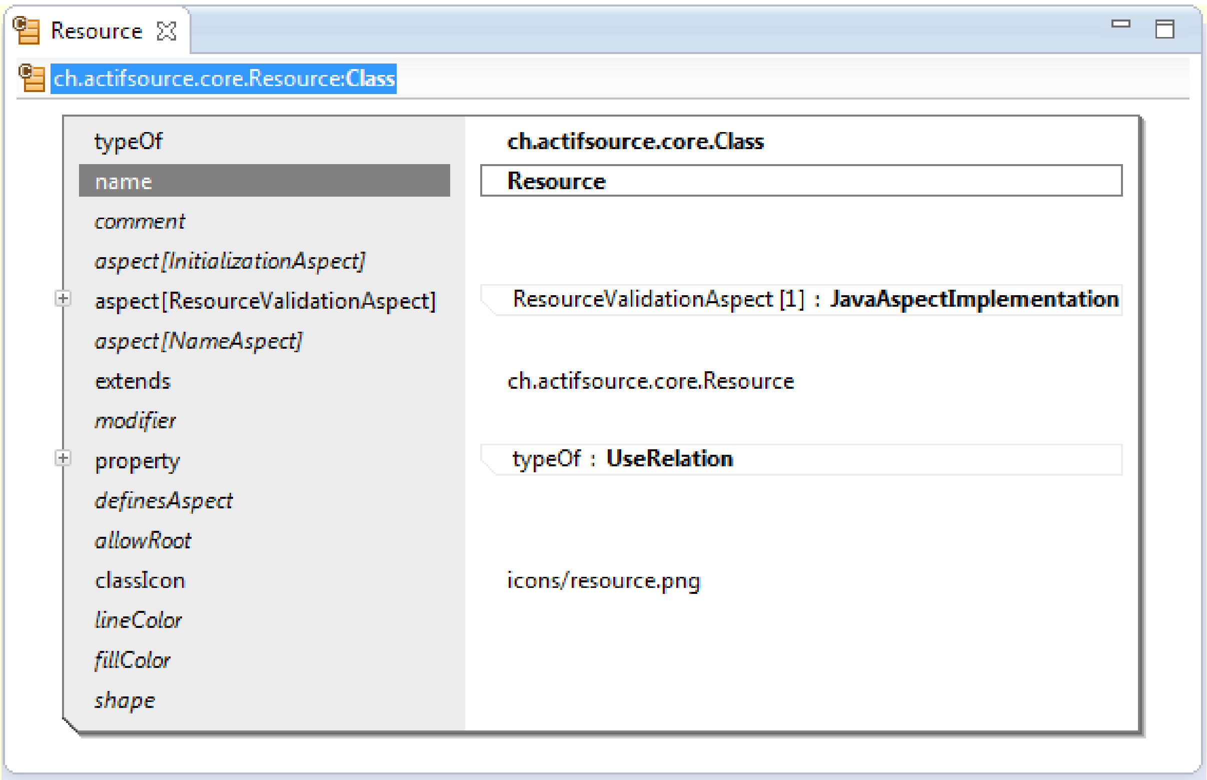
com.actifsource.statemachine.specific.Statemachine1:Statemachine

typeOf	com.actifsource.statemachine.generic.Statemachine
name	Statemachine1
event	
state	

Actifsource only allows a **typeOf** relation to resources of type **Class**.



3.2 Read Only View



Resource

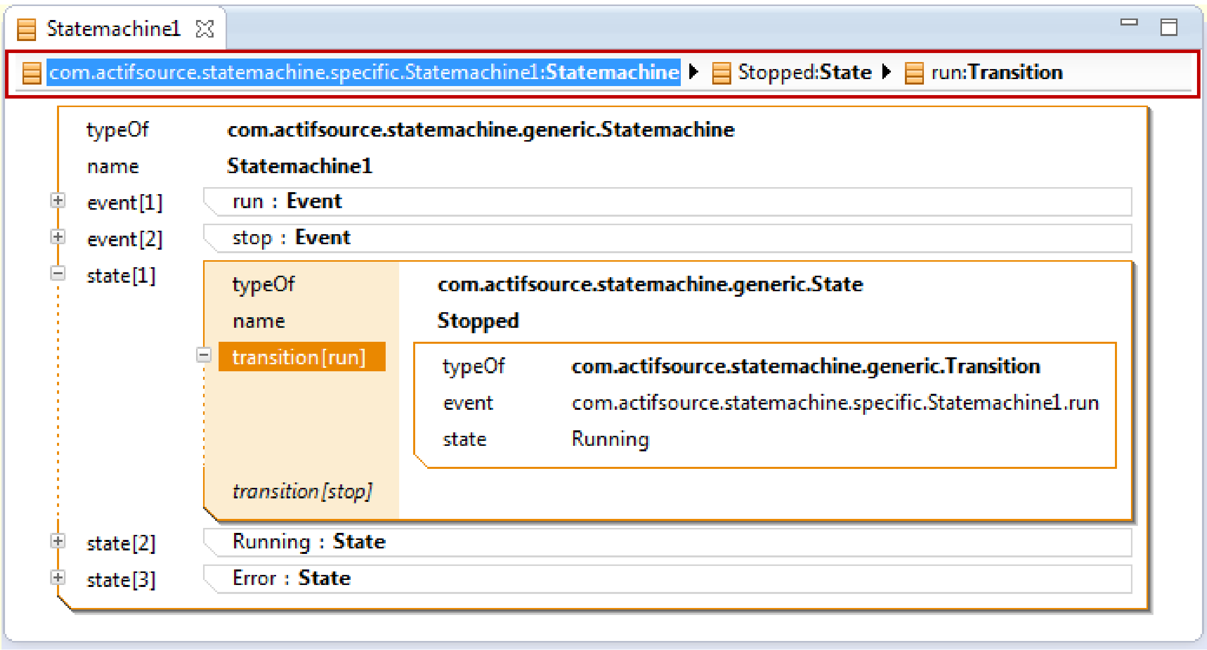
ch.actifsource.core.Resource:Class

typeOf	ch.actifsource.core.Class
name	Resource
comment	
aspect [InitializationAspect]	
aspect [ResourceValidationAspect]	ResourceValidationAspect [1] : JavaAspectImplementation
aspect [NameAspect]	
extends	ch.actifsource.core.Resource
modifier	
property	typeOf : UseRelation
definesAspect	
allowRoot	
classIcon	icons/resource.png
lineColor	
fillColor	
shape	

Deploying Actifsource Models as an Eclipse Plugin (see Chapter 15 Plugin Project) leads to a read only view of the models. Models that are read-only are displayed in gray colors. Note that the Actifsource Core Model is read-only for you.

3.3 Breadcrumb

The breadcrumb helps you navigating large resources.



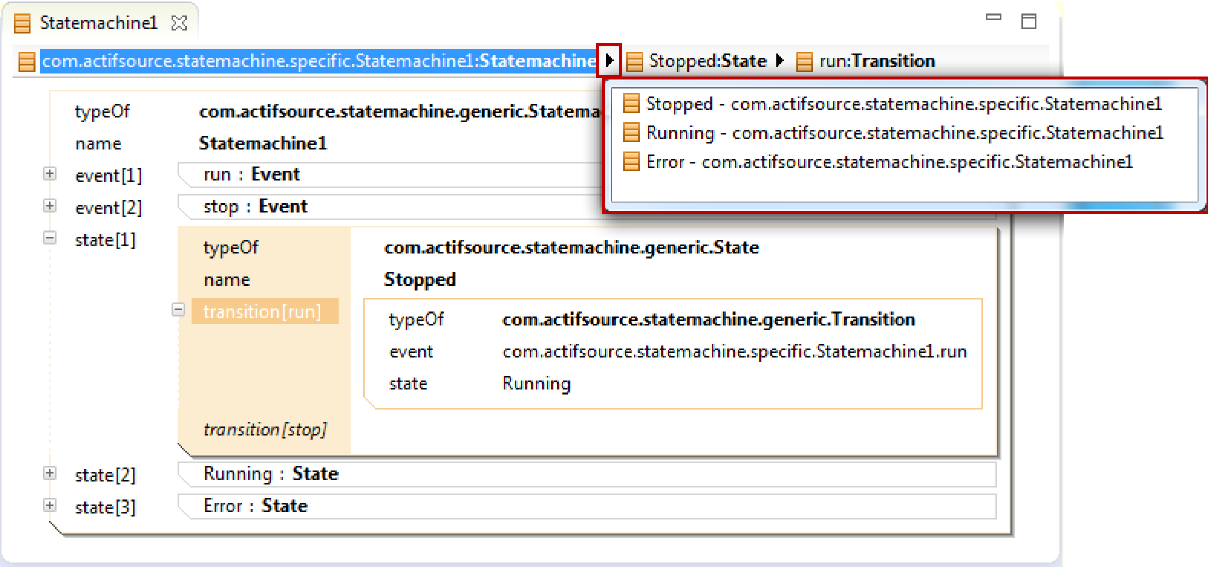
Statemachine1

com.actifsource.statemachine.specific.Statemachine1:**Statemachine** ▸ Stopped:**State** ▸ run:**Transition**

typeOf **com.actifsource.statemachine.generic.Statemachine**
name **Statemachine1**
event[1] run : **Event**
event[2] stop : **Event**
state[1]
 typeOf **com.actifsource.statemachine.generic.State**
 name **Stopped**
 transition[run]
  typeOf **com.actifsource.statemachine.generic.Transition**
  event com.actifsource.statemachine.specific.Statemachine1.run
  state Running
 transition[stop]
state[2] Running : **State**
state[3] Error : **State**

3.3.1 Navigating resources

Clicking on the arrow in the breadcrumb allows you navigating all resources from the same property.



Statemachine1

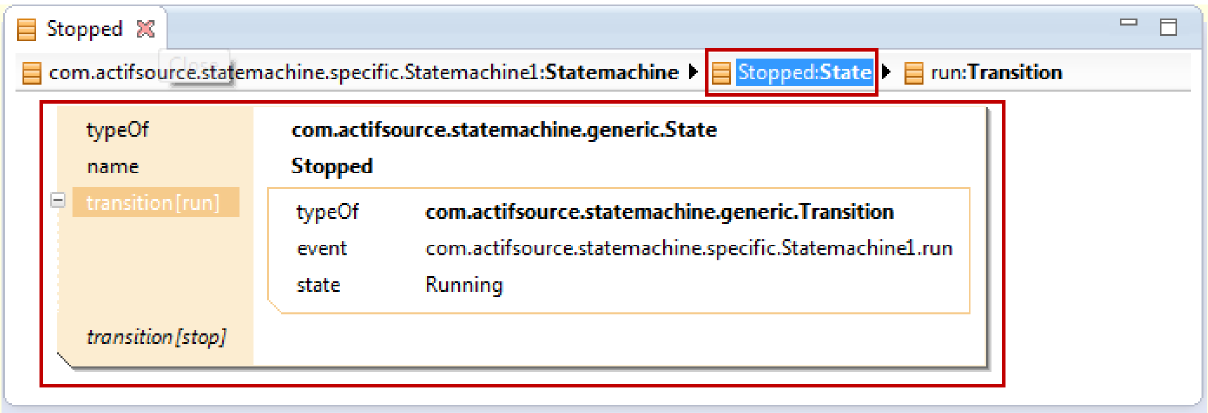
com.actifsource.statemachine.specific.Statemachine1:**Statemachine** ▸ Stopped:**State** ▸ run:**Transition**

Stopped - com.actifsource.statemachine.specific.Statemachine1
Running - com.actifsource.statemachine.specific.Statemachine1
Error - com.actifsource.statemachine.specific.Statemachine1

typeOf **com.actifsource.statemachine.generic.Statema**
name **Statemachine1**
event[1] run : **Event**
event[2] stop : **Event**
state[1]
 typeOf **com.actifsource.statemachine.generic.State**
 name **Stopped**
 transition[run]
  typeOf **com.actifsource.statemachine.generic.Transition**
  event com.actifsource.statemachine.specific.Statemachine1.run
  state Running
 transition[stop]
state[2] Running : **State**
state[3] Error : **State**

3.3.2 Focusing aggregated resources

Clicking on a resource in the breadcrumb allows you to focus only on this aggregated resource. Use this feature to work with large resource files.



Stopped

com.actifsource.statemachine.specific.Statemachine1:Statemachine ▸ Stopped:State ▸ run:Transition

typeOf com.actifsource.statemachine.generic.State

name Stopped

transition[run]

typeOf com.actifsource.statemachine.generic.Transition

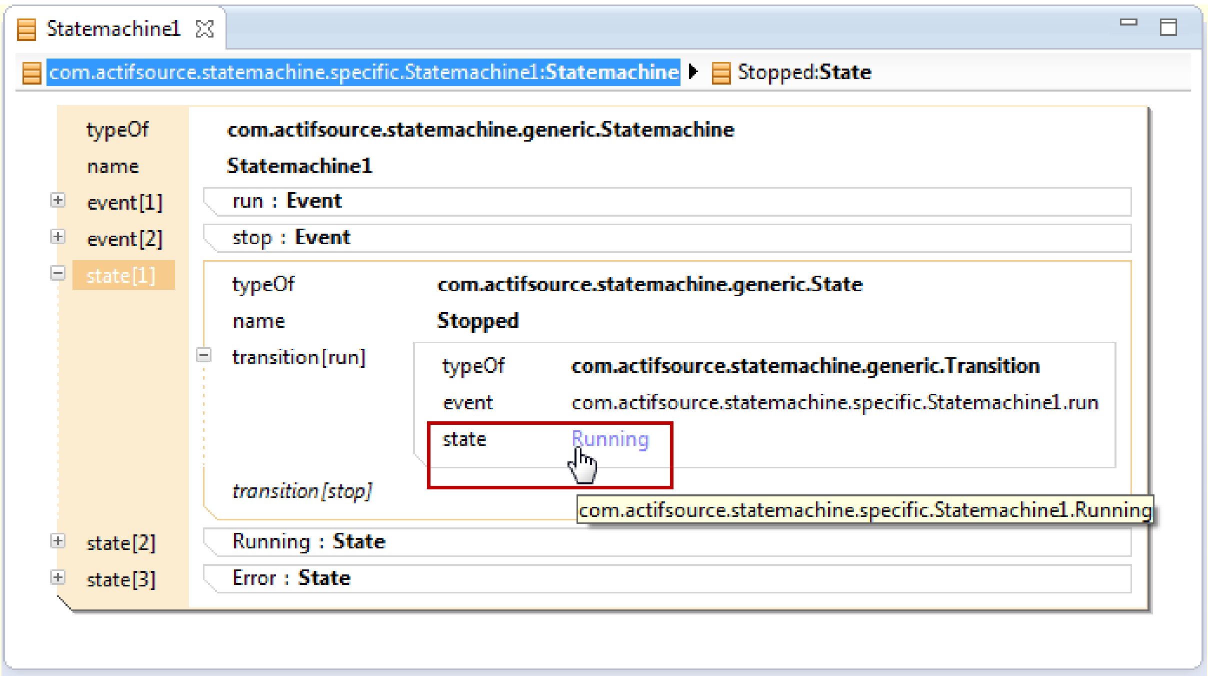
event com.actifsource.statemachine.specific.Statemachine1.run

state Running

transition[stop]

3.4 Browse resource

Actifsource allows you to browse any resource in any editor.



Statemachine1

com.actifsource.statemachine.specific.Statemachine1:Statemachine ▸ Stopped:State

typeOf com.actifsource.statemachine.generic.Statemachine

name Statemachine1

event[1] run : Event

event[2] stop : Event

state[1]

typeOf com.actifsource.statemachine.generic.State

name Stopped

transition[run]

typeOf com.actifsource.statemachine.generic.Transition

event com.actifsource.statemachine.specific.Statemachine1.run

state Running

com.actifsource.statemachine.specific.Statemachine1.Running

transition[stop]

state[2] Running : State

state[3] Error : State

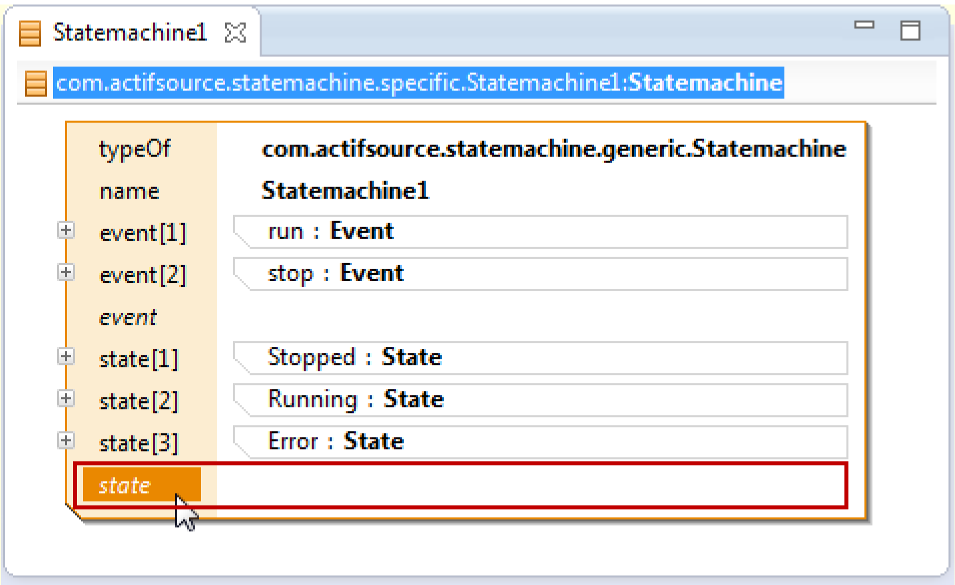
To browse any resources in actifsource use the following possibilities.

Device	Action
Mouse	Ctrl+LeftClick
Context menu	Browse Into
Keyboard	F3

3.5 Insert resource

3.5.1 Insert on the empty line

Actifsource shows an *empty line* for all properties which might have another instance (depends on the subject cardinality; see Chapter 4.4.1 Property).

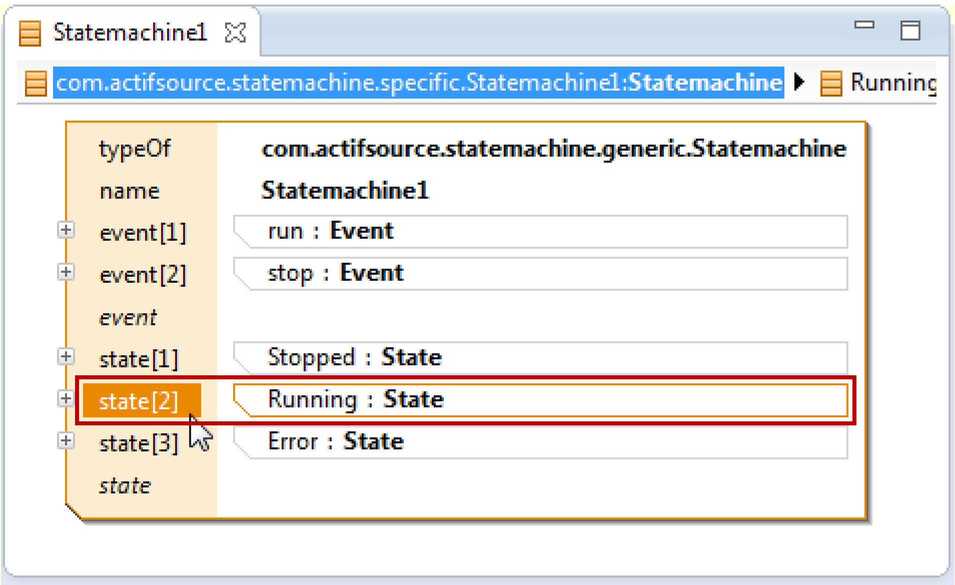


To insert a resources on the empty line use the following possibilities.

Device	Action
Mouse	Ctrl+DoubleLeftClick on the property
Context menu	Insert here
Keyboard	Enter

3.5.2 Insert before or after properties

Actifsource allows inserting resources before or after existing properties.

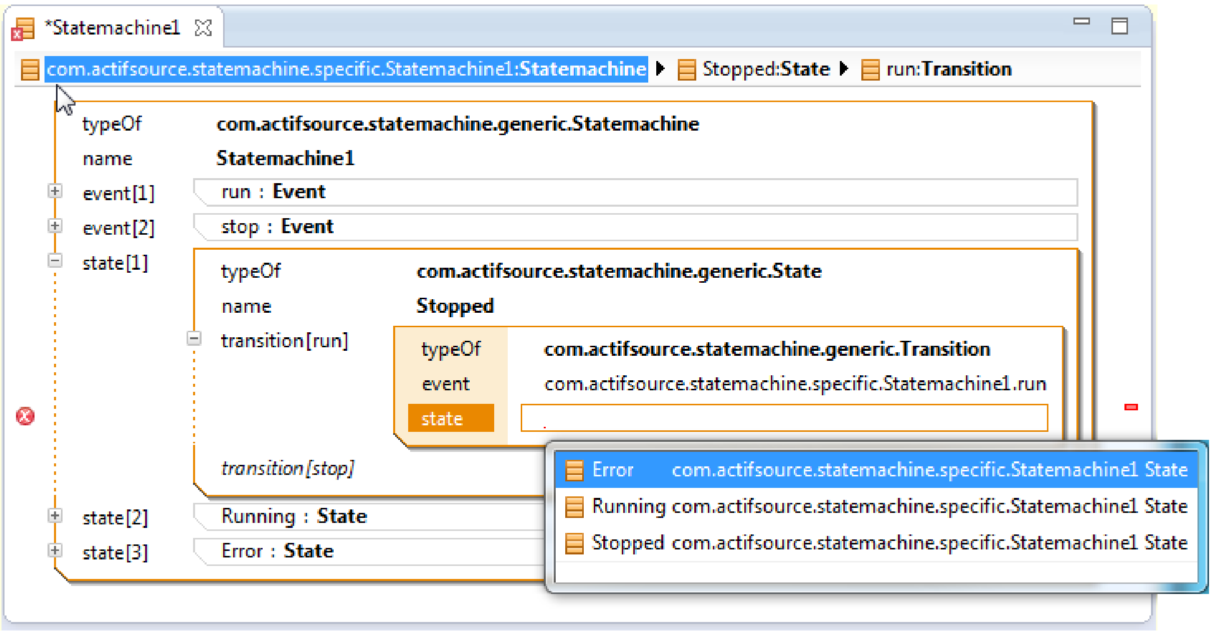


To insert an aggregated resource before or after an existing resource use the following possibilities.

Device	Action
Mouse	Ctrl+DoubleLeftClick to insert after the current resource
	Ctrl+Shift+DoubleLeftClick to before after the current resource
Context menu	Insert after to insert after the current resource
	Insert before to insert before the current resource
Keyboard	Ctrl+Enter to insert after the current resource

Ctrl+Shift+Enter to insert before the current resource

3.6 Reference resource



*Statemachine1

com.actifsource.statemachine.specific.Statemachine1:Statemachine ▸ Stopped:State ▸ run:Transition

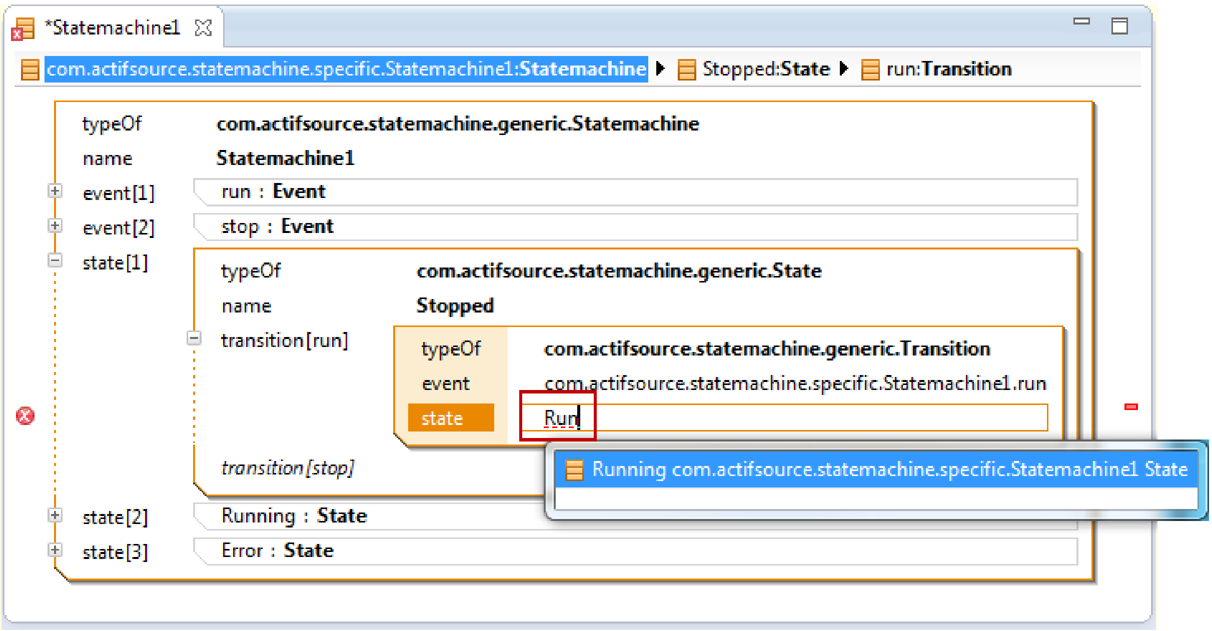
typeOf com.actifsource.statemachine.generic.Statemachine
name Statemachine1
event[1] run : Event
event[2] stop : Event
state[1]
typeOf com.actifsource.statemachine.generic.State
name Stopped
transition[run]
typeOf com.actifsource.statemachine.generic.Transition
event com.actifsource.statemachine.specific.Statemachine1.run
state
transition[stop]
state[2] Running : State
state[3] Error : State

Error com.actifsource.statemachine.specific.Statemachine1 State
Running com.actifsource.statemachine.specific.Statemachine1 State
Stopped com.actifsource.statemachine.specific.Statemachine1 State

To reference any resource use the *content assist*. Note that Actifsource supports content assist in many different situations. Just try *Ctrl+Space* to activate content assist via keyboard.

Device	Action
Context menu	Open Content Assist
Keyboard	Ctrl+Space

You may also type some letters to filter the resources.



*Statemachine1

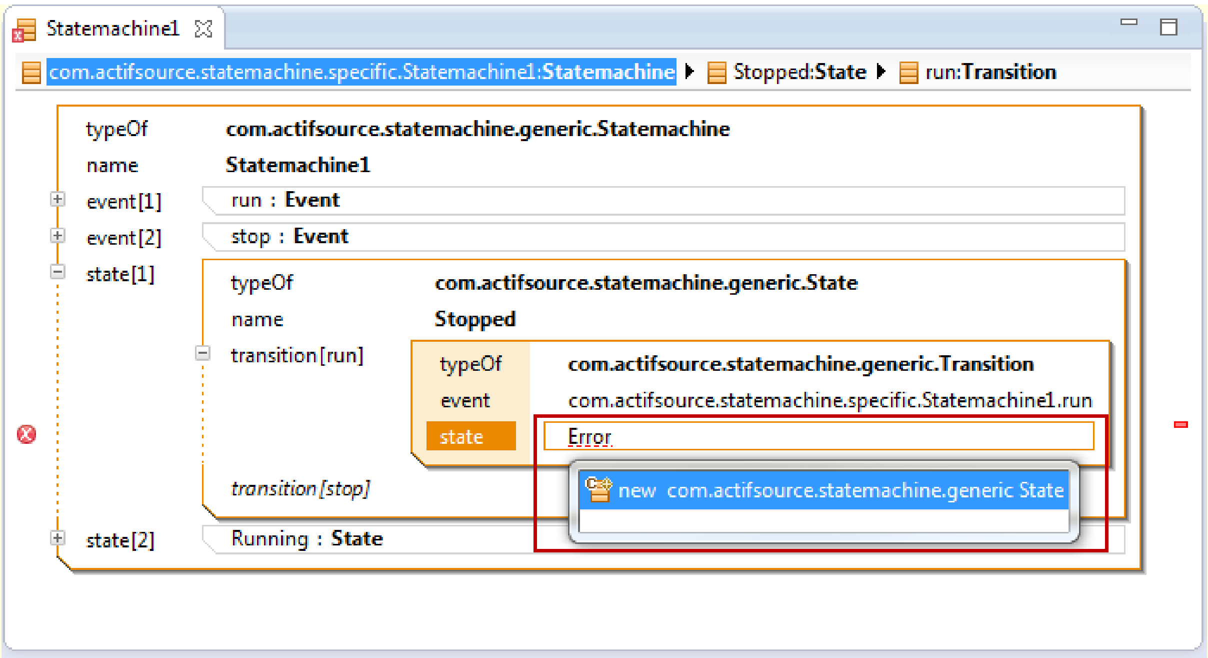
com.actifsource.statemachine.specific.Statemachine1:Statemachine ▸ Stopped:State ▸ run:Transition

typeOf com.actifsource.statemachine.generic.Statemachine
name Statemachine1
event[1] run : Event
event[2] stop : Event
state[1]
typeOf com.actifsource.statemachine.generic.State
name Stopped
transition[run]
typeOf com.actifsource.statemachine.generic.Transition
event com.actifsource.statemachine.specific.Statemachine1.run
state Run
transition[stop]
state[2] Running : State
state[3] Error : State

Running com.actifsource.statemachine.specific.Statemachine1 State

3.7 New referenced resource

You are able to create new referenced resources just by typing the named and select new from the content assist.

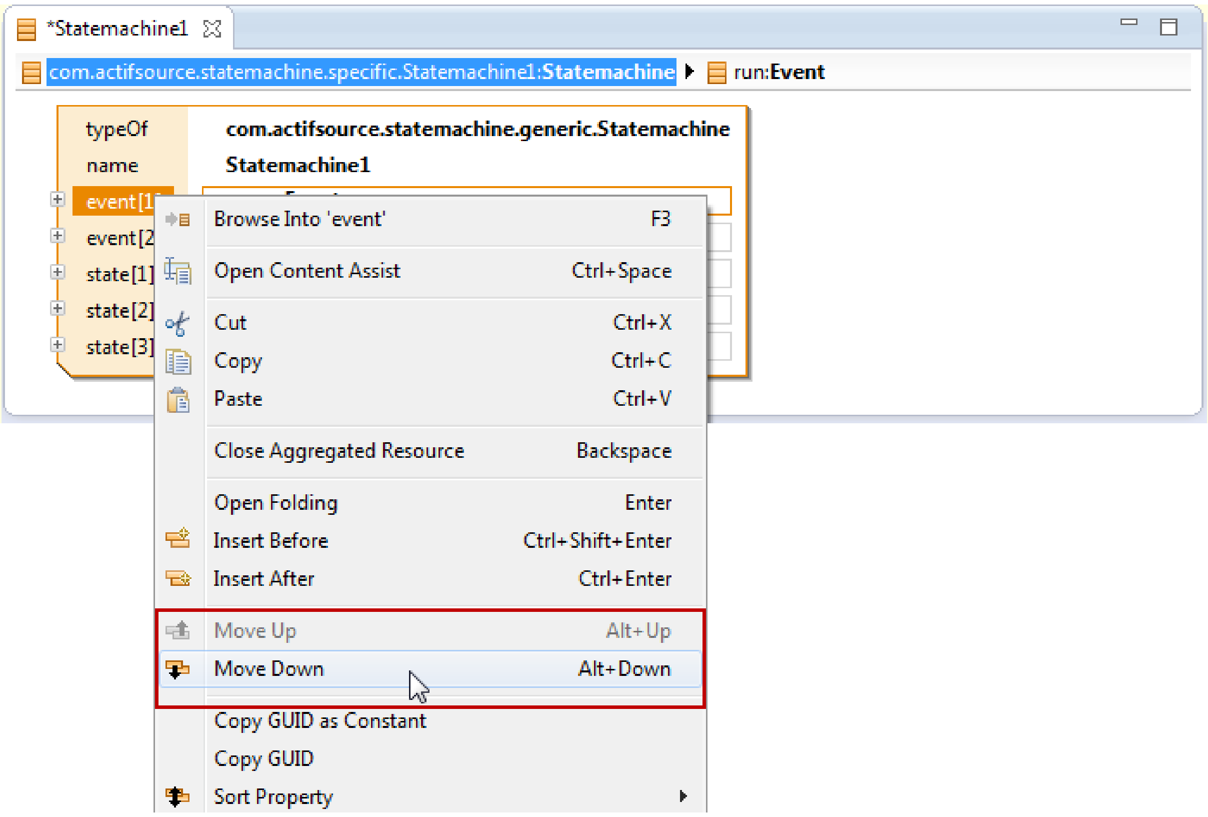

Statemachine1
com.actifsource.statemachine.specific.Statemachine1:Statemachine
Stopped:State
run:Transition

typeOf com.actifsource.statemachine.generic.Statemachine
name Statemachine1
event[1] run : Event
event[2] stop : Event
state[1]
typeOf com.actifsource.statemachine.generic.State
name Stopped
transition[run]
typeOf com.actifsource.statemachine.generic.Transition
event com.actifsource.statemachine.specific.Statemachine1.run
state Error
transition[stop]
state[2] Running : State

The place where the resource is created depends on the UseRangeRestrictionAspect (see Chapter 4.4.4 UseRelation). If there is no UseRangeRestrictionAspect defined, the new resource is created in the same package as the referencing resource.

3.8 Move resource

If you need to change the order you can simple move resources up and down within the same property.


*Statemachine1
com.actifsource.statemachine.specific.Statemachine1:Statemachine
run:Event

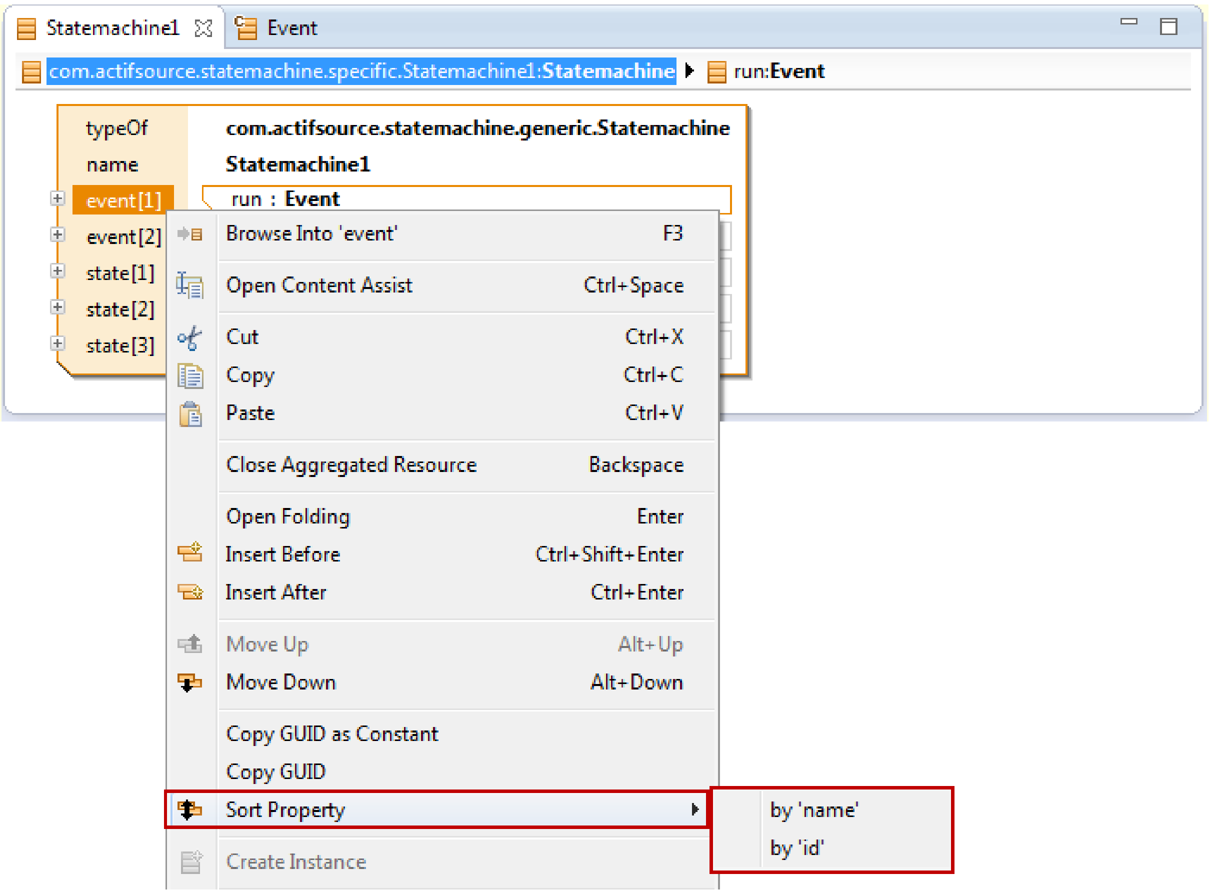
typeOf com.actifsource.statemachine.generic.Statemachine
name Statemachine1
event[1
event[2
state[1]
state[2]
state[3]

To move resources up and down use the following possibilities.

Device	Action
Context menu	Move Down
	Move Up
Keyboard	Alt+CursorDown
	Alt+CursorUp

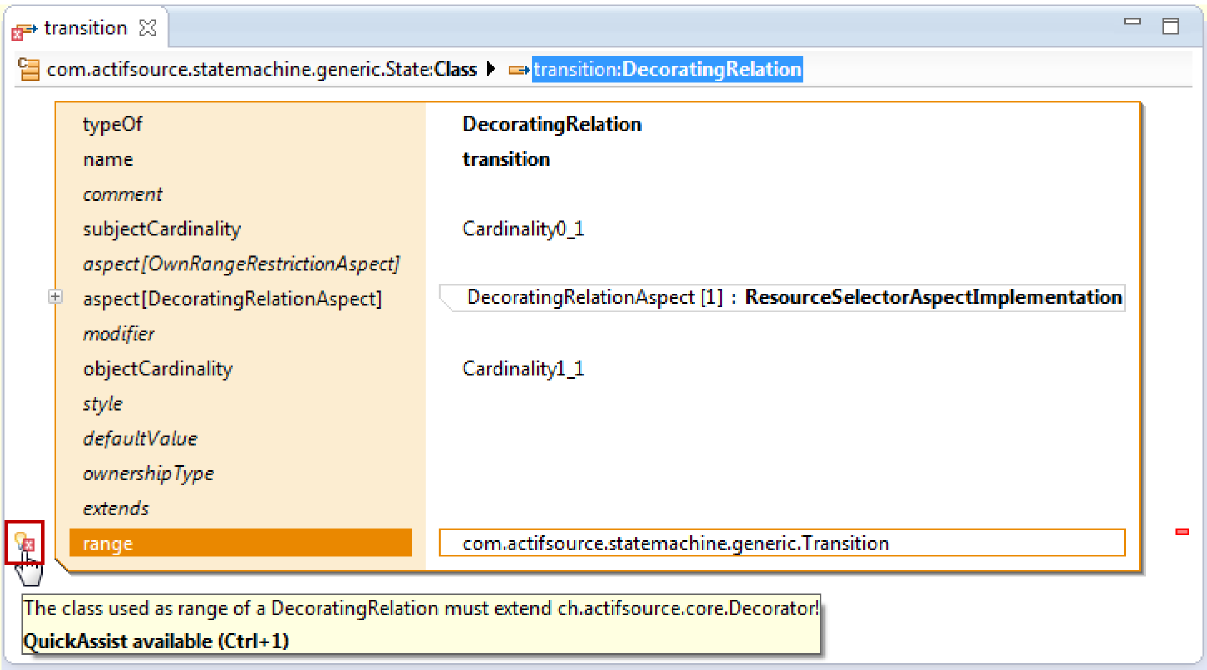
3.9 Sort property

Actifsource lets you sort all resources of the same property by any literal (i.e. name property). Note that the sort algorithm is just applied once. Just call sort any time if needed.



3.10 Quick Assist

Sometimes Actifsource offers a quick assist to fix common issues. Hoover the mouse pointer over the light bulb symbol 💡 to get a tooltip with a short description of the problem.



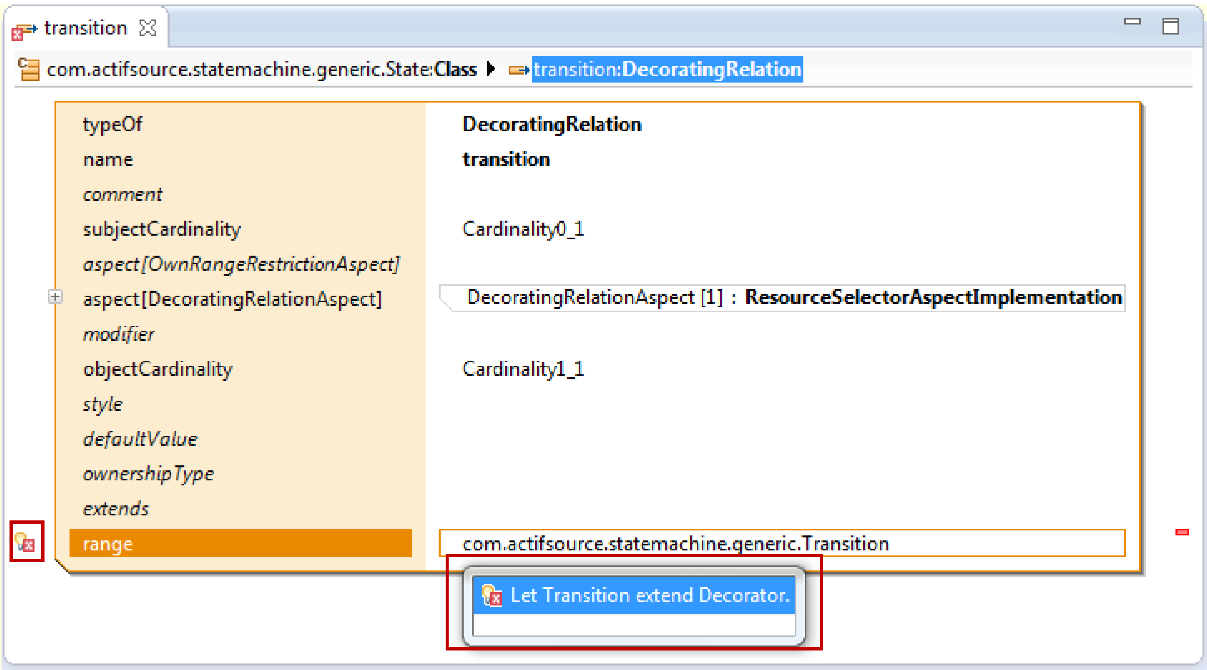
transition

com.actifsource.statemachine.generic.State:Class ▸ transition:DecoratingRelation

typeOf	**DecoratingRelation**
name	**transition**
comment	
subjectCardinality	Cardinality0_1
aspect[OwnRangeRestrictionAspect]	
aspect[DecoratingRelationAspect]	DecoratingRelationAspect [1] : **ResourceSelectorAspectImplementation**
modifier	
objectCardinality	Cardinality1_1
style	
defaultValue	
ownershipType	
extends	
range	com.actifsource.statemachine.generic.Transition

The class used as range of a DecoratingRelation must extend ch.actifsource.core.Decorator!
QuickAssist available (Ctrl+1)

Click on the bulb symbol to get the possible solutions.



transition

com.actifsource.statemachine.generic.State:Class ▸ transition:DecoratingRelation

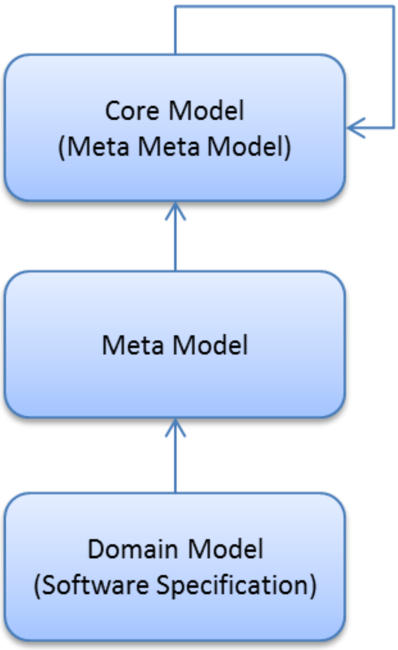
typeOf	**DecoratingRelation**
name	**transition**
comment	
subjectCardinality	Cardinality0_1
aspect[OwnRangeRestrictionAspect]	
aspect[DecoratingRelationAspect]	DecoratingRelationAspect [1] : **ResourceSelectorAspectImplementation**
modifier	
objectCardinality	Cardinality1_1
style	
defaultValue	
ownershipType	
extends	
range	com.actifsource.statemachine.generic.Transition

Device	Action
Mouse	Click on the bulb symbol
Keyboard	Ctrl+1

4 Core Model (Meta Meta-Model)

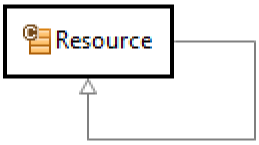
4.1 Overview

Every meta-model is based on the *Actifsource Core Model*. Make sure that you understand the concepts of **Resource**, **Class** and **Property** before you start creating your own meta-model.

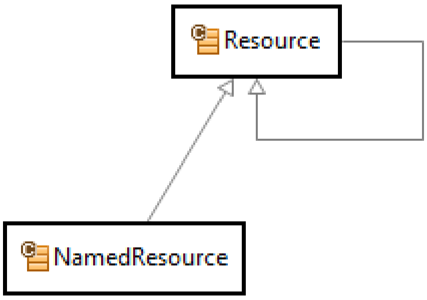


4.2 Resource

In Actifsource everything is a **Resource**. That means that every **Resource** you create extends the **Resource**. Even **Resource** extends **Resource**, so that the core model can describe itself.



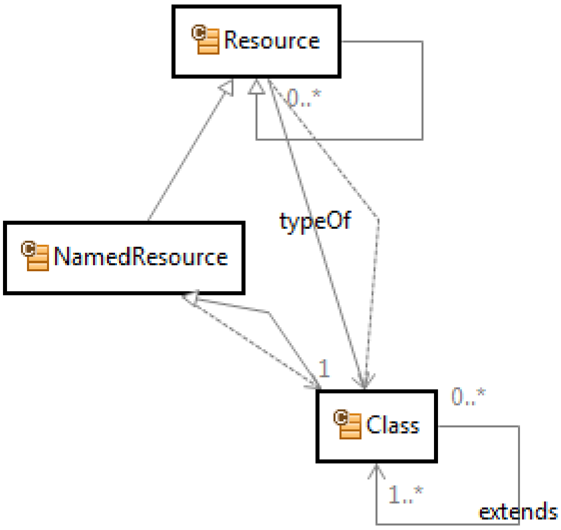
If the **Resource** shall has a name you can extend from **NamedResource**.



4.3 Class

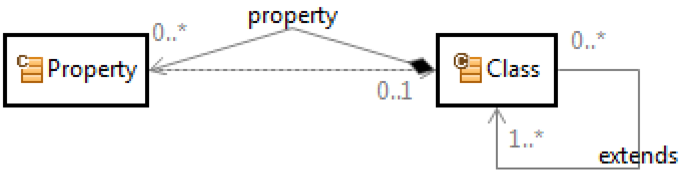
Creating your own meta-model you have to specify a set of classes and their relationships. The Core Model therefore provides you a **NamedResource** called **Class**. **Class** is the only **Resource** in Actifsource that can be instantiated where in this context we mean by instantiate that there is a **typeOf** relation. For that reason **Resource** is of type **Class** and **Class** extends **NamedResource**.

This recursive definition becomes necessary as the Actifsource core model has to describe itself. You can say that the Actifsource core model is meta-model of itself.

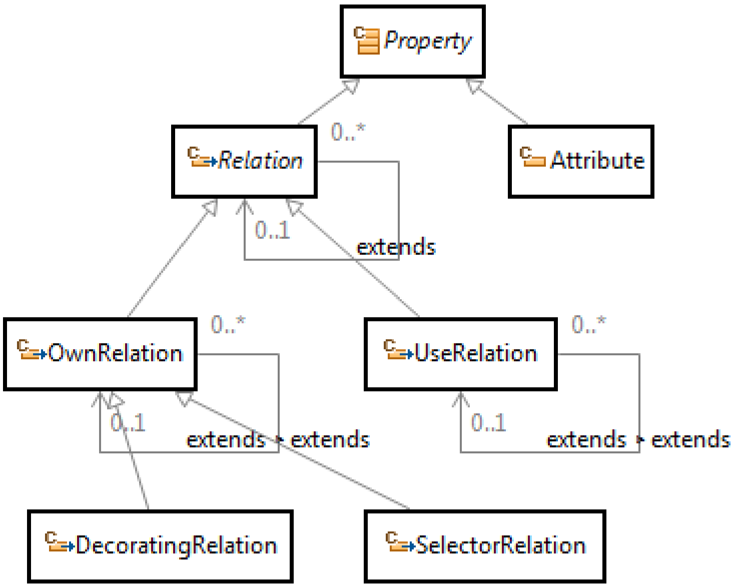


4.4 Property

Any information is modelled by a property. Specifying your own class therefore also means specifying all properties of this class.

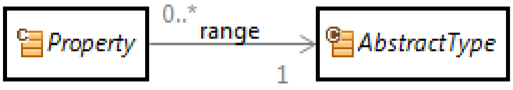


There are different types of properties.

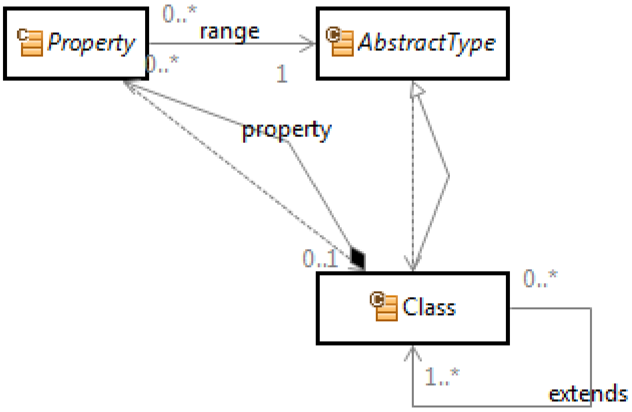


4.4.1 Property

The property acts as an abstract base class. Please note that properties are fully typed. In Actifsource the type of a property is named range.

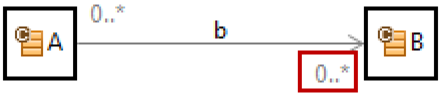


Every resource that can be referenced by a range has to be of type **AbstractType**. One abstract type you already know by now is **Class**.

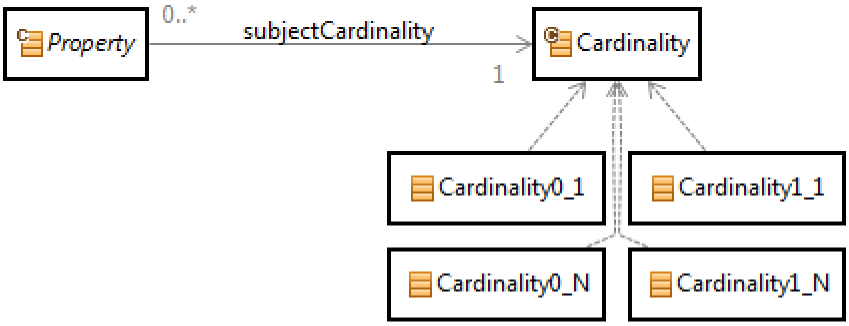


SubjectCardinality

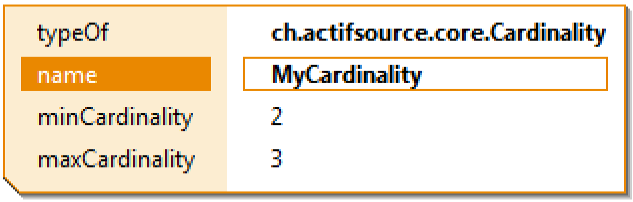
The Property defines the subject cardinality which determines how many resources of type B can be referenced by a resource of type A via relation b.



You may choose one of the following predefined cardinality instances.



It is also possible to define any other cardinality just by specifying minCardinality and maxCardinality.



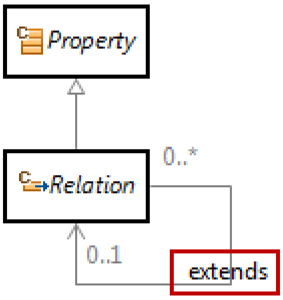
typeOf	ch.actifsource.core.Cardinality
name	MyCardinality
minCardinality	2
maxCardinality	3

4.4.2 Relation

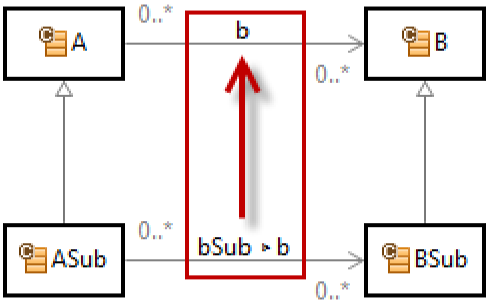
The relation acts as an abstract base class for any relation property between classes.

4.4.3 Extends

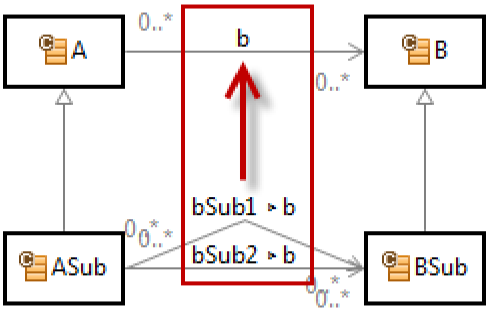
The relation extends allows you to define sub-properties of existing properties in a super class.



ASub.bSub extends A.b in the following example. That means that instances of ASub can have only references to resources of type BSub while it is still possible to access the ASub.bSub resources via A.b as type of B.



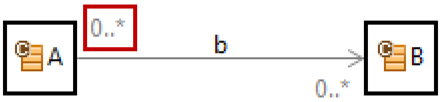
It is also possible to have more than one sub-property that extends the same super-property, as long as the sum of cardinalities of the sub-properties complies with the cardinality of the super-property.



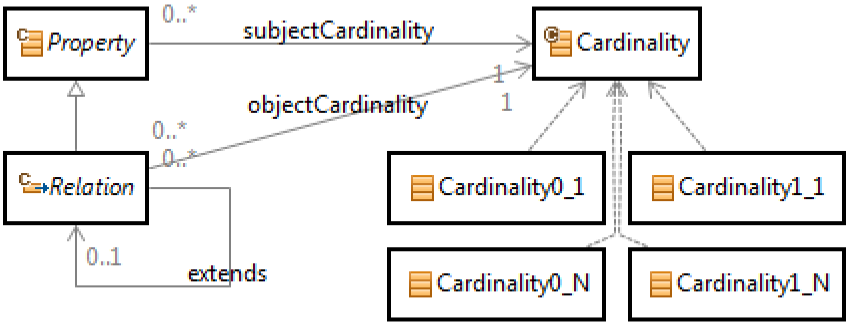
Please note that is it only allowed to extend a property of the same type (i.e. **UseRelation** extends **UseRelation**, **OwnRelation** extends **OwnRelation**).

ObjectCardinality

The Relation defines the object cardinality which determines how many resources of type A can reference a resource of type B via relation b.

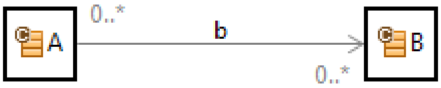


You may choose one of the following predefined cardinality instances.

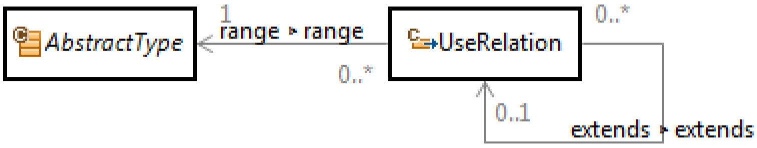


4.4.4 UseRelation

The **UseRelation** (UML: Association) references another resource without affecting the lifetime of this resource. Deleting the referencing resource will not delete the referenced resource.

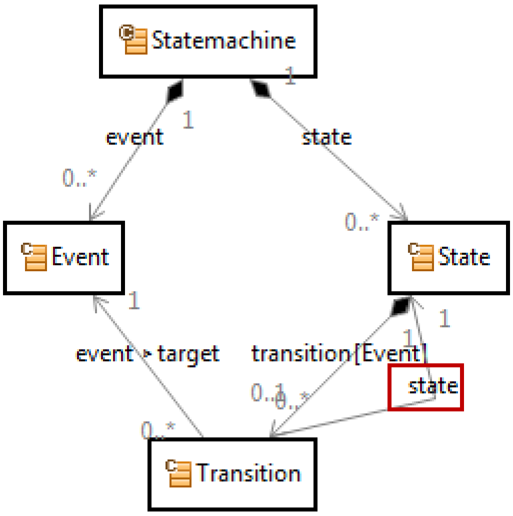


The range of **UseRelation** is **AbstractType**.

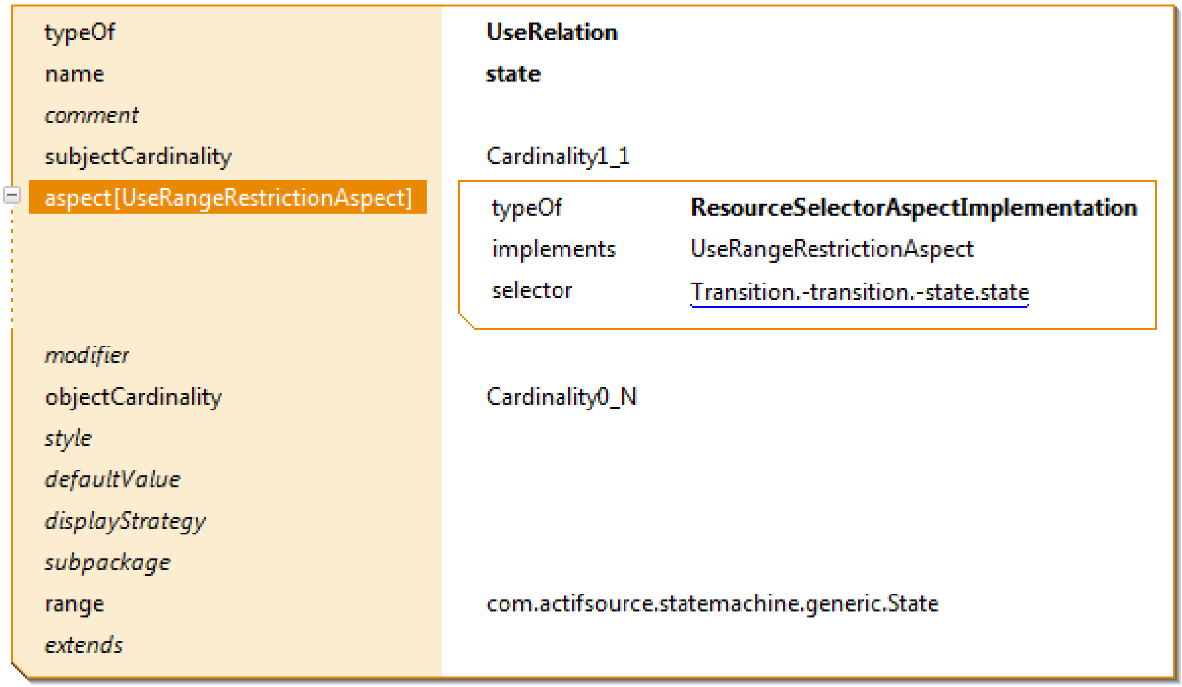


UseRangeRestrictionAspect

You may want to restrict resources that can be referenced by a **UseRelation** depending on their context. Consider a state machine where only target states of the own state machine shall be selected.



Let's define a **UseRangeRestriction** aspect for Transition.state that allows only states of the own state machine. Select the **ResourceSelectorAspectImplementation** for the easy to use Selector-Syntax (see Chapter 9.3.2 SelectorFunction) or the **JavaAspectImplementation** for a powerful Java implementation.

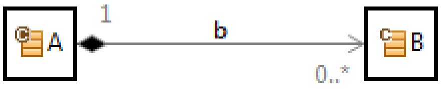


typeOf	**UseRelation**	
name	**state**	
comment		
subjectCardinality	Cardinality1_1	
aspect[UseRangeRestrictionAspect]	typeOf	**ResourceSelectorAspectImplementation**
	implements	UseRangeRestrictionAspect
	selector	Transition.-transition.-state.state
modifier		
objectCardinality	Cardinality0_N	
style		
defaultValue		
displayStrategy		
subpackage		
range	com.actifsource.statemachine.generic.State	
extends		

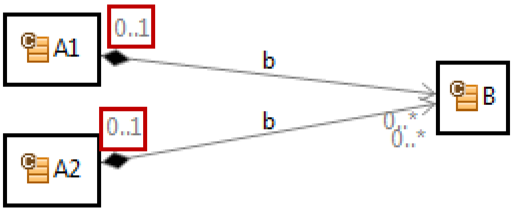
Starting from Transition navigating backwards via transition and state (note the minus sign for backward navigation) to Statemachine where navigating forward to all States.

4.4.5 OwnRelation

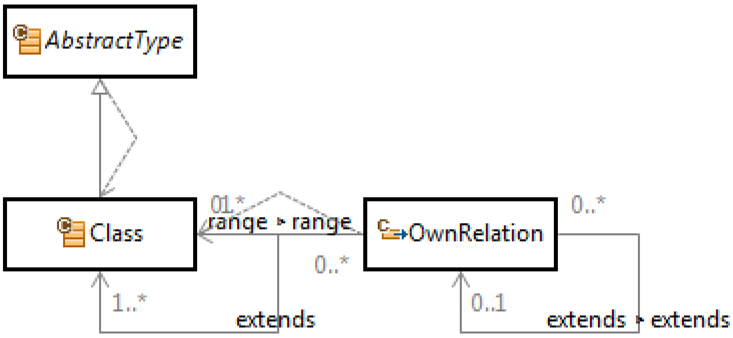
The **OwnRelation** (UML: Aggregation, Composition) aggregates another resource. Deleting the aggregating resource of type A will delete the referenced resources of type B.



Please note that you have to adjust the **ObjectCardinality** if B may be owned *either* from A1 *or* A2.



The range of **OwnRelation** is **Class**.

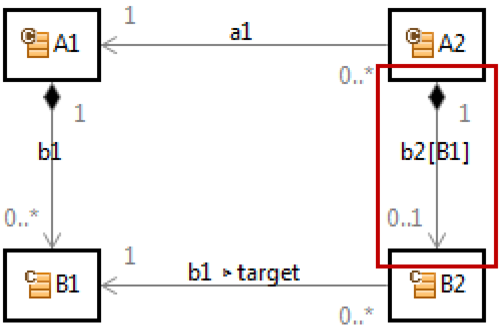


4.4.6 DecoratingRelation

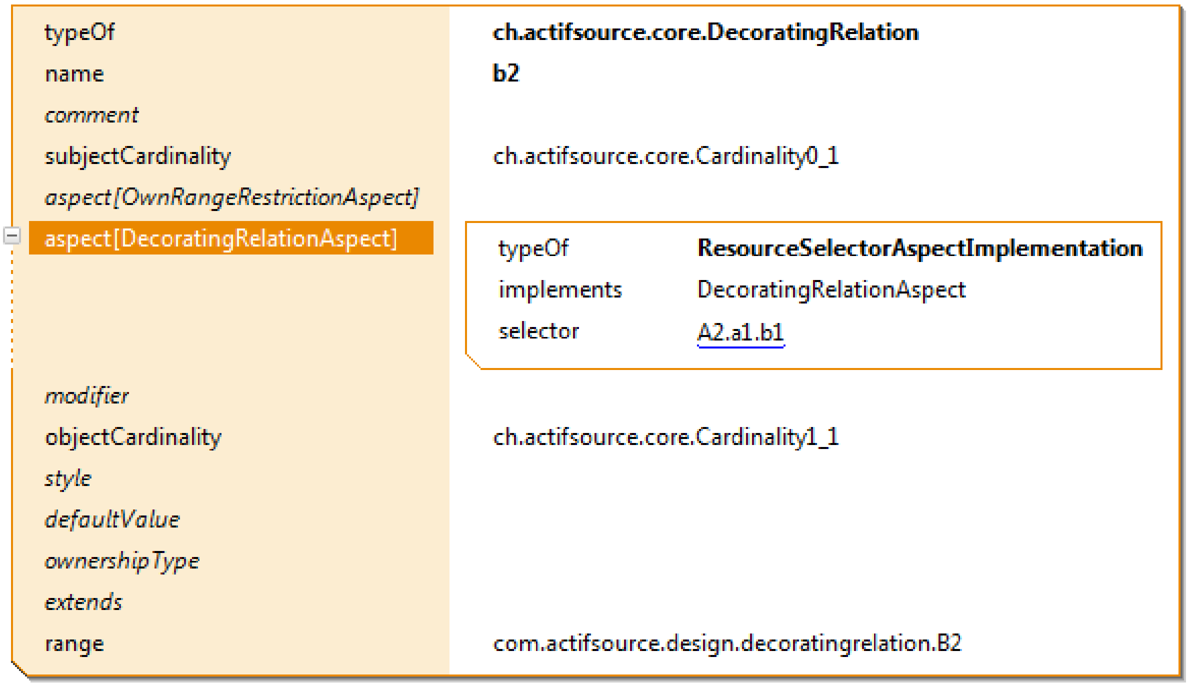
Combining two existing models together is a very important task. Sometimes this is done by so called model transformation. Actifsource chooses another way, because model transformation often only works in one direction. Actifsource allows decorating existing resources with other ones. That means that we can add any auxiliary information to already existing information. You will find out that this is a very powerful concept.

The **DecoratingRelation** allows building a homomorphism. A homomorphism is a structure-preserving map between two structures. The word homomorphism comes from the ancient Greek language: ὁμός (homos) meaning "same" and μορφή (morphe) meaning "shape".

Consider a resource A1 with a list b1 of resources of type B1. Consider a second List A2 that has a reference a1 to a resource of type A1. The decorating relation A2.b2 shall have 0..1 resources of type B2 for any resources reached via A2.a1.b1.

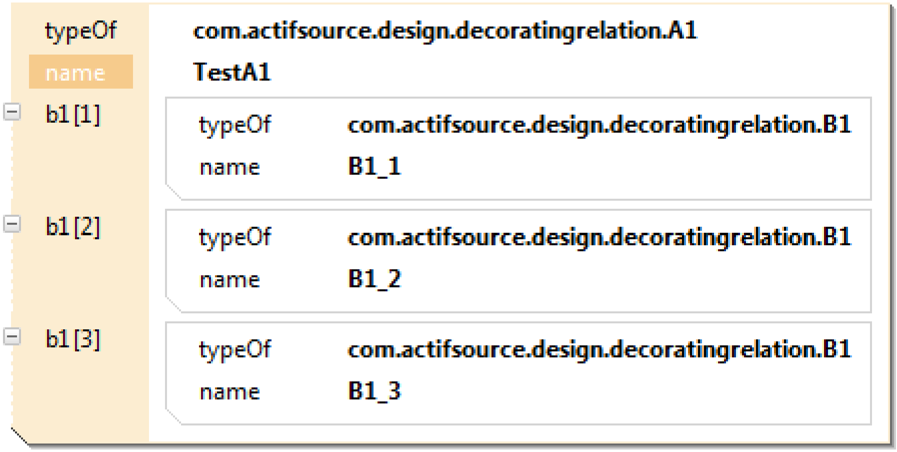


The decorating relation A2.b2 needs a decorating aspect which defines where to find the resources that shall be decorated.



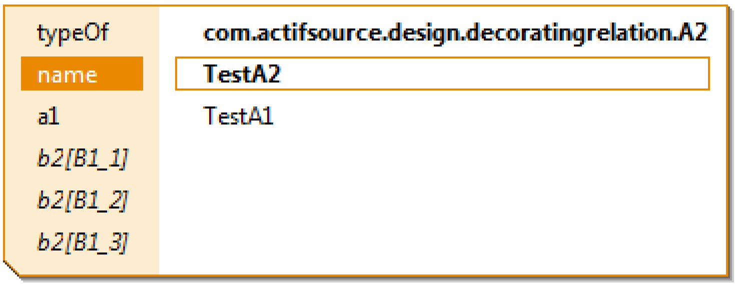
typeOf	**ch.actifsource.core.DecoratingRelation**
name	**b2**
comment	
subjectCardinality	ch.actifsource.core.Cardinality0_1
aspect [OwnRangeRestrictionAspect]	
aspect [DecoratingRelationAspect]	typeOf **ResourceSelectorAspectImplementation** implements DecoratingRelationAspect selector A2.a1.b1
modifier	
objectCardinality	ch.actifsource.core.Cardinality1_1
style	
defaultValue	
ownershipType	
extends	
range	com.actifsource.design.decoratingrelation.B2

Let us create a resource TestA1 of type A1 first.



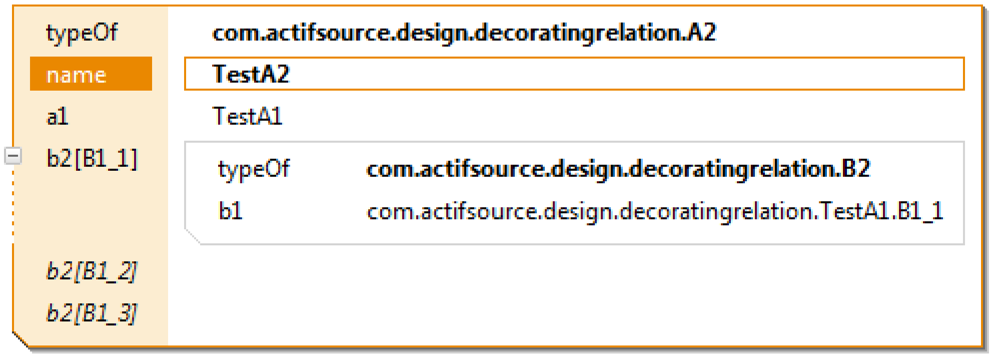
typeOf	**com.actifsource.design.decoratingrelation.A1**
name	**TestA1**
b1 [1]	typeOf **com.actifsource.design.decoratingrelation.B1** name **B1_1**
b1 [2]	typeOf **com.actifsource.design.decoratingrelation.B1** name **B1_2**
b1 [3]	typeOf **com.actifsource.design.decoratingrelation.B1** name **B1_3**

Now let us create a resource TestA2 of type A2 with a reference to TestA1. Please note that A2.b2 is now decorating A2.a1.b1 what is indicated by the brackets (i.e. b1[B1_1]).



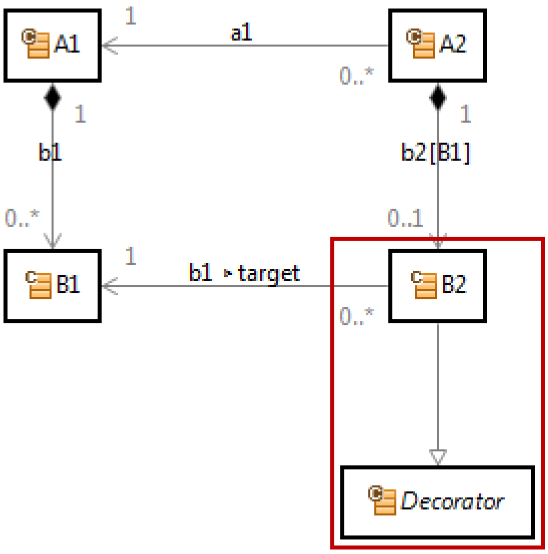
typeOf	**com.actifsource.design.decoratingrelation.A2**
name	**TestA2**
a1	TestA1
b2[B1_1]	
b2[B1_2]	
b2[B1_3]	

Creating a decorated resource will automatically fill in the target property b1.

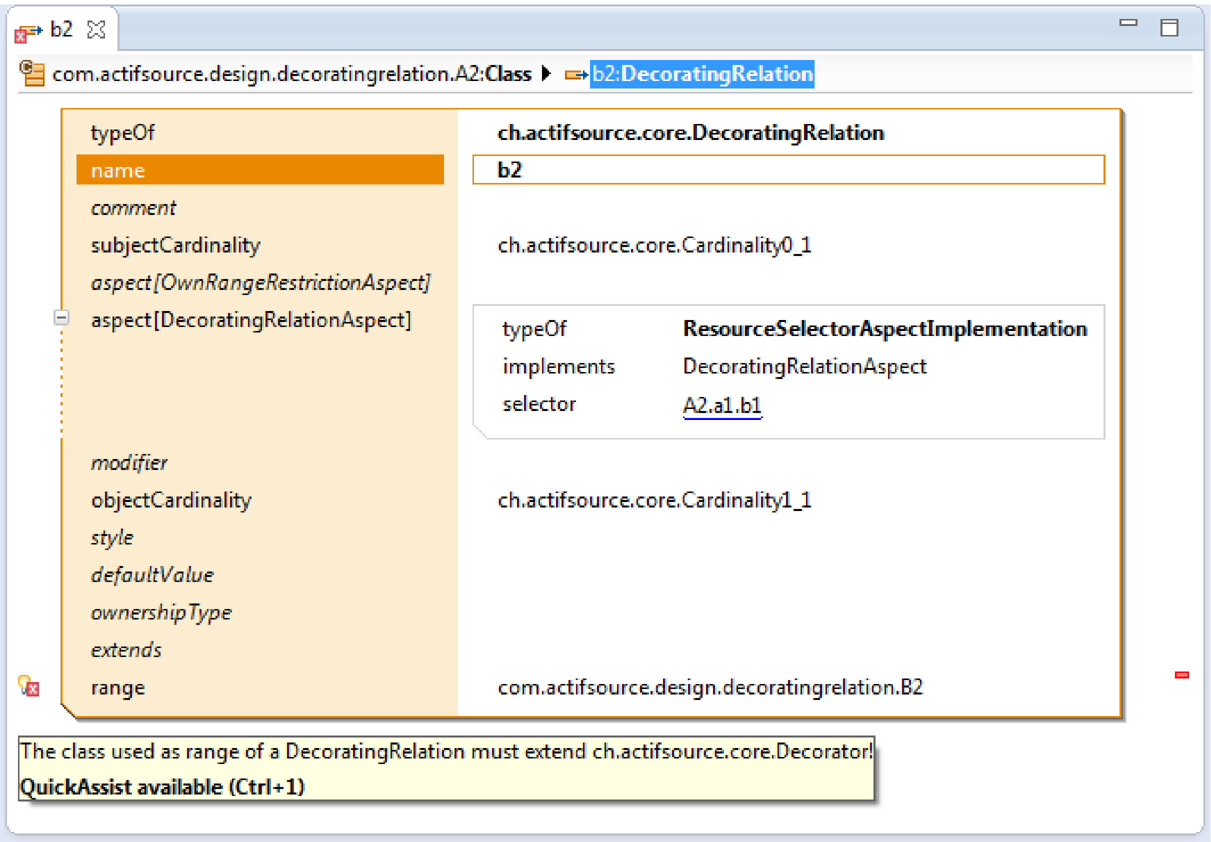


typeOf	**com.actifsource.design.decoratingrelation.A2**
name	**TestA2**
a1	TestA1
b2[B1_1]	typeOf **com.actifsource.design.decoratingrelation.B2** b1 com.actifsource.design.decoratingrelation.TestA1.B1_1
b2[B1_2]	
b2[B1_3]	

Please note that the decorating relation needs a subclass of type **Decorator**.



If your decorating relation does not point to a resource which fits the requirements Actifsource comes up with a *quick assist* (see also Chapter 3.10 Quick Assist).



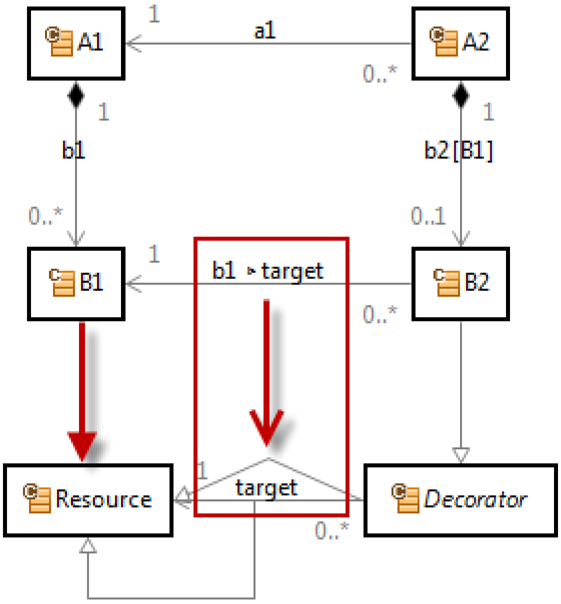
b2

com.actifsource.design.decoratingrelation.A2:Class ▸ b2:DecoratingRelation

typeOf	**ch.actifsource.core.DecoratingRelation**
name	**b2**
comment	
subjectCardinality	ch.actifsource.core.Cardinality0_1
aspect[OwnRangeRestrictionAspect]	
aspect[DecoratingRelationAspect]	typeOf **ResourceSelectorAspectImplementation** implements DecoratingRelationAspect selector A2.a1.b1
modifier	
objectCardinality	ch.actifsource.core.Cardinality1_1
style	
defaultValue	
ownershipType	
extends	
range	com.actifsource.design.decoratingrelation.B2

The class used as range of a DecoratingRelation must extend ch.actifsource.core.Decorator!
QuickAssist available (Ctrl+1)

The *quick assist* extends the range of the decorating relation from **Decorator** and adds a sub relation to **Decorator.target** with type of the target of your decorating aspect.

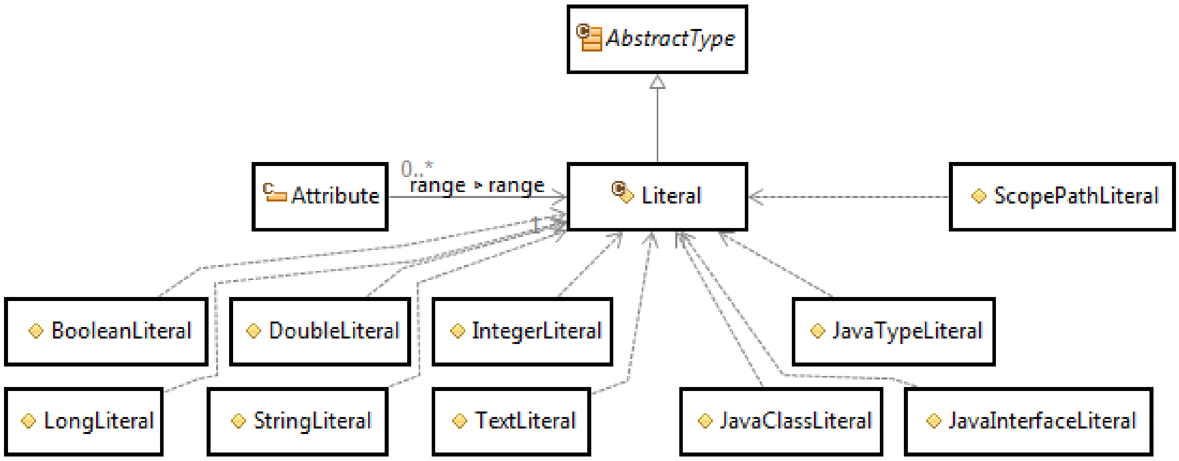


4.4.7 SelectorRelation

[TBD]

4.4.8 Attribute

The Attribute allows creating simple literals. There are some predefined literal instances that you might use. Please note that the range of an attribute is **Literal** where **Literal** is also an **AbstractType** as already seen for **Class** (see Chapter 4.4.1 Property).



BooelanLiteral

The Boolean literal allows the values true or false only.

DoubleLiteral

The double literal is a 64 bit floating point value.

IntegerLiteral

The integer literal is a 32 bit integral value.

LongLiteral

The long literal is a 64 bit integral value.

StringLiteral

The string literal is a single-line string value.

TextLiteral

The text literal is a multi-line string value.

ScopePathLiteral

The scope literal allows referencing any file in the current project.

JavaTypeLiteral

The Java type literal allows referencing a Java class or interface.

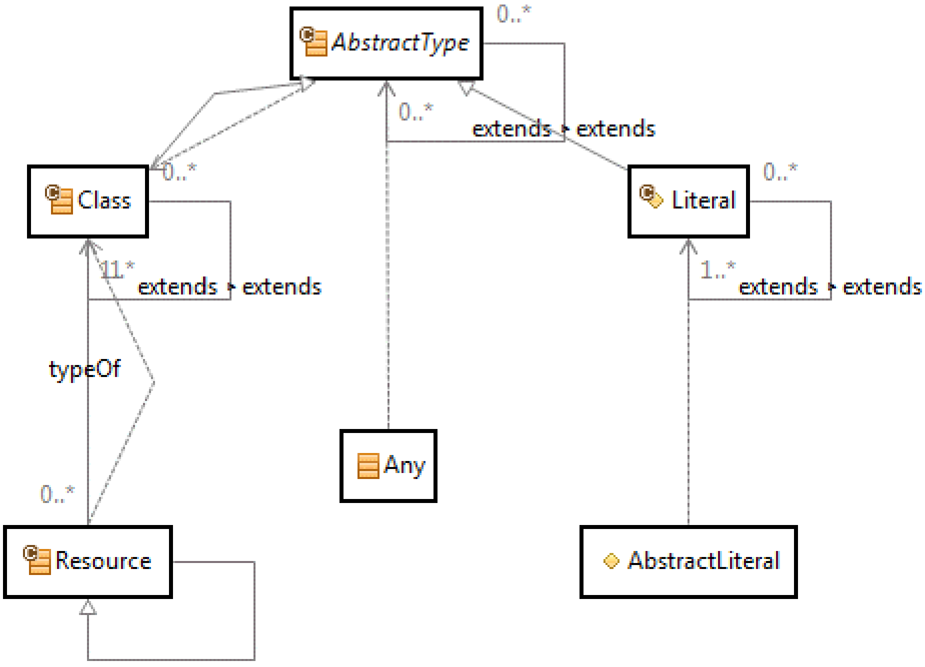
JavaClassLiteral

The Java type literal allows referencing a Java class.

JavaInterfaceLiteral

The Java type literal allows referencing a Java interface.

4.5 AbstractType

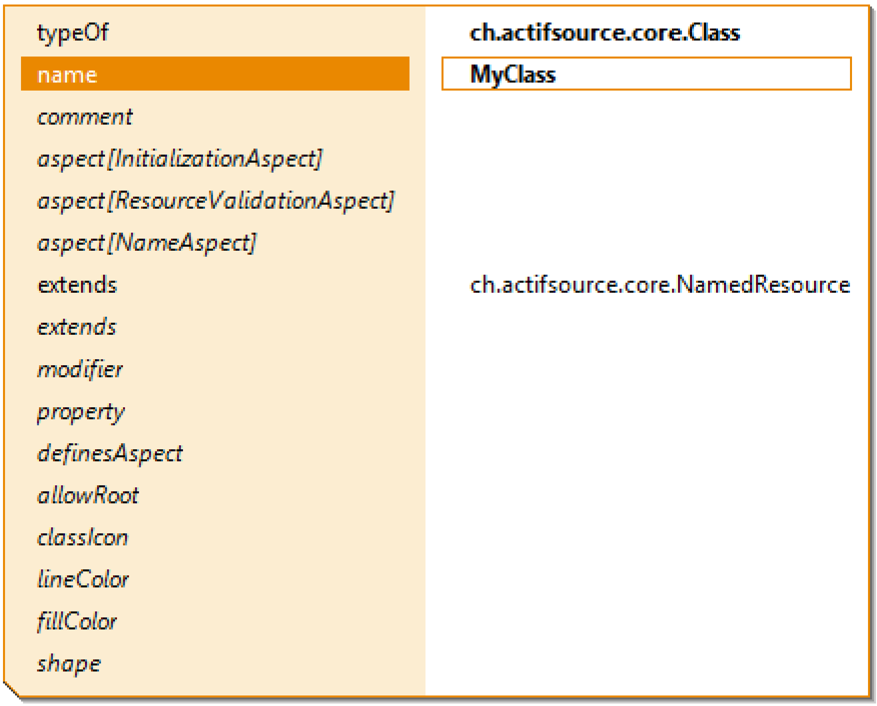


As we have seen before, **Class** and **Literal** both extend **AbstractType**. Furthermore, **AbstractLiteral** is of type **Literal** and **Resource** is of type **Class**. To complete this picture, we can add **Any** which is of type **AbstractType**.

4.6 Core Resources

The Actifsource Core provides a set of resources that allows you to build your own meta-model. The most important resources are **Class** with its **Properties** and **Enum**.

4.6.1 ch.actifsource.core.Class



typeOf	**ch.actifsource.core.Class**
name	**MyClass**
comment	
aspect [InitializationAspect]	
aspect [ResourceValidationAspect]	
aspect [NameAspect]	
extends	ch.actifsource.core.NamedResource
extends	
modifier	
property	
definesAspect	
allowRoot	
classIcon	
lineColor	
fillColor	
shape	

typeOf

To act as a class a resource has to be of type **Class**. To be type something of means to be an instance of something.

comment

The comment text literal let you comment your classes and literals. Comments are shown in tooltips when hovering the mouse pointer over class and property names. Note that the comment property is inherited by extending from **Commentable**.

Extending from **Commentable** allows writing comment for any resource and activates the tooltip functionality.

aspect[InitializationAspect]

The initialization aspect initializes a resource during creation. To be provided as Java class.

aspect[ResourceValidationAspect]

The resource validation aspect defines specific validation rules for a resource. To be provided as Java class.

aspect[NameAspect]

The name aspect defines the name of a resource. To be provided as Java class or in the simple selector syntax. Note that you might define a selector aspect pointing to a TemplateFunction (see Chapter 9.3.6 TemplateFunction) or a TemplateLineFunction (see Chapter 9.3.7 TemplateLineFunction).

extends

Defines the base class and inherits all properties when instantiating.

modifier

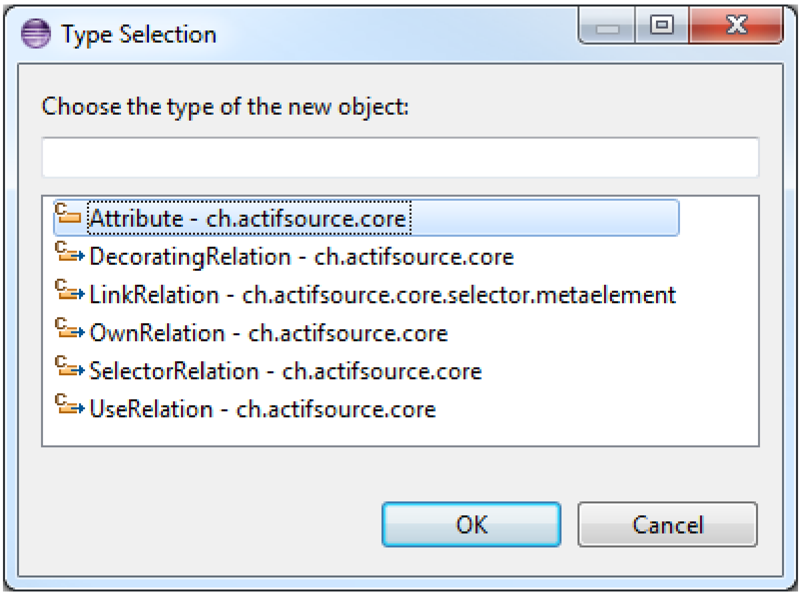
The modifier defines if a class can be instantiated or sub classed.

Modifier	Description
Abstract	No instance allowed of this class.
Final	No subclass allowed of this class.

property

Defines the property of this class. Properties carry the data when instantiating the class. Choose the property type you need (see also Chapter 0

Property).



Type Selection

Choose the type of the new object:

- Attribute - ch.actifsource.core
- DecoratingRelation - ch.actifsource.core
- LinkRelation - ch.actifsource.core.selector.metaelement
- OwnRelation - ch.actifsource.core
- SelectorRelation - ch.actifsource.core
- UseRelation - ch.actifsource.core

definesAspect
Defines aspects for instances of this class (see also aspect[InitializationAspect], aspect[ResourceValidationAspect], aspect[NameAspect] which are defined in the aspect property of the class Class.

allowRoot
A **Class** is a root class if it is not aggregated exclusively (that mean owned by an **ownRelation** with **ObjectCardinality1_1**). Set this flag to true or false if you want to overwrite the Actifsource logic.

Please note that only root classes can be created directly via the New Actifsource Resource Tool (see Chapter 2.9.1 New Actifsource Resource).

classIcon
Sets an icon for this class and all instances. You can select an icon project by using content assist to browse the current project. The icon size shall be 16 x 16 pixels.

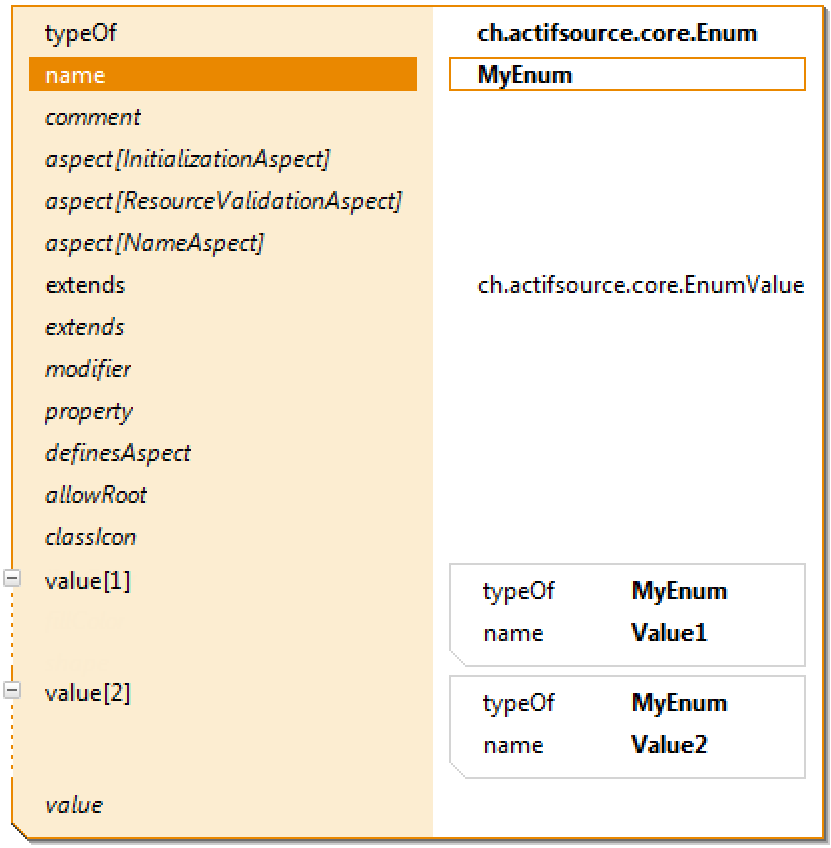
lineColor
Deprecated.

fillColor
Deprecated.

shape
Deprecated.

4.6.2 ch.actifsource.core.Enum

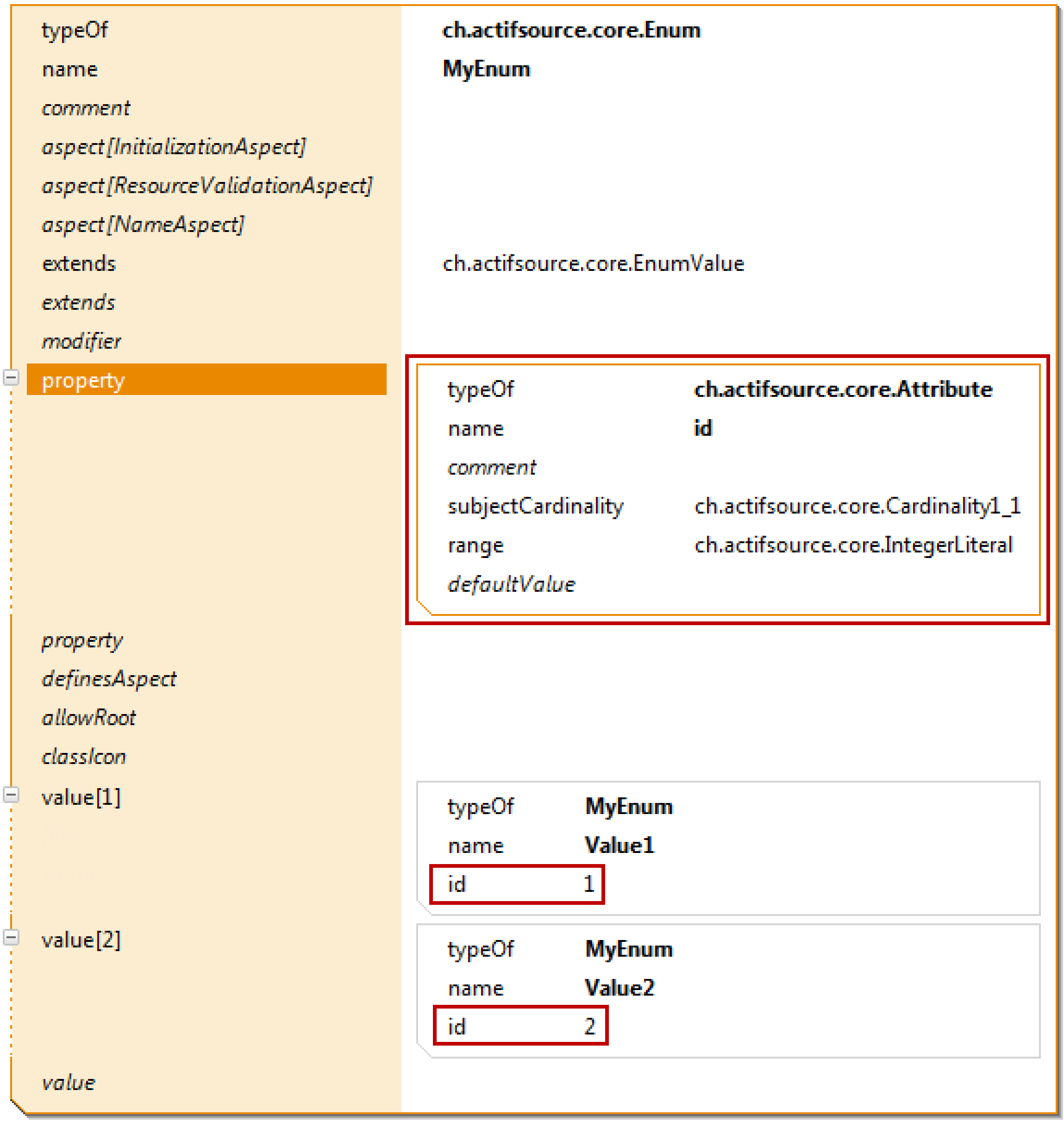
The *enum* is a class with a set of *values*.



typeOf	**ch.actifsource.core.Enum**
name	**MyEnum**
comment	
aspect[InitializationAspect]	
aspect[ResourceValidationAspect]	
aspect[NameAspect]	
extends	ch.actifsource.core.EnumValue
extends	
modifier	
property	
definesAspect	
allowRoot	
classIcon	
value[1]	typeOf **MyEnum** name **Value1**
value[2]	typeOf **MyEnum** name **Value2**
value	

Value

Technically the value is an instance of the enum and therefore fully typed. Note that you might define properties in your enum which are than accessible in the values.



typeOf	**ch.actifsource.core.Enum**
name	**MyEnum**
comment	
aspect[InitializationAspect]	
aspect[ResourceValidationAspect]	
aspect[NameAspect]	
extends	ch.actifsource.core.EnumValue
extends	
modifier	
property	typeOf **ch.actifsource.core.Attribute** name **id** *comment* subjectCardinality ch.actifsource.core.Cardinality1_1 range ch.actifsource.core.IntegerLiteral *defaultValue*
property	
definesAspect	
allowRoot	
classIcon	
value[1]	typeOf **MyEnum** name **Value1** id 1
value[2]	typeOf **MyEnum** name **Value2** id 2
value	

5 Diagram Editor

5.1 Overview

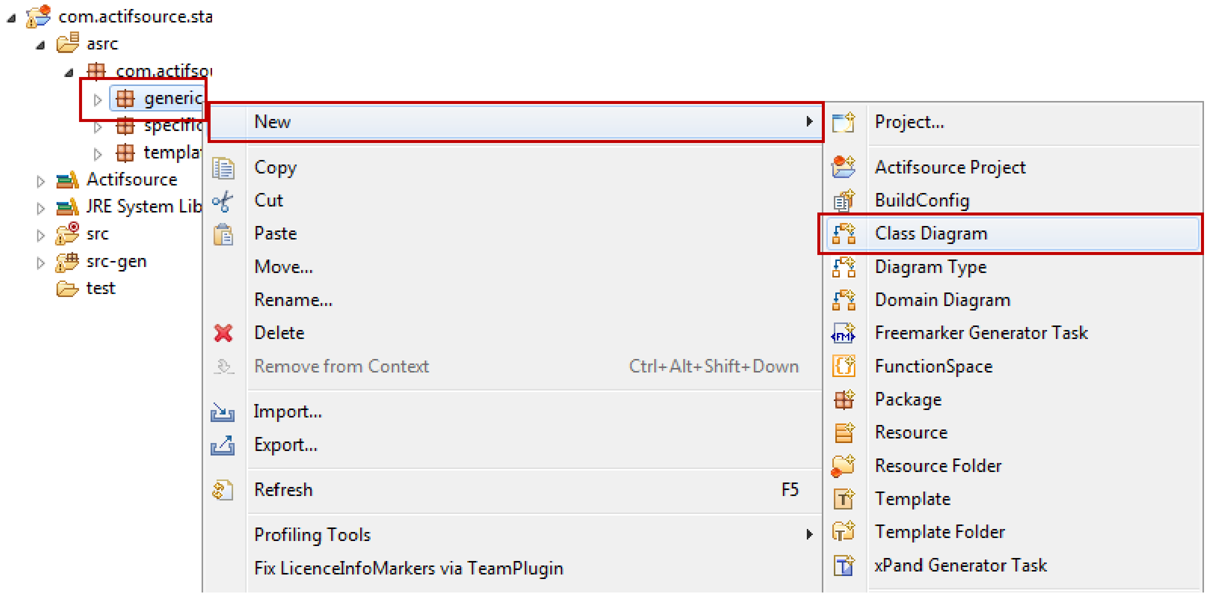
Actifsource offers graphical editors to view and edit the meta-model and model. For editing the model it is possible to define domain-specific editors.

5.2 Class Diagram Editor

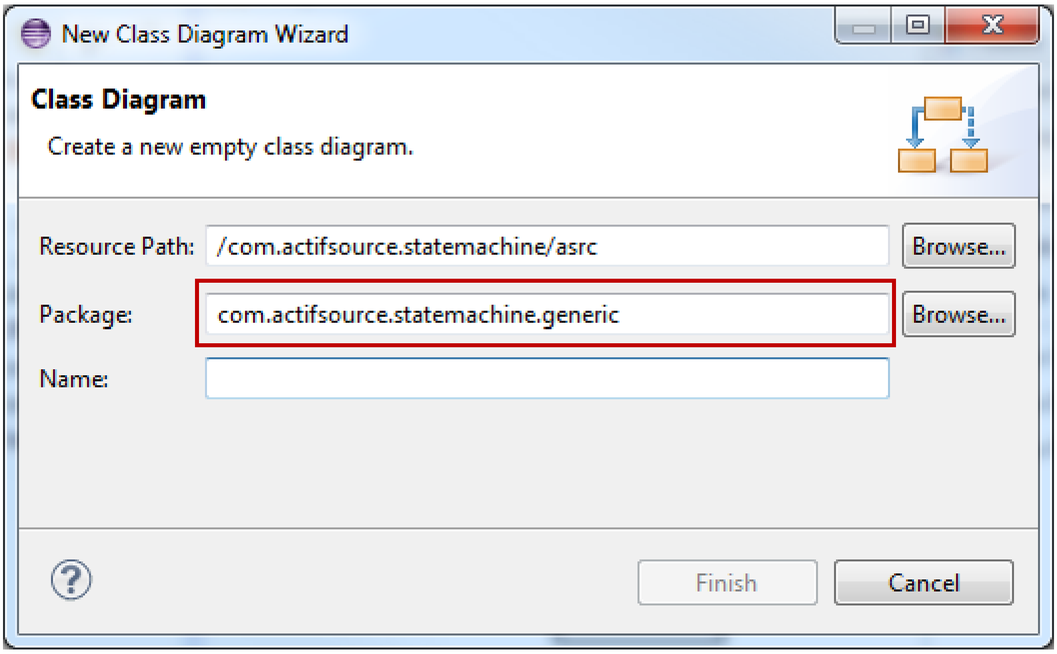
The class diagram editor allows you to create new meta-models based on Actifsource classes.

5.2.1 New Class Diagram

You can create a new class diagram via context menu in the project explorer or File/New.



Preselecting a package will directly fill in the resource path and package in the wizard.



Class Diagram

Create a new empty class diagram.

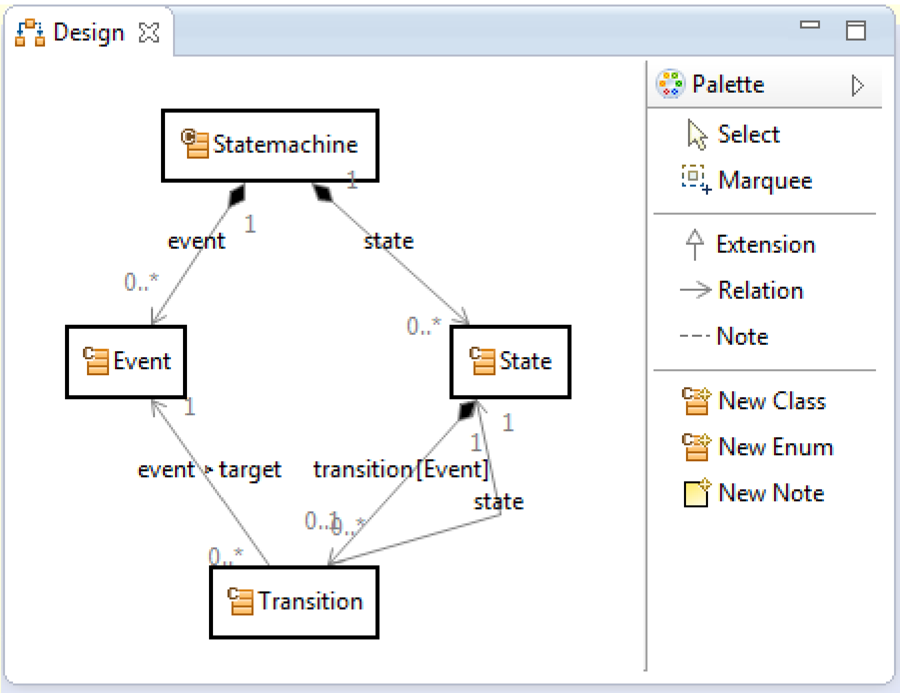
Resource Path: /com.actifsource.statemachine/asrc

Package: com.actifsource.statemachine.generic

Name:

5.2.2 Palette

Use the palette to edit your diagram.



Select

Selects one or many classes from the diagram. Use Ctrl+Click for multi select.

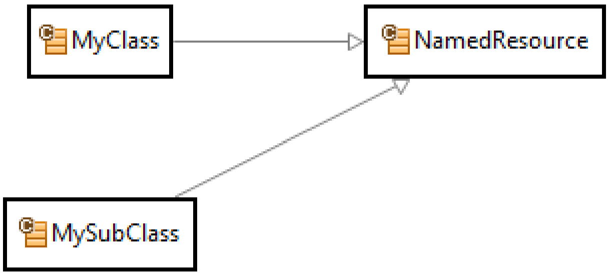
Marquee

Selects classes from the diagram within a rectangle.

Extension

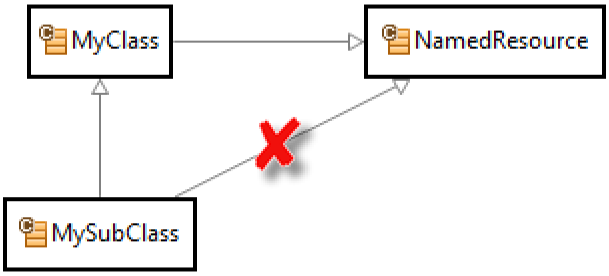
Derives a class from another class by adding a **Class.extend** statement in the sub class.

Note that are special rules for your convenience. Consider creating two new Classes MyClass and MySubClass. Both are extending **NamedResource** by default.



Inserting an extends-relation from MySubClass to MyClass would add an extends-relation from MySubClass to MyClass as expected. At the same time the extends-relation from MySubSubClass to **NamedResource** is removed because MyClass already extends **NamedResource**.

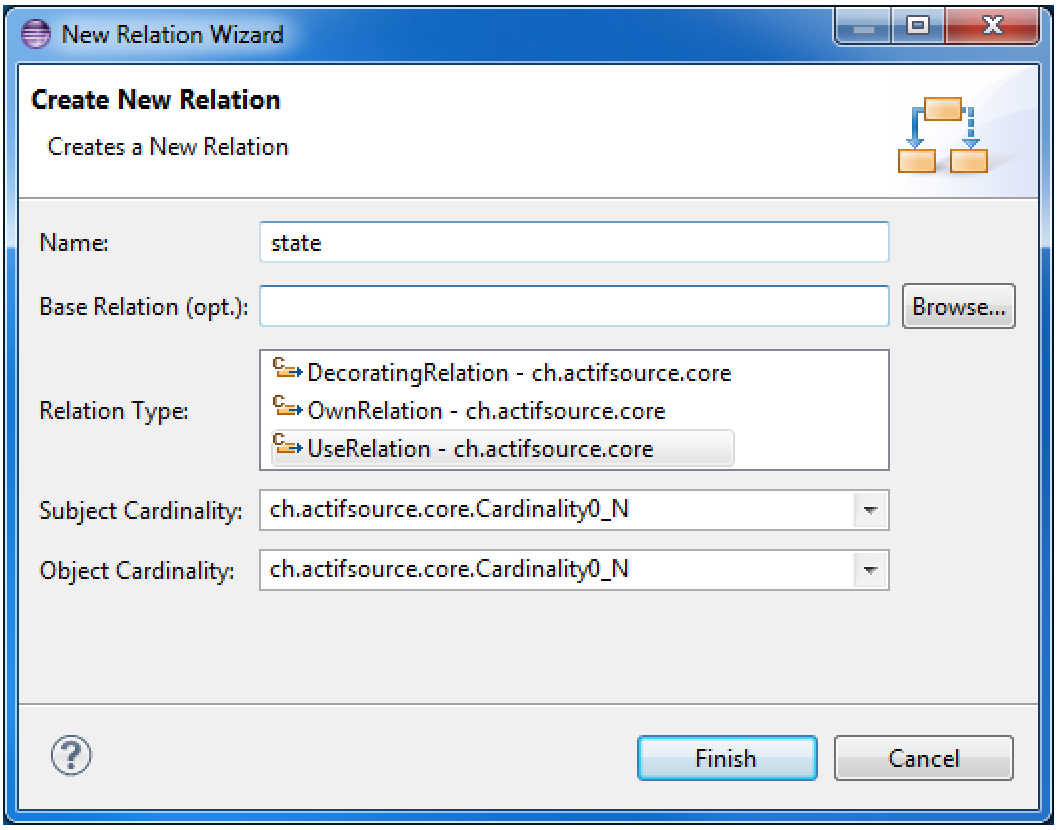
The same rule also applies when extending from **Resource**.



Relation

Inserts a relation from a class to another. Select one of the relation types or a base relation. Selecting a base relation automatically selects the relation type needed (see Chapter 0

Property).



Create New Relation

Creates a New Relation

Name: state

Base Relation (opt.):

Relation Type:
- DecoratingRelation - ch.actifsource.core
- OwnRelation - ch.actifsource.core
- UseRelation - ch.actifsource.core

Subject Cardinality: ch.actifsource.core.Cardinality0_N

Object Cardinality: ch.actifsource.core.Cardinality0_N

Note

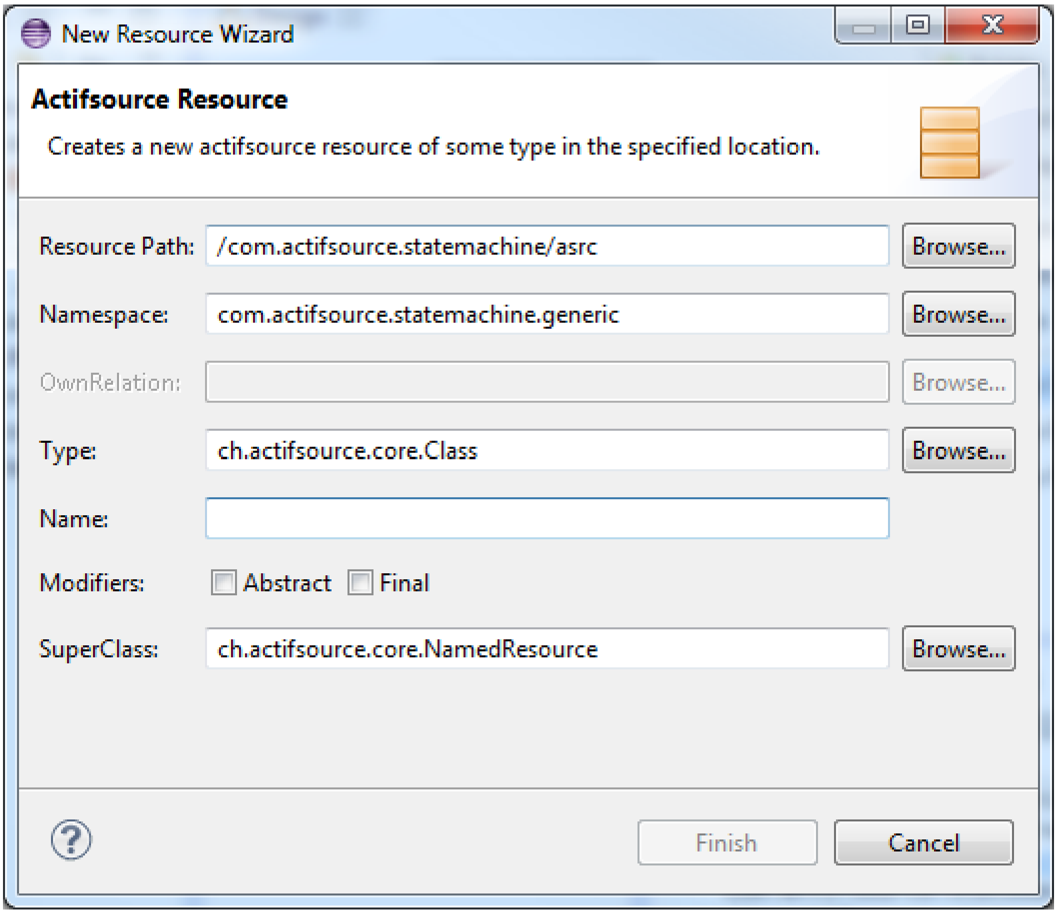
Connects class to any note. See also New Note.

New Class

Inserts a new class on the diagram and in the model. Note that the preselected namespace (package and/or containing resource) is the same as the package of the class diagram.

Note that you can directly select the super class in this dialog. While **NamedResource** is the default you may select **Resource** to create an unnamed resource, or any other **Class**.

For other properties see also Chapter 4.6.1 ch.actifsource.core.Class.



New Resource Wizard

Actifsource Resource

Creates a new actifsource resource of some type in the specified location.

Resource Path: /com.actifsource.statemachine/asrc [Browse...]

Namespace: com.actifsource.statemachine.generic [Browse...]

OwnRelation: [Browse...]

Type: ch.actifsource.core.Class [Browse...]

Name:

Modifiers: □ Abstract □ Final

SuperClass: ch.actifsource.core.NamedResource [Browse...]

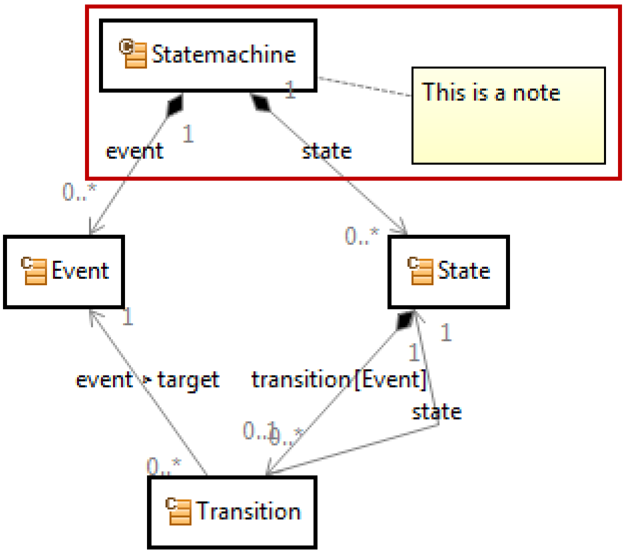
[Finish] [Cancel]

New Enum

Inserts a new enum on the diagram and in the model. For other properties see also Chapter 4.6.2 ch.actifsource.core.Enum.

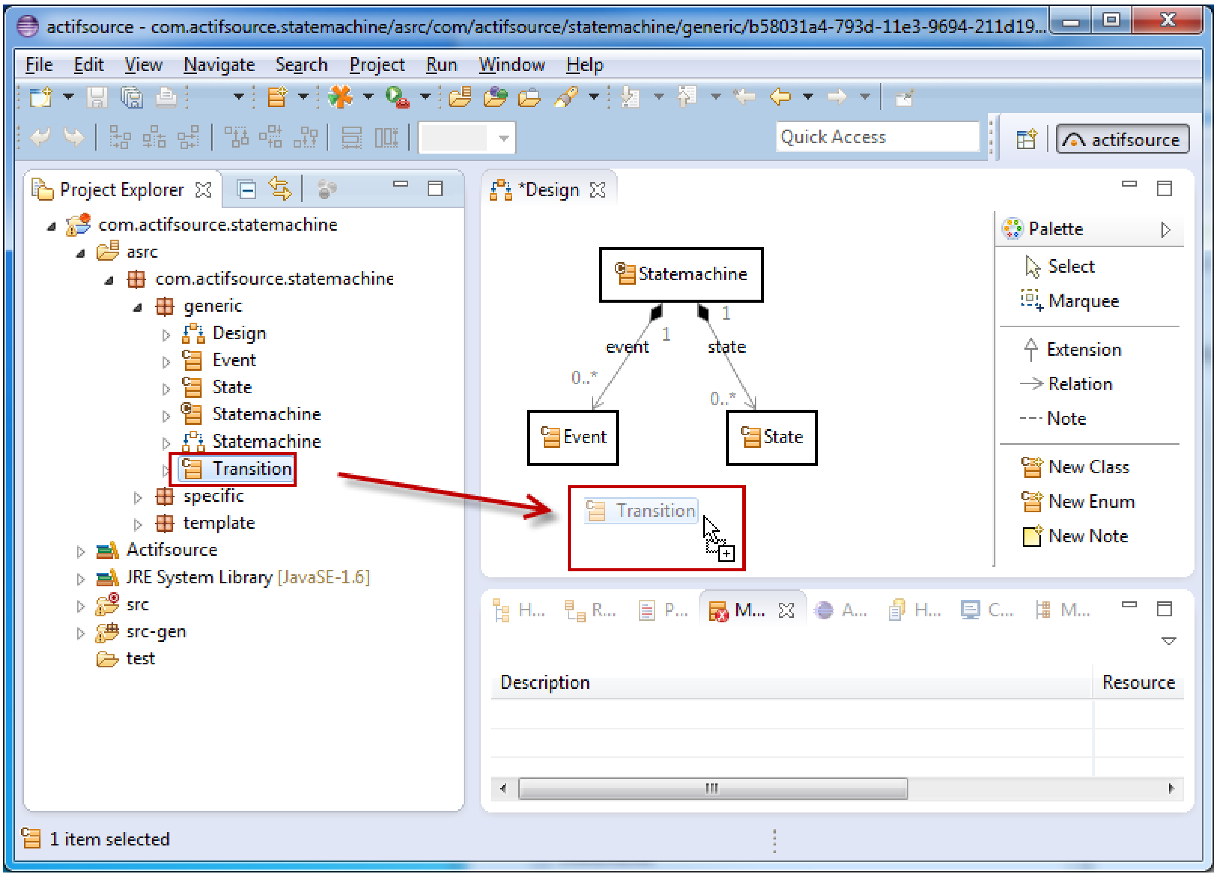
New Note

Inserts a new note on the diagram. Click on the first text line of the note to enter the edit mode.



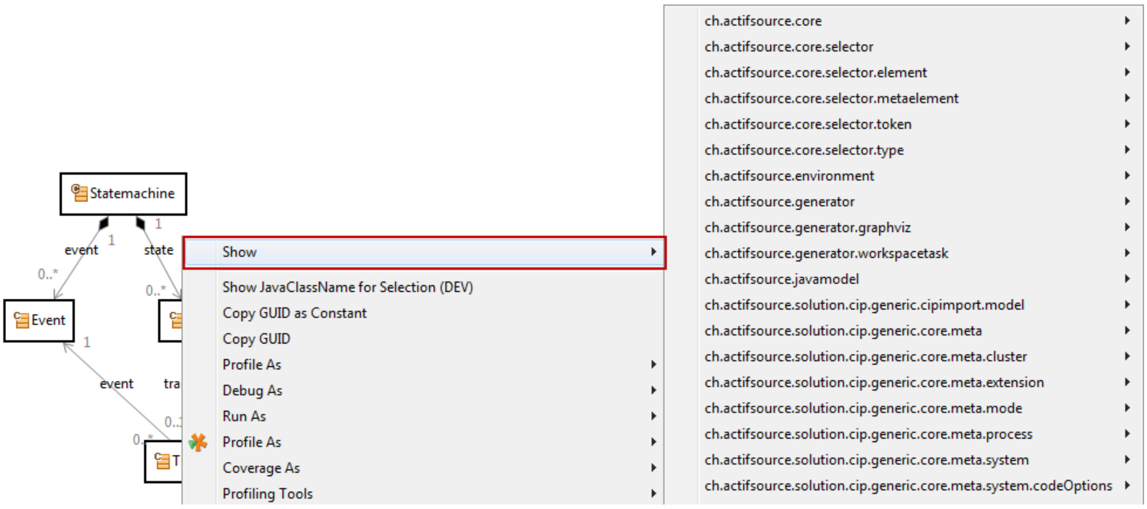
5.2.3 Drag and Drop

Use the *drag and drop* feature from the project explorer to add an existing class to your diagram.



5.2.4 Context Menu

Use the context menu on the diagram background.

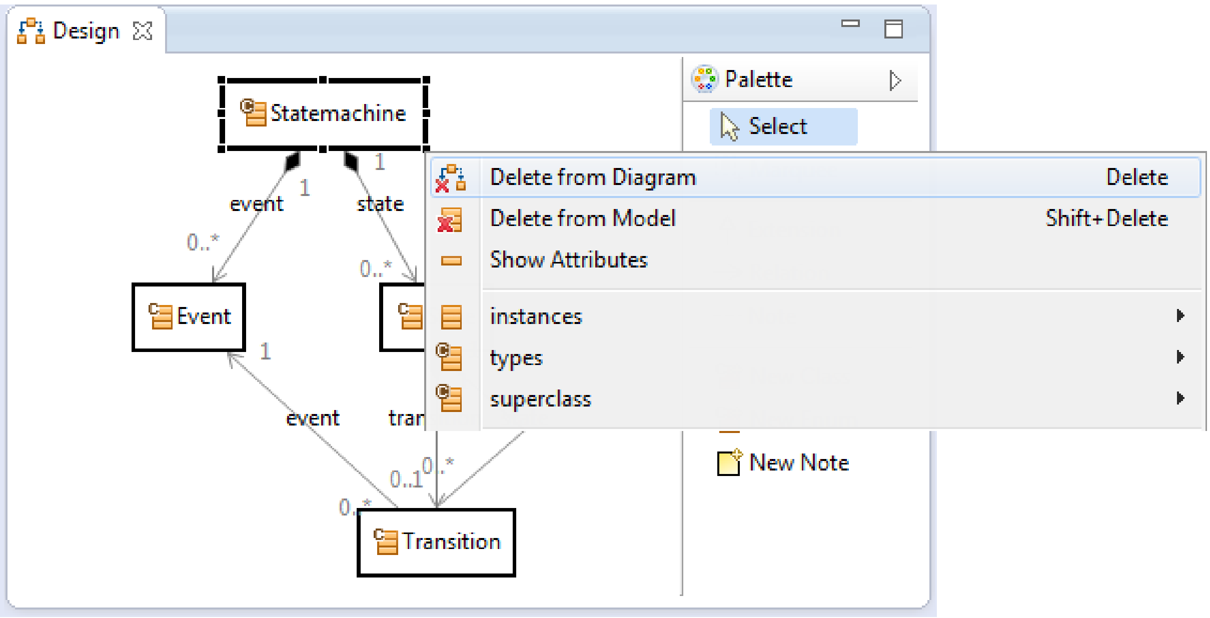

Statemachine
event
state
Event
Show
Show JavaClassName for Selection (DEV)
Copy GUID as Constant
Copy GUID
Profile As
Debug As
Run As
Profile As
Coverage As
Profiling Tools
ch.actifsource.core
ch.actifsource.core.selector
ch.actifsource.core.selector.element
ch.actifsource.core.selector.metaelement
ch.actifsource.core.selector.token
ch.actifsource.core.selector.type
ch.actifsource.environment
ch.actifsource.generator
ch.actifsource.generator.graphviz
ch.actifsource.generator.workspacetask
ch.actifsource.javamodel
ch.actifsource.solution.cip.generic.cipimport.model
ch.actifsource.solution.cip.generic.core.meta
ch.actifsource.solution.cip.generic.core.meta.cluster
ch.actifsource.solution.cip.generic.core.meta.extension
ch.actifsource.solution.cip.generic.core.meta.mode
ch.actifsource.solution.cip.generic.core.meta.process
ch.actifsource.solution.cip.generic.core.meta.system
ch.actifsource.solution.cip.generic.core.meta.system.codeOptions

Show

Shows any existing resource in the scope of your project. Use this feature to insert resources from third party or Actifsource models to your diagram (i.e. Actifsource Core Model).

5.2.5 Class Context Menu

Use the content menu on any class.



Delete from Diagram

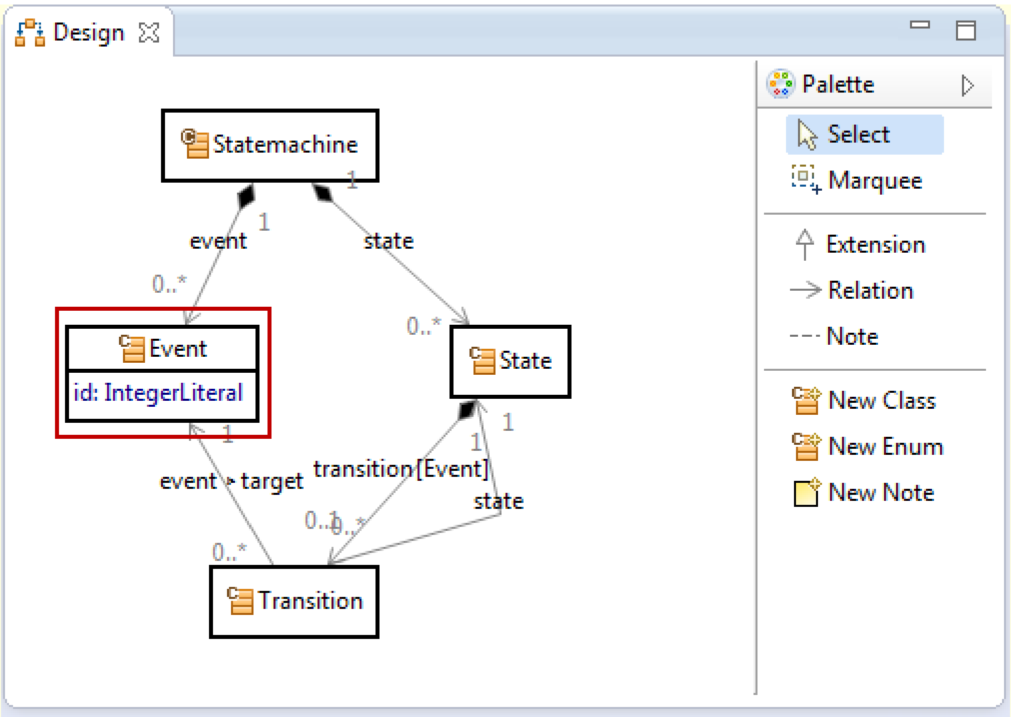
Deletes this class from the diagram only but not from the model. Use Delete on the keyboard to delete selected classes from the diagram.

Delete from Model

Deletes this class from the diagram and from the model. Use Shift+Delete on the keyboard to delete selected classes from the diagram and from the model.

Show Attributes

Shows attributes (literals) in an UML-like style.

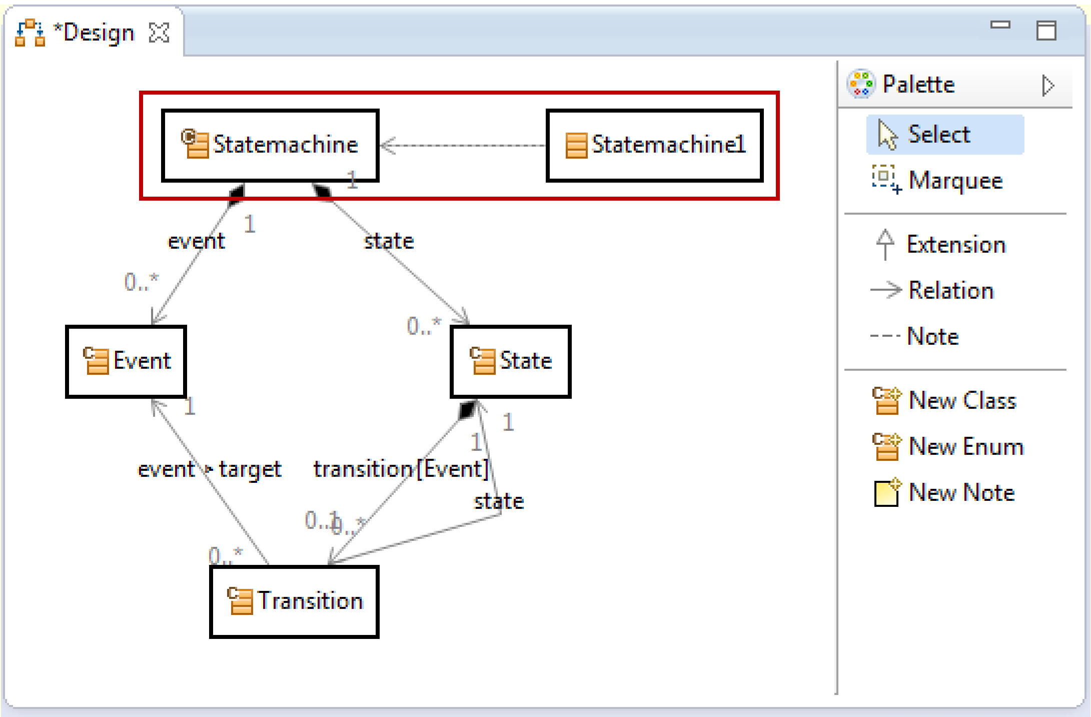


Hide Attributes

Hides shown attribute (see also Show Attributes).

Instances

Inserts instances of a specific class on the diagram.



Types

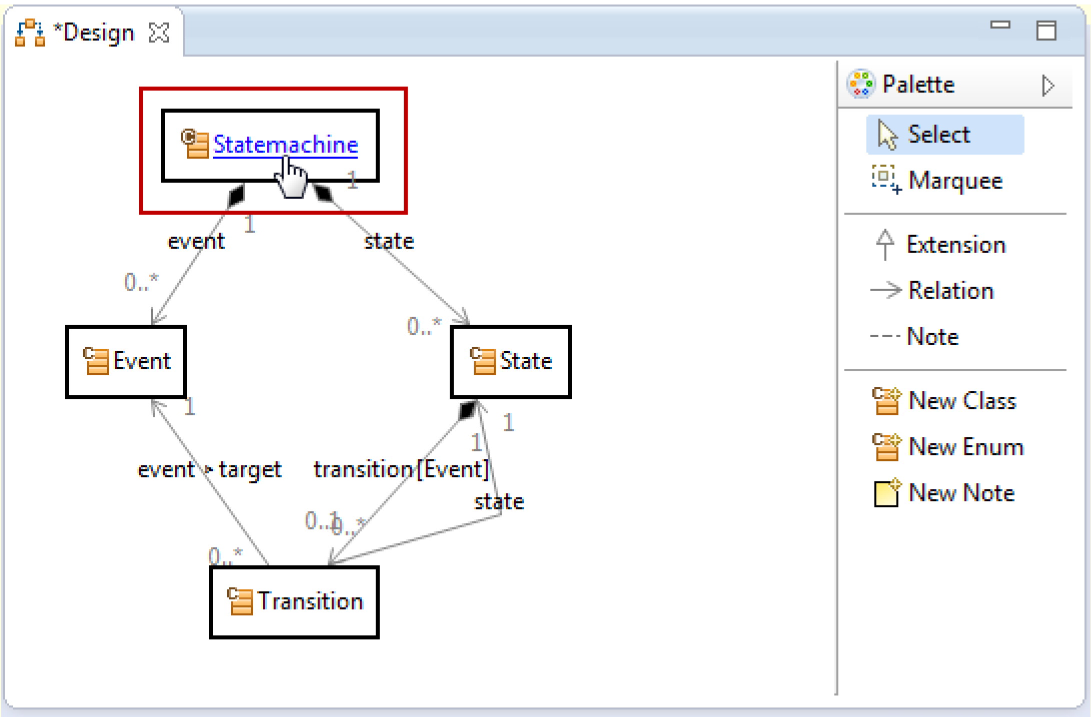
Shows the types (**Class.typeOf** statement) of a specific class.

Super Class

Shows the superclass (**Class.extend** statement) of a specific class. You this feature repeatedly to show the inheritance hierarchy.

5.2.6 Browse Resource

Open any class in the resource editor by Ctrl+Click on the class name.

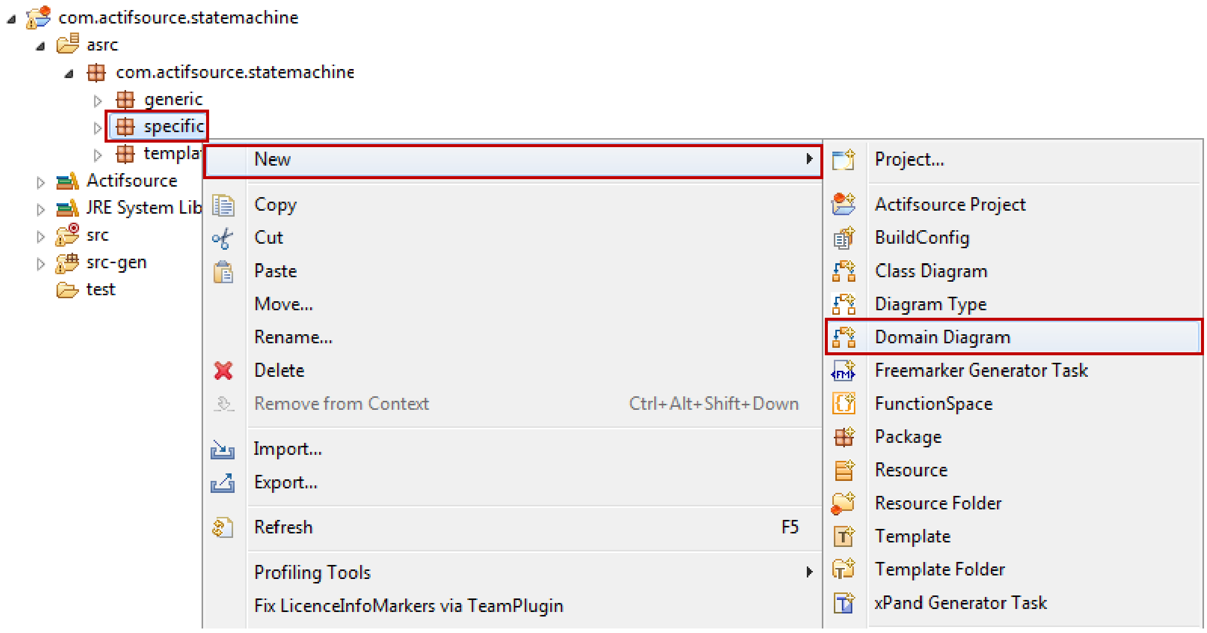


5.3 Domain Diagram Editor

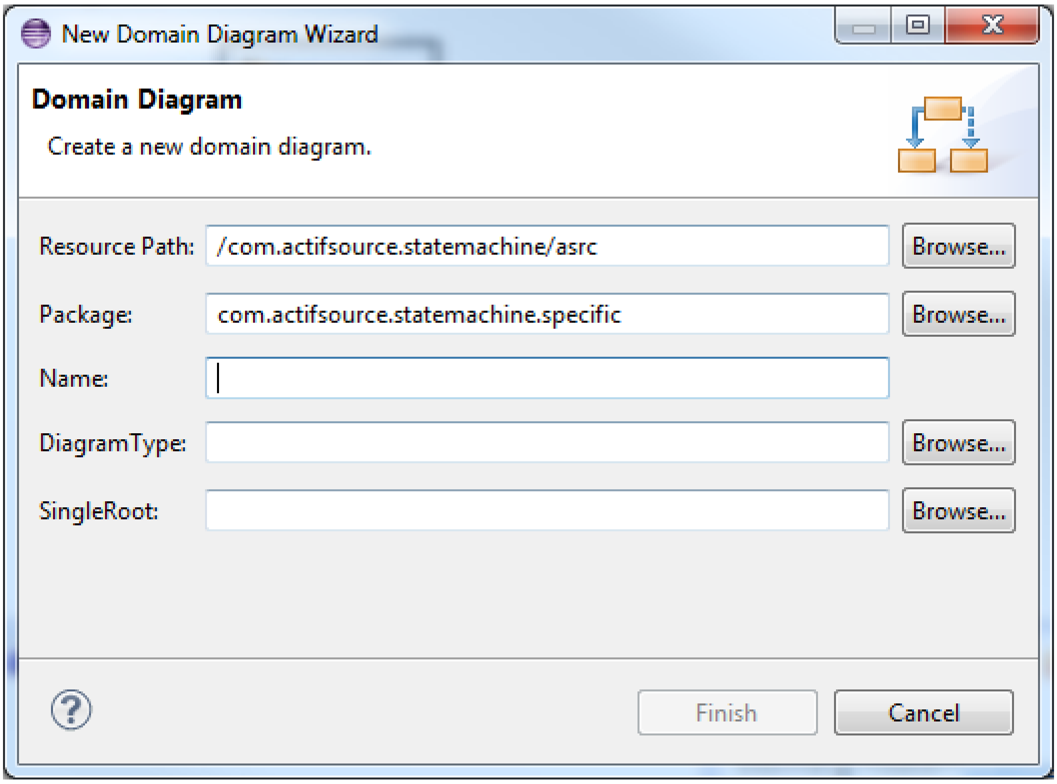
The domain diagram editor allows you to create new models based on your meta-model.

5.3.1 New Domain Diagram

You can create a new class diagram via context menu in the project explorer or *File/New*.



Preselecting a package will directly fill in the resource path and package in the wizard.


New Domain Diagram Wizard

Domain Diagram

Create a new domain diagram.

Resource Path: /com.actifsource.statemachine/asrc

Package: com.actifsource.statemachine.specific

Name:

DiagramType:

SingleRoot:

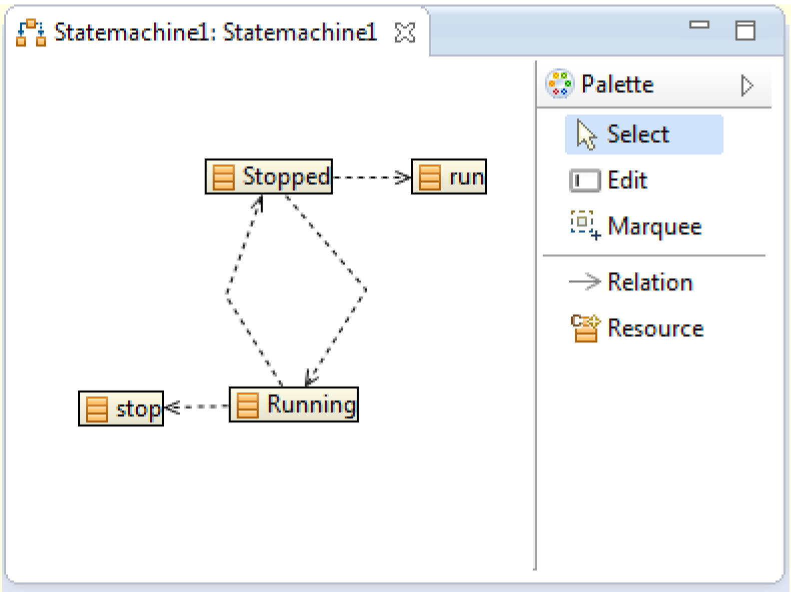
Name

The name of the domain diagram.

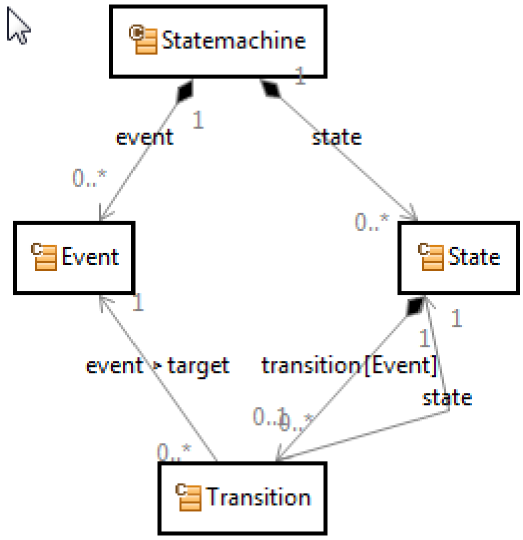
Diagram Type

Every domain uses its own diagram styles. The *DiagramType* lets you define your domain-specific domain diagrams (see Chapter 6 Domain Diagram Type).

If there is no diagram type defined, the diagram editor has a default behavior and shows resources and their relationships (use and own relation, dependencies).

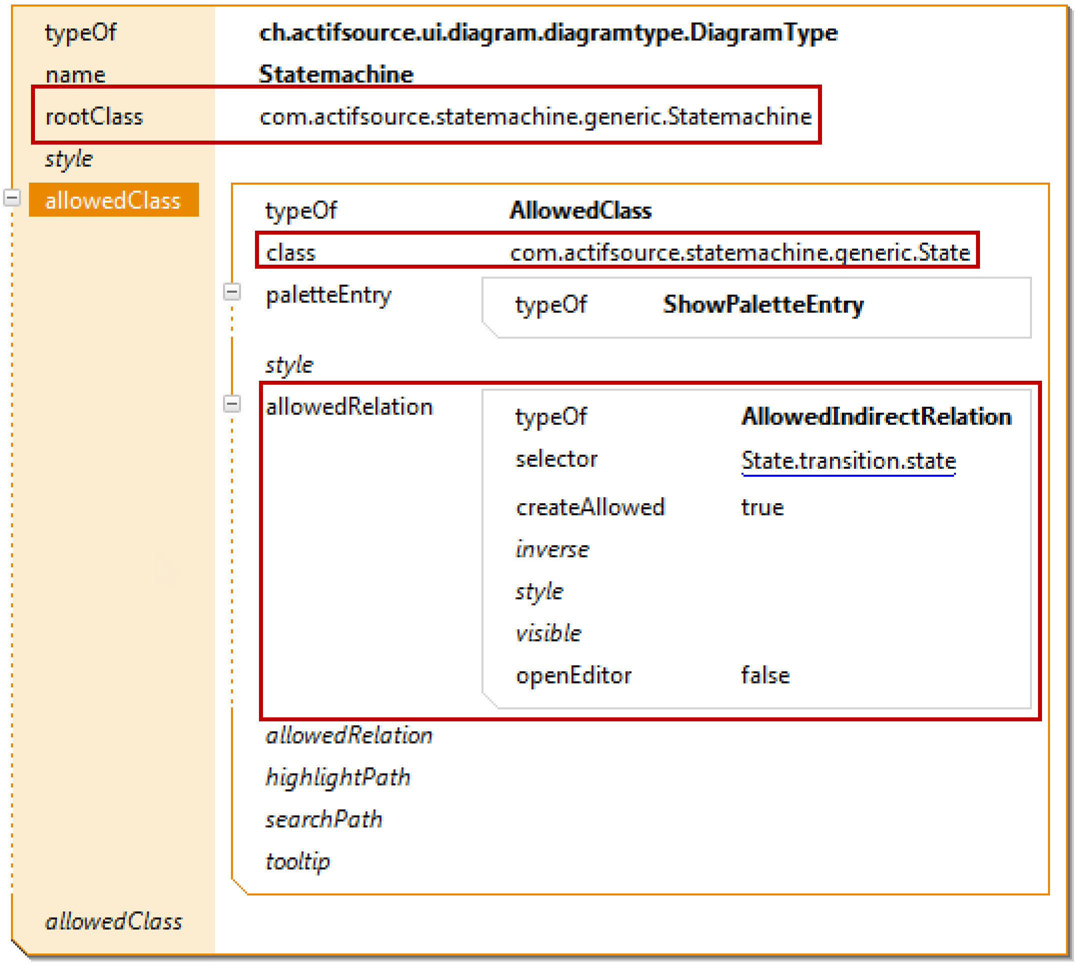


As a very simple example, let us define a diagram type for a state machine meta-model.



As the root class (see also Single Root) we choose Statemachine. This means that we can only edit elements that are part of the state machine.

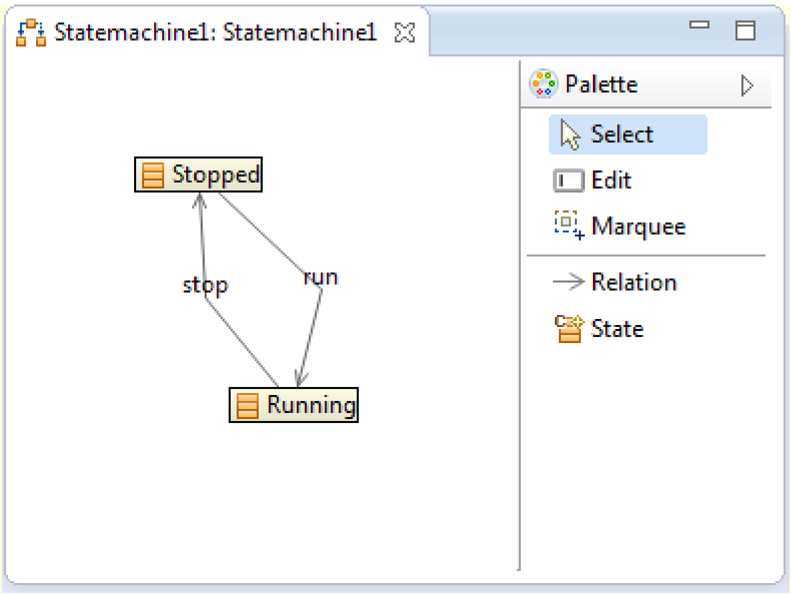
The allowed class is State since we like to edit states. Between states there are transitions. A Transition is an indirect relation from State via Transition to State.



typeOf **ch.actifsource.ui.diagram.diagramtype.DiagramType**
name **Statemachine**
rootClass com.actifsource.statemachine.generic.Statemachine
style
allowedClass
- typeOf **AllowedClass**
- class com.actifsource.statemachine.generic.State
- paletteEntry
  - typeOf **ShowPaletteEntry**
- *style*
- allowedRelation
  - typeOf **AllowedIndirectRelation**
  - selector State.transition.state
  - createAllowed true
  - *inverse*
  - *style*
  - *visible*
  - openEditor false
- *allowedRelation*
- *highlightPath*
- *searchPath*
- *tooltip*

allowedClass

The minimal diagram type shown above leads to a domain specific state event diagram.

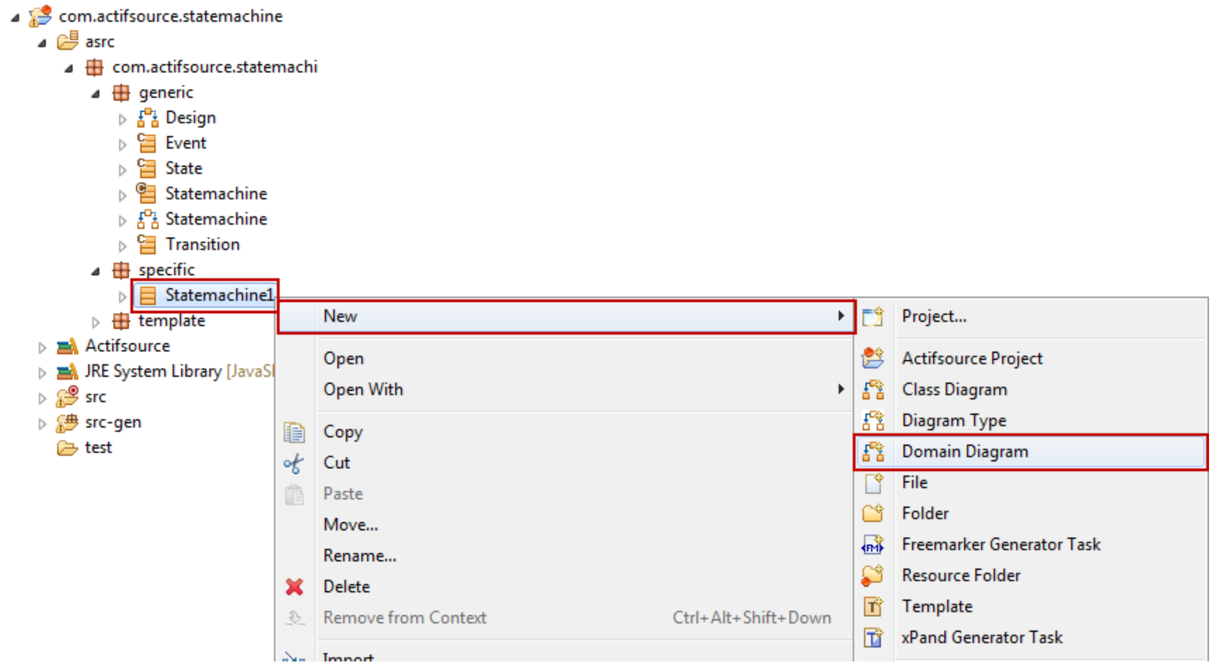


SingleRoot

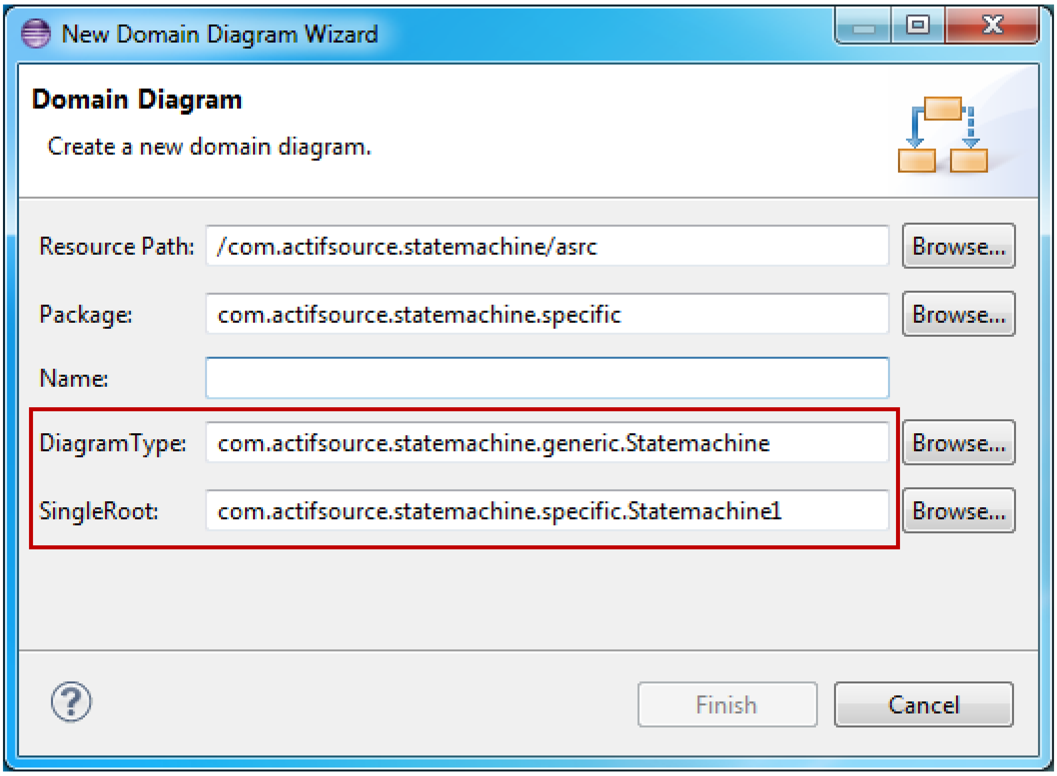
If there is a single root defined, every resource is created in the context of the single root. If no single root is defined, resources are created in the same package as the domain diagram.

Note that defining a **rootClass** in the **DiagramType** demands for a **singleRoot** in the domain diagram.

5.3.2 New Domain Diagram for Resource



You can create a new domain diagram for a single root directly by calling *New/Domain Diagram* on a resource.



Domain Diagram

Create a new domain diagram.

Resource Path: /com.actifsource.statemachine/asrc

Package: com.actifsource.statemachine.specific

Name:

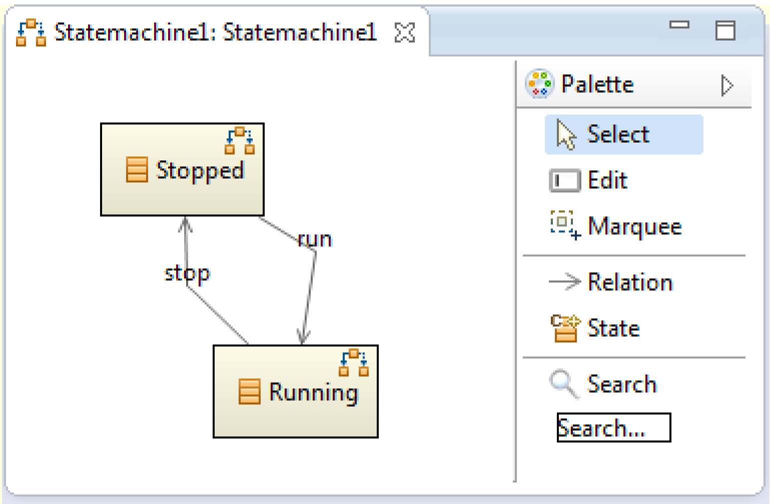
DiagramType: com.actifsource.statemachine.generic.Statemachine

SingleRoot: com.actifsource.statemachine.specific.Statemachine1

Note that the single root is preselected in the wizard and the diagram type is automatically detected if there is a diagram type which has a root class of the same type as the chosen single root.

5.3.3 Palette

Use the palette to edit the domain diagram.



Select

Selects one or many classes from the diagram. Use Ctrl+Click for multi select.

Edit

Edits figures with a **FigureEditableLabelSelector**.

[REF]

Note that you can also enter the edit mode with the Select tool. Click on the text to alter – wait for one second until the cursor changes to text mode – and click again.

Marquee

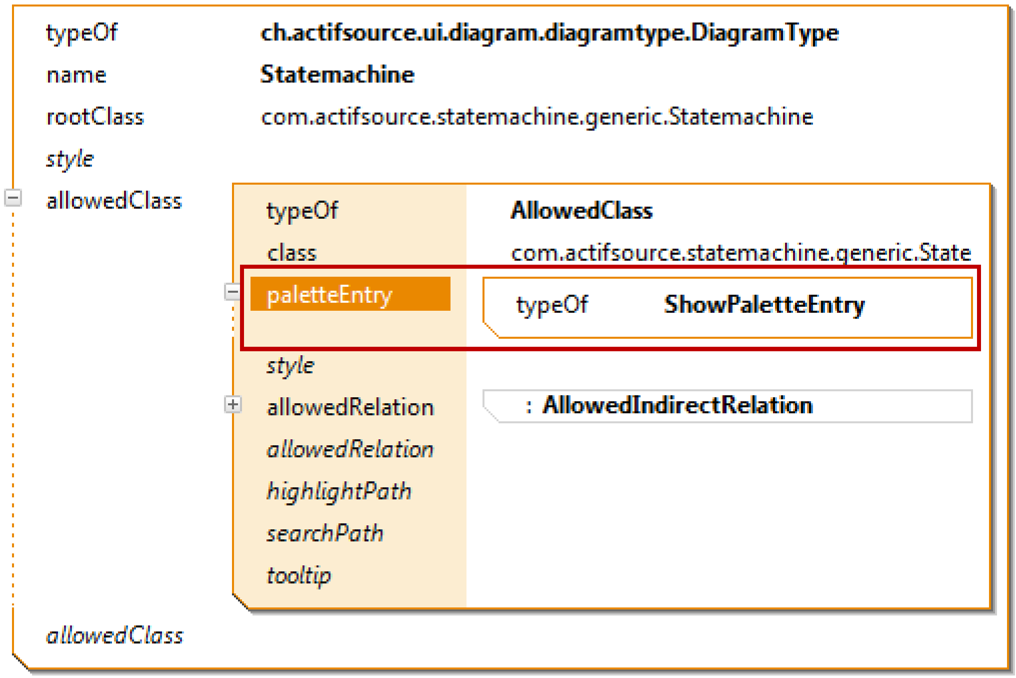
Selects classes from the diagram within a rectangle.

Relation

Inserts a relation from a resource to another.

Resources

Inserts a new resource on the diagram and in the model. Note that you control the palette by **AllowedClass.paletteEntry** in your diagram type.



typeOf	**ch.actifsource.ui.diagram.diagramtype.DiagramType**
name	**Statemachine**
rootClass	com.actifsource.statemachine.generic.Statemachine
style	
allowedClass	typeOf **AllowedClass** class com.actifsource.statemachine.generic.State paletteEntry typeOf **ShowPaletteEntry** *style* allowedRelation : **AllowedIndirectRelation** *allowedRelation* *highlightPath* *searchPath* *tooltip*
allowedClass	

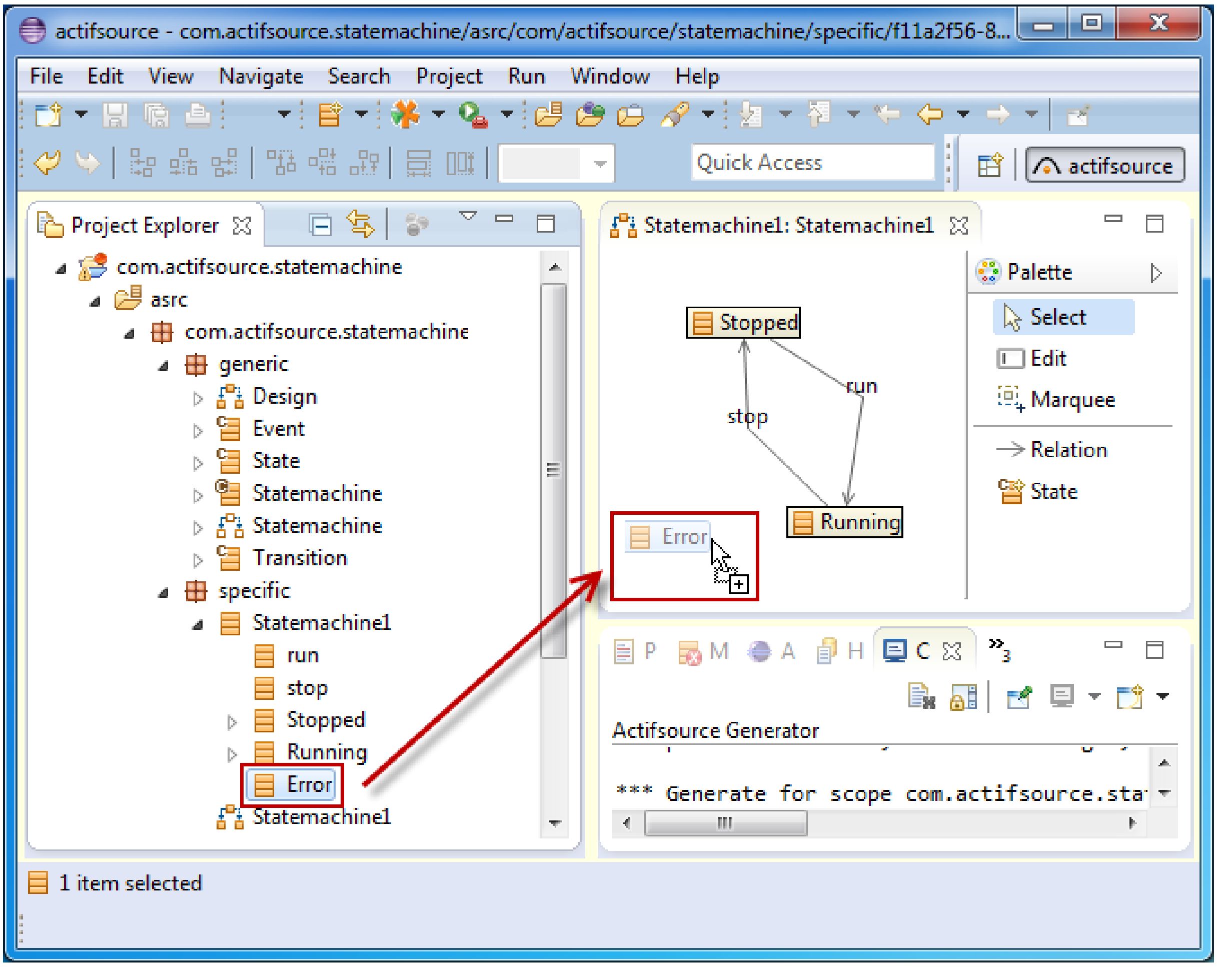
Search

Searches allowed classes with a defined search path.

[REF]

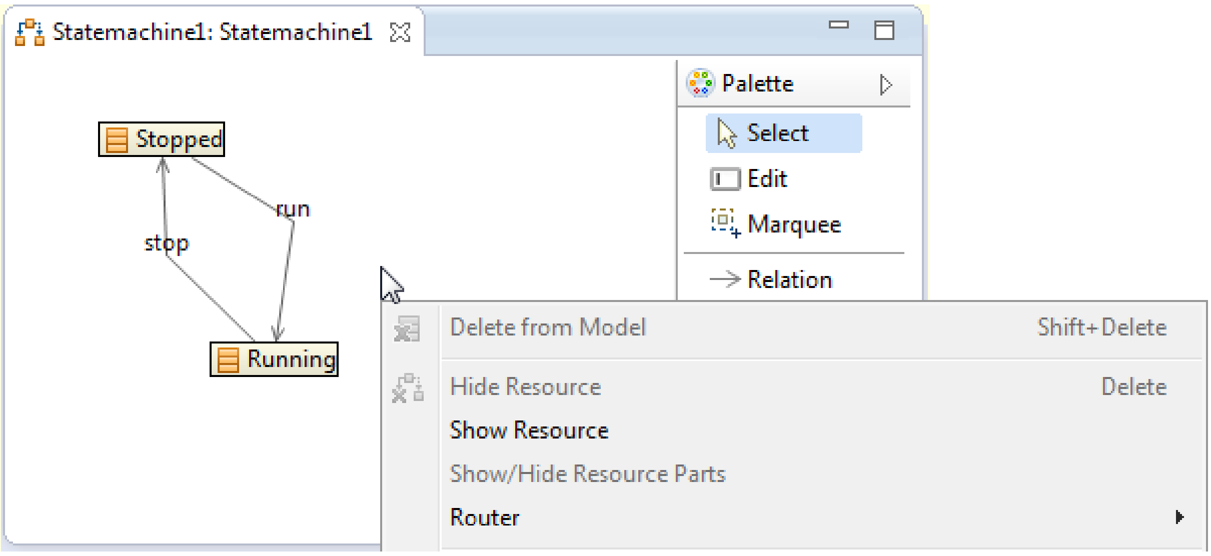
5.3.4 Drag and Drop

Use the *drag and drop* feature from the project explorer to add an existing resource to your diagram.



```
*** Generate for scope com.actifsource.sta
```

5.3.5 Context Menu



Delete from Model

Deletes this resource from the diagram and from the model. Use Shift+Delete on the keyboard to delete selected resources from the diagram and from the model.

Hide Resource

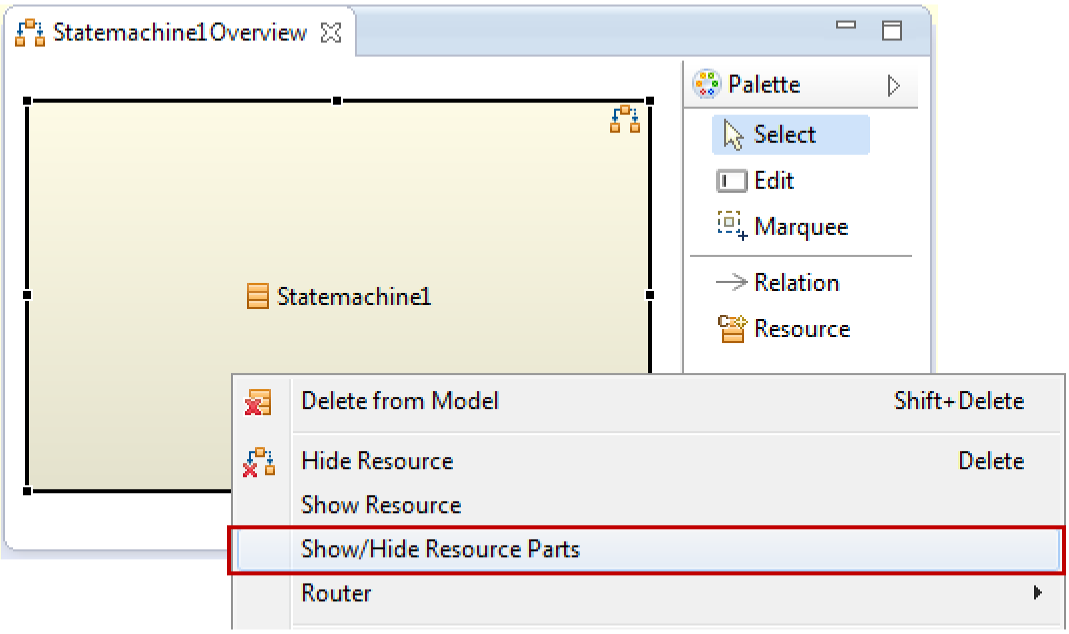
Deletes this resource from the diagram but not from the model. Use Delete on the keyboard to hide selected resources from the diagram.

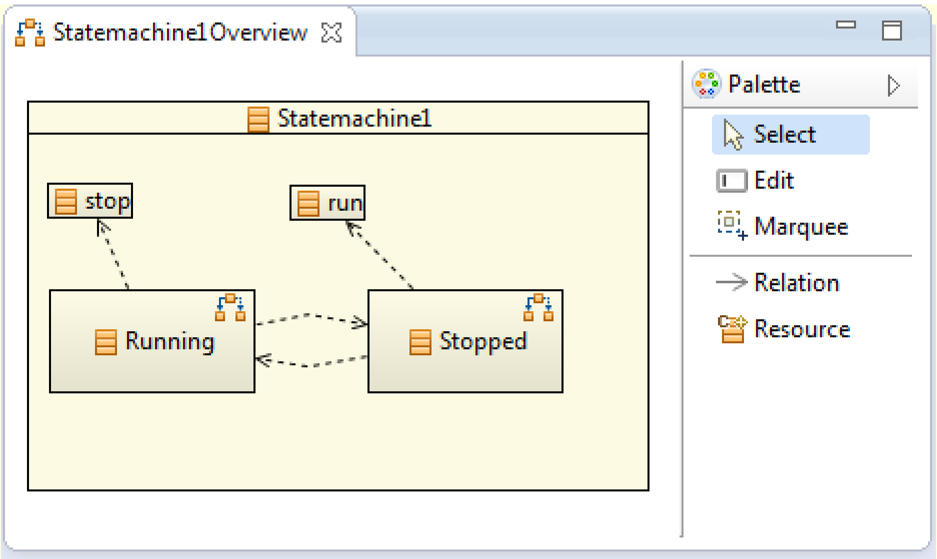
Show Resource

Shows any hidden resource in the scope of the selected resource. If this action is called on the background of the diagram, the scope is your single root.

Show/Hide Resource Parts

Shows or hides aggregated parts (see Chapter 4.4.5 OwnRelation).

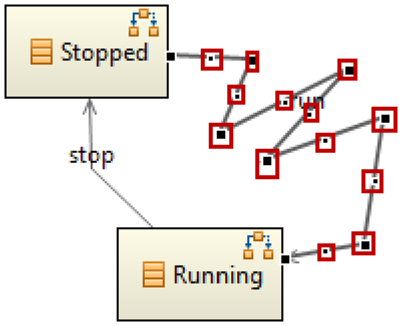
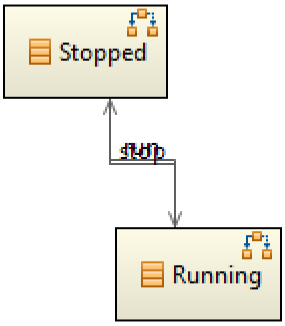
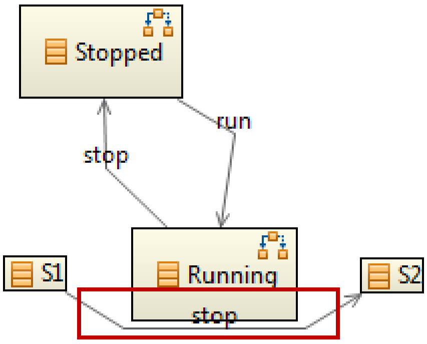




Router

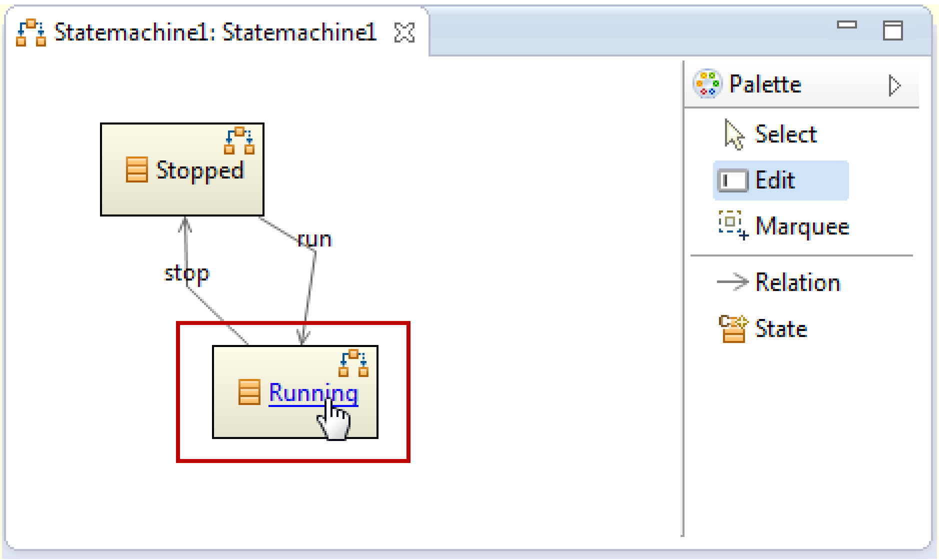
Selects between different routing algorithms.

Router	Description	Image

Default	Manual routing by dragging the line on the drag points.	
Manhatten	Lines are routed with 90° angles.	
NoIntersection	As few intersections as possible.	

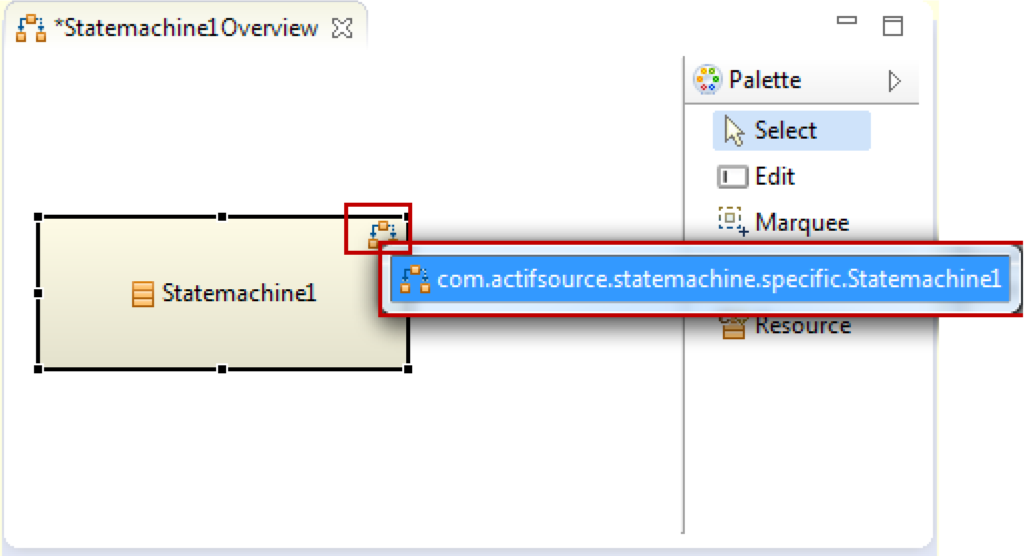
5.3.6 Browse Resource

Open any class in the resource editor by Ctrl+Click on the class name.



5.3.7 Browse Diagram

Actifsource automatically detects resources that are also shown on other diagrams. Simply click on the diagram symbol to list and browse the other diagrams.



6 Domain Diagram Type

6.1 Overview

Domain diagrams are domain-specific by definition. This means that you can define your own domain-specific diagrams.

6.2 Shape

6.3 Figure

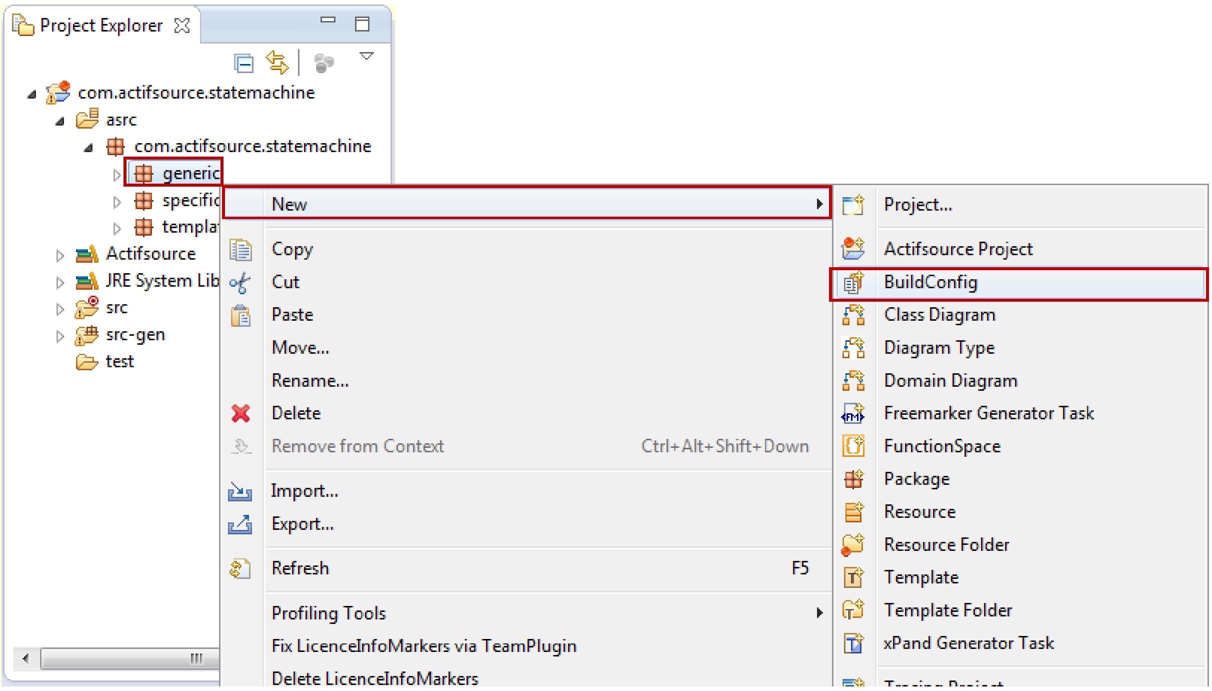
7 Build Config

7.1 Overview

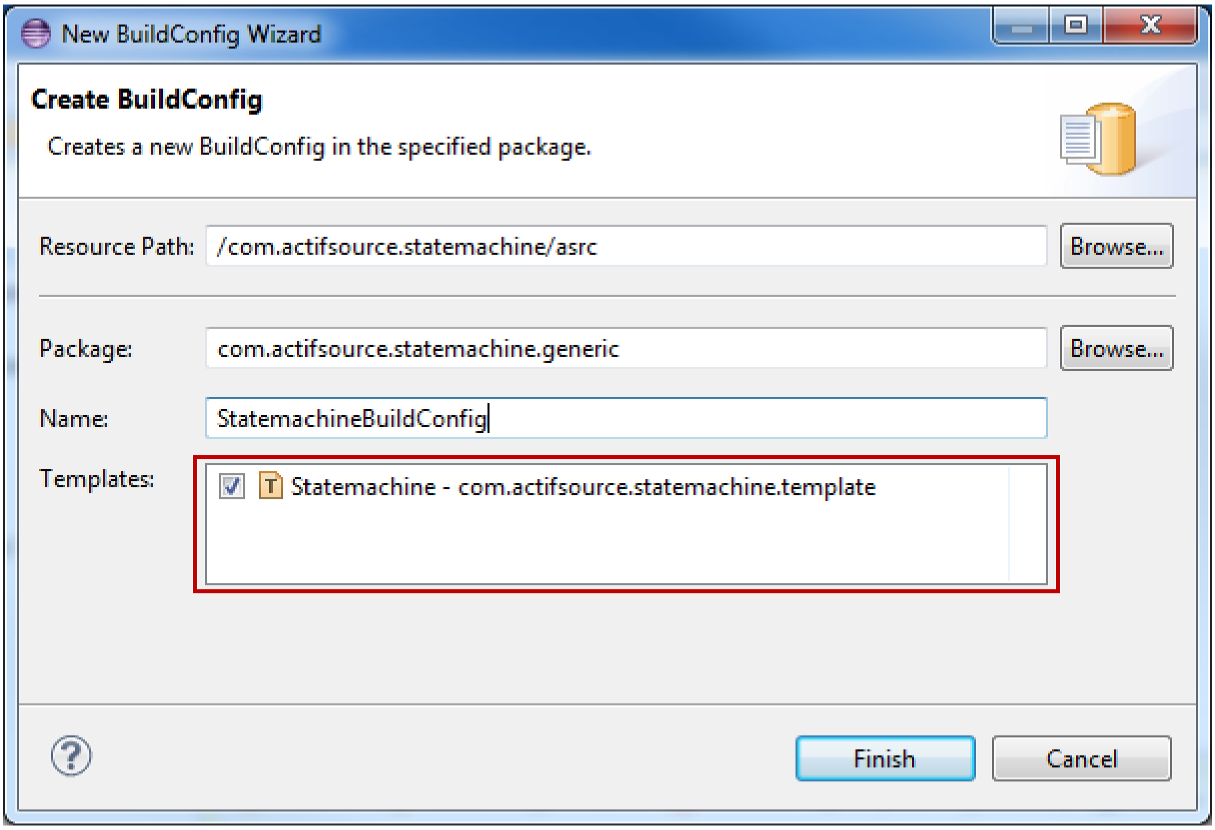
The Actifsource **BuildConfig** acts like a make file. It tells Actifsource which *build tasks* shall be executed. The most important build task for code generation is the **TemplateGeneratorTask**.

7.2 New BuildConfig

You can create a new build configuration via context menu in the project explorer or File/New.



Actifsource suggests adding templates to the build configuration which are not assigned to any other build configuration yet.



Create BuildConfig

Creates a new BuildConfig in the specified package.

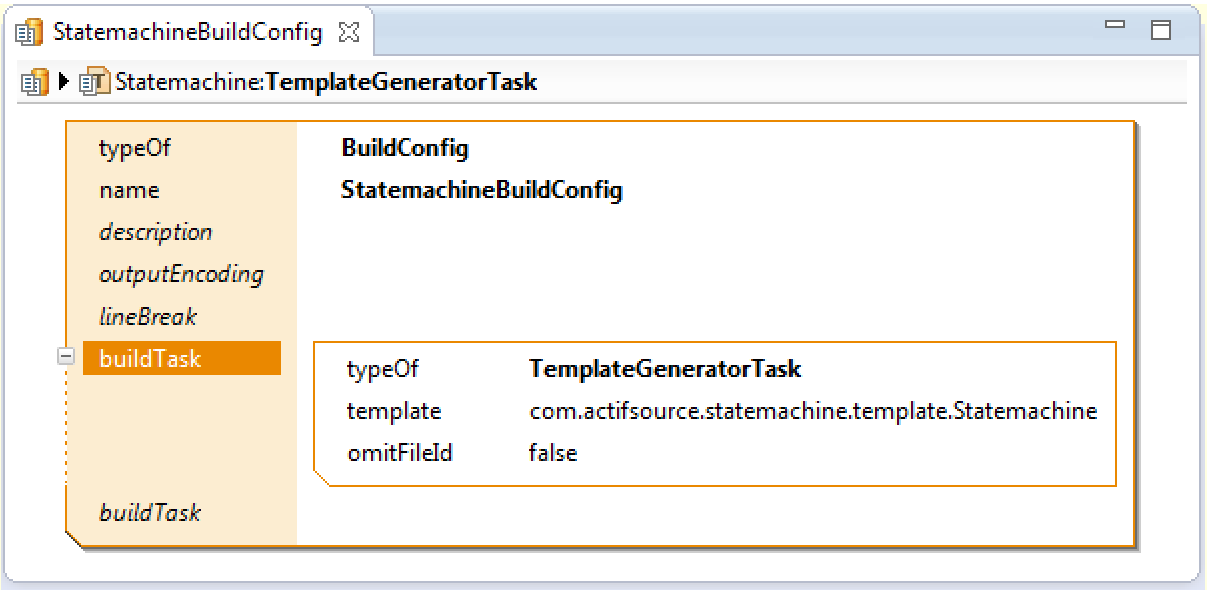
Resource Path: /com.actifsource.statemachine/asrc

Package: com.actifsource.statemachine.generic

Name: StatemachineBuildConfig

Templates: ☑ Statemachine - com.actifsource.statemachine.template

For every selected template, Actifsource creates a so called a **TemplateGeneratorTask** as shown below.

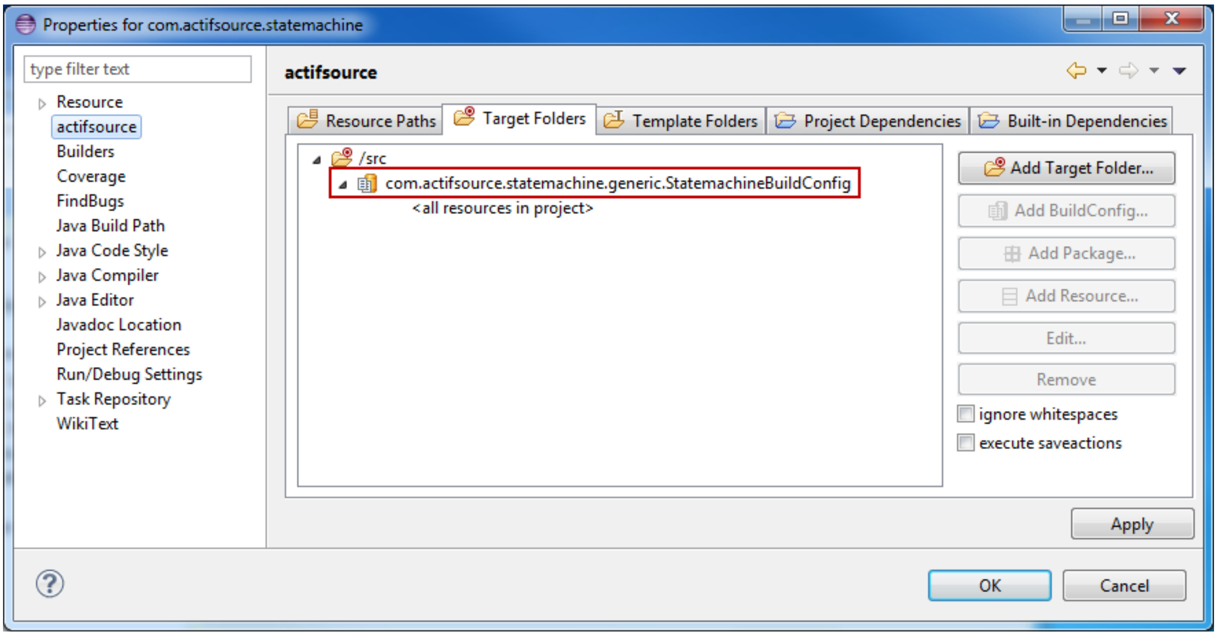


Statemachine:**TemplateGeneratorTask**

typeOf	**BuildConfig**
name	**StatemachineBuildConfig**
description	
outputEncoding	
lineBreak	
buildTask	typeOf **TemplateGeneratorTask** template com.actifsource.statemachine.template.Statemachine omitFileId false
buildTask	

7.3 BuildConfig and TargetFolder

Build configurations have to be registered with target folders to take any effect (see also Chapter 2.5.3 Target Folder and Chapter 2.6 Project Properties).

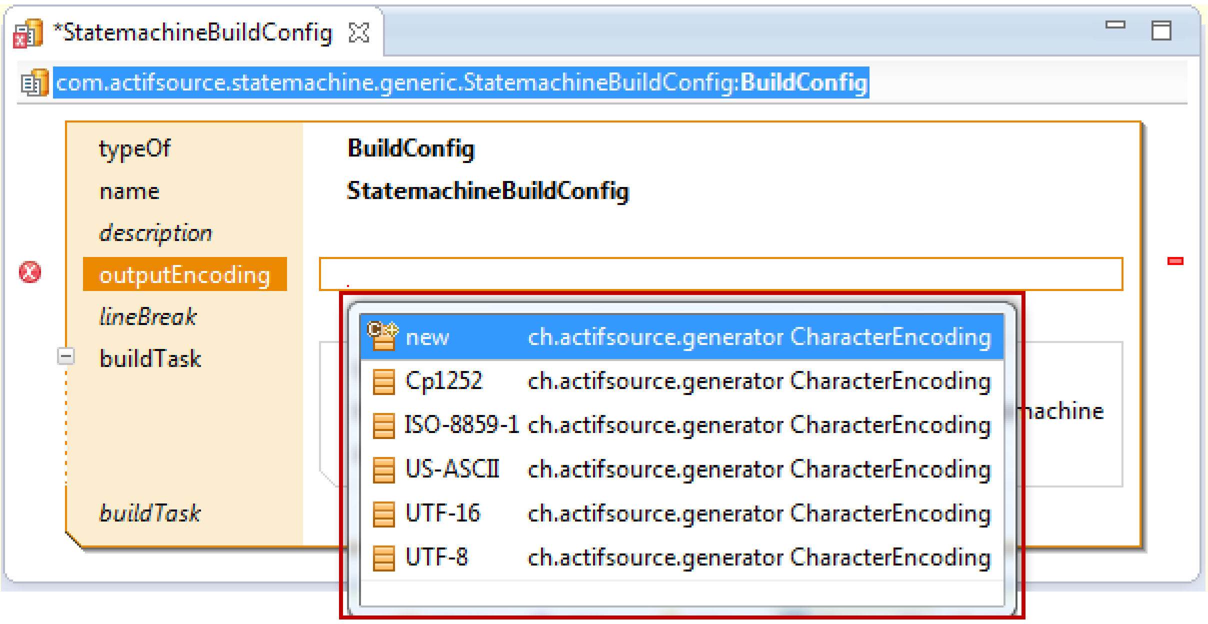


/src
com.actifsource.statemachine.generic.StatemachineBuildConfig
<all resources in project>

□ ignore whitespaces
□ execute saveactions

7.4 Output Encoding

For every build configuration you may select the output encoding.

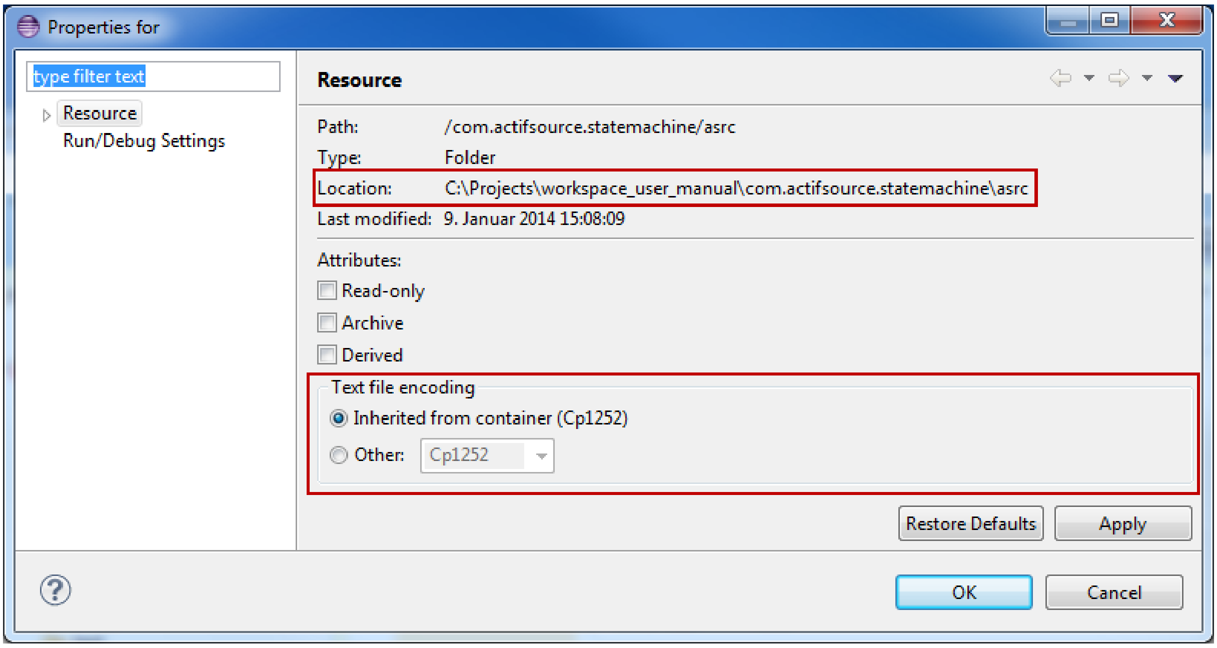


typeOf	BuildConfig
name	StatemachineBuildConfig

new	ch.actifsource.generator CharacterEncoding
Cp1252	ch.actifsource.generator CharacterEncoding
ISO-8859-1	ch.actifsource.generator CharacterEncoding
US-ASCII	ch.actifsource.generator CharacterEncoding
UTF-16	ch.actifsource.generator CharacterEncoding
UTF-8	ch.actifsource.generator CharacterEncoding

If no output encoding has been set, the one from the parent build configuration (see also Chapter 7.6.2 NestedBuildConfigGeneratorTask), folder, parent folder, project, or workspace is taken (in this order).

Check *Properties/Resource/Text file encoding* on folder or project.



Properties for

type filter text

- Resource
- Run/Debug Settings

Resource

Path: /com.actifsource.statemachine/asrc

Type: Folder

Location: C:\Projects\workspace_user_manual\com.actifsource.statemachine\asrc

Last modified: 9. Januar 2014 15:08:09

Attributes:

□ Read-only

□ Archive

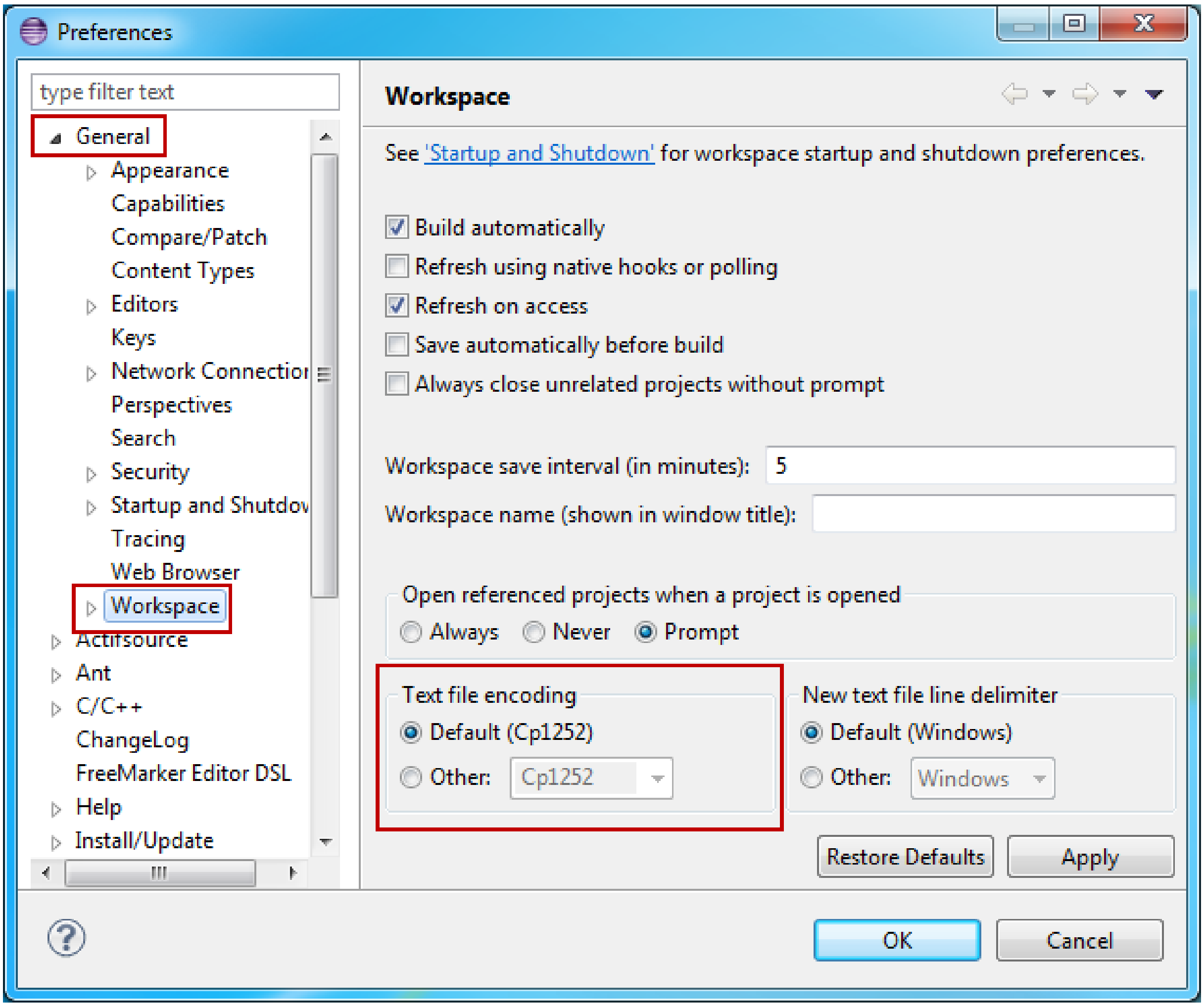
□ Derived

Text file encoding

● Inherited from container (Cp1252)

○ Other: Cp1252

Check *Window/Preferences/General/Workspace/Text file encoding* on workspace.



Preferences

type filter text

- General
  - Appearance
  - Capabilities
  - Compare/Patch
  - Content Types
  - Editors
  - Keys
  - Network Connectior
  - Perspectives
  - Search
  - Security
  - Startup and Shutdov
  - Tracing
  - Web Browser
  - Workspace
- Actifsource
- Ant
- C/C++
- ChangeLog
- FreeMarker Editor DSL
- Help
- Install/Update

Workspace

See 'Startup and Shutdown' for workspace startup and shutdown preferences.

☑ Build automatically

□ Refresh using native hooks or polling

☑ Refresh on access

□ Save automatically before build

□ Always close unrelated projects without prompt

Workspace save interval (in minutes): 5

Workspace name (shown in window title):

Open referenced projects when a project is opened

○ Always ○ Never ● Prompt

Text file encoding

● Default (Cp1252)

○ Other: Cp1252

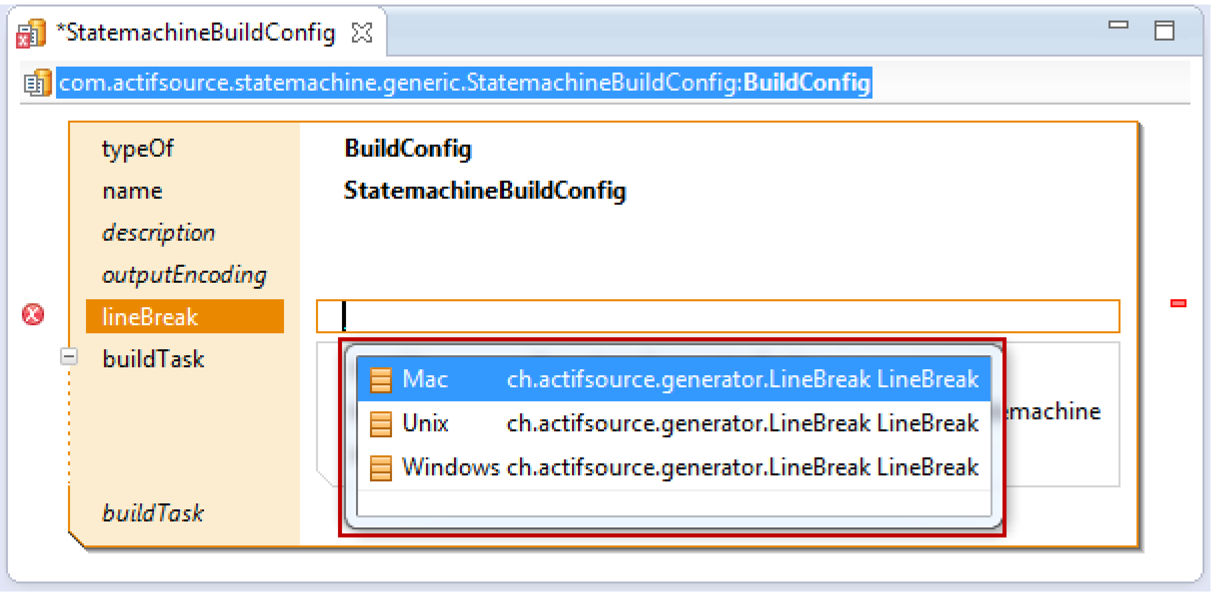
New text file line delimiter

● Default (Windows)

○ Other: Windows

7.5 Line Break

For every build configuration you may select the line break style.



*StatemachineBuildConfig

com.actifsource.statemachine.generic.StatemachineBuildConfig:**BuildConfig**

typeOf **BuildConfig**

name **StatemachineBuildConfig**

description

outputEncoding

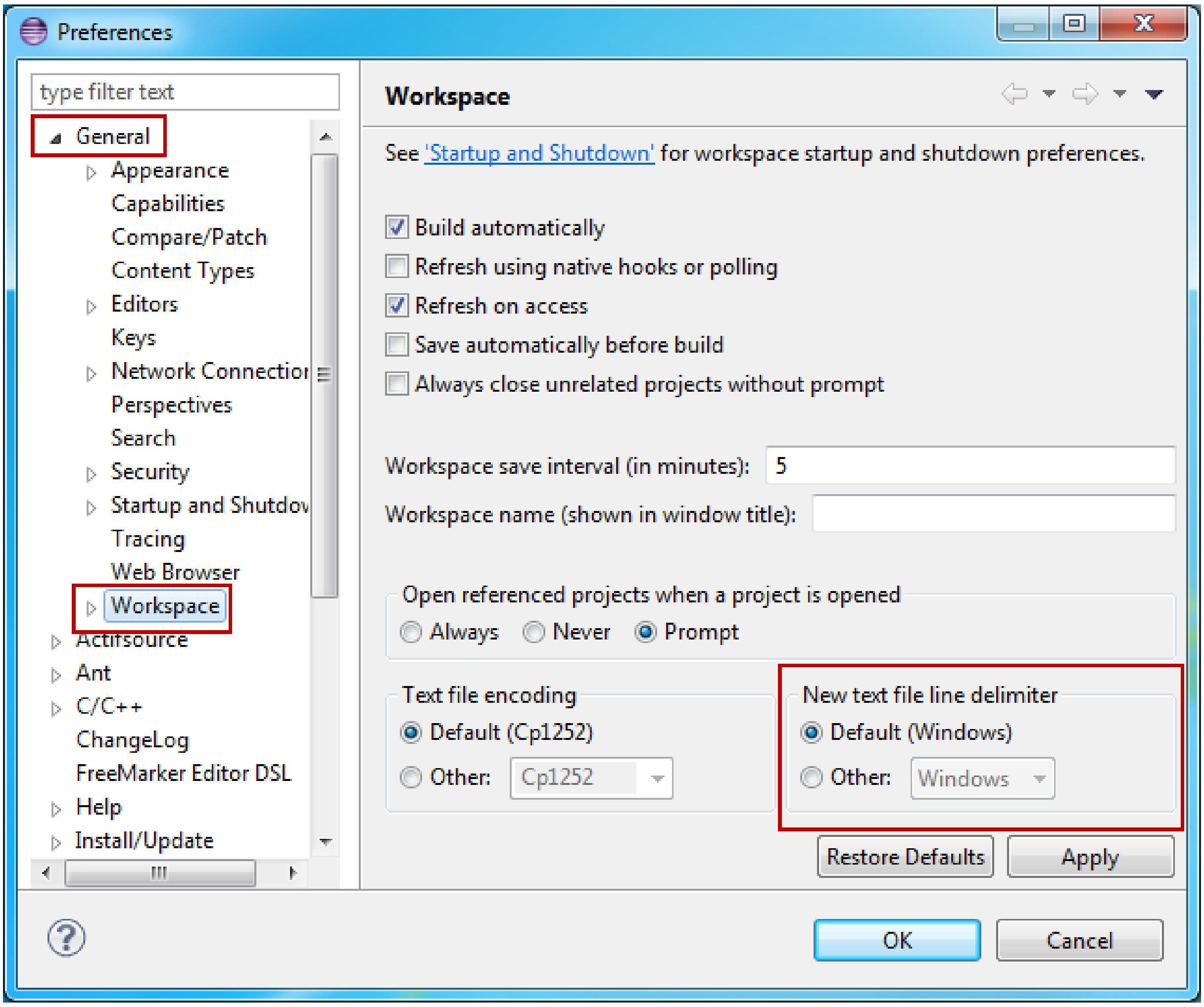
lineBreak

buildTask

buildTask

Mac	ch.actifsource.generator.LineBreak LineBreak
Unix	ch.actifsource.generator.LineBreak LineBreak
Windows	ch.actifsource.generator.LineBreak LineBreak

If no line break has been set, the one from the parent build configuration (see also Chapter 7.6.2 NestedBuildConfigGeneratorTask), project, or workspace is taken.



Workspace

See 'Startup and Shutdown' for workspace startup and shutdown preferences.

☑ Build automatically
□ Refresh using native hooks or polling
☑ Refresh on access
□ Save automatically before build
□ Always close unrelated projects without prompt

Workspace save interval (in minutes): 5

Workspace name (shown in window title):

Open referenced projects when a project is opened
○ Always ○ Never ● Prompt

Text file encoding
● Default (Cp1252)
○ Other: Cp1252

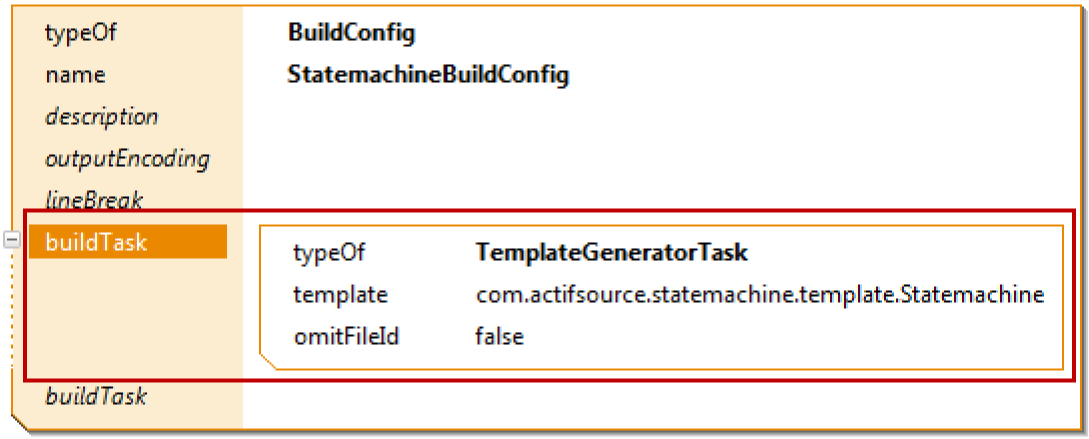
New text file line delimiter
● Default (Windows)
○ Other: Windows

7.6 BuildTask

The build configuration lists all build tasks. Build tasks are executed in the order as listed. There are different types of build tasks. The most important one is the **TemplateGeneratorTask**.

7.6.1 Template Generator Task

The template generator task defines which templates have to be built.



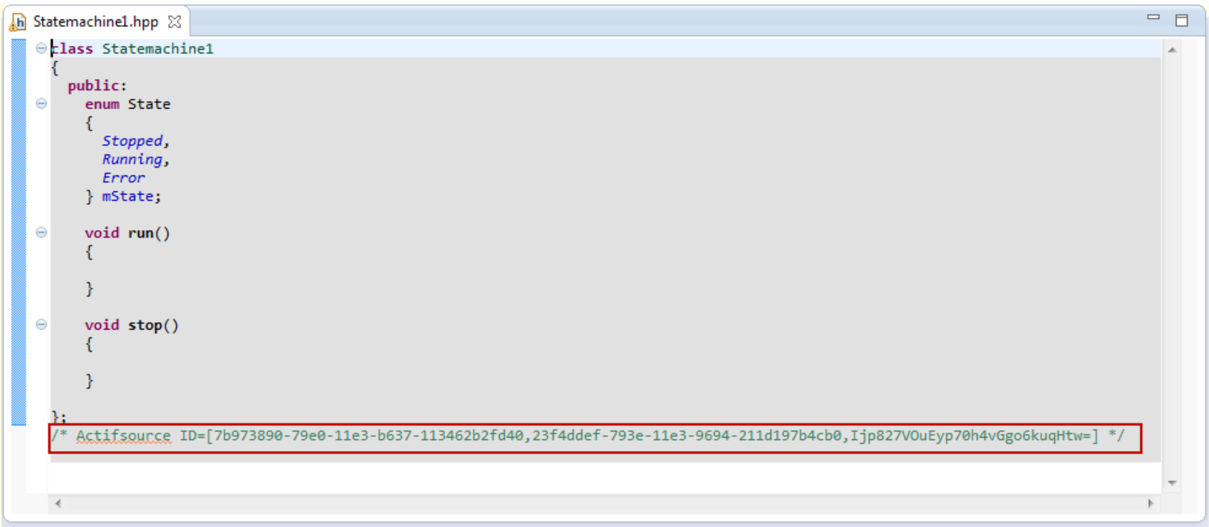
typeOf	**BuildConfig**
name	**StatemachineBuildConfig**
description	
outputEncoding	
lineBreak	
buildTask	typeOf **TemplateGeneratorTask** template com.actifsource.statemachine.template.Statemachine omitFileId false
buildTask	

Template

References the template.

Omit File Id

Actifsource normally inserts an id at the end of every generated file. This file id helps identify and track generated code.



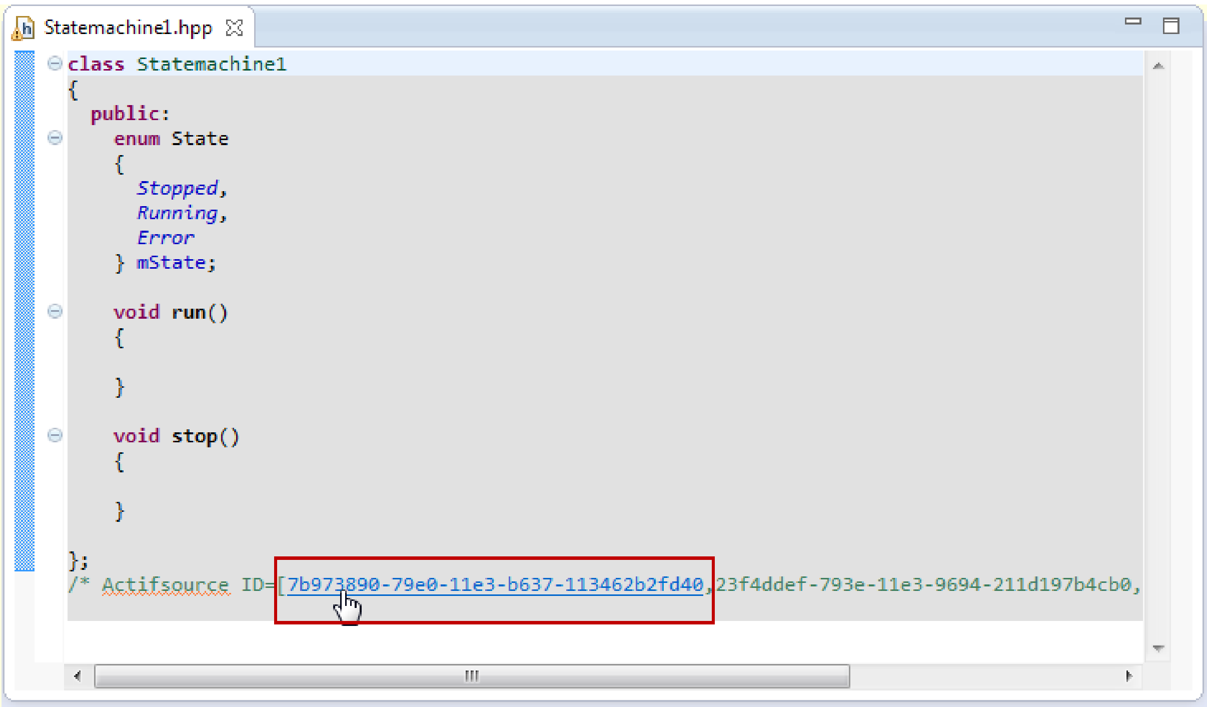
```
class Statemachine1
{
  public:
    enum State
    {
      Stopped,
      Running,
      Error
    } mState;

    void run()
    {

    }

    void stop()
    {

    }

};
/* Actifsource ID=[7b973890-79e0-11e3-b637-113462b2fd40,23f4ddef-793e-11e3-9694-211d197b4cb0,Ijp827VOuEyp70h4vGgo6kuqHtw=] */
```

The Actifsource ID is assembled as follows.

/* Actifsource ID=[TemplateGUID,SuperContextGUID*,BaseContextGUID,MD5Hash] */

Element	Descriptiom
Comment Tags	The comment tags (i.e. /* */) are given by the language
Actifsource ID	Static identifier
TemplateGUID	GUID of the template which created this file
SuperContextGUID	GUID of the resources which contains the base resource
BaseContextGUID	GUID of the base resources of this file
MD5Hash	A hash code over the generated code but not including protected regions to detect if the generated code has been changed manually. To ignore white spaces when building the MD5 hash check Chapter 2.5.3 Target Folder.

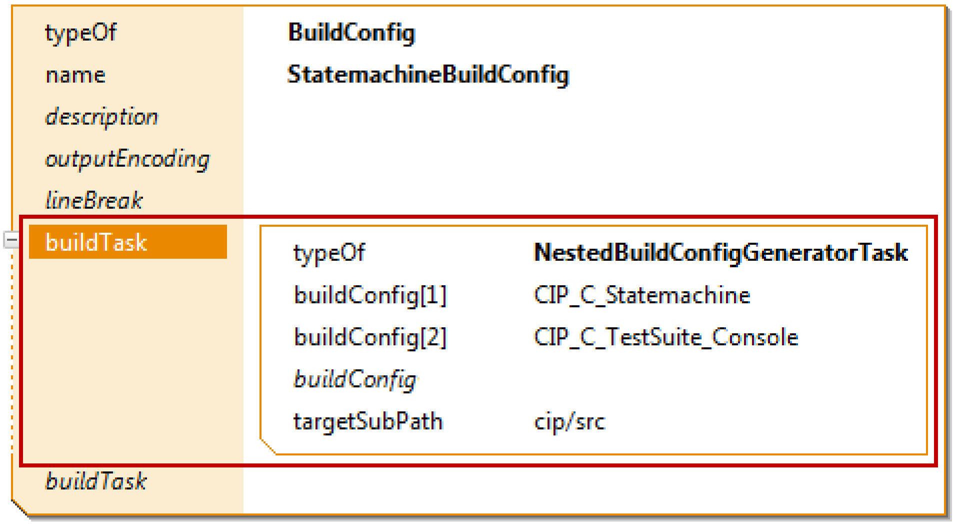
Note that you can open the resource for any GUID in an Eclipse text editor or in the Eclipse console by Ctrl+Click on the GUID.



```
class Statemachine1
{
  public:
    enum State
    {
      Stopped,
      Running,
      Error
    } mState;

    void run()
    {

    }

    void stop()
    {

    }
};
/* Actifsource ID=[7b973890-79e0-11e3-b637-113462b2fd40,23f4ddef-793e-11e3-9694-211d197b4cb0,
```

7.6.2 NestedBuildConfigGeneratorTask

The nested build configurator task let you reference and execute existing build configurations.



typeOf	BuildConfig	
name	StatemachineBuildConfig	
description		
outputEncoding		
lineBreak		
buildTask	typeOf	NestedBuildConfigGeneratorTask
	buildConfig[1]	CIP_C_Statemachine
	buildConfig[2]	CIP_C_TestSuite_Console
	buildConfig	
	targetSubPath	cip/src
buildTask		

Build Config

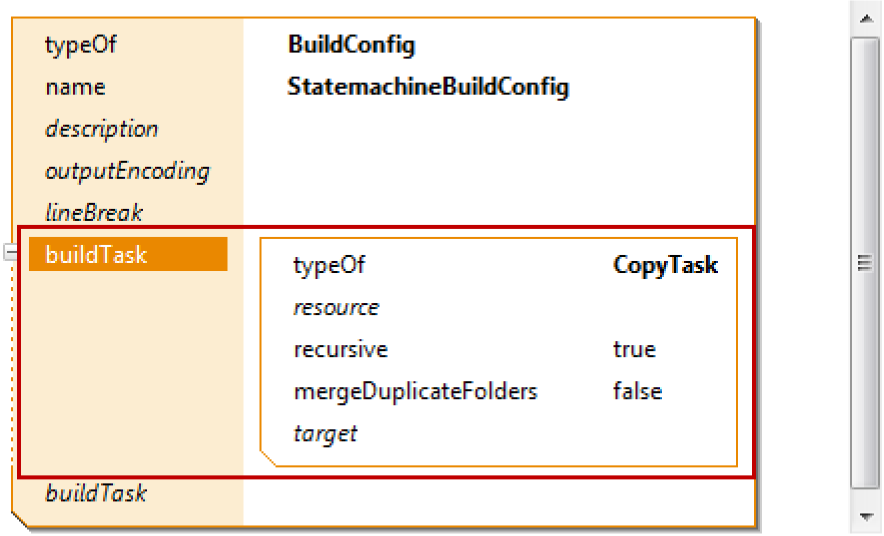
References any existing build configuration.

Target Sub Path

Defines a sub path to the target folder (see also Chapter 2.5.3 Target Folder).

7.6.3 CopyTask

Copies a file or a folder to a specified target path. Please note that this tasks needs a built-in dependency to WORKSPACE (see Chapter 2.5.6 Built-in Dependencies).



typeOf	BuildConfig
name	StatemachineBuildConfig
description	
outputEncoding	
lineBreak	
buildTask	typeOf CopyTask *resource* recursive true mergeDuplicateFolders false *target*
buildTask	

Resource

The file or folder to copy. There are different resource types.

Resource Type	Description
BundleResource	File or folders found in a bundle (plugin project).
OutputScopeResource	File or folders found in the target folder.
TemplateScopeResource	File or folders found in the template folder.
WorkspaceResource	File or folders found in the workspace.

Recursive

All subfolders are copied if set to true.

Merge Duplicate Folders

When enabled, the generator allows merging content from different folders into one folder. Otherwise an error will occur.

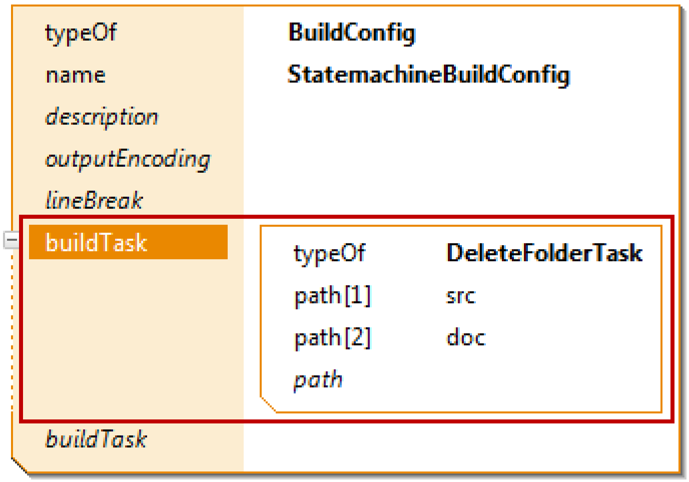
Target

The target to copy the files or folders. There are different target types.

Target Type	Description
ResourcePathTarget	Target path relative to copied resources.
ZipTarget	File and folders are copied into a zip file.

7.6.4 DeleteFolderTask

Deletes the specified folders relative to the target folder. Please note that this tasks needs a built-in dependency to WORKSPACE (see Chapter 2.5.6 Built-in Dependencies).



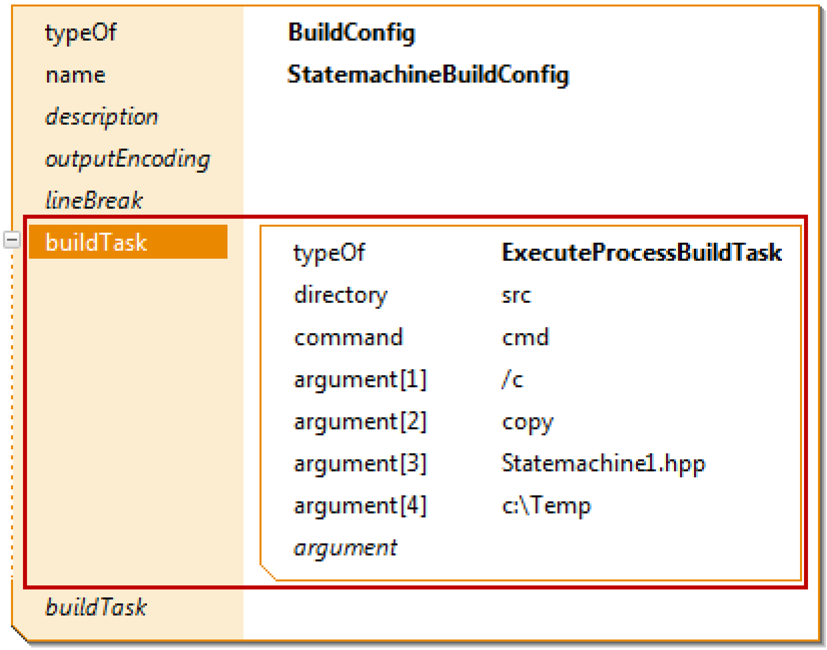
typeOf BuildConfig
name StatemachineBuildConfig
description
outputEncoding
lineBreak
buildTask
typeOf DeleteFolderTask
path[1] src
path[2] doc
path
buildTask

Path
A path relative to the target folder.

7.6.5 ExecuteProcessBuildTask

Executes any process on your operating system. Please note that this tasks needs a built-in dependency to WORKSPACE (see also Chapter 2.5.6 Built-in Dependencies).

To execute a shell command on windows, choose cmd as shell command, /c as first argument, and your shell command and parameters as subsequent arguments.



typeOf BuildConfig
name StatemachineBuildConfig
description
outputEncoding
lineBreak
buildTask
typeOf ExecuteProcessBuildTask
directory src
command cmd
argument[1] /c
argument[2] copy
argument[3] Statemachine1.hpp
argument[4] c:\Temp
argument
buildTask

Directory
Directory to execute the process relative to the target project.

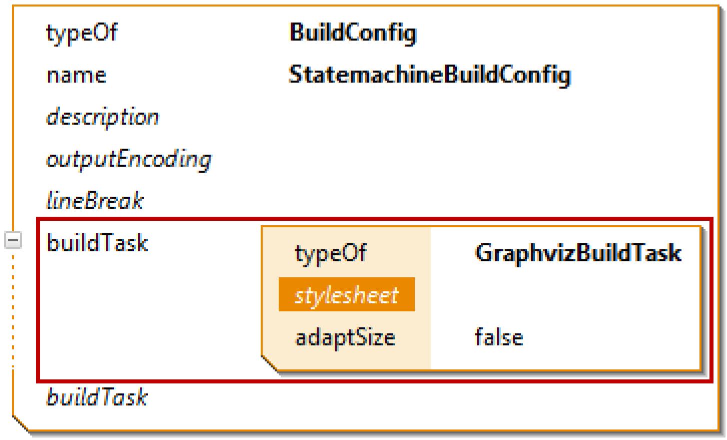
Command
The command (without arguments) to execute.

Argument
The arguments of the command.

7.6.6 GraphvizBuiltTask

Runs the graphviz dot command on all .dot files in the target folder. Please note that this tasks needs a built-in dependency to GRAPHVIZ (see also Chapter 2.5.6 Built-in Dependencies).

Make sure that you have graphviz installed (see http://www.graphviz.org/) and reachable in your path.



typeOf	**BuildConfig**
name	**StatemachineBuildConfig**
description	
outputEncoding	
lineBreak	
buildTask	typeOf **GraphvizBuildTask** *stylesheet* adaptSize false
buildTask	

Styleheet

A css stylesheet if needed.

Adapt Size

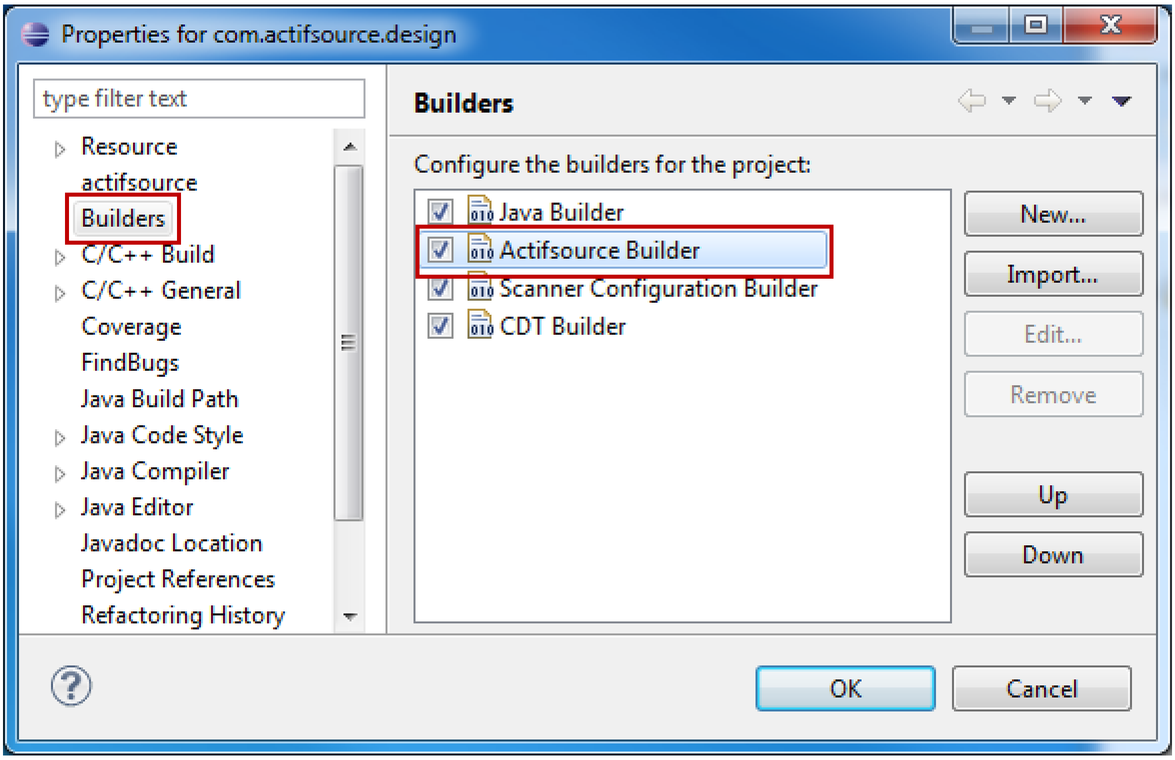
If set to true, the generated diagram's width is set to 100%.

7.7 Eclipse Builder

Eclipse supports so called *Builders* to build anything. In C/C++ there is the CDT Builder to build executables and libraries from header and source files. In Java there is the Java Builder to build .class files from .java files.

In Actifsource there is the Actifsource Builder to generate code from the model (.asr files).

Make sure that the builders are arranged in the correct order. You will find the settings in *Project/Properties/Builder*.



Properties for com.actifsource.design

type filter text

- Resource
- actifsource
- Builders
- C/C++ Build
- C/C++ General
- Coverage
- FindBugs
- Java Build Path
- Java Code Style
- Java Compiler
- Java Editor
- Javadoc Location
- Project References
- Refactoring History

Builders

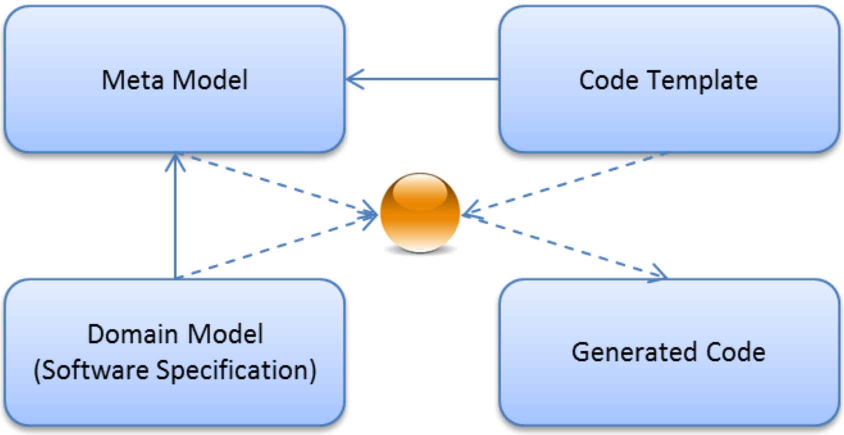
Configure the builders for the project:

☑ Java Builder
☑ Actifsource Builder
☑ Scanner Configuration Builder
☑ CDT Builder

8 Template Editor

8.1 Overview

As already seen in Chapter 1.1 Working with models, the *Actifsource Template Editor* allows you to write *meta-code* based on the *meta-model*. Writing meta-code means to write code along the structures which are given by the meta-model without knowing the specific domain model.



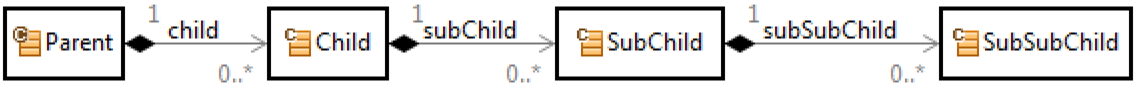
8.2 New Template

A template is either based on a type (**Class**, **Enum**) or not.

Template Type	Description
Based on types	Based on a type means that the Template is applied for every instance of that type. The result is one file per instance.
Build once	Build once means that the Template is applied exactly once. The result is one file.

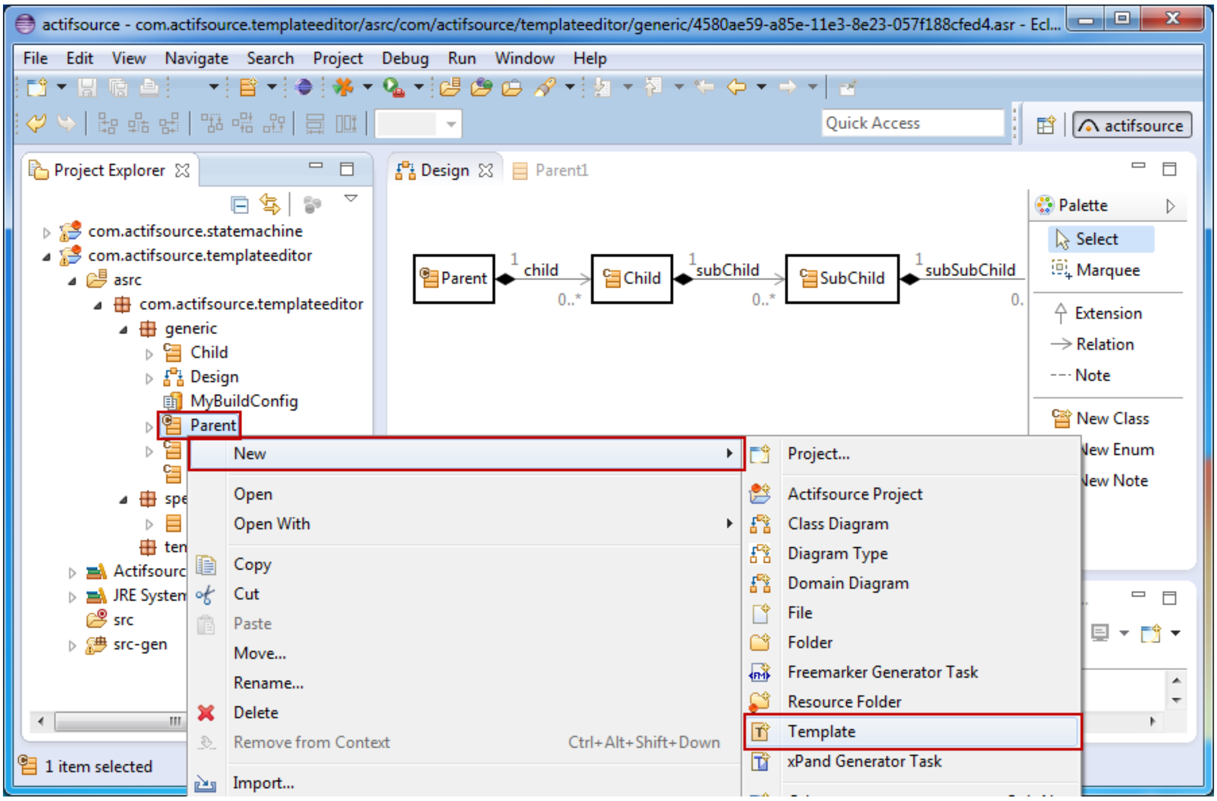
8.2.1 Create a template based on type

Creating a template based on a specific type (**Class**) is the normal case. Consider a nested Parent-Child structure with the following meta-model.

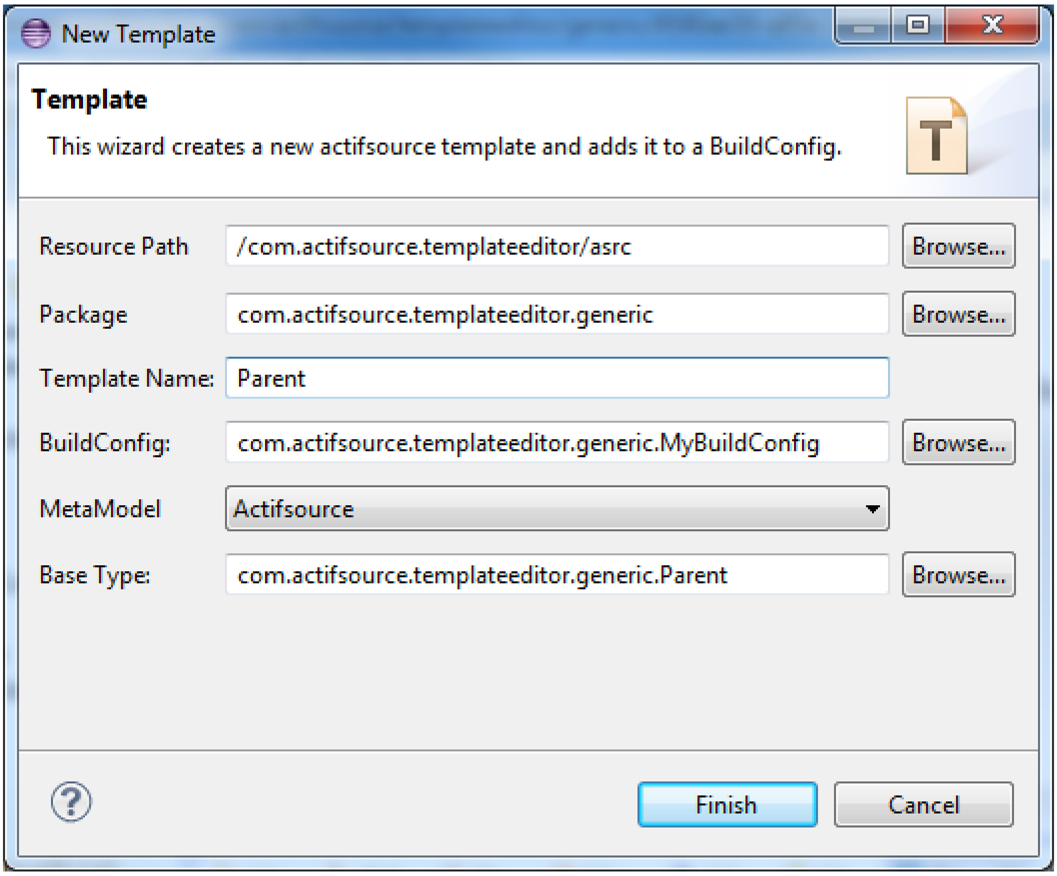


For every specific parent-child structure there is at least a resource of type *Parent* to start with. So let's start writing meta-code based on the class Parent.

To create a template based on the class Parent simply choose *New/Template* from the context menu of the class Parent.



The New Template Wizard helps to configure the template settings.


New Template

Template

This wizard creates a new actifsource template and adds it to a BuildConfig.

Resource Path: /com.actifsource.templateeditor/asrc

Package: com.actifsource.templateeditor.generic

Template Name: Parent

BuildConfig: com.actifsource.templateeditor.generic.MyBuildConfig

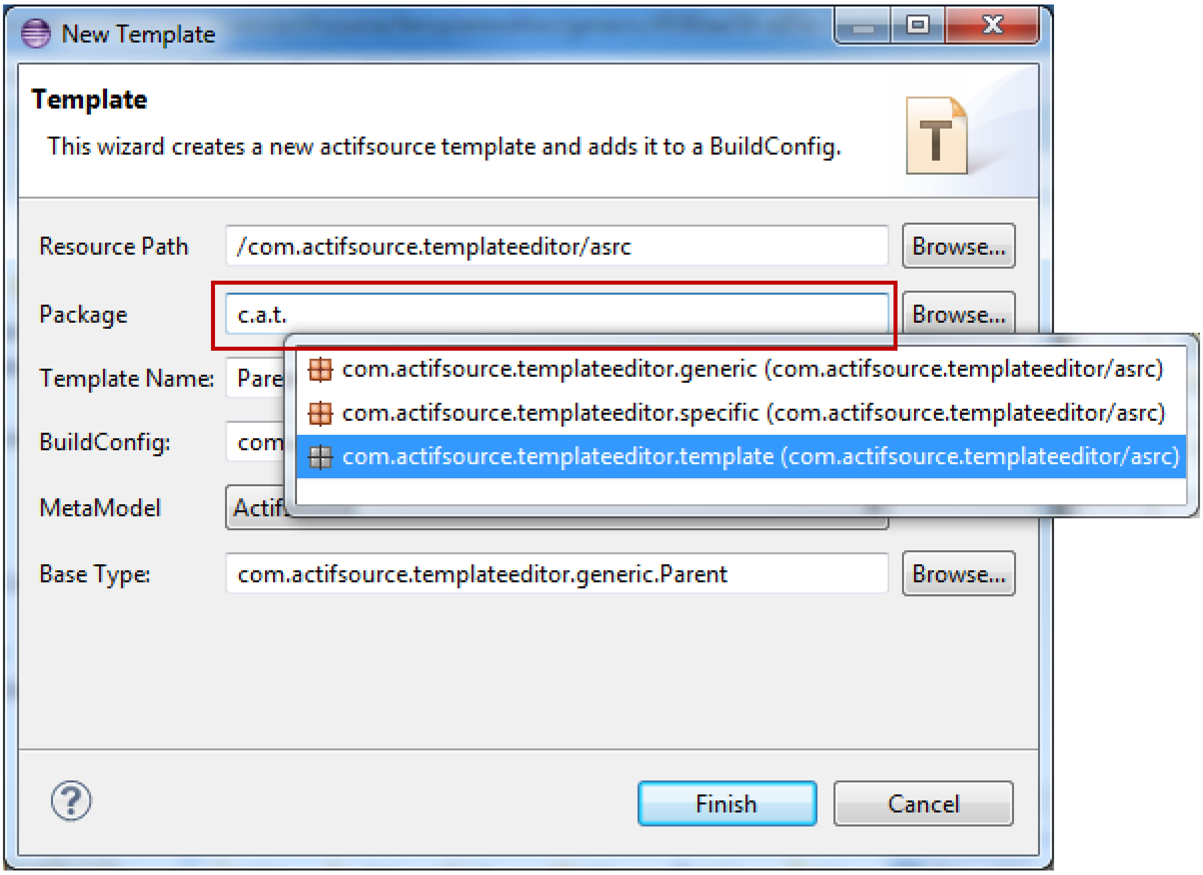
MetaModel: Actifsource

Base Type: com.actifsource.templateeditor.generic.Parent

Set the options as needed.

Option	Description
Resource Path	The resource path where the template is located (see Chapter 18 Resource Paths). This option is automatically filled in.
Package	The package where the template is located. The package is derived from the location where the context menu was called.
Template Name	The name of the template. The template name is automatically derived from the *Base Type*.
BuildConfig	The build configuration where this template is referenced (see Chapter 7 Build Config).
MetaModel	Make sure to choose Actifsource unless you know exactly what you do.
Base Type	The base type is derived from the location where the context menu was called.

Please note that there is a short way for choosing the package. Just type the first few letters of a package followed by a dot. Using content assist (Ctrl+Space) shows the matching packages.



New Template

Template

This wizard creates a new actifsource template and adds it to a BuildConfig.

Resource Path: /com.actifsource.templateeditor/asrc

Package: c.a.t.

- com.actifsource.templateeditor.generic (com.actifsource.templateeditor/asrc)
- com.actifsource.templateeditor.specific (com.actifsource.templateeditor/asrc)
- com.actifsource.templateeditor.template (com.actifsource.templateeditor/asrc)

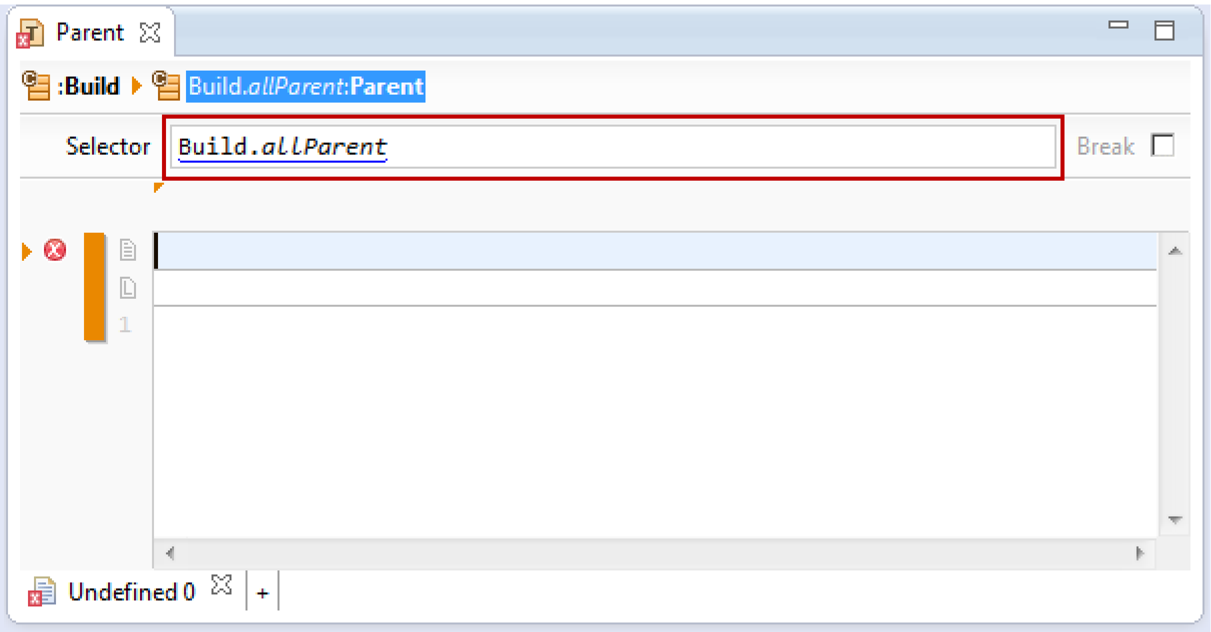
Template Name: Pare

BuildConfig: com

MetaModel: Actif

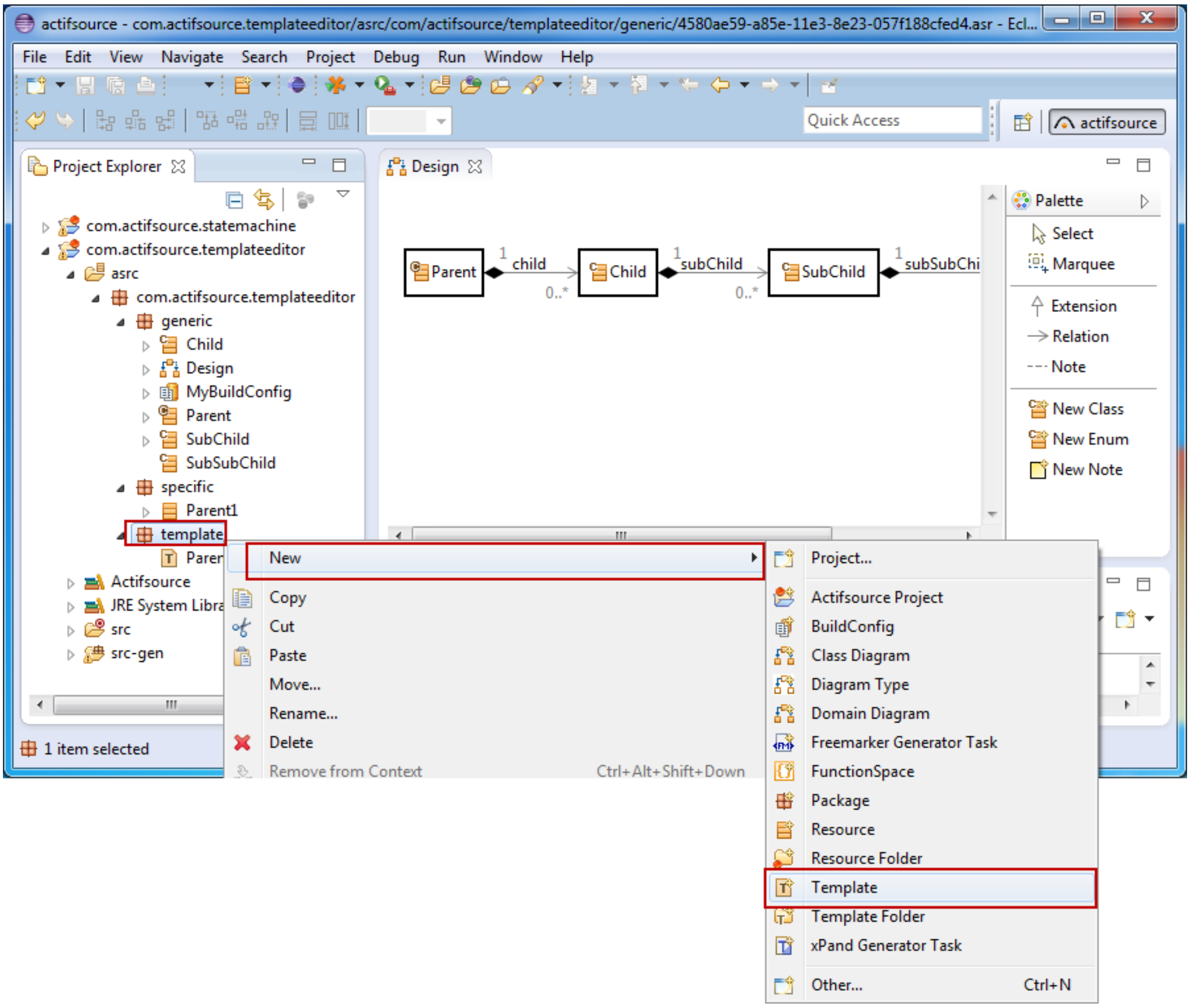
Base Type: com.actifsource.templateeditor.generic.Parent

Creating a template based on a type (i.e. class Parent) opens an editor with a predefined selector **Build.*allParent***. This means that this template is executed for all resources of type Parent.

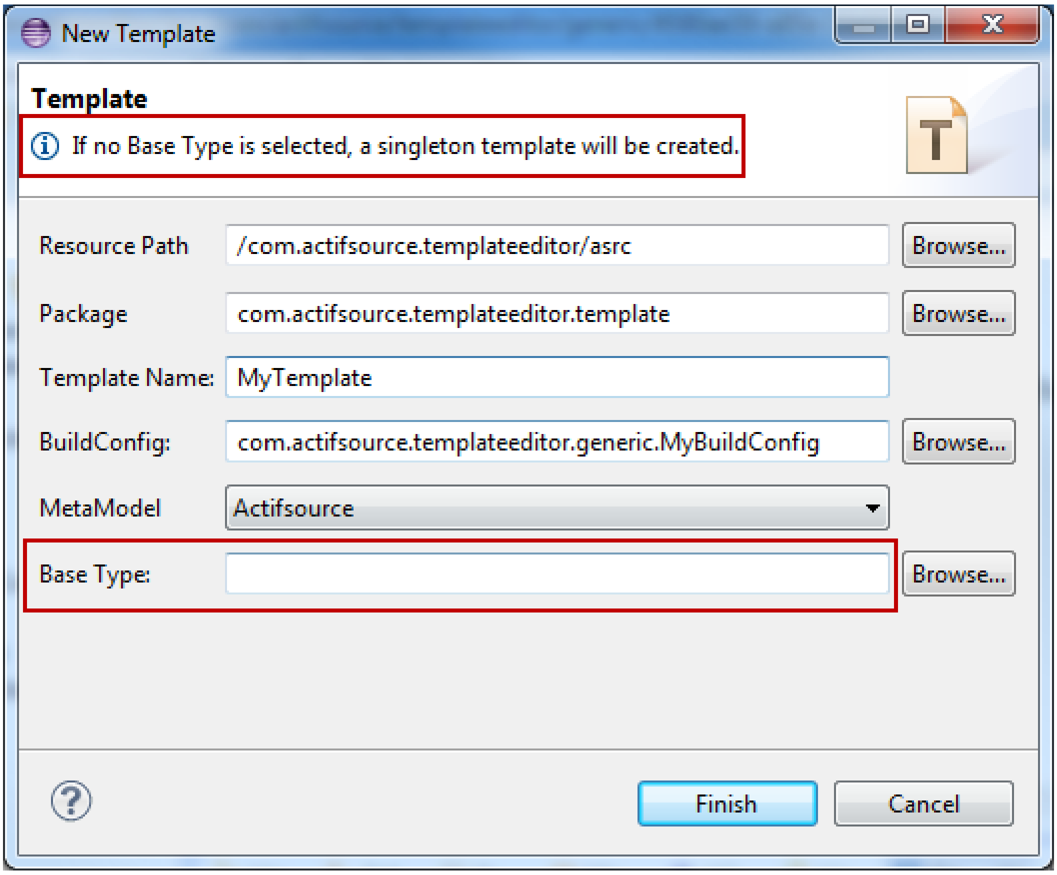


8.2.2 Create a Build.once Template

To create a **Build.*once*** template simply choose *New/Template* from the context menu of a *package*. Please note that the template is created in the chosen package.



The *New Template Wizard* asks for the template name and even allows you to add a base type afterwards using the content assist (Ctrl+Space). Adding a base type leads to a template based on a type (see Chapter 8.2.1 Create a template based on type).



New Template

Template

If no Base Type is selected, a singleton template will be created.

Resource Path: /com.actifsource.templateeditor/asrc

Package: com.actifsource.templateeditor.template

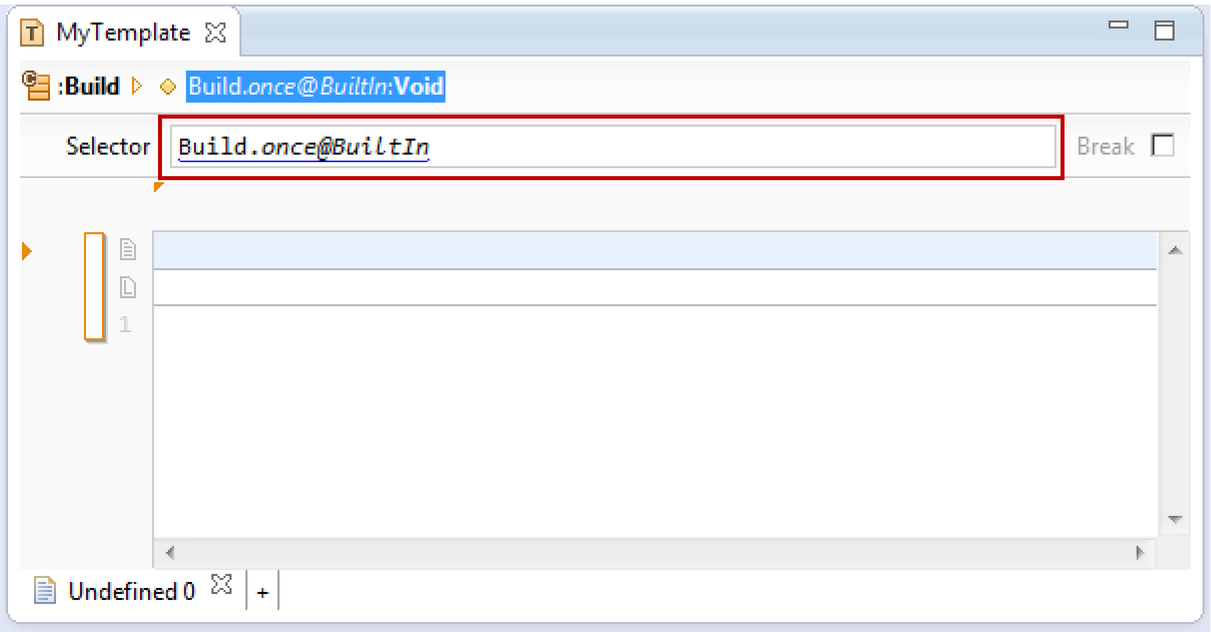
Template Name: MyTemplate

BuildConfig: com.actifsource.templateeditor.generic.MyBuildConfig

MetaModel: Actifsource

Base Type:

Creating a Build.once template opens an editor with a predefined selector **_Build.once@BuiltIn_**. That means that this template is executed only once.



MyTemplate

:Build ▷ ◇ Build.once@BuiltIn:Void

Selector: Build.once@BuiltIn

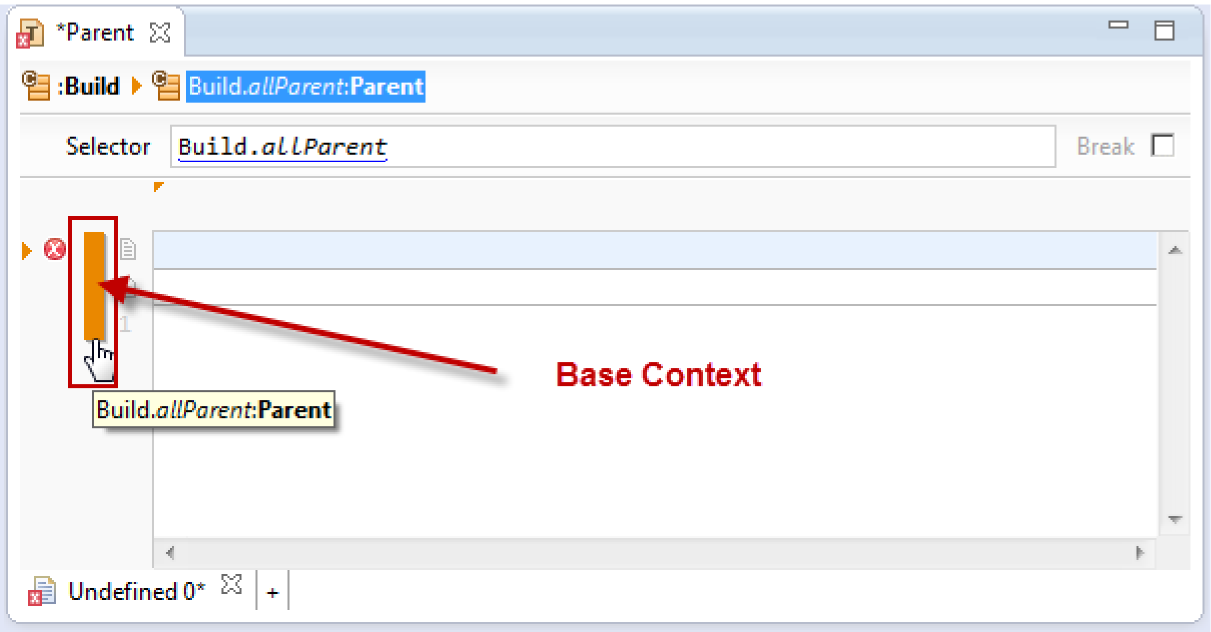
Break □

8.3 Writing template code

Writing template code is nearly as easy as writing common code – thanks to the *Actifsource Template Editor*.

8.3.1 Base Context

The *Actifsource Template Editor* lets you write code in the context of the meta-model.

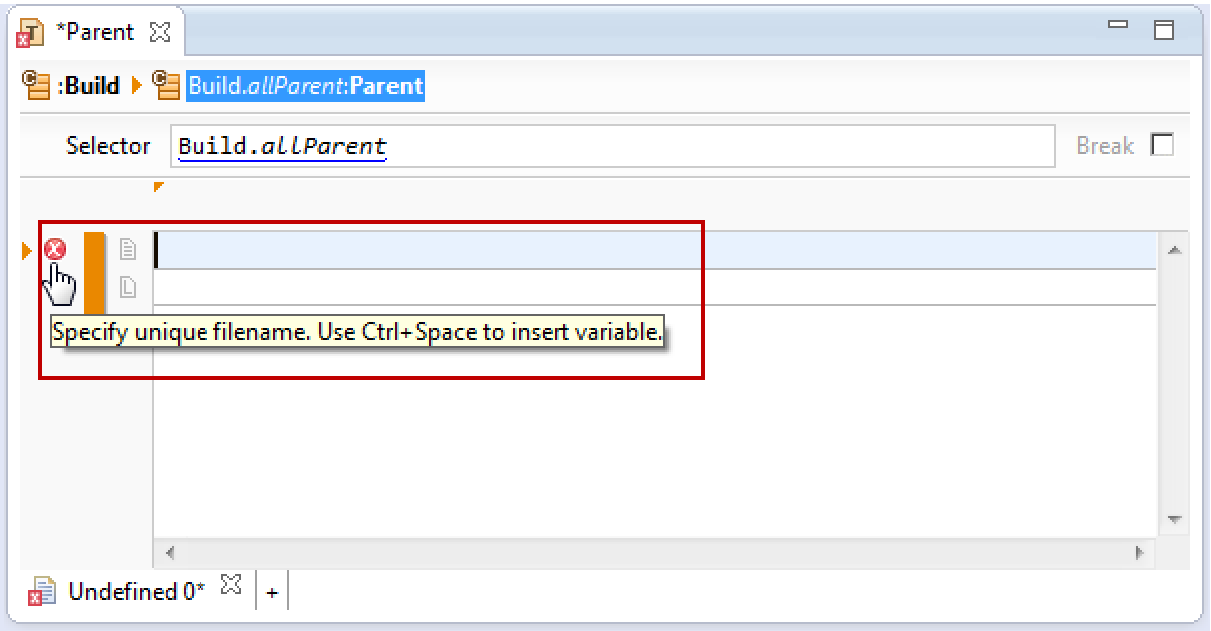


The orange bar on the left is the context you are in. Creating a template for the class Parent lets you work in the context of this class.

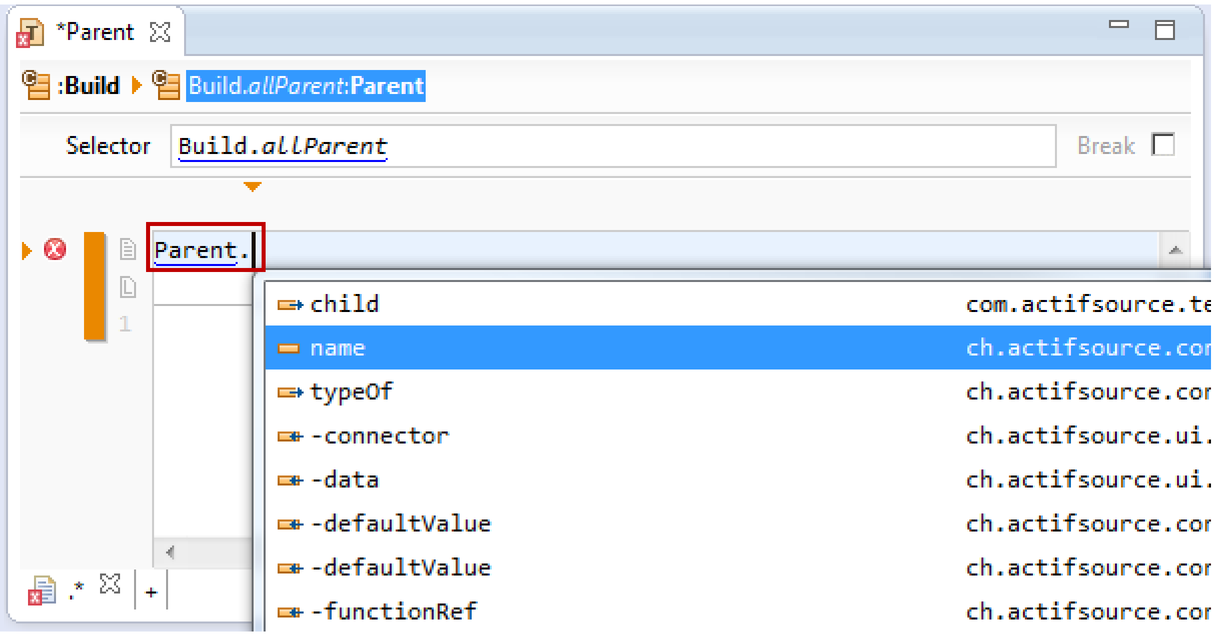
The context derived from the base type (see Chapter 8.2.1 Create a template based on type) is called *base context*.

8.3.2 File Line

First of all you have to specify a proper name. Since we want to generate a file for any instance of the class Parent, we have to specify a file name that is unique for every Parent instance.



The name of the generated files is derived from the specific resource instance for which code is generated. Use content assist (Ctrl+Space) to access the properties of the class which is bound to the base context by the *Selector*.



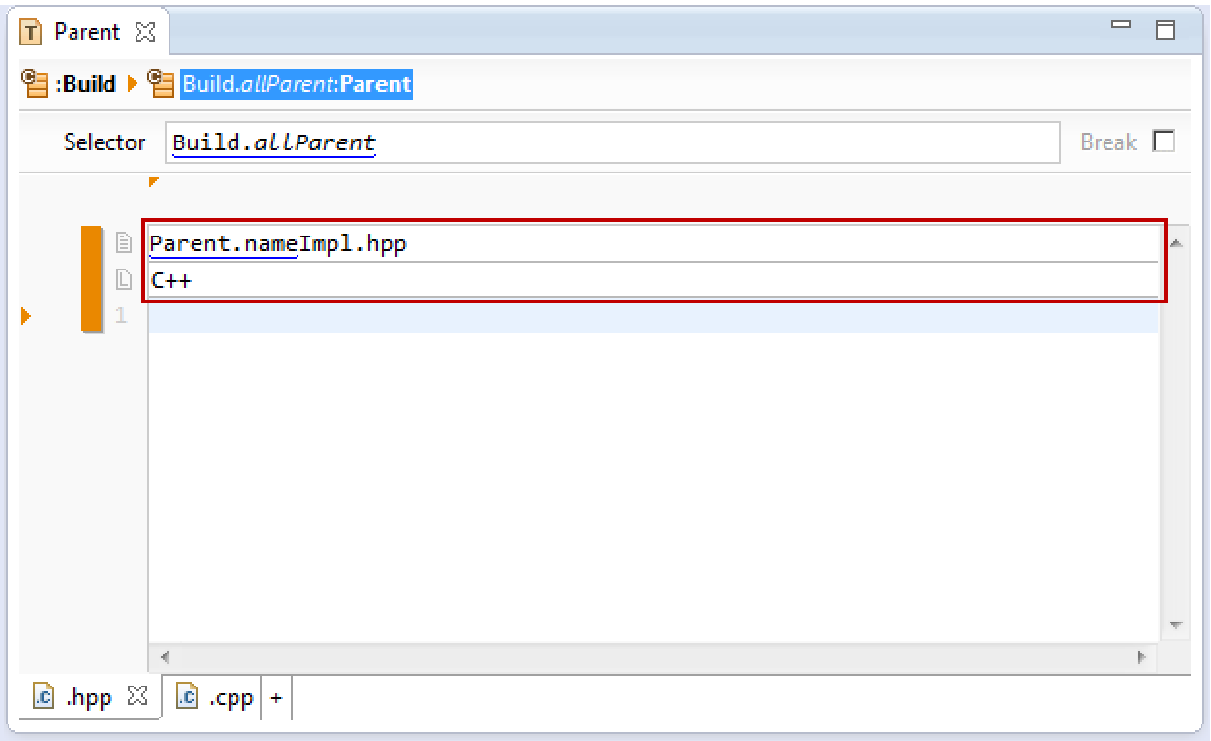
Selector Build.allParent

```
Parent.
```

```
child            com.actifsource.te
name             ch.actifsource.cor
typeOf           ch.actifsource.cor
-connector       ch.actifsource.ui.
-data            ch.actifsource.ui.
-defaultValue    ch.actifsource.cor
-defaultValue    ch.actifsource.cor
-functionRef     ch.actifsource.cor
```

The following file name will create files named Parent.nameImpl.hpp while Parent.name is replaced by the name of the specific instance of class Parent. Text elements referring to the model are called *links* and displayed underlined.

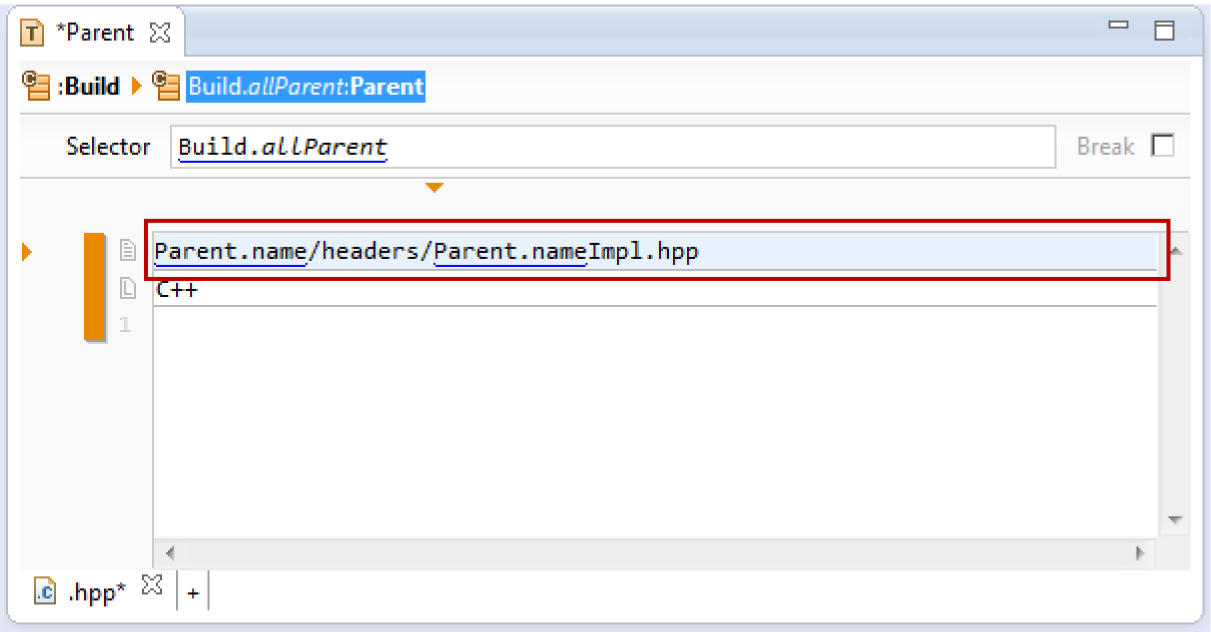
Please note that the file extension .hpp automatically selects the Language C++ (see Chapter 8.3.3 Language Line and Chapter 8.4 Declaring a Programming Language).



Selector Build.allParent

```
Parent.nameImpl.hpp
C++
```

It is also possible to define a folder structure in the file line. The generated files will be placed in the defined folders.



```
Build.allParent
Parent.name/headers/Parent.nameImpl.hpp
C++
```

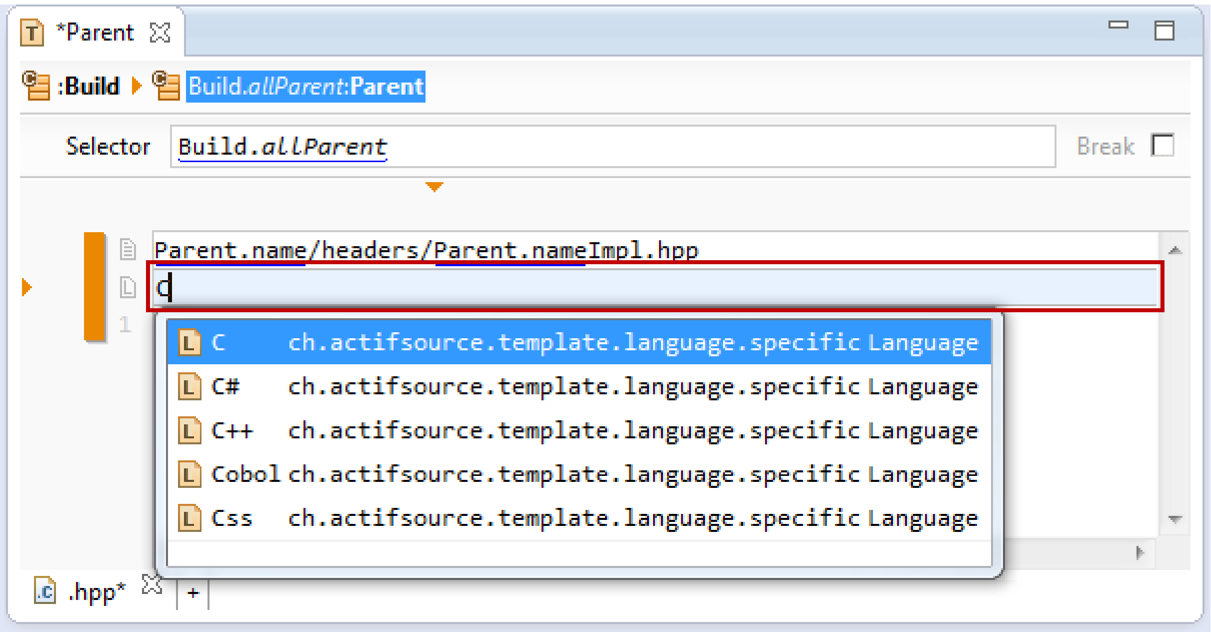
8.3.3 Language Line

The language line defines the programming language for

- Syntax Highlighting
- Comment Style
- String Style including escape rules

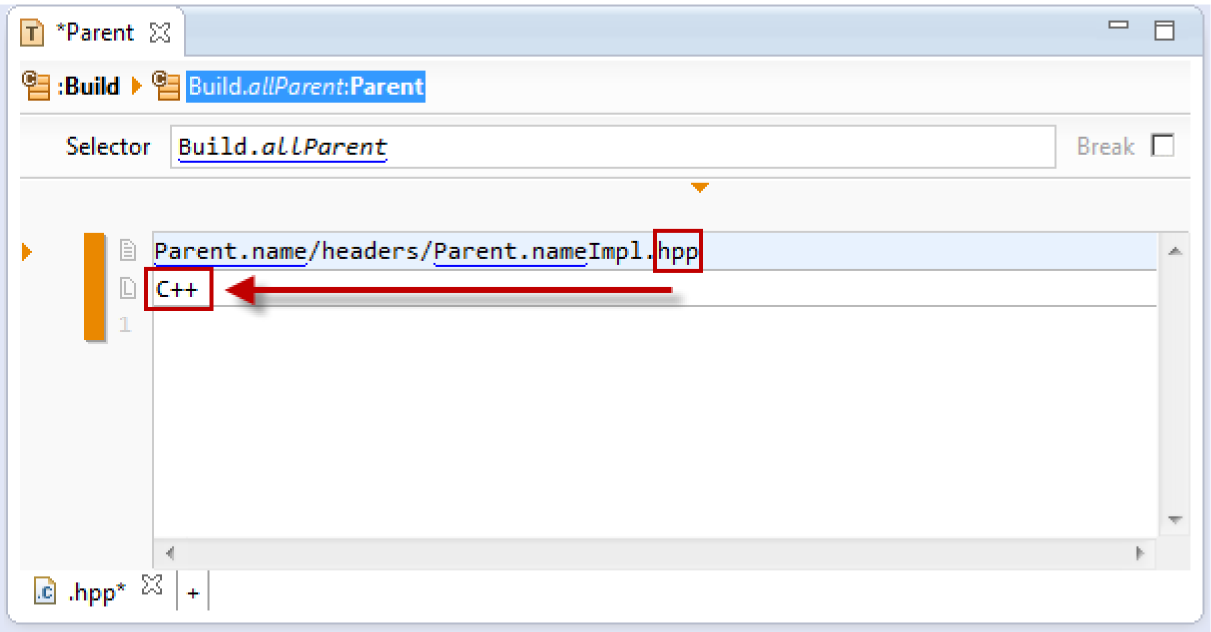
Actifsource defines the most common languages. If you are using a language which is not defined by default (see Chapter 8.4.1 Supported Programming Languages), do not hesitate to create one by your own (see Chapter 8.4 Declaring a Programming Language).

You may change the language at any time by using the content assist (Ctrl+Space) on the *language line*.

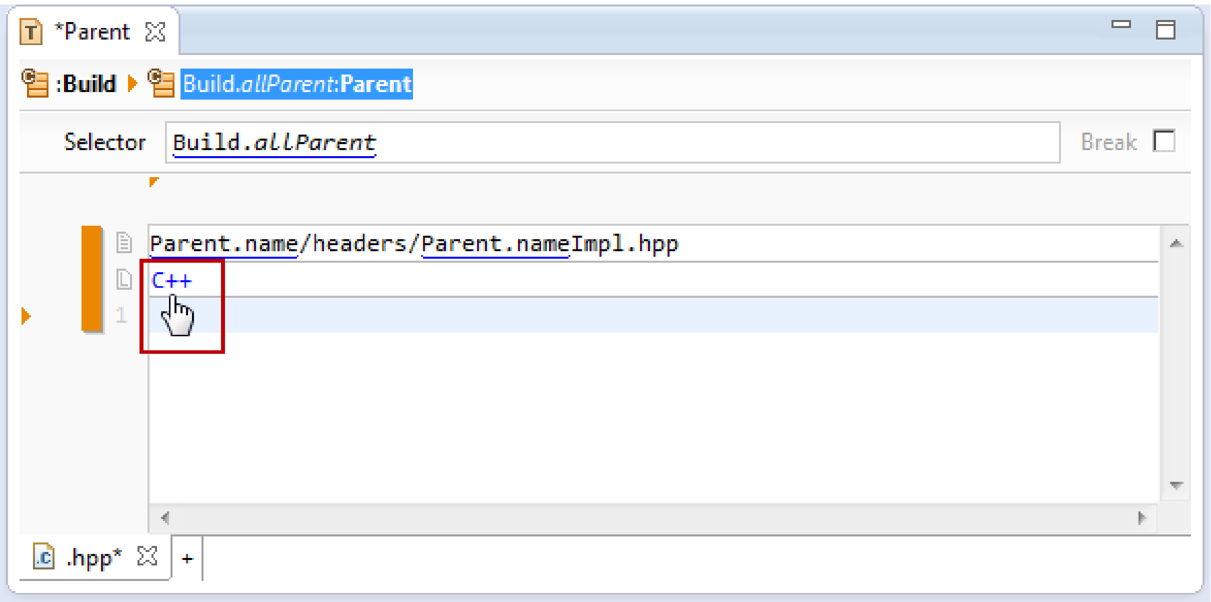


```
Build.allParent
Parent.name/headers/Parent.nameImpl.hpp
c
C     ch.actifsource.template.language.specific Language
C#    ch.actifsource.template.language.specific Language
C++   ch.actifsource.template.language.specific Language
Cobol ch.actifsource.template.language.specific Language
Css   ch.actifsource.template.language.specific Language
```

Selecting or changing a *file extension* in the *file line* (see Chapter 8.3.2 File Line) automatically selects the corresponding language.



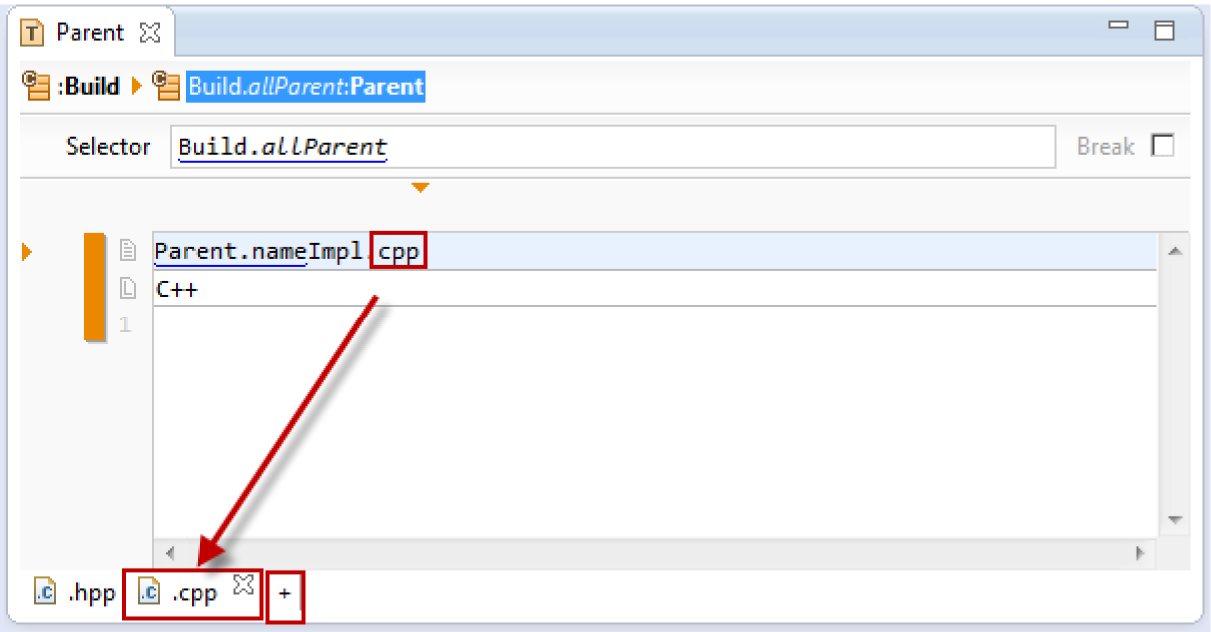
Use Ctrl+Click on the language to show the underlying language model (see Chapter 8.4 Declaring a Programming Language).



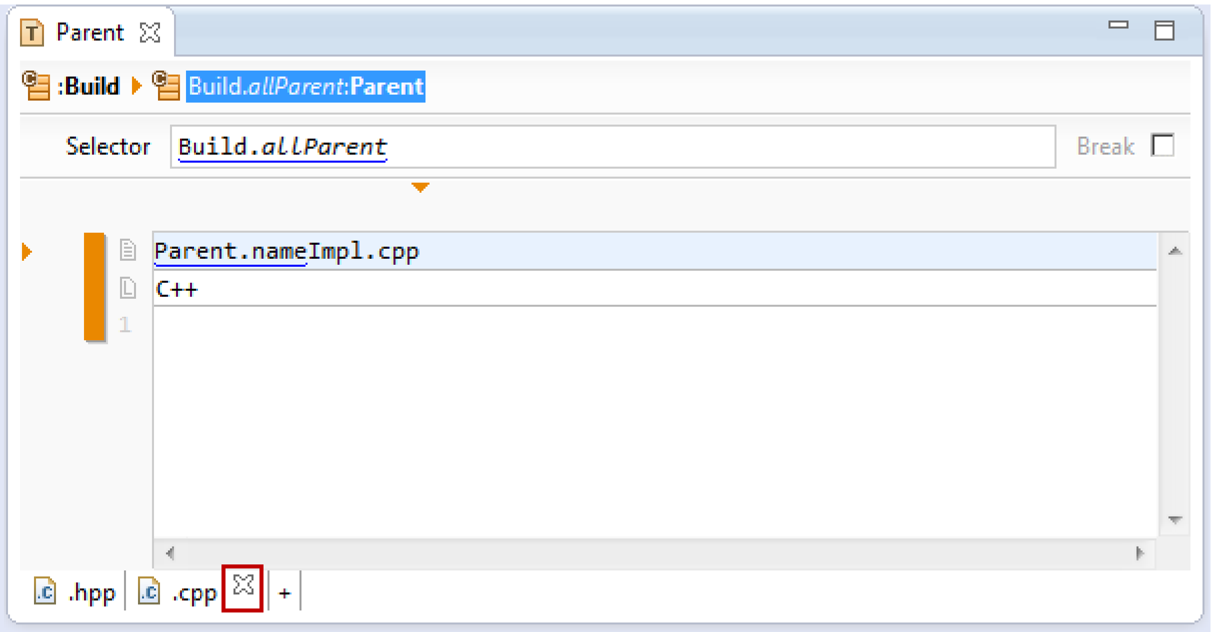
8.3.4 File Tab

There are always situations where two or more files belong to each other (i.e. hpp/cpp in C++). Actifsource therefore supports *file tabs*.

Just press the [+] button right next to the *file tabs* to add a new *file tab*. Note that *files tabs* are always automatically named the same as the *file extension*.



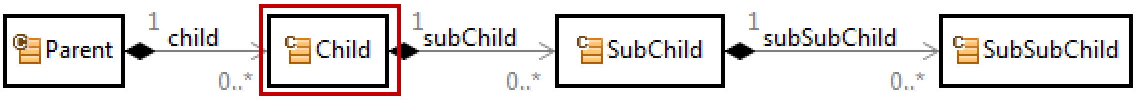
Press the X button to delete the active file tab.



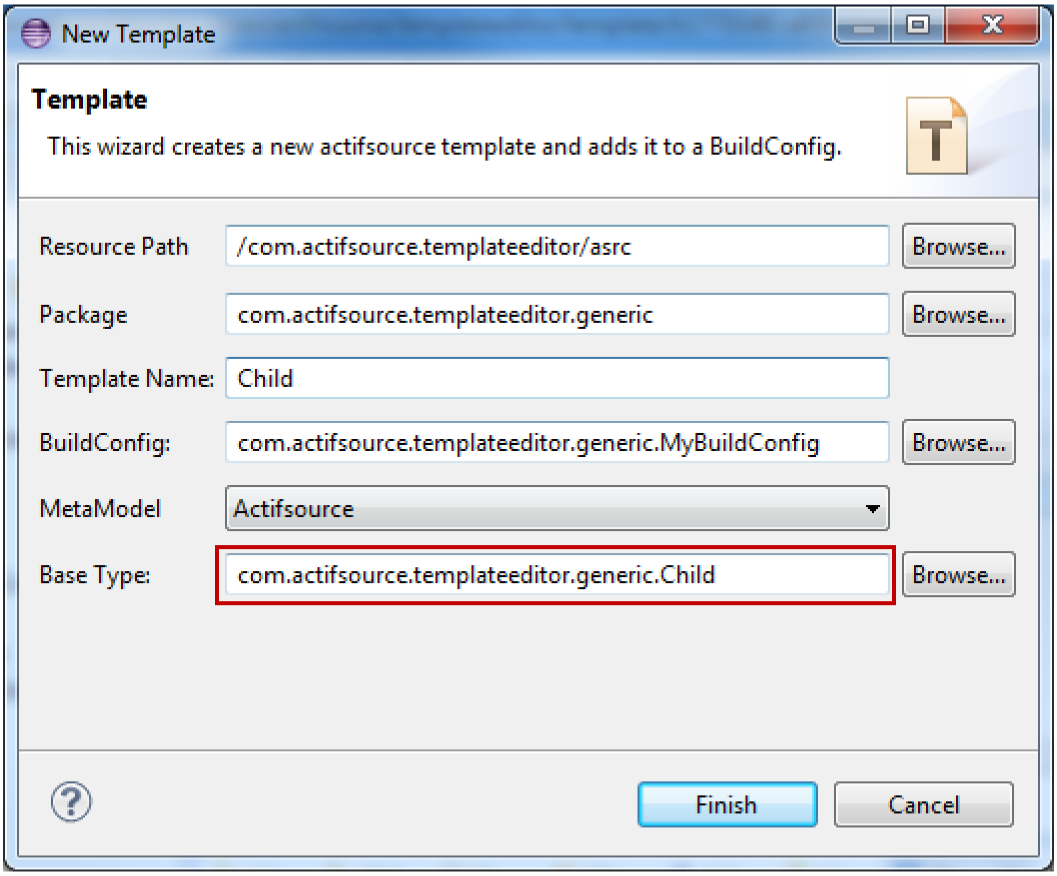
Press Ctrl+Tab to select next tab from within the code section.

8.3.5 SuperContext

Let’s assume that we want to generate a file for every Child instance.



For that reason we create a template with Child as *base type*.



This wizard creates a new actifsource template and adds it to a BuildConfig.

Resource Path: /com.actifsource.templateeditor/asrc

Package: com.actifsource.templateeditor.generic

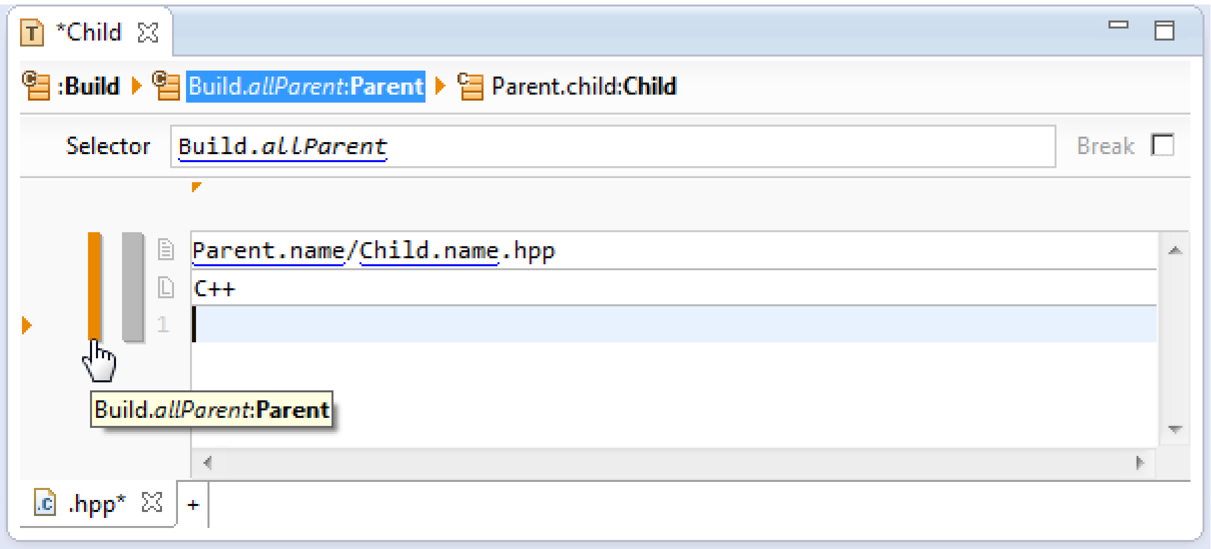
Template Name: Child

BuildConfig: com.actifsource.templateeditor.generic.MyBuildConfig

MetaModel: Actifsource

Base Type: com.actifsource.templateeditor.generic.Child

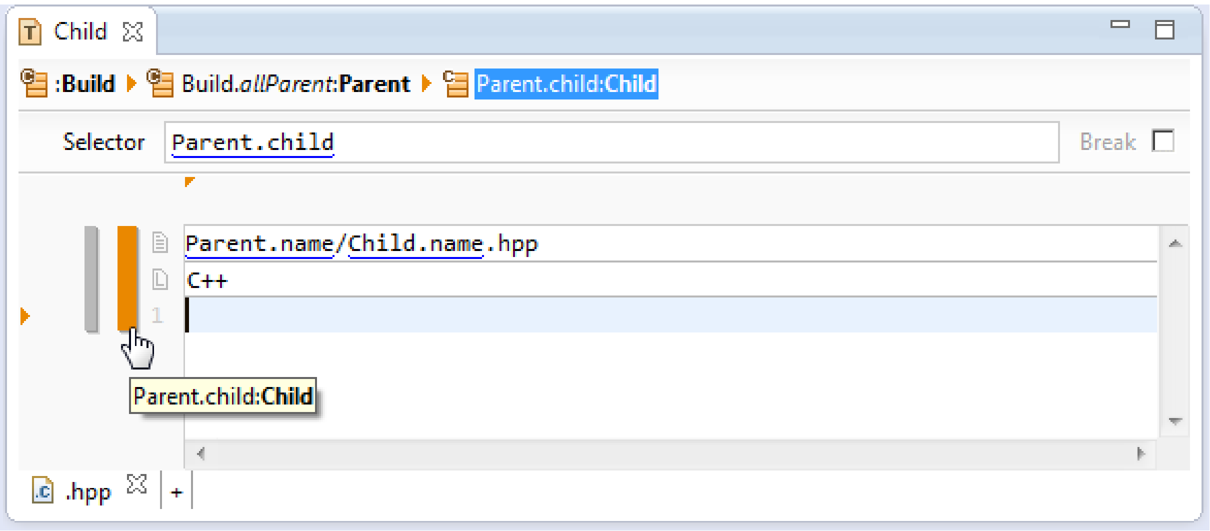
Since Child is owned by Parent.child, Actifsource automatically provides you with a super context of type Parent.



Selector: Build.allParent

```
Parent.name/Child.name.hpp
C++
```

Please note that the base context (i.e. Child in this example) is the widest bar (▮ 🟧).



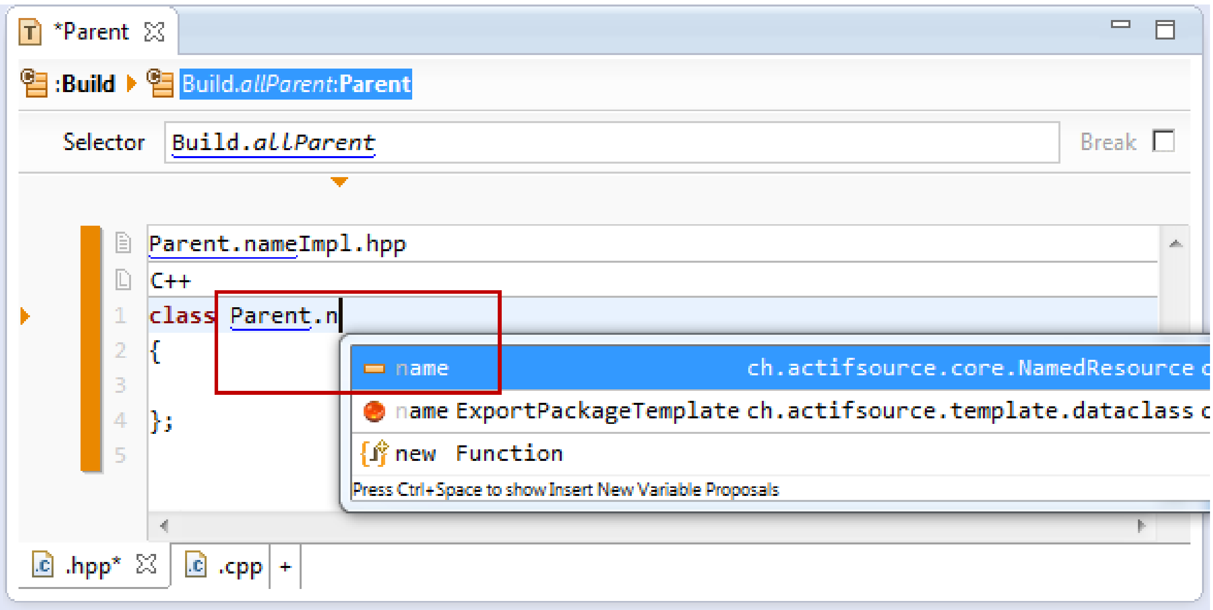
Selector Parent.child

```
Parent.name/Child.name.hpp
C++
```

8.3.6 Writing Code

Let's start writing code. First we write a C++ class named Parent.nameImpl while Parent.name is replaced by the name of the specific instance of class Parent. Note that the keyword *class* is bold and has a special color as defined in the language *C++* (see Chapter 8.4 Declaring a Programming Language).

To insert a reference to the meta-model just use content assist (Ctrl+Space) at any time.

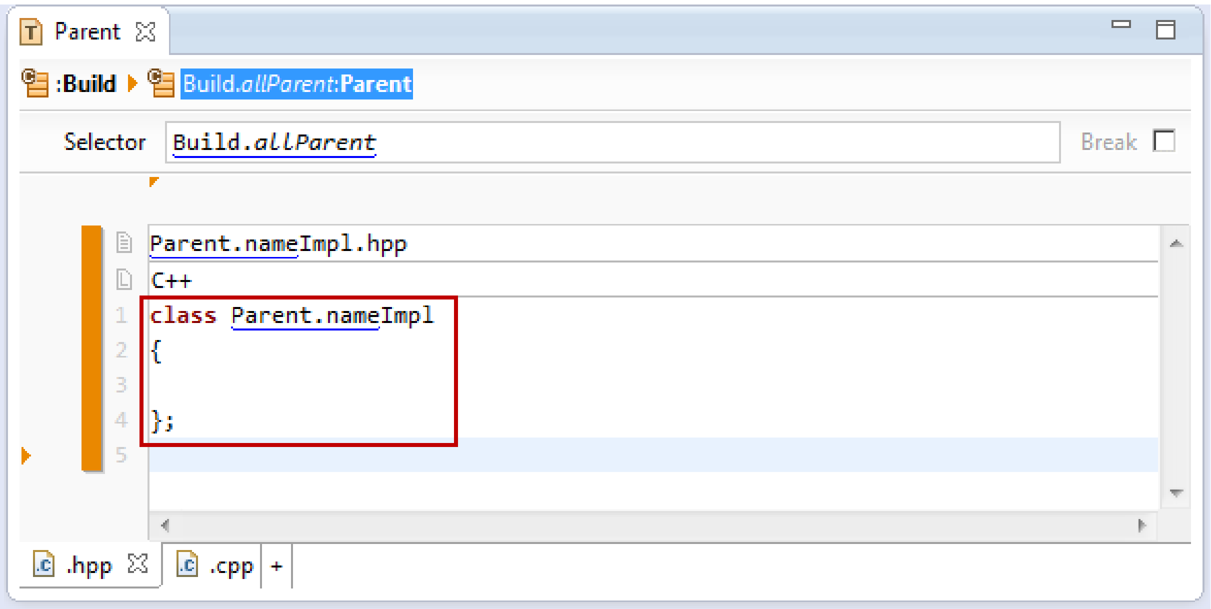


Selector Build.allParent

```
Parent.nameImpl.hpp
C++
class Parent.n
{

};
```

```
name ch.actifsource.core.NamedResource c
name ExportPackageTemplate ch.actifsource.template.dataclass c
new Function
Press Ctrl+Space to show Insert New Variable Proposals
```

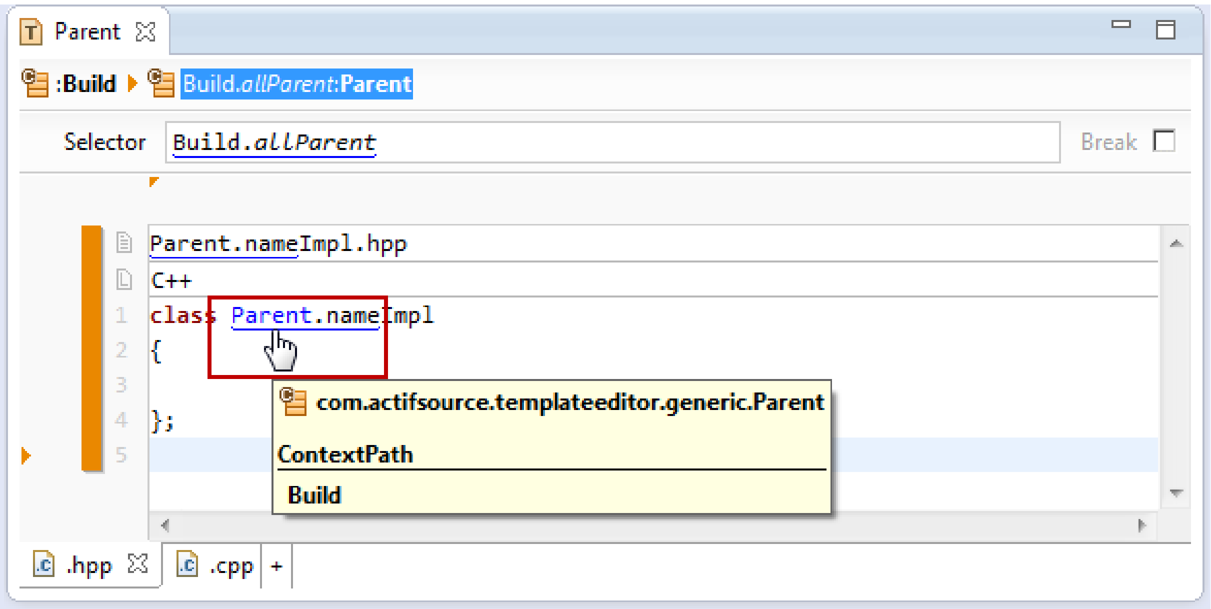
Underlined words are so called *links* which are directly linked with your model. Note that renaming resources in the meta-model automatically renames all links in the template synchronously.



Build.allParent

```
Parent.nameImpl.hpp
C++
class Parent.nameImpl
{

};
```

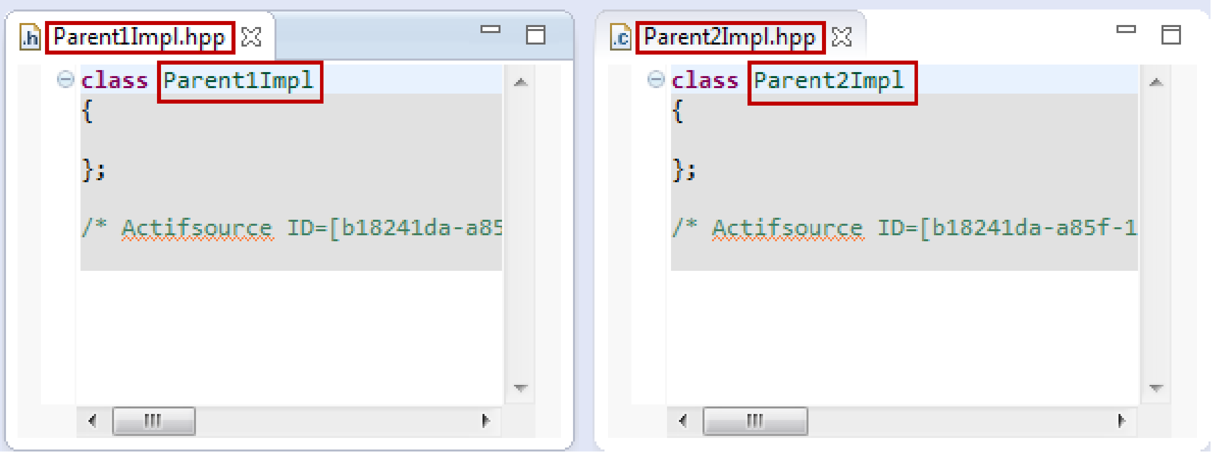
You can always navigate to the corresponding resource in the model by using Ctrl+Click on the links as shown below or the tool from the toolbar.



Build.allParent

```
Parent.nameImpl.hpp
C++
class Parent.nameImpl
{

};
```

Saving the above template leads to one file for every resource of type Parent in your project.



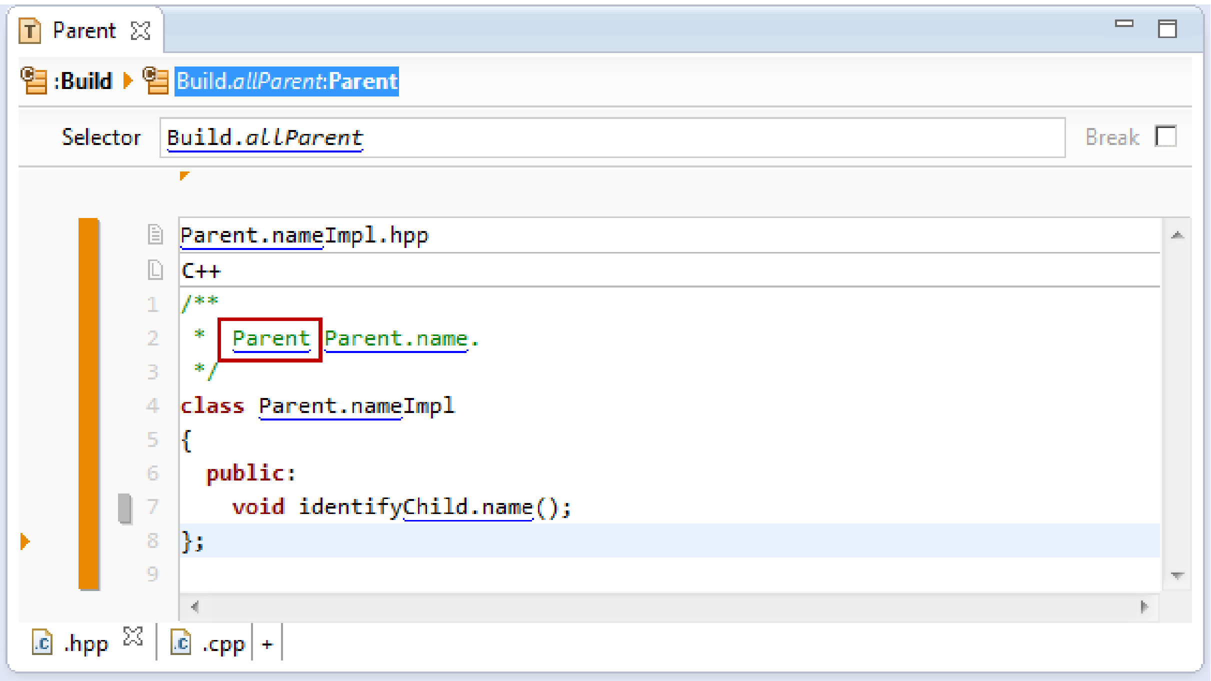
```
class Parent1Impl
{

};

/* Actifsource ID=[b18241da-a85
```

```
class Parent2Impl
{

};

/* Actifsource ID=[b18241da-a85f-1
```

A *build config* is needed to work with resources from other projects (see Chapter 7 Build Config).

8.3.7 Using type names in the template code

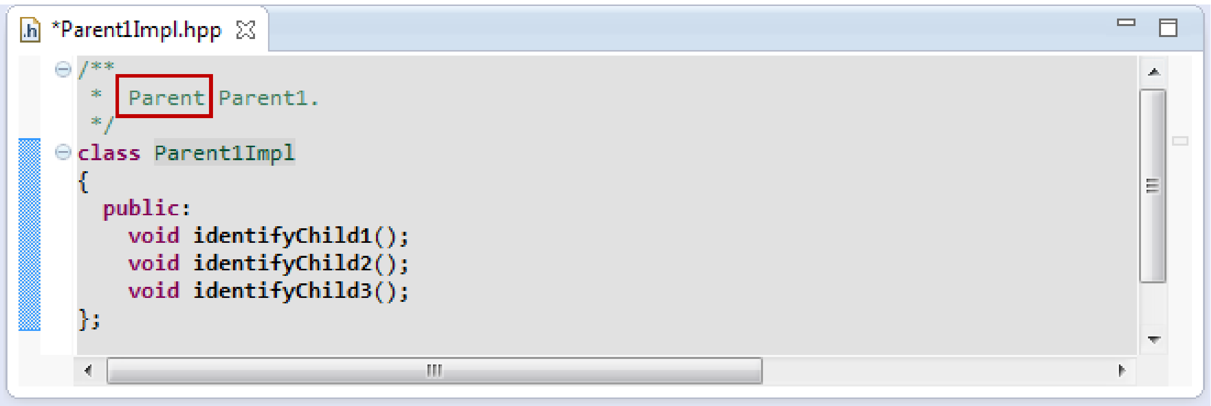
Please note that you might insert type names directly in the template code by using the content assist (Ctrl+Space). If the desired type name is not available, press Ctrl+Space again to get all available type names.



Selector: Build.allParent

```
Parent.nameImpl.hpp
C++
/**
 * Parent Parent.name.
 */
class Parent.nameImpl
{
  public:
    void identifyChild.name();
};
```

The type name is inserted just as given. The advantage of using type names in the template is the automatic renaming if the name of the type is changed.



```
/**
 * Parent Parent1.
 */
class Parent1Impl
{
  public:
    void identifyChild1();
    void identifyChild2();
    void identifyChild3();
};
```

8.3.8 Open Link

You can always open a resource link in the template editor.

Open Link with Default Editor

Use one of the following methods to open a link in the default editor (see Chapter 2.11.2 Open with).

Action	Opens	Description
Ctrl+Click	Default Editor	Press Ctrl+Click on the link to open the resource
F3	Default Editor	Press F3 on the current cursor position to open the resource
➜{	Default Editor	Click *Open Link in ResourceEditor* from the toolbar on the current cursor position to open the resource

To open a function link in the function editor use the default editor. To open the function model use the resource editor (see below).

Open Link with Resource Editor

Use one of the following methods to open a link in the resource editor (see Chapter 2.11.2 Open with).

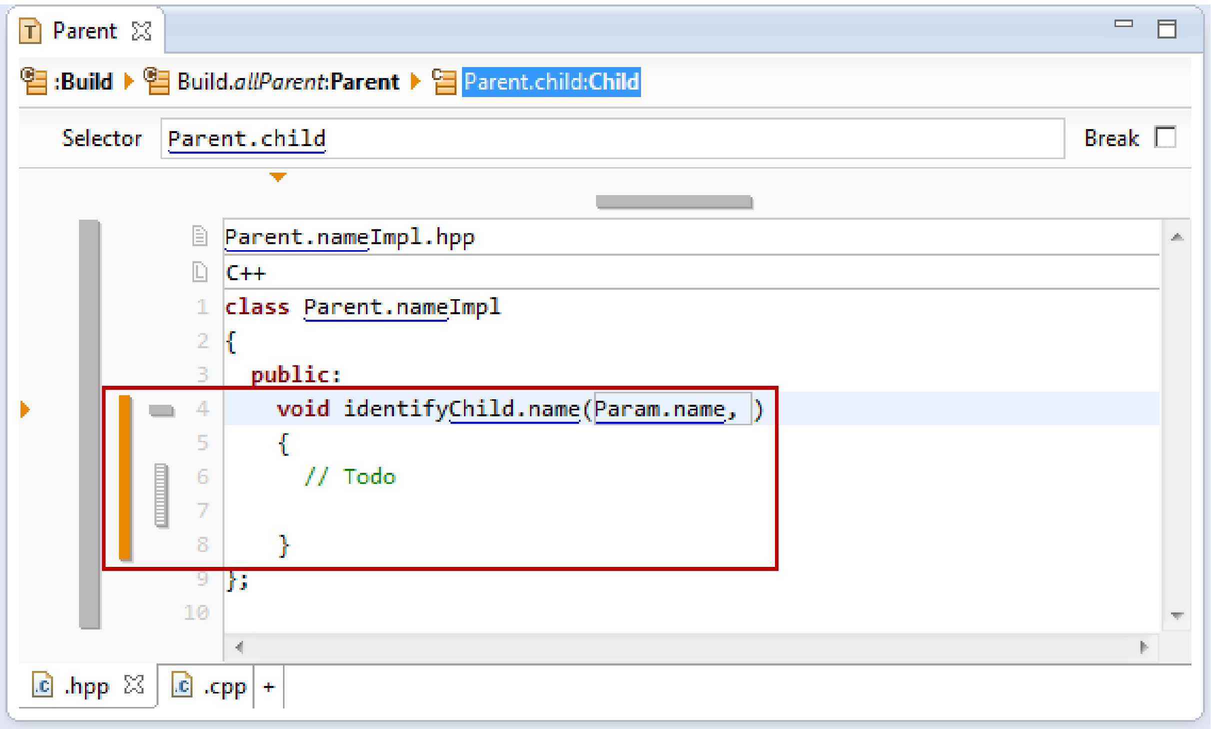
Action	Opens	Description
Ctrl+Alt+Click	Resource Editor	Press Ctrl+Click on the link to open the resource
Alt+F3	Resource Editor	Press F3 on the current cursor position to open the resource
➜▤	Resource Editor	Click *Open Link in ResourceEditor* from the toolbar on the current cursor position to open the resource

8.3.9 Line Context, Column Context, Protected Context

Actifsource knows three different types of contexts.

Line Context

The *line context* consists of one or more lines in a file. The text in the *line context* is repeated for any resource reached by the selector (see chapter 8.3.10 Working with Context). To insert a *line context* use the *Insert Line-Context* tool from the toolbar or press Alt+Insert.



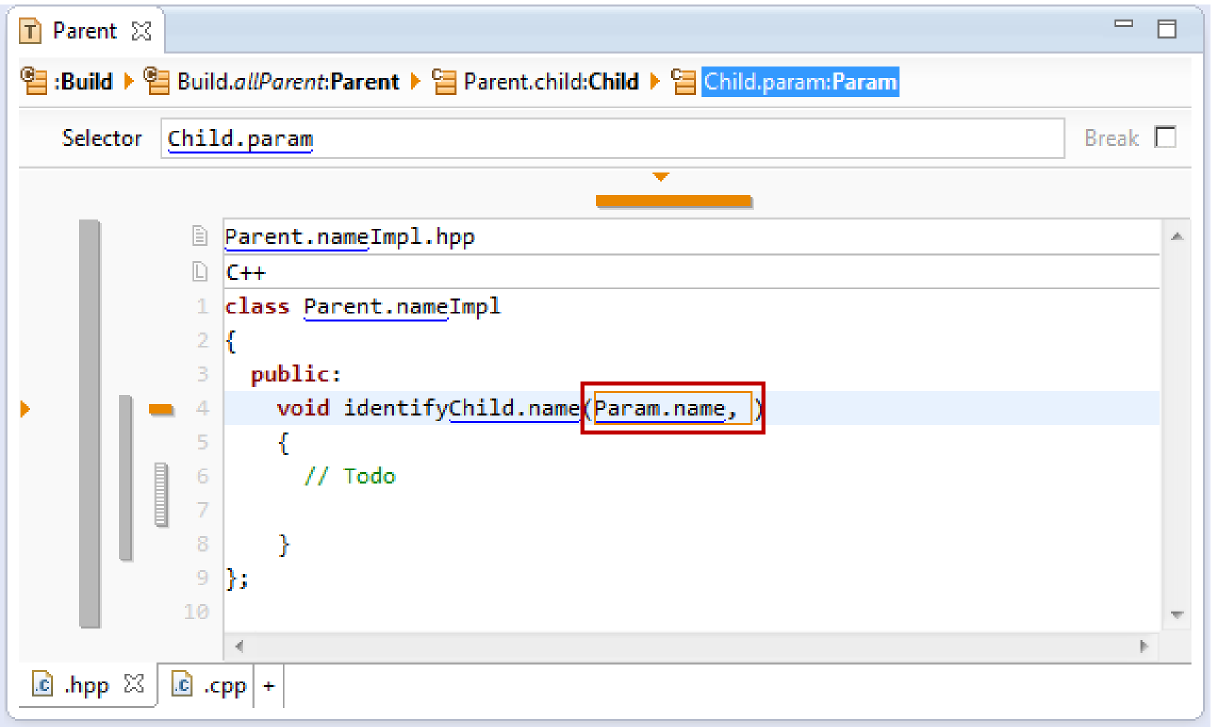
Selector Parent.child

Break □

```
Parent.nameImpl.hpp
C++
class Parent.nameImpl
{
  public:
    void identifyChild.name(Param.name, )
    {
      // Todo

    }
};
```

Column Context

The *column context* consists of one or more columns of a line. The text in the *column context* is repeated for any resource reached by the selector (see Chapter 8.3.10 Working with Context). To insert a *column context* use the *Insert ColumnContext* tool from the toolbar or press Alt+Shift+Insert.


Parent
:Build ▸ Build.allParent:Parent ▸ Parent.child:Child ▸ Child.param:Param
.hpp
.cpp
+

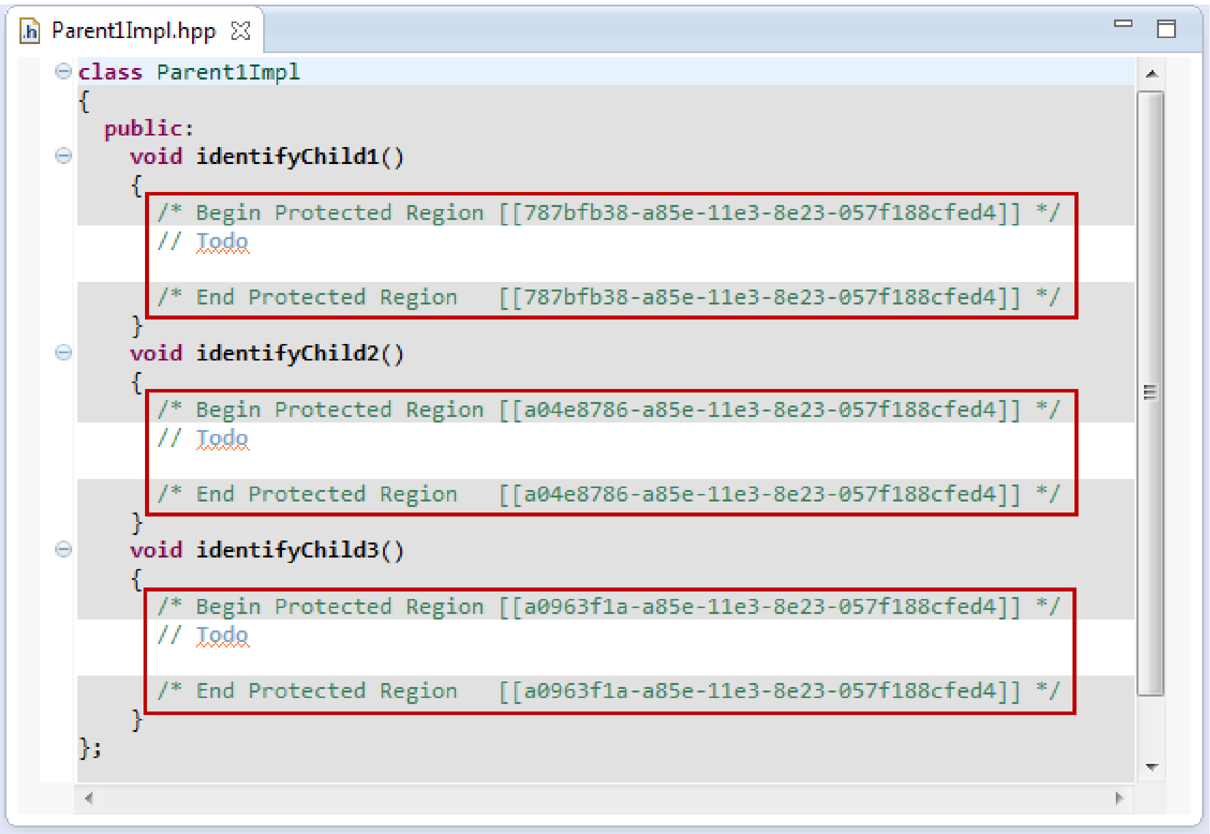
Selector Child.param

Break □

```
Parent.nameImpl.hpp
C++
class Parent.nameImpl
{
  public:
    void identifyChild.name(Param.name, )
    {
      // Todo

    }
};
```

Protected Context

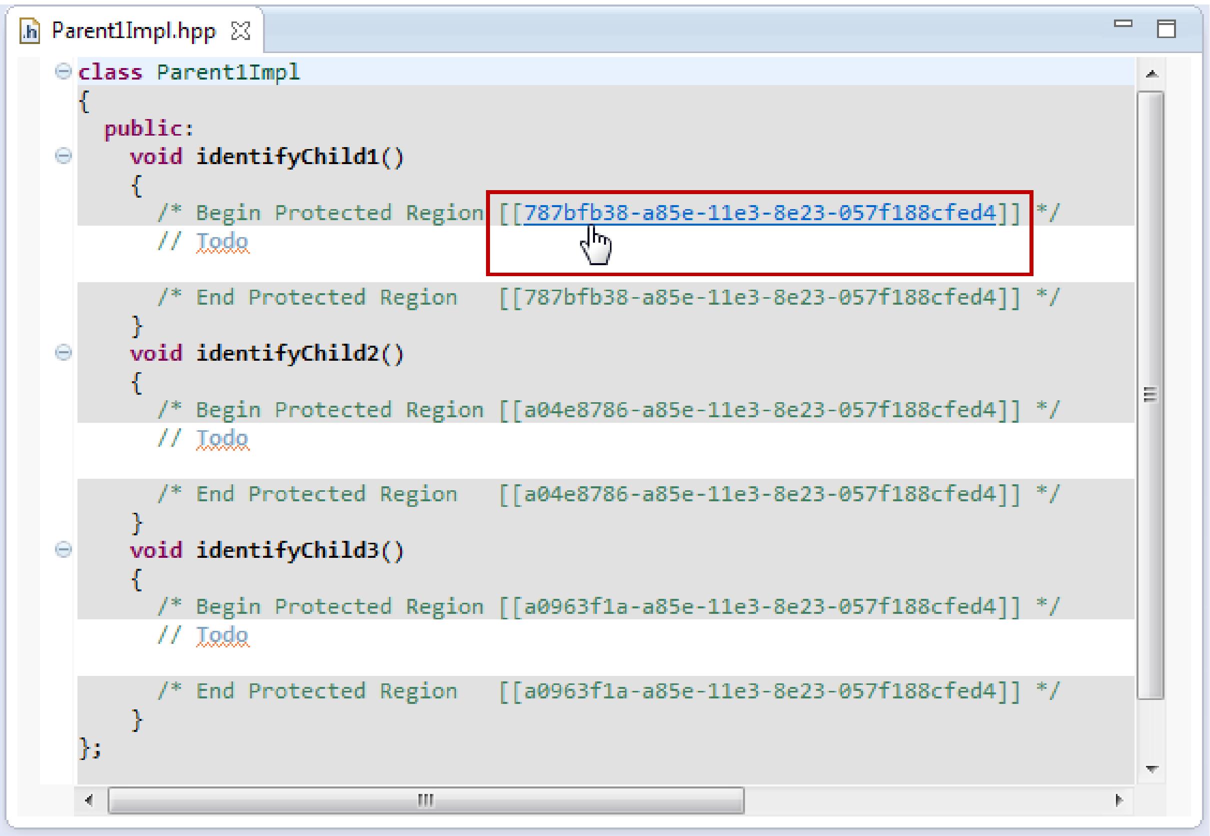
The *protected context* allows inserting so called *protected regions* into the generated files.



```
class Parent1Impl
{
  public:
    void identifyChild1()
    {
      /* Begin Protected Region [[787bfb38-a85e-11e3-8e23-057f188cfed4]] */
      // Todo

      /* End Protected Region   [[787bfb38-a85e-11e3-8e23-057f188cfed4]] */
    }
    void identifyChild2()
    {
      /* Begin Protected Region [[a04e8786-a85e-11e3-8e23-057f188cfed4]] */
      // Todo

      /* End Protected Region   [[a04e8786-a85e-11e3-8e23-057f188cfed4]] */
    }
    void identifyChild3()
    {
      /* Begin Protected Region [[a0963f1a-a85e-11e3-8e23-057f188cfed4]] */
      // Todo

      /* End Protected Region   [[a0963f1a-a85e-11e3-8e23-057f188cfed4]] */
    }
};
```

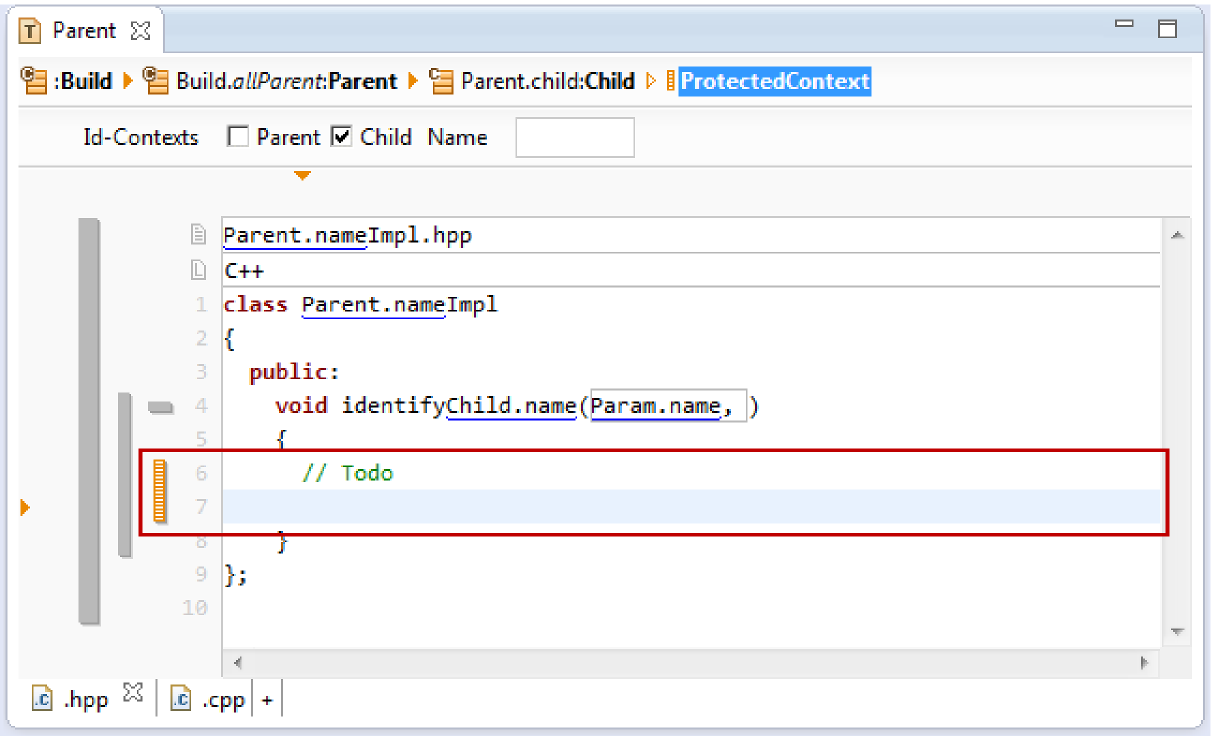
The content of the *Protected Regions* is saved before regenerating and inserted in the newly generated file. Use *Protected Regions* to insert handwritten code into generated files.

Note that *Protected Regions* are identified by the GUID of the resource of the current context. Use Ctrl+Click on the GUID to navigate to the corresponding resource.



```
class Parent1Impl
{
  public:
    void identifyChild1()
    {
      /* Begin Protected Region [[787bfb38-a85e-11e3-8e23-057f188cfed4]] */
      // Todo

      /* End Protected Region   [[787bfb38-a85e-11e3-8e23-057f188cfed4]] */
    }
    void identifyChild2()
    {
      /* Begin Protected Region [[a04e8786-a85e-11e3-8e23-057f188cfed4]] */
      // Todo

      /* End Protected Region   [[a04e8786-a85e-11e3-8e23-057f188cfed4]] */
    }
    void identifyChild3()
    {
      /* Begin Protected Region [[a0963f1a-a85e-11e3-8e23-057f188cfed4]] */
      // Todo

      /* End Protected Region   [[a0963f1a-a85e-11e3-8e23-057f188cfed4]] */
    }
};
```

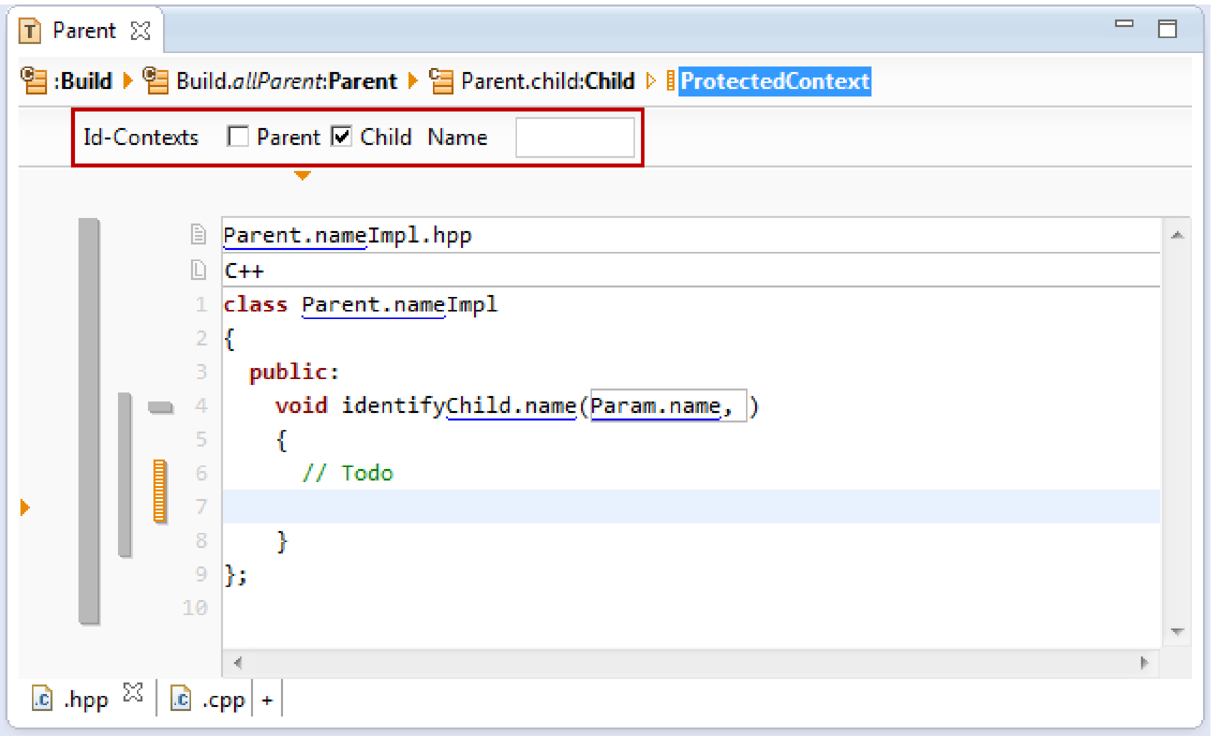
The *protected context* consists of one or more lines in a file. The text in the *protected context* is repeated for any resource reached by the selector (see Chapter 8.3.10 Working with Context) and generated. To insert a *protected context* use the *Insert ProtectedContext* tool from the toolbar.


Parent
:Build ▸ Build.allParent:Parent ▸ Parent.child:Child ▸ ProtectedContext
.hpp
.cpp
+

Id-Contexts ☐ Parent ☑ Child Name

```
Parent.nameImpl.hpp
C++
class Parent.nameImpl
{
  public:
    void identifyChild.name(Param.name, )
    {
      // Todo

    }
};
```

Note that you can control the GUIDs that identify the protected regions by checking the resources in the context path. Just make sure that the resulting set of GUIDs is unique in your generated file. You might also define a name for the *Protected Region*.


Parent
:Build ▸ Build.allParent:Parent ▸ Parent.child:Child ▸ ProtectedContext
.hpp
.cpp
+

Id-Contexts ☐ Parent ☑ Child Name

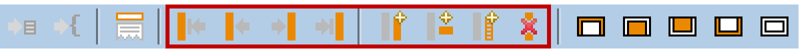
```
Parent.nameImpl.hpp
C++
class Parent.nameImpl
{
  public:
    void identifyChild.name(Param.name, )
    {
      // Todo

    }
};
```

Please note that changing the name of the Protected Region or the involved resources leads to new Protected Regions while the old once are moved to the end of the file.

8.3.10 Working with Contexts

Adding and removing a context is one of the most important operations when working with the Template Editor.

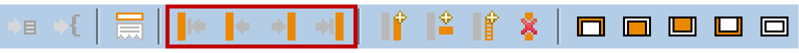
Use the Template Editor Toolbar to add, remove or navigate contexts.



Context Operation	Icon	Shortcut	Description
Select TopContext	⇤	Alt+Home	Selects the top context
Select ParentContext	←	Alt+PgUp	Selects the parent context from the actual context
Select ChildContext	→	Alt+PgDown	Selects the child context from the actual context
Select BottomContext	⇥	Alt+End	Selects the bottom context
Insert LineContext	↑	Alt+Insert	Inserts a line context in the actual context
Insert ColumnContext	↑	Alt+Shift+Insert	Inserts a column context in the actual context
Insert ProtectedContext	↑		Inserts a protected context in the actual context
Delete Context	✗	Alt+Delete	Delete the actual context

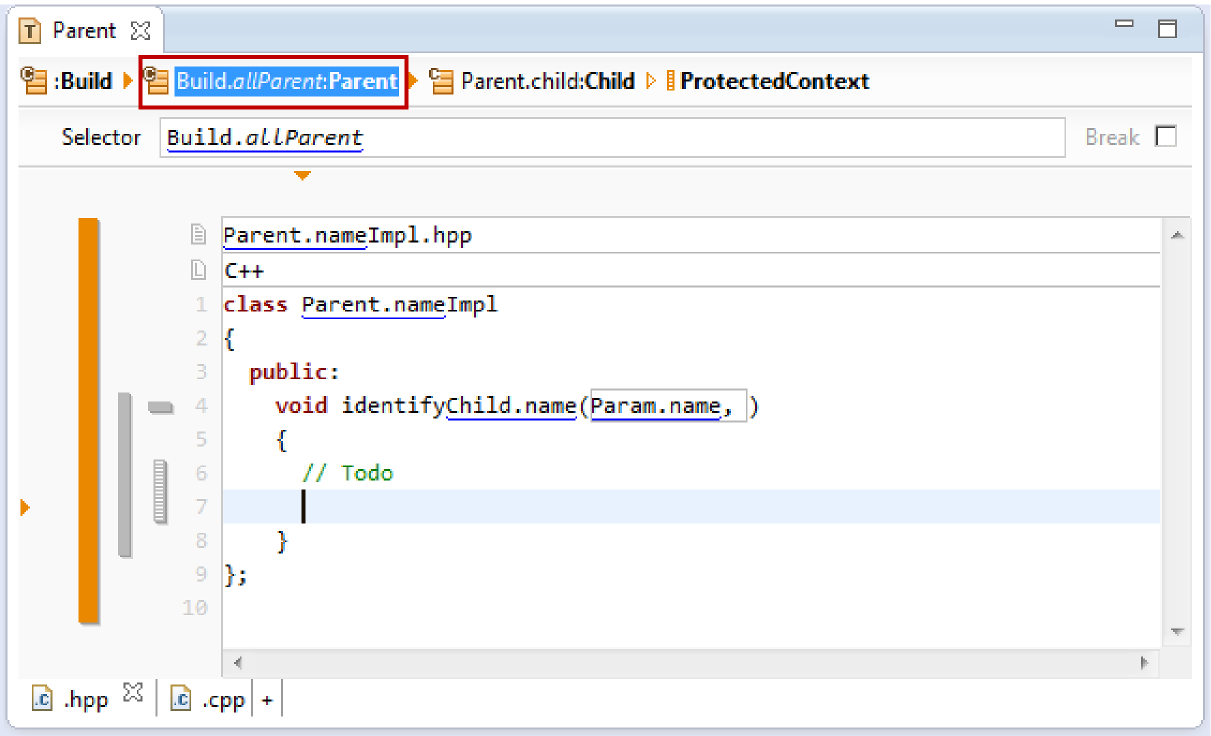
Navigate Context via Select Tools

Using the Context Select Tools in the Toolbar you might change the selection of the context from parent to child and vice versa.



Navigate Context via Breadcrumb

A context can be selected by clicking on the *Breadcrumb*.

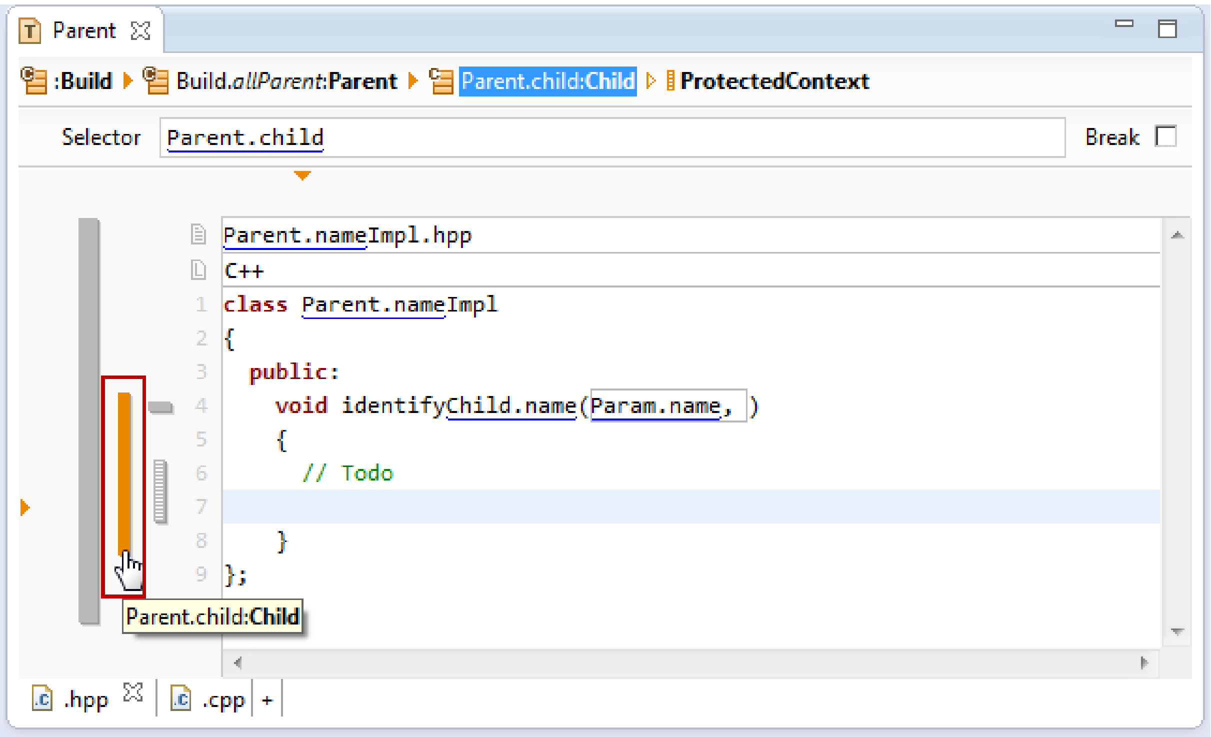


Selector: Build.allParent

```
Parent.nameImpl.hpp
C++
class Parent.nameImpl
{
  public:
    void identifyChild.name(Param.name, )
    {
      // Todo
      
    }
};
```

Navigate Context via Context Bar

A context can also be selected by clicking on the context bar.


Parent
:Build ▸ Build.allParent:Parent ▸ Parent.child:Child ▸ ProtectedContext
Selector
Break □
Parent.child:Child
.hpp
.cpp
+

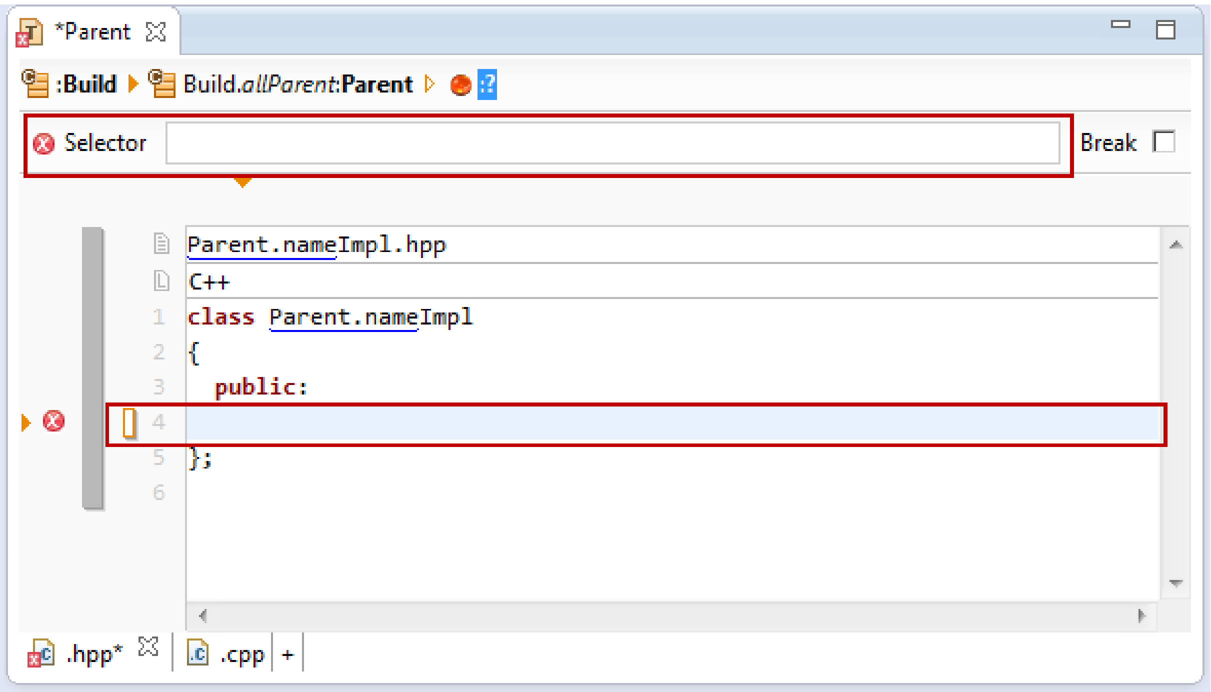
Selector: Parent.child

```
Parent.nameImpl.hpp
C++
class Parent.nameImpl
{
  public:
    void identifyChild.name(Param.name, )
    {
      // Todo

    }
};
```

Add Context

A new context is always added after the actually selected context. Navigate to a certain context before inserting a new context as shown above.

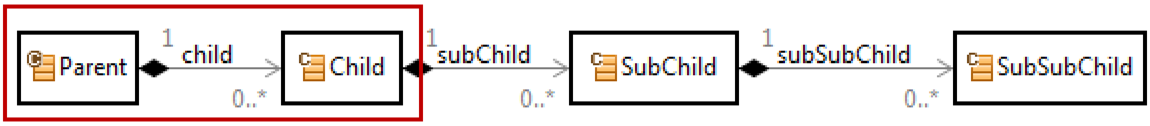
Let’s assume that we want create a function identifyChild.name() for every child in the parent context. For that reason we insert a new context using the *Insert LineContext* tool from the toolbar.



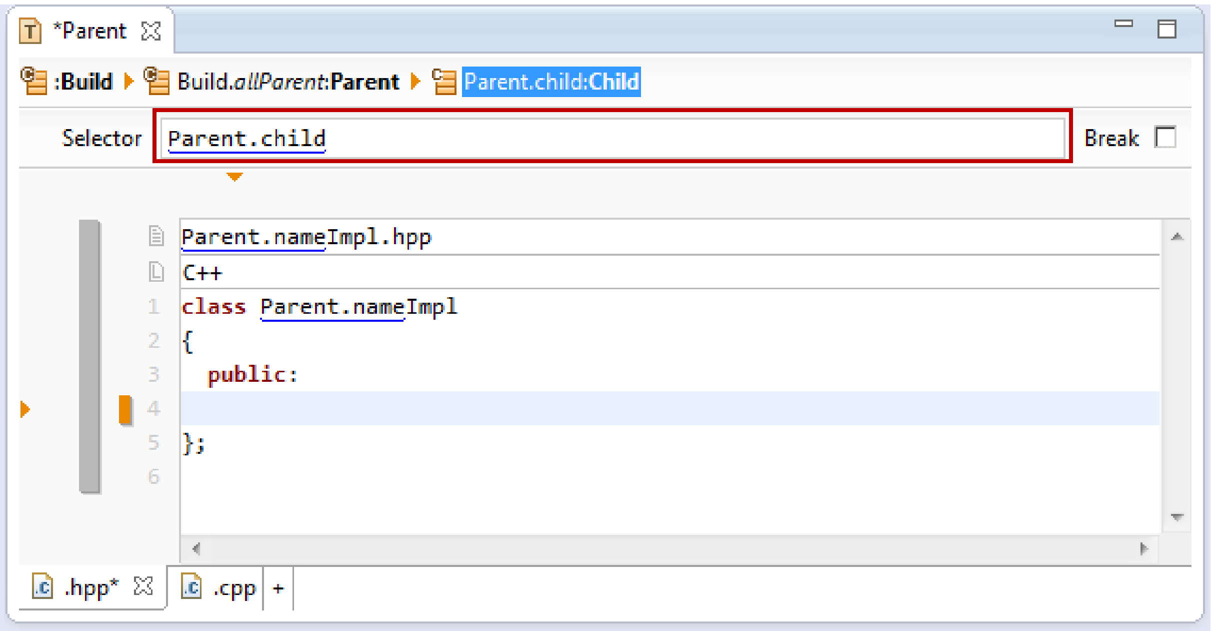
```
Parent.nameImpl.hpp
C++
class Parent.nameImpl
{
  public:

};
```

As a second step you have to declare a *selector* (see Chapter 8.3.12 Selector) to define the context.

Since our *base context* is Parent, we have to traverse the relation Parent.child to reach all children from parent.



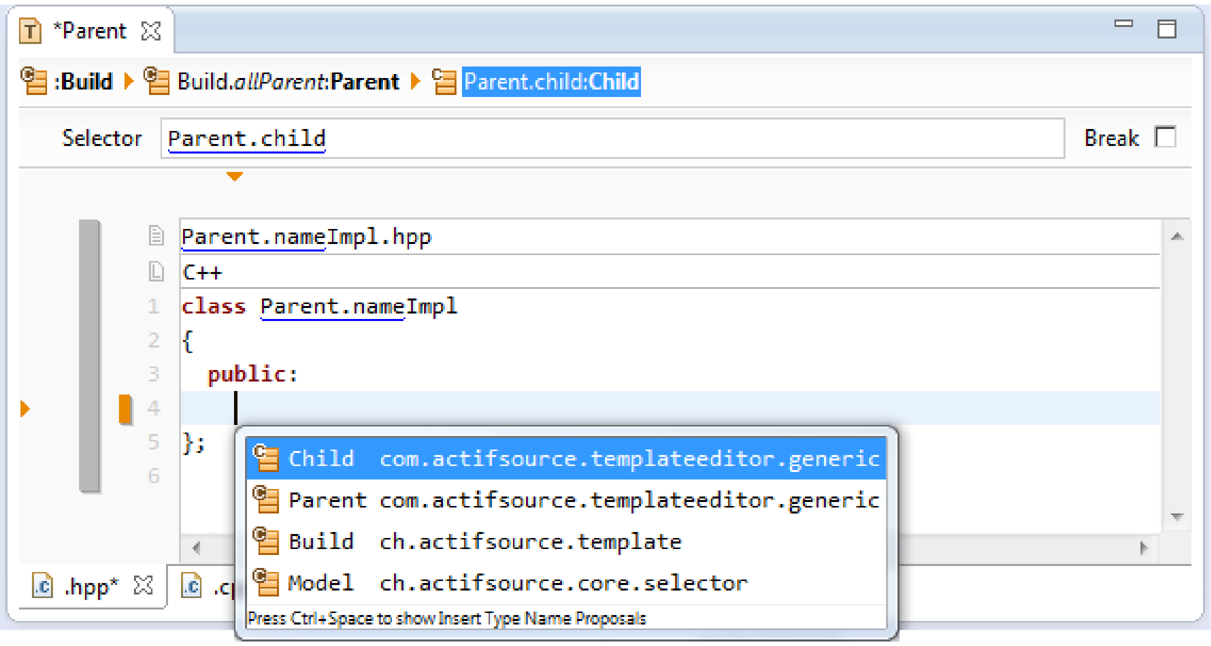
Choose the relation Parent.child for the selector using content assist (Ctrl+Space).



Selector: Parent.child

```
Parent.nameImpl.hpp
C++
class Parent.nameImpl
{
  public:

};
```

Using content assist in the new context you are now able to use links on resources of type Child.

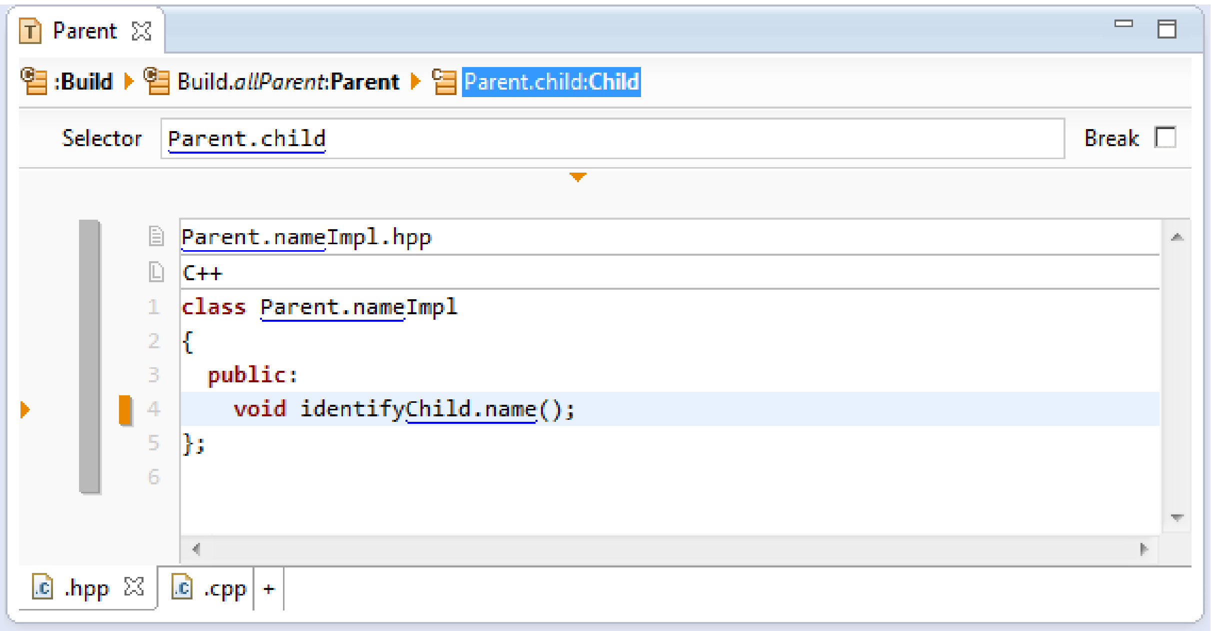


Selector: Parent.child

```
Parent.nameImpl.hpp
C++
class Parent.nameImpl
{
  public:

};
```

```
Child   com.actifsource.templateeditor.generic
Parent  com.actifsource.templateeditor.generic
Build   ch.actifsource.template
Model   ch.actifsource.core.selector
Press Ctrl+Space to show Insert Type Name Proposals
```

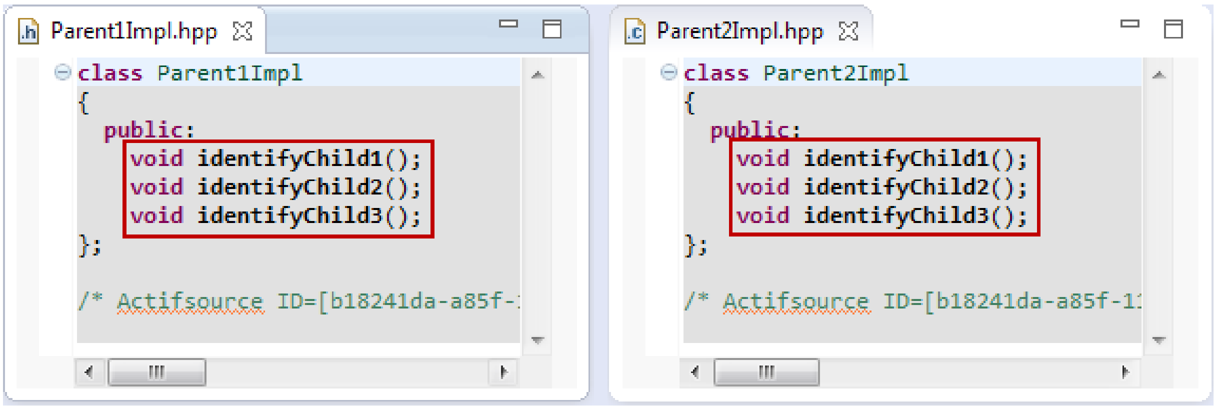
To complete the task from above insert a function named identifyChild.name().



Selector: Parent.child

```
  Parent.nameImpl.hpp
  C++
1 class Parent.nameImpl
2 {
3   public:
4     void identifyChild.name();
5 };
6
```

Line 4 is now repeated for any resource of type Child reached by the relation Parent.child.



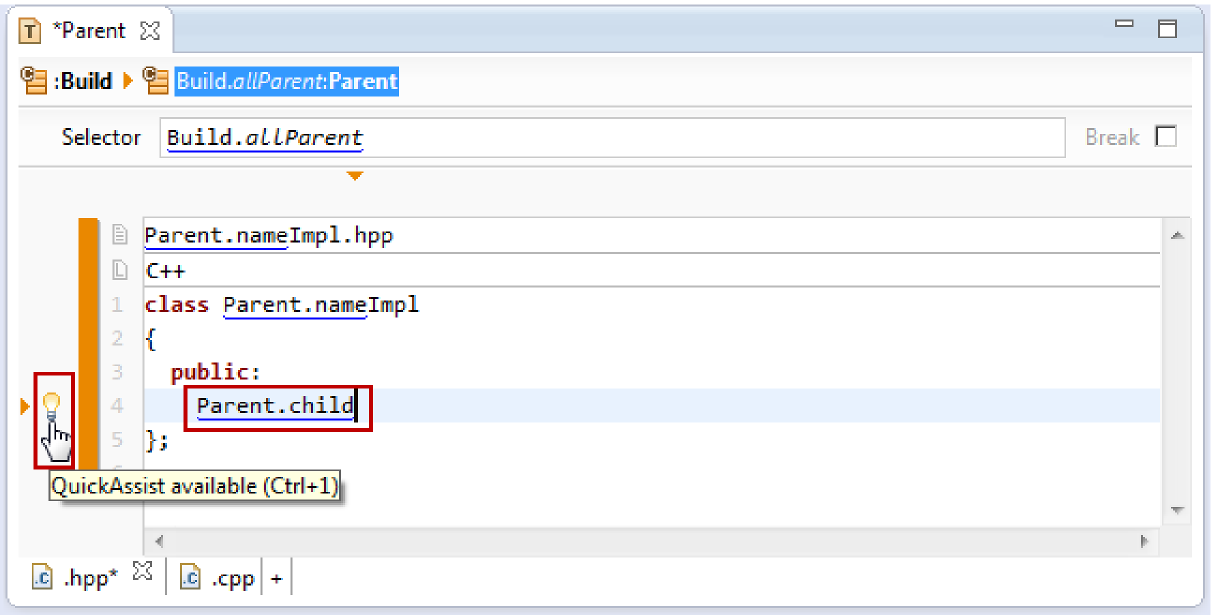
```
class Parent1Impl
{
  public:
    void identifyChild1();
    void identifyChild2();
    void identifyChild3();
};

/* Actifsource ID=[b18241da-a85f-
```

```
class Parent2Impl
{
  public:
    void identifyChild1();
    void identifyChild2();
    void identifyChild3();
};

/* Actifsource ID=[b18241da-a85f-1
```

Add Context via Quick-Assist

Using the quick assist is the most efficient way to add a new context.

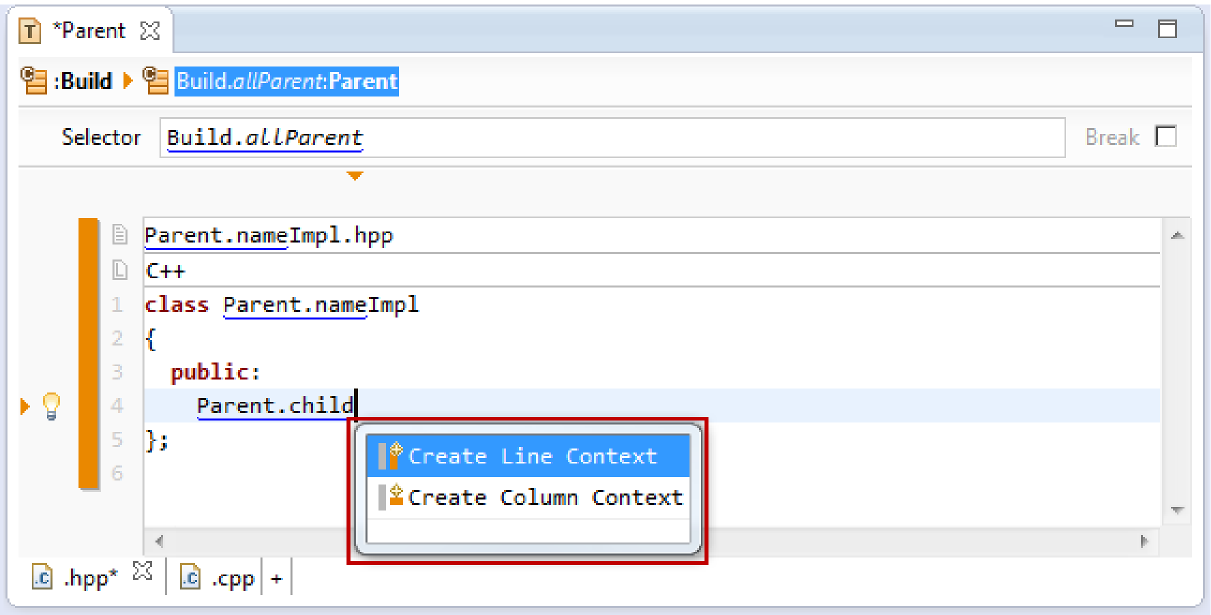
To create a new context with the selector Parent.child just insert the link Parent.child using context assist (Ctrl+Space). A light bulb 💡 indicates that there is a quick assist available. Click on the light bulb or press Ctrl+1 to open the quick assist.



Selector Build.allParent Break □

```
Parent.nameImpl.hpp
C++
class Parent.nameImpl
{
  public:
    Parent.child
};
```

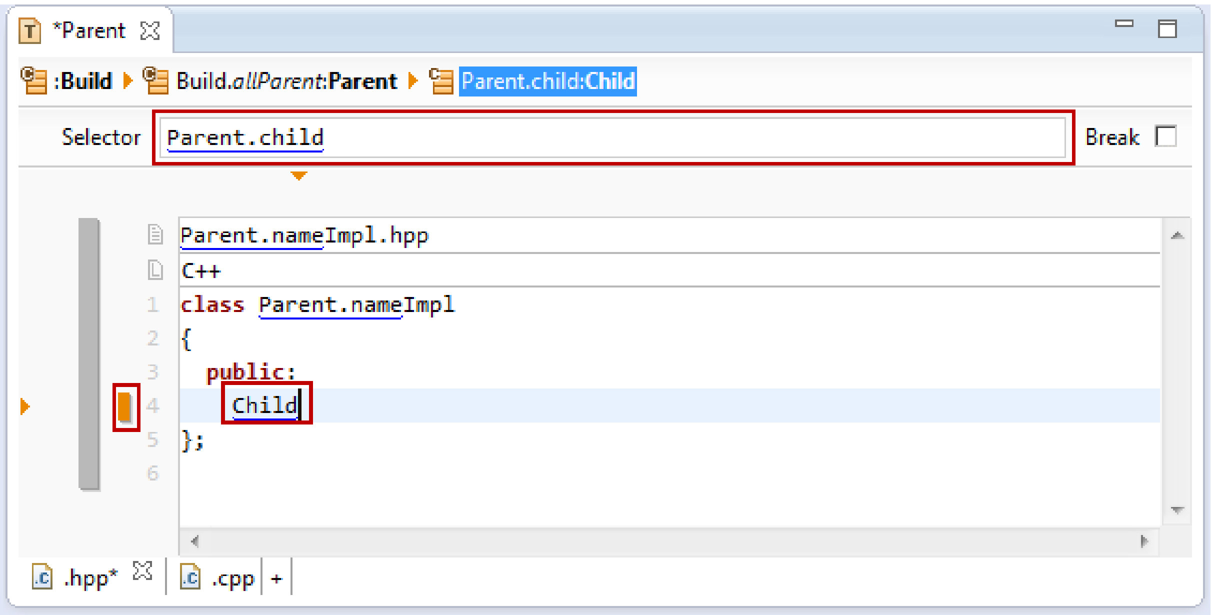
You are now allowed to create a *line context* or a *column context* directly with Parent.child as the selector.



Selector Build.allParent Break □

```
Parent.nameImpl.hpp
C++
class Parent.nameImpl
{
  public:
    Parent.child
};
```

A new context is inserted with the desired selector. Parent.child is automatically replaced by Child which is the result of the selector.



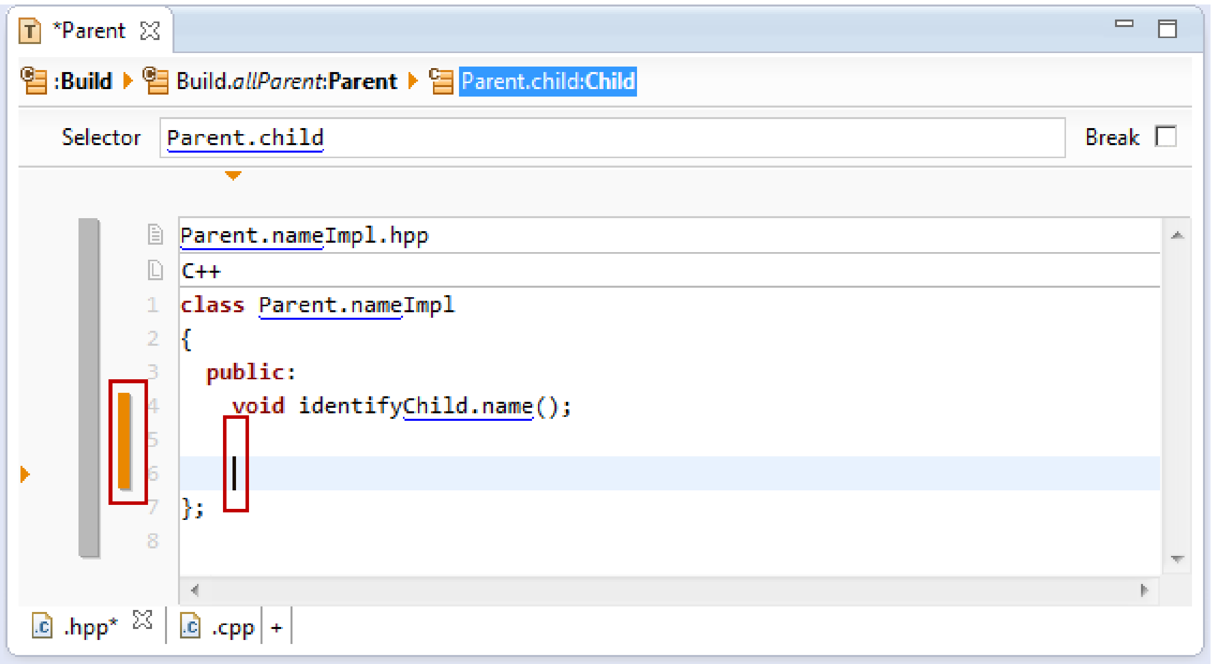
Selector Parent.child

Break □

```
Parent.nameImpl.hpp
C++
class Parent.nameImpl
{
  public:
    Child
};
```

Automatic Context growth

Adding new lines (pressing return) automatically lets the context grow.



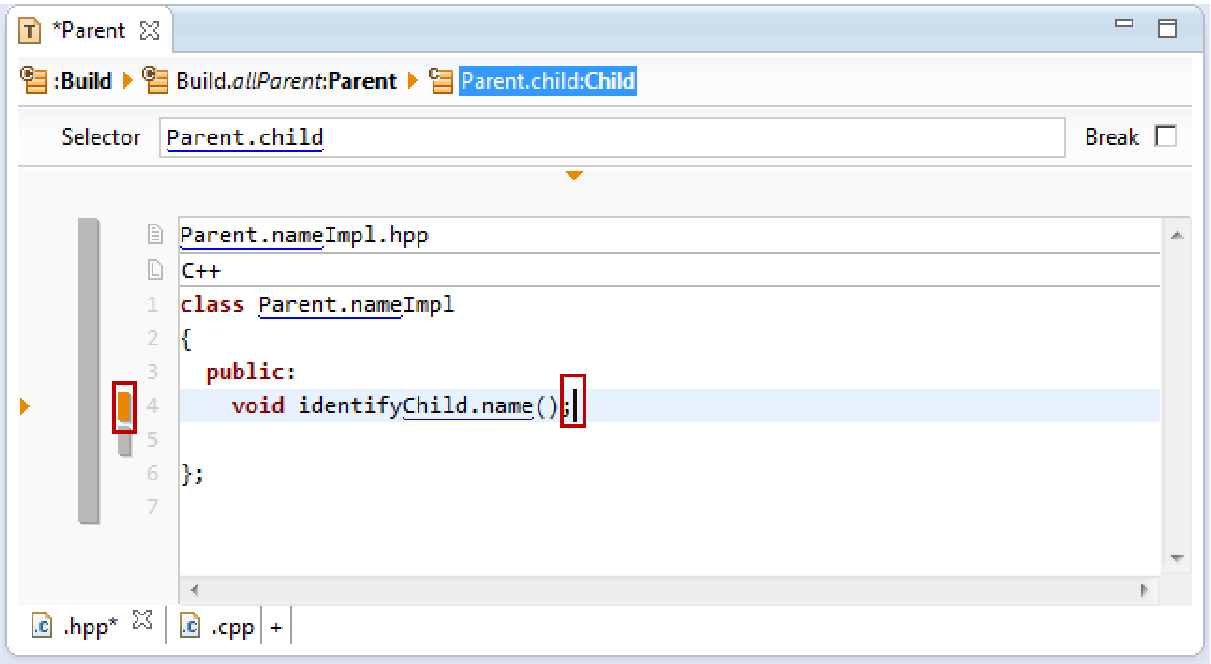
Selector Parent.child

Break □

```
Parent.nameImpl.hpp
C++
class Parent.nameImpl
{
  public:
    void identifyChild.name();


};
```

Add Content between existing Contexts

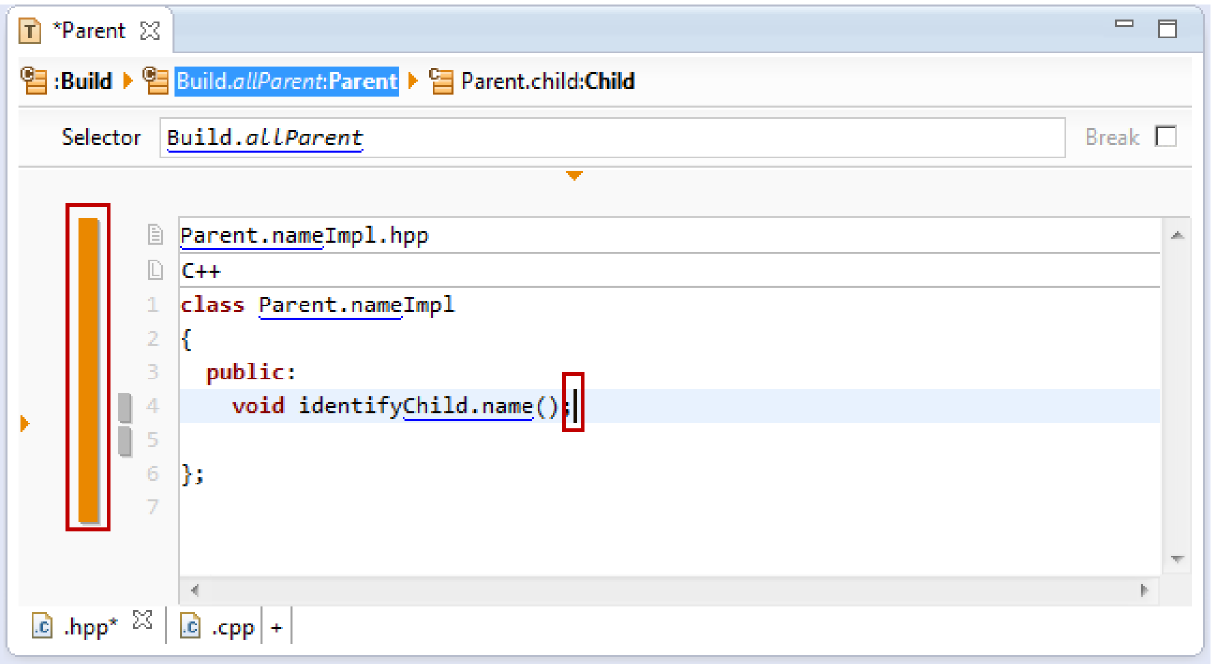
Consider two contexts that follow each other (line 4 and 5 in the following example). How to insert new content between line 4 and line 5 but in the base context?



Selector Parent.child Break □

```
Parent.nameImpl.hpp
C++
1 class Parent.nameImpl
2 {
3   public:
4     void identifyChild.name();
5
6 };
7
```

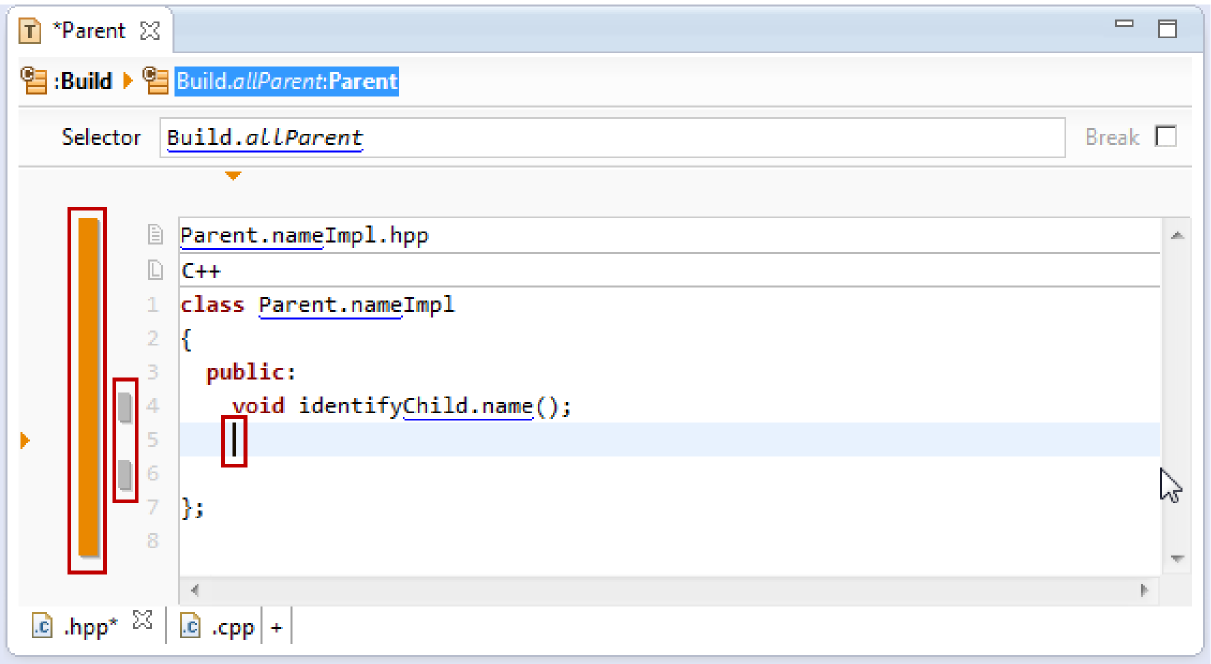
Place the cursor on the end of line 4 as shown above and press cursor right. The cursor will still remain at the same position but the context selection will change to the parent context.



Selector Build.allParent Break □

```
Parent.nameImpl.hpp
C++
1 class Parent.nameImpl
2 {
3   public:
4     void identifyChild.name();
5
6 };
7
```

Entering a new line is done in the selected parent context and results in a new line between the existing contexts.



Selector: Build.allParent

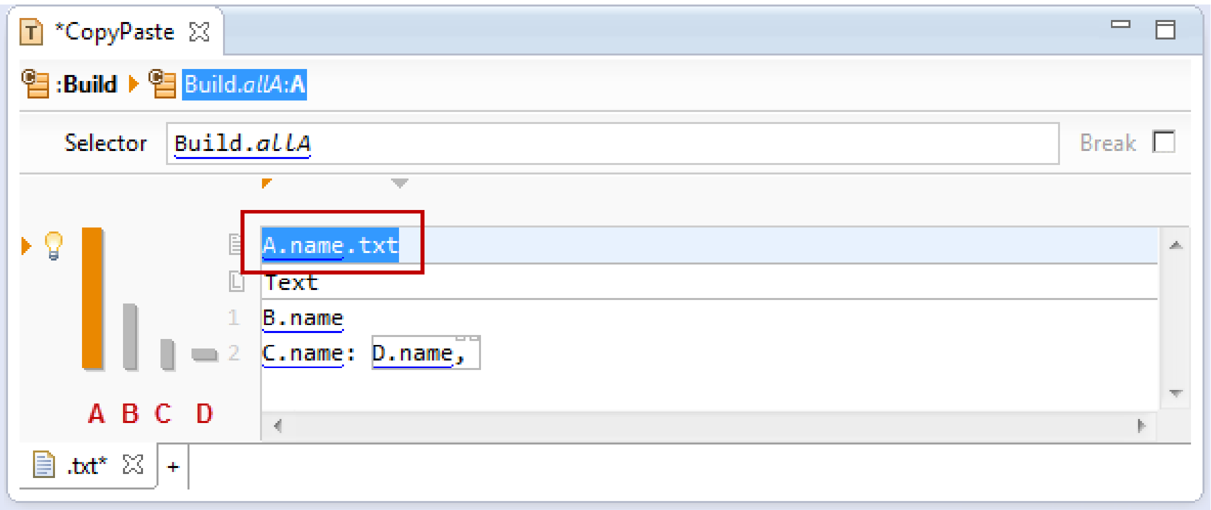
```
Parent.nameImpl.hpp
C++
class Parent.nameImpl
{
  public:
    void identifyChild.name();

};
```

8.3.11 Copy/Paste

Copy (Ctrl+C) and Paste (Ctrl+V) in the Template Editor has some special features to work with links and contexts.

Copy/Paste with Link

Links can be easily copied like text.

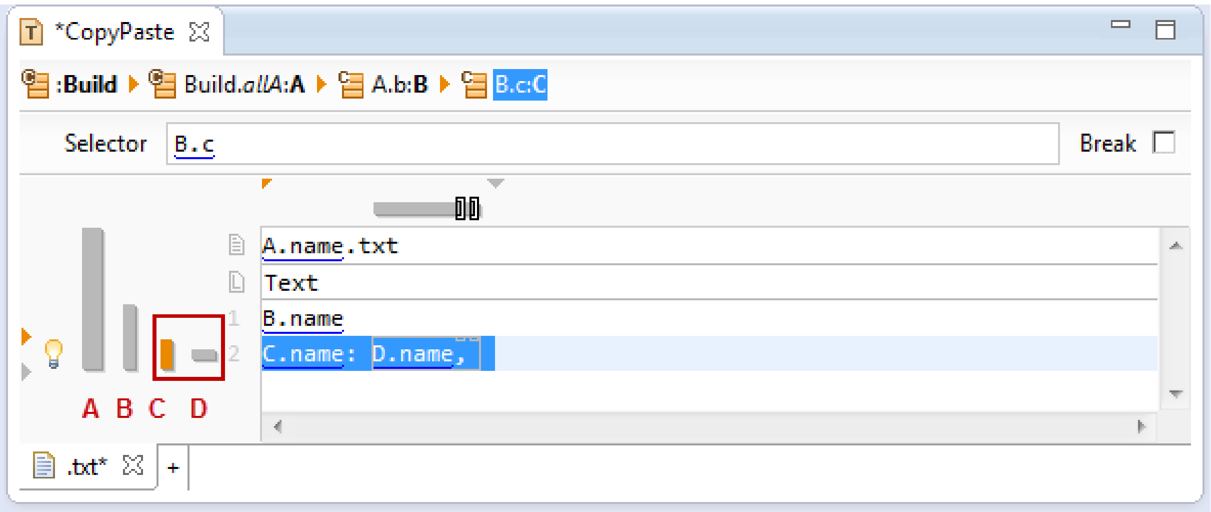


Selector: Build.allA

```
A.name.txt
Text
B.name
C.name: D.name,
```

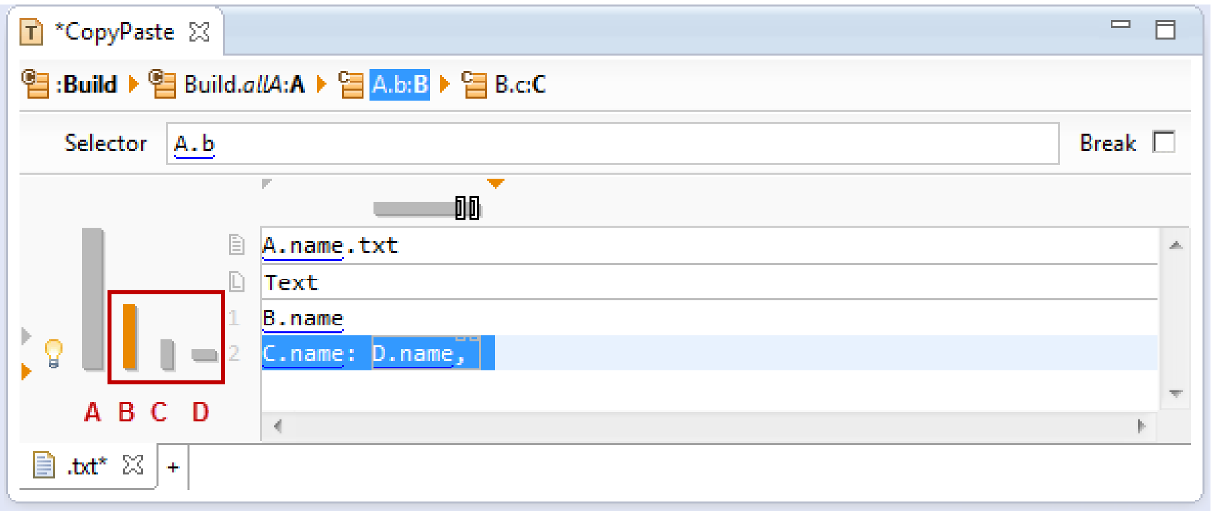
Copy/Paste with Context

The Copy action takes care on all *nested contexts* in the current context. The following situation won't copy the selected *line context* C on line 2 but the nested *column context* D.



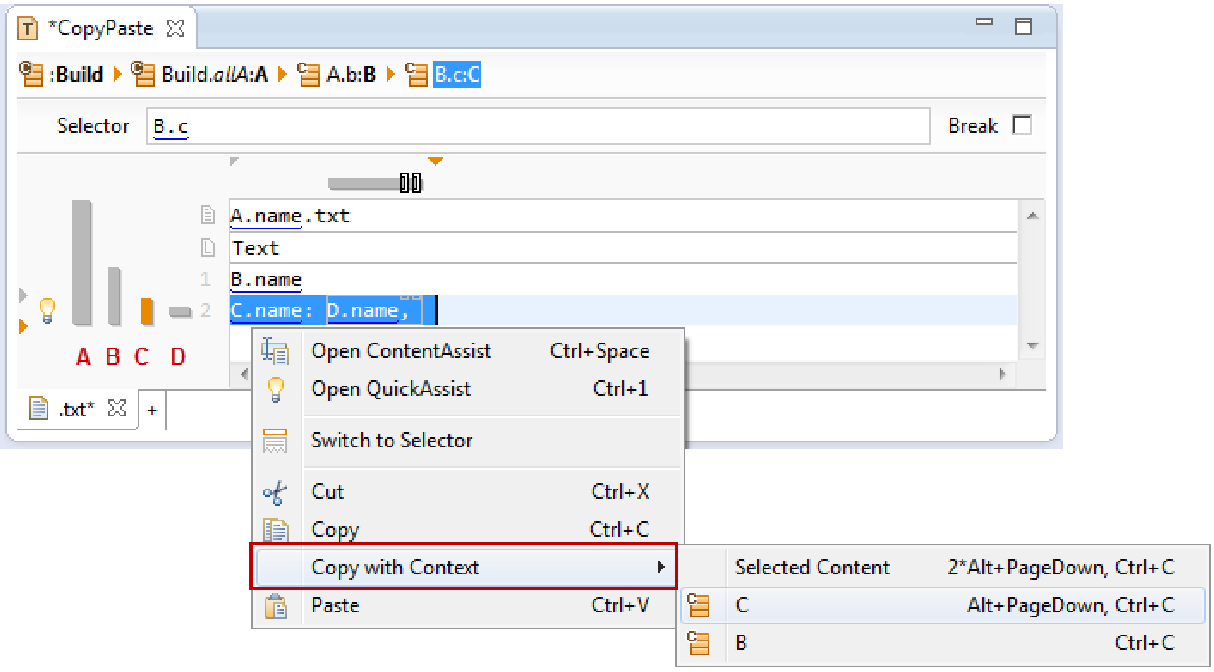
```
A.name.txt
Text
B.name
C.name: D.name,
```

If you have to copy the line context C on line 2 just navigate to the base context (see Chapter 8.3.10 Working with Contexts) what makes the line context a nested context.



```
A.name.txt
Text
B.name
C.name: D.name,
```

There is also a command *Copy with Context* in the context menu which allows to specify the position from which context are copied. The following situation shows a selection on line 2 where the copy operation allows to copy with context C or B.



*CopyPaste

:Build ▸ Build.allA:A ▸ A.b:B ▸ B.c:C

Selector B.c Break □

```
A.name.txt
Text
B.name
C.name: D.name,
```

Please note that it makes no sense to copy the base context (see Chapter 8.3.1 Base Context) or even the super context (see Chapter 8.3.5 SuperContext) because they are part of the whole template.

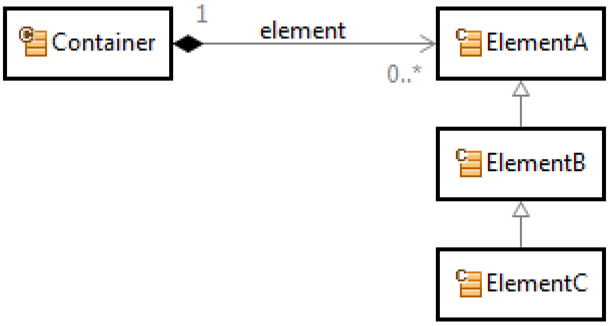
8.3.12 Selector

The selector allows navigating the meta-model and is extremely powerful. Please consult Chapter 9.3.2 SelectorFunction for details.

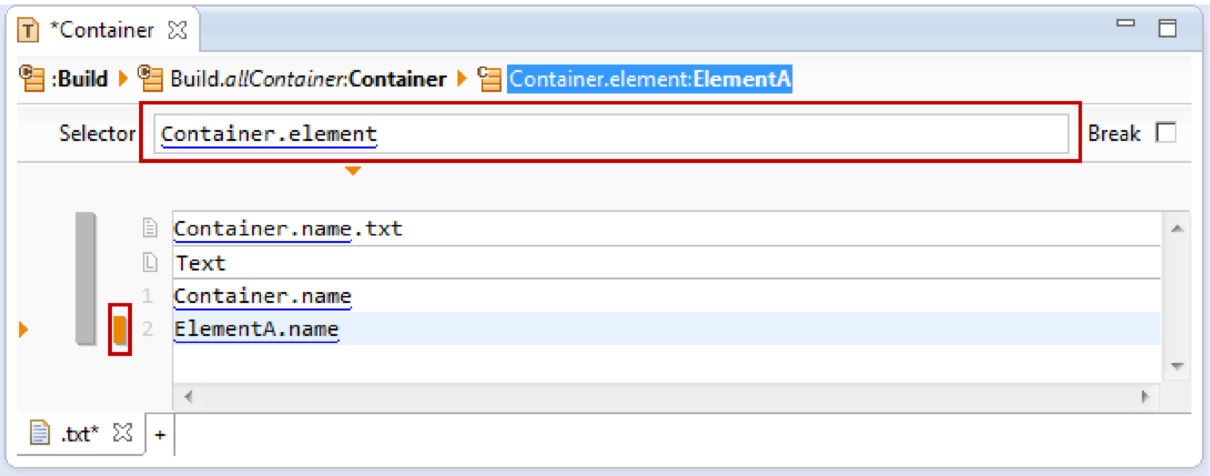
Use the *Switch to Selector* tool from the tool bar or Alt+Enter to navigate from the code to the selector. Use Enter in the selector to jump back to the code.

Break Flag

Consider the following situation:



The subsequent template iterates over Container.element and prints the name of every element.

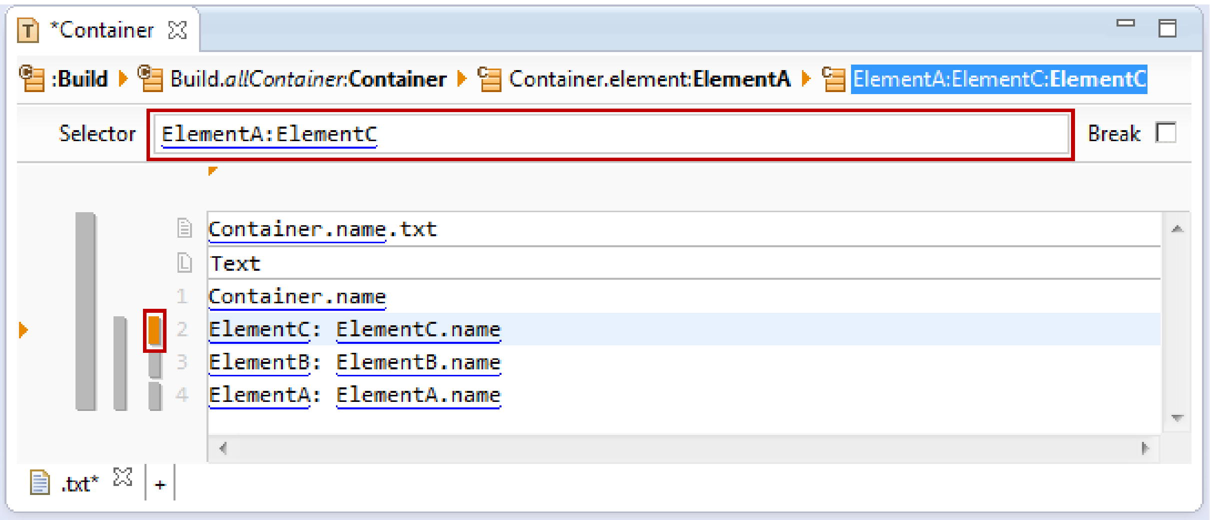


Selector: Container.element

```
Container.name.txt
Text
Container.name
ElementA.name
```

Let's assume that we write template code that shall produce different code depending whether Container.element is of type ElementA, ElementB or ElementC.

A straightforward solution is introducing a context with a type cast for type in the inheritance hierarchy.



*Container
:Build
Build.allContainer:Container
Container.element:ElementA
ElementA:ElementC:ElementC
Selector
Break ☐
.txt*
+

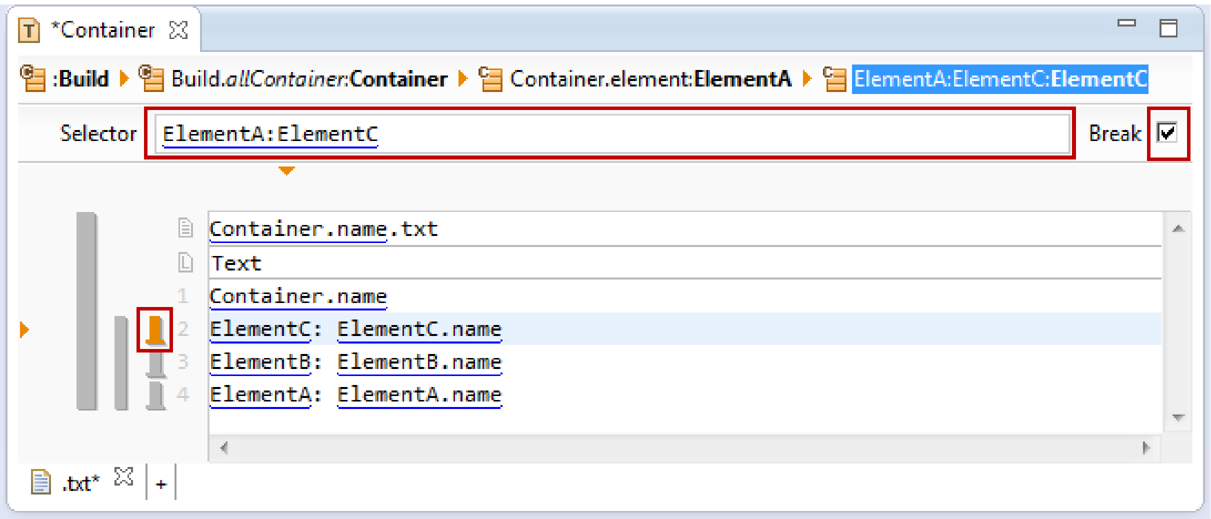
Selector: ElementA:ElementC

```
Container.name.txt
Text
1 Container.name
2 ElementC: ElementC.name
3 ElementB: ElementB.name
4 ElementA: ElementA.name
```

The problem is that a resource type of ElementC is also of type ElementB and ElementA. Therefore the above template prints lines 2, 3 and 4 for resources of type ElementC. But the intention is that only line 2 is printed.

Use the break flag in the selector for the desired behavior. If the break flag is set all subsequent context of the same level are skipped. Users familiar with programming language C or C++ can think of the switch/case/break statement.

The following template prints line 2 for resource types of ElementC and then breaks the current iteration to continue with the next resource for Container.element.


*Container
:Build
Build.allContainer:Container
Container.element:ElementA
ElementA:ElementC:ElementC
.txt*
+

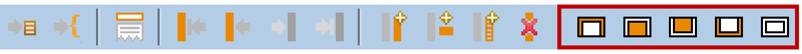
Selector ElementA:ElementC Break ☑

```
Container.name.txt
Text
Container.name
ElementC: ElementC.name
ElementB: ElementB.name
ElementA: ElementA.name
```

Please note that a context with a break flag is displayed with a ground beam.

8.3.13 Line Attributes

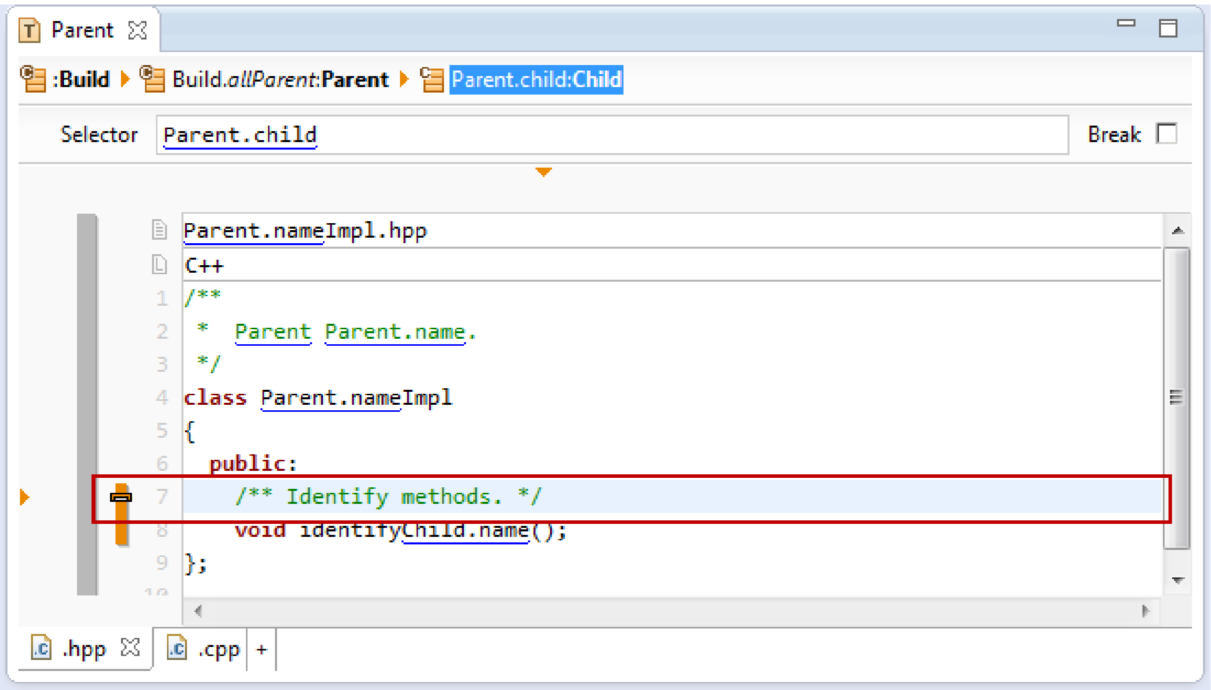
Use *line attributes* on a *line context* to control the output specific positions of a resource in a list. Place the cursor on the desired line to apply a line attribute.



There are five different types of *line attributes* which might be applied to a *line context*.

Context Operation	Icon	Shortcut	Description
First	⬒	Alt+1	The first element of the iteration
Not First	▣	Alt+2	All elements of the iteration except the first
Not Last	▣	Alt+3	All elements of the iteration except the last
Last	⬓	Alt+4	The last element of the iteration
Empty	▭	Alt+5	For empty iterations

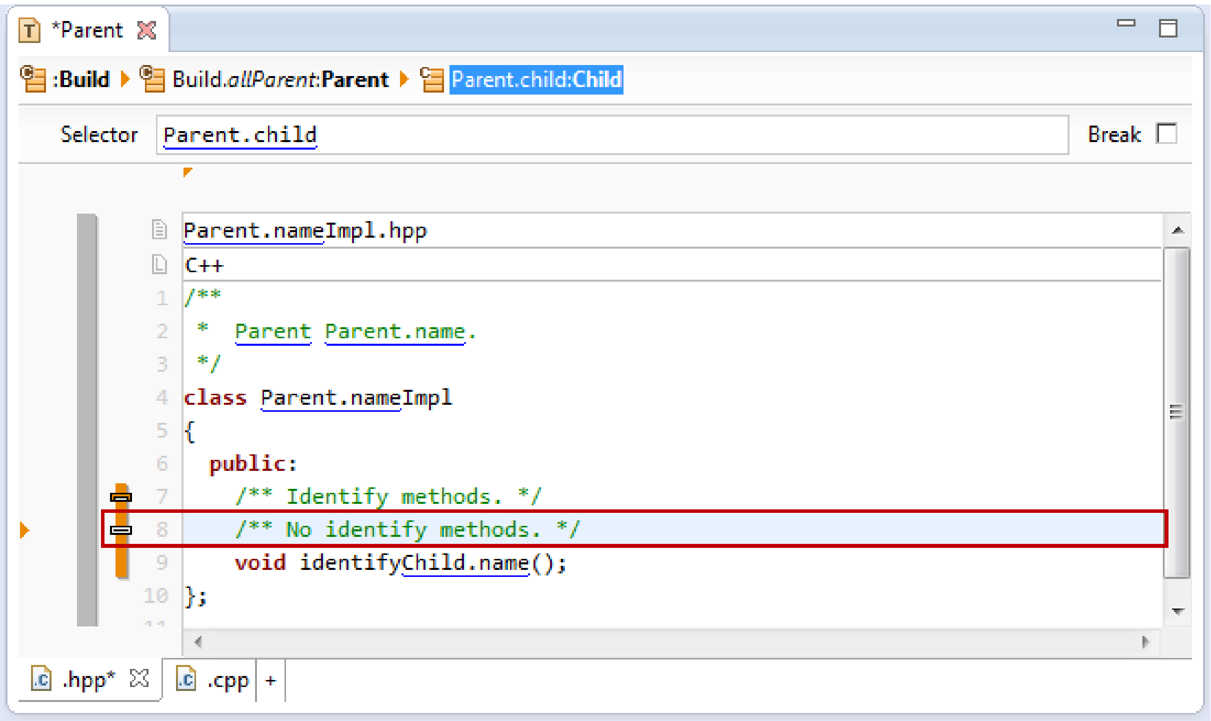
The following template prints the comment on line 7 only for the first element of the iteration over the list Parent.child. Please note that line 7 is not printed if Parent.child is empty.



Selector Parent.child

```
   Parent.nameImpl.hpp
   C++
1  /**
2   *  Parent Parent.name.
3   */
4  class Parent.nameImpl
5  {
6    public:
7      /** Identify methods. */
8      void identifyChild.name();
9  };
```

The following example prints a comment on line 8 if Parent.child is empty.

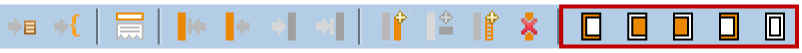


Selector Parent.child

```
    Parent.nameImpl.hpp
    C++
1   /**
2    *  Parent Parent.name.
3    */
4   class Parent.nameImpl
5   {
6     public:
7       /** Identify methods. */
8       /** No identify methods. */
9       void identifyChild.name();
10  };
```

8.3.14 Column Attributes

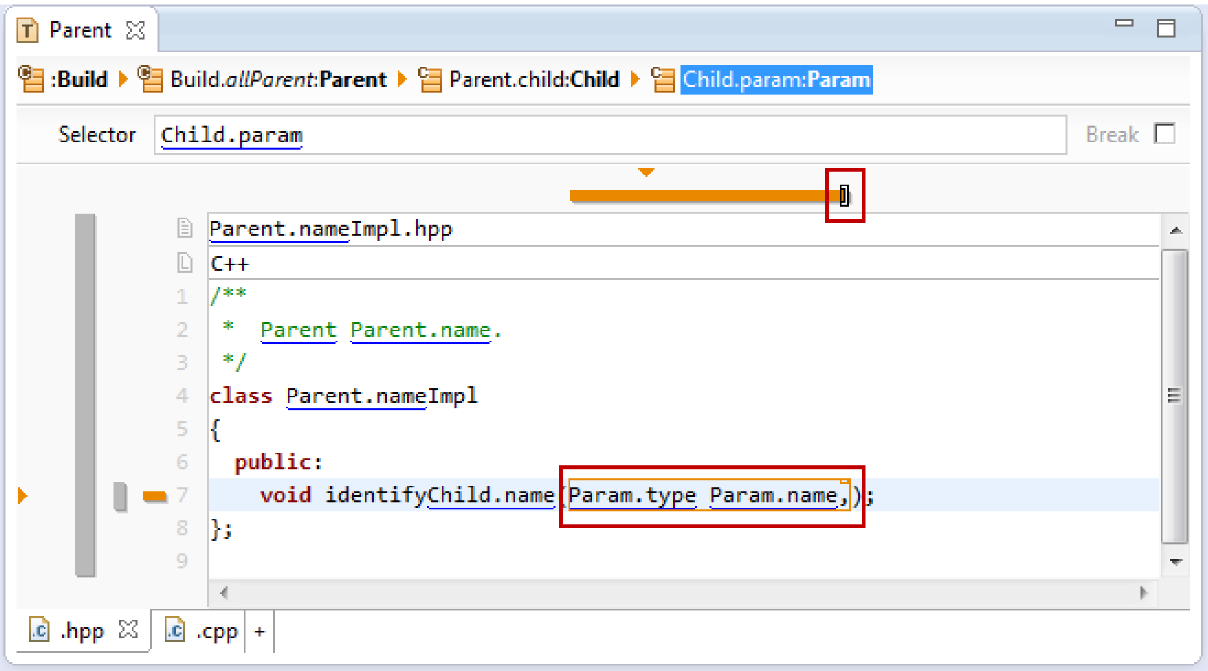
Use *column attributes* on a *column context* to control the output specific positions of a resource in a list. Select the desired characters to apply a line attribute.



There are five different types of *column attributes* which might be applied to a *column context*.

Context Operation	Icon	Shortcut	Description
First	▯	Alt+1	The first element of the iteration
Not First	▮	Alt+2	All elements of the iteration except the first
Not Last	▮	Alt+3	All elements of the iteration except the last
Last	▯	Alt+4	The last element of the iteration
Empty	▯	Alt+5	For empty iterations

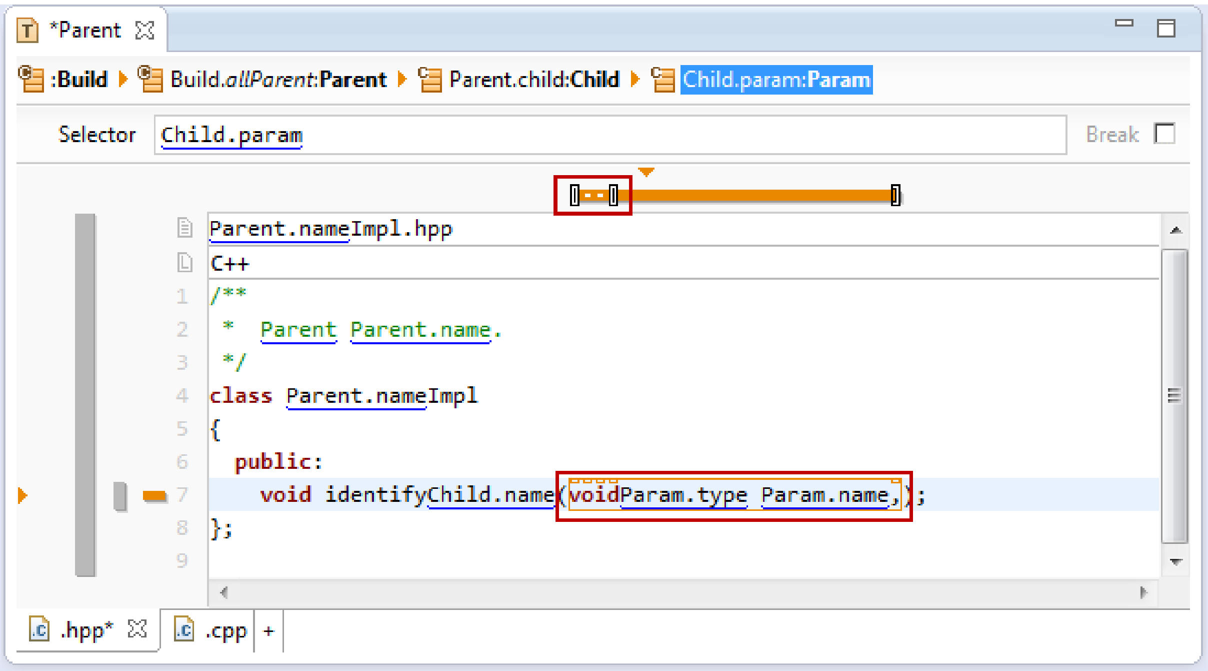
The following example prints the comma after Param.name for all elements of the iteration except the last.


Parent
:Build ▸ Build.allParent:Parent ▸ Parent.child:Child ▸ Child.param:Param
Break ☐
.hpp
.cpp
+

Selector Child.param

```
Parent.nameImpl.hpp
C++
1  /**
2   *  Parent Parent.name.
3   */
4  class Parent.nameImpl
5  {
6    public:
7      void identifyChild.name(Param.type Param.name,);
8  };
9
```

The next example prints void on line 7 if Parent.child is empty.

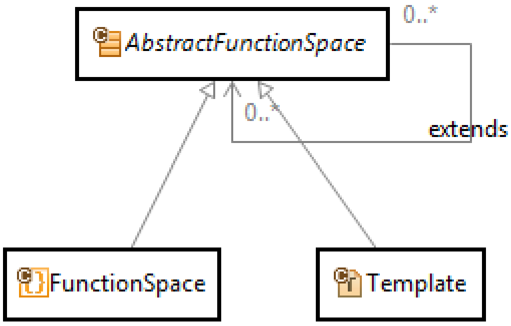

*Parent
:Build ▸ Build.allParent:Parent ▸ Parent.child:Child ▸ Child.param:Param
Break ☐
.hpp*
.cpp
+

Selector Child.param

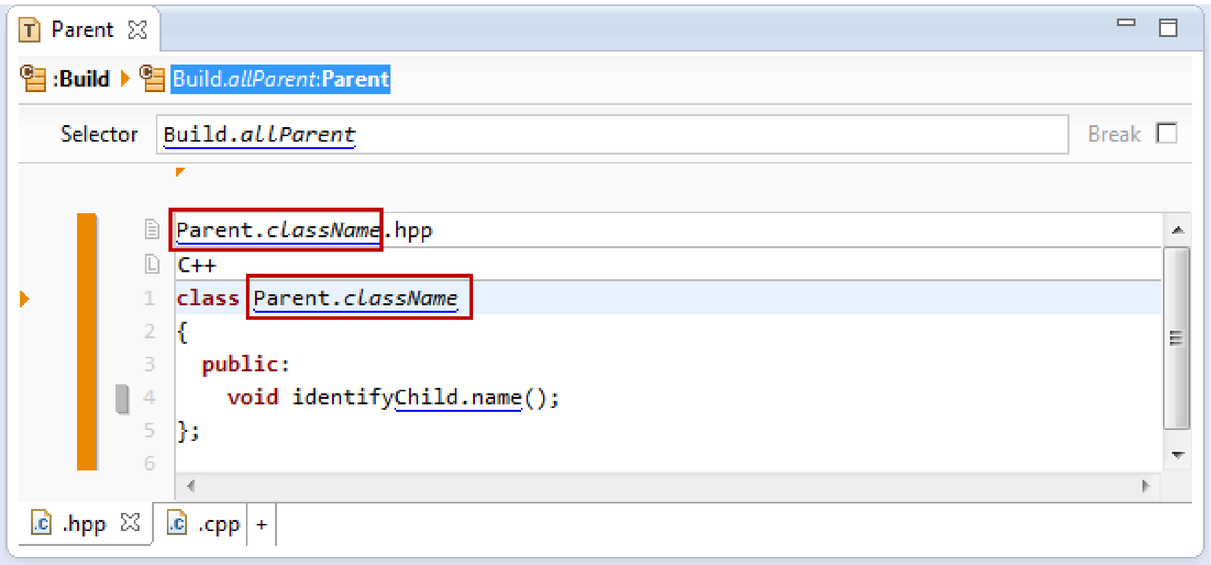
```
Parent.nameImpl.hpp
C++
1  /**
2   *  Parent Parent.name.
3   */
4  class Parent.nameImpl
5  {
6    public:
7      void identifyChild.name(voidParam.type Param.name,);
8  };
9
```

8.3.15 FunctionSpace

As shown in Chapter 9.2 Function Space the *Template* acts as a *Functions Space*. Therefore functions might be placed directly in the template (see Chapter 9 Functions for details).



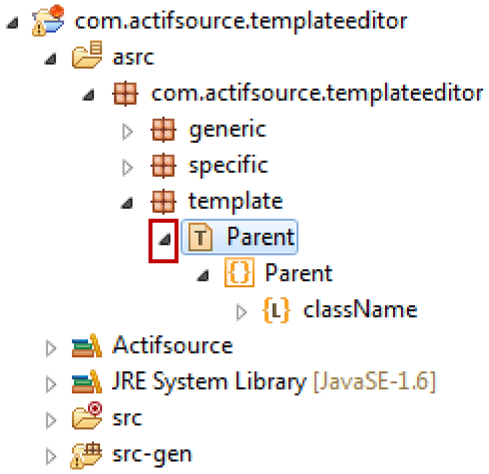
Function calls are displayed in italic. In the subsequent example there is a call to Parent.*className* where *className* is the function.



```
Build.allParent
```

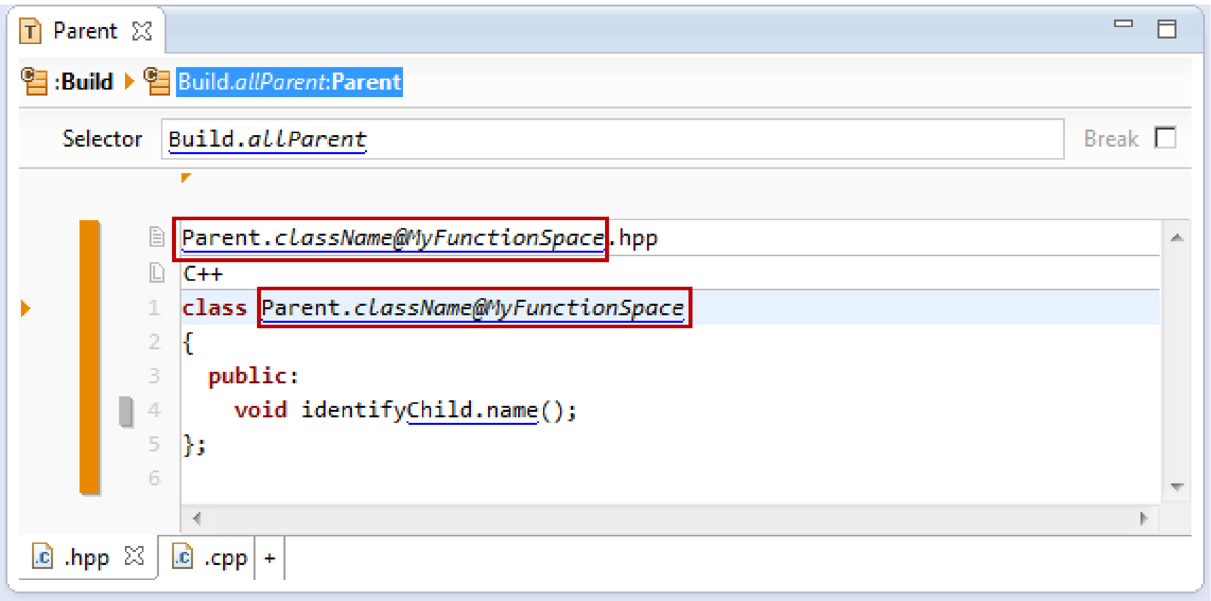
```
Parent.className.hpp
C++
class Parent.className
{
  public:
    void identifyChild.name();
};
```

To see the model of a function within a template just open the folding ▷ on the template.



```
◢ com.actifsource.templateeditor
  ◢ asrc
    ◢ com.actifsource.templateeditor
      ▷ generic
      ▷ specific
      ◢ template
        ◢ Parent
          ◢ Parent
            ▷ className
  ▷ Actifsource
  ▷ JRE System Library [JavaSE-1.6]
  ▷ src
  ▷ src-gen
```

If a function is placed in a *function space* other than the own template the function call is displayed with the name of the function space after the @ sign. Parent.*className@MyFunctionSpace* indicates a function call where the function *className* is located in the function space *MyFunctionSpace*.

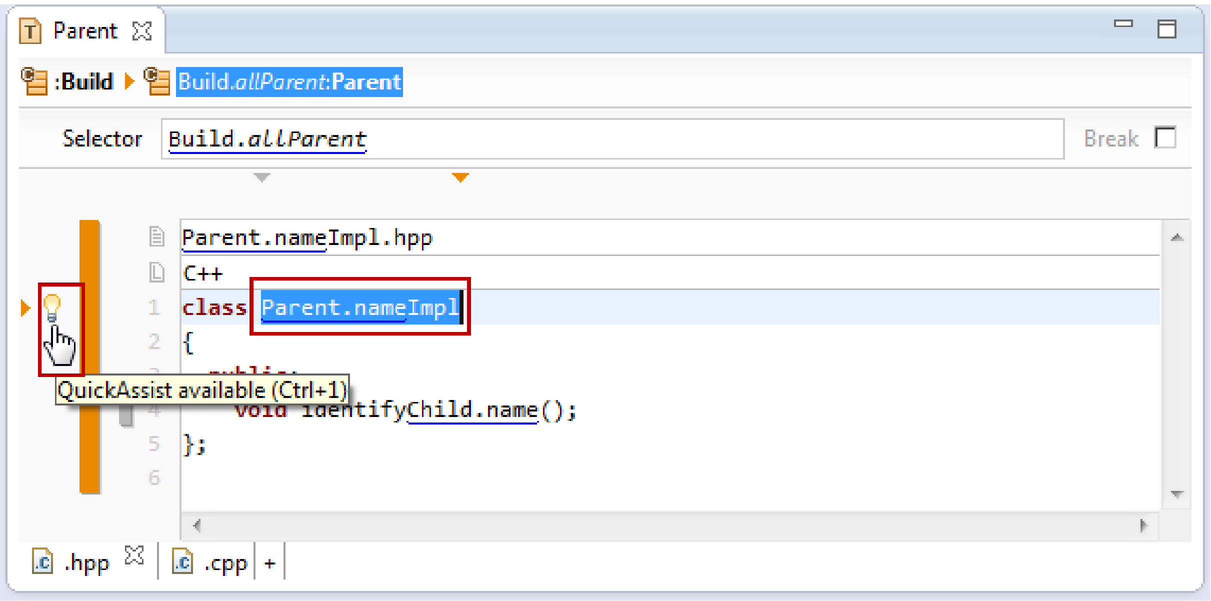


Selector: `Build.allParent`

```
Parent.className@MyFunctionSpace.hpp
C++
class Parent.className@MyFunctionSpace
{
  public:
    void identifyChild.name();
};
```

8.3.16 Extract Function

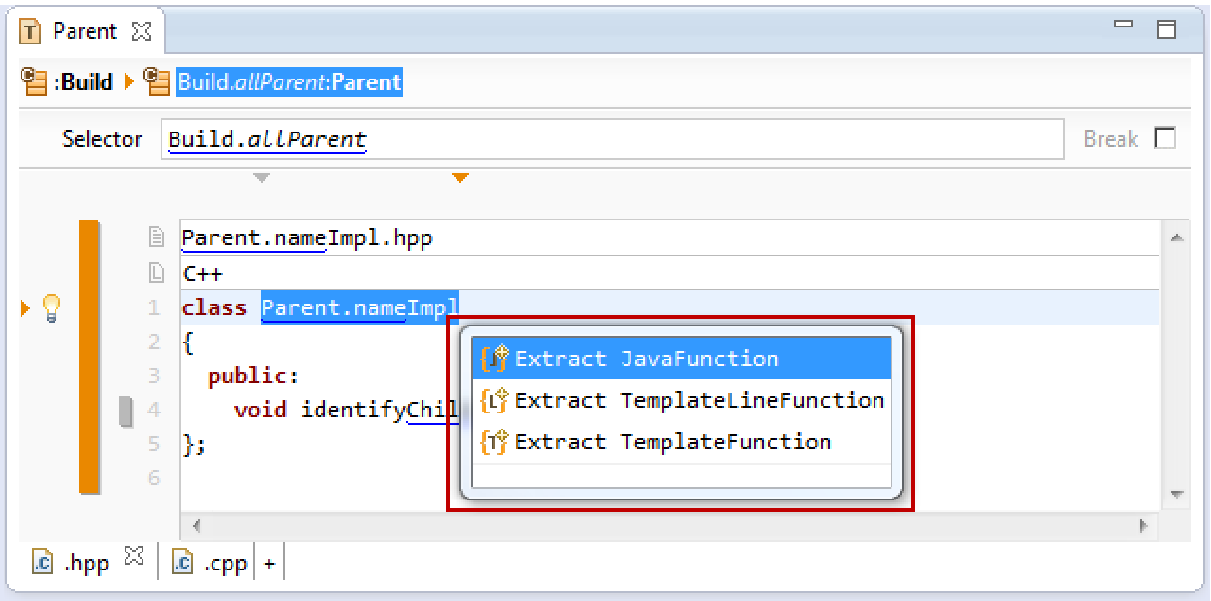
The Actifsource Template Editor allows you to extract selected expressions as functions. Please consider extracting complex expression if you use them more than once.



Selector: `Build.allParent`

```
Parent.nameImpl.hpp
C++
class Parent.nameImpl
{
  public:
    void identifyChild.name();
};
```

Selecting an expression which might also contain links leads to a 💡 light bulb on the left side which indicates that there is a Quick Assist available. Click on the light bulb or press Ctrl+1 to open the Quick Assist.



Parent

:Build ▸ Build.*allParent*:Parent

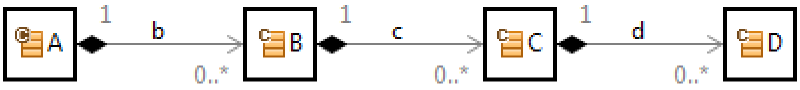
Selector Build.*allParent* Break □

```
Parent.nameImpl.hpp
C++
class Parent.nameImpl
{
  public:
    void identifyChil
};
```

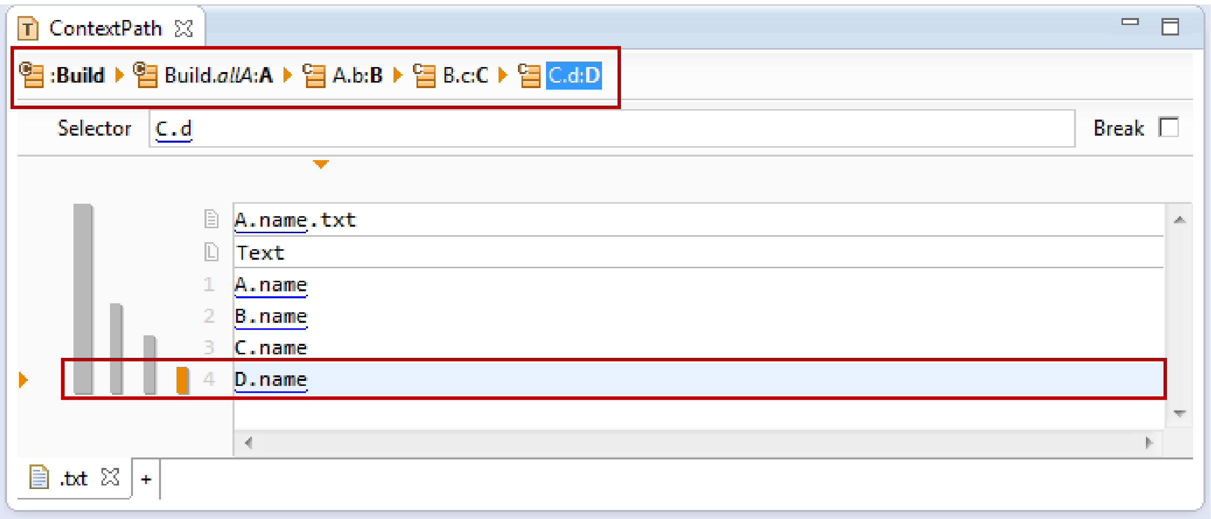
8.3.17 Context Path

The path from the outermost to the innermost context is called *Context Path*. Actifsource uses the *Context Path* to determine the parameters of a function (see Chapter 9.2.1 Function Parameters).

Consider the following meta-model:



The subsequent template shows nested contexts based on the above meta-model. Please note that the breadcrumb displays the context path for the actual cursor position.



ContextPath

:Build ▸ Build.*allA*:A ▸ A.b:B ▸ B.c:C ▸ C.d:D

Selector C.d Break □

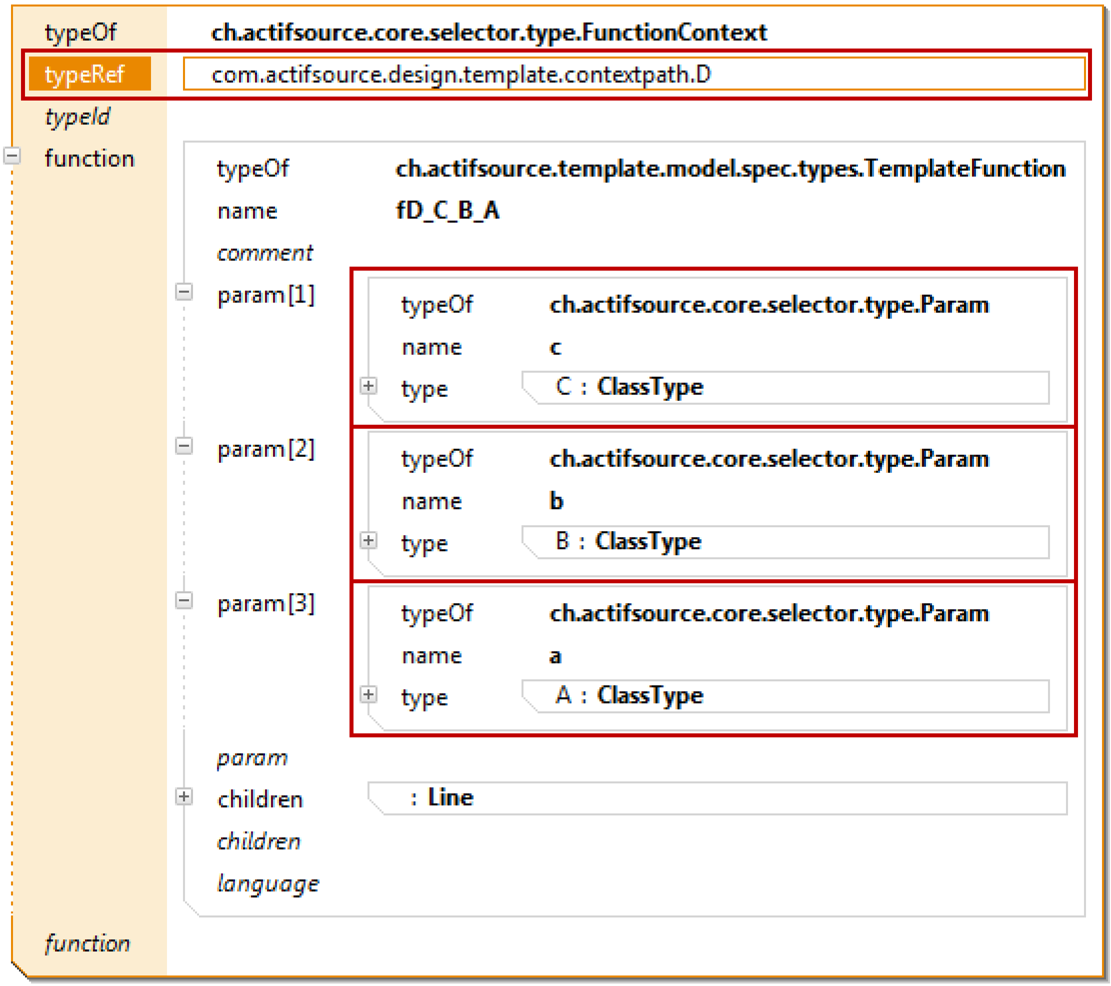
```
A.name.txt
Text
1 A.name
2 B.name
3 C.name
4 D.name
```

The template shows the following context paths.

Line	Context Path
Line 1	A
Line 2	A, B
Line 3	A, B, C
Line 4	A, B, C, D

Calling a function with parameters is only allowed if the context path is matching.

The following function *fD_C_B_A* is based on Class D and declaring the parameters c of type C, b of type B, and a of type A.

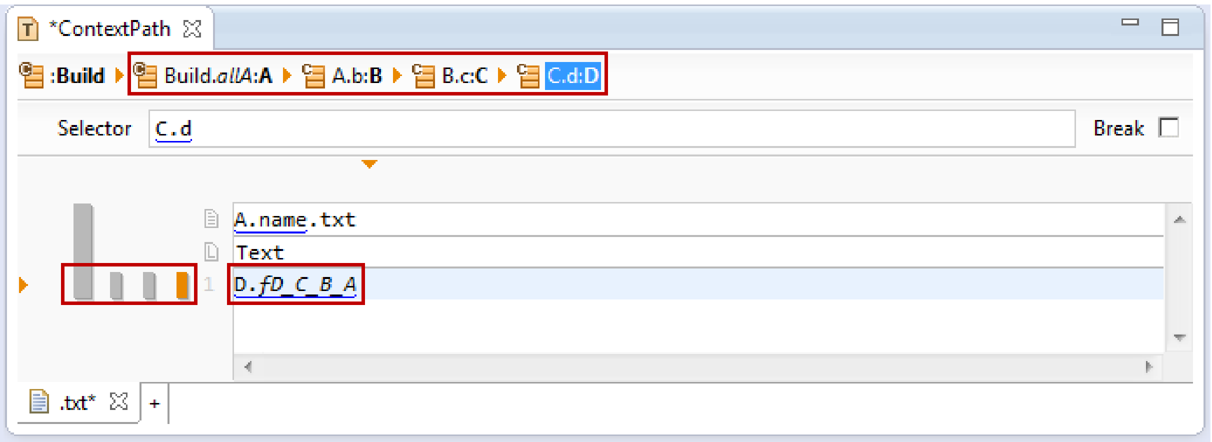


typeOf	ch.actifsource.core.selector.type.FunctionContext
typeRef	com.actifsource.design.template.contextpath.D
typeId	
function	

typeOf	ch.actifsource.template.model.spec.types.TemplateFunction
name	fD_C_B_A
comment	
param[1]	typeOf ch.actifsource.core.selector.type.Param name c type C : ClassType
param[2]	typeOf ch.actifsource.core.selector.type.Param name b type B : ClassType
param[3]	typeOf ch.actifsource.core.selector.type.Param name a type A : ClassType
param	
children	: Line
children	
language	

function

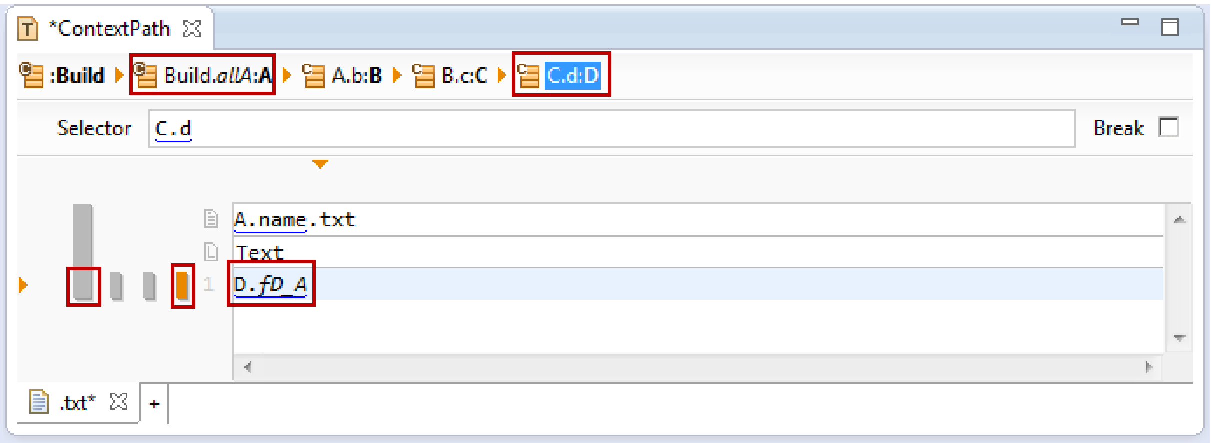
Calling *fD_C_B_A* is only allowed if the context path contains at least A, B, C, and D in the given order.



Selector: C.d

```
A.name.txt
Text
D.fD_C_B_A
```

Consider a function *fD_A* which is based on Class D and declaring a parameter a of type A. It is allowed to call this function on the context path A, B, C, and *D* because it contains *A* and *D* in the correct order.



Selector: C.d

```
A.name.txt
Text
D.fD_A
```

8.4 Declaring a Programming Language

The Actifsource Template editor does syntax highlighting for keywords, comments, and strings. The actual selected language is determined by the Language Line (see Chapter 8.3.3 Language Line).

8.4.1 Supported Programming Languages

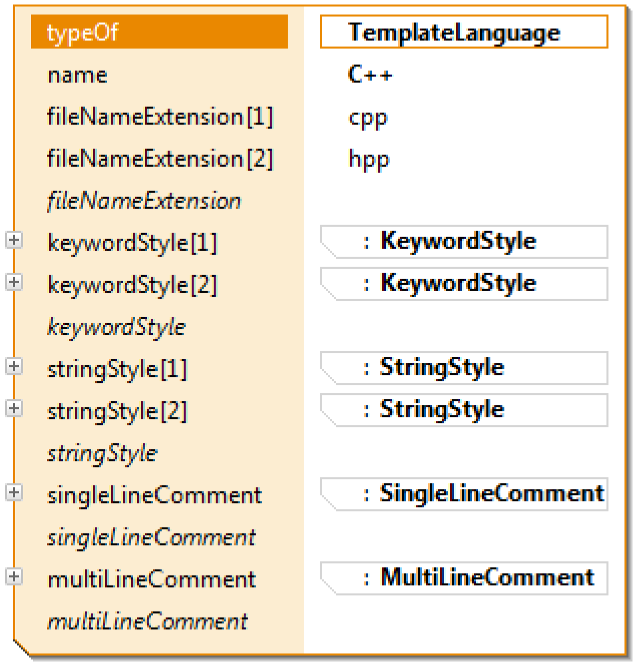
Currently Actifsource supports syntax highlighting for the following programming, script, markup, or domain languages.

Programming Language	Description	File Name Extension
Ada		ada
C		c, h
C#	Microsoft C Sharp	cs
C++		cpp, hpp
Cobol		cob
Css	Cascading Style Sheet	css
D		d
Delphi		dfm
Eiffel		e
Erlang		erl, hrl
GraphViz	http://www.graphviz.org/	dot
Groovy		groovy
Haskell		hs
Html	Hyper Text Markup Language	html, xtml
Java		java
JavaScript		js
Modula2		mod
Oberon		pas
OmgIdl	http://www.omg.org/gettingstarted/omg_idl.htm	idl
Pascal		pas
Perl		pl
Php		php
Python		py
Ruby		rb
Scala		scala
Sql	Structured Query Language for RDBMS	sql
StructuredText	http://en.wikipedia.org/wiki/Structured_text	st
Svg	Scalable Vector Graphics	svg
Text	Plain text	txt

VisualBasic	Microsoft Visual Basic	vb, vba
Xml	Extensible Markup Language	xml

8.4.2 TemplateLanguage Model

To create your own *template language* model just instantiate the class **TemplateLanguage**.



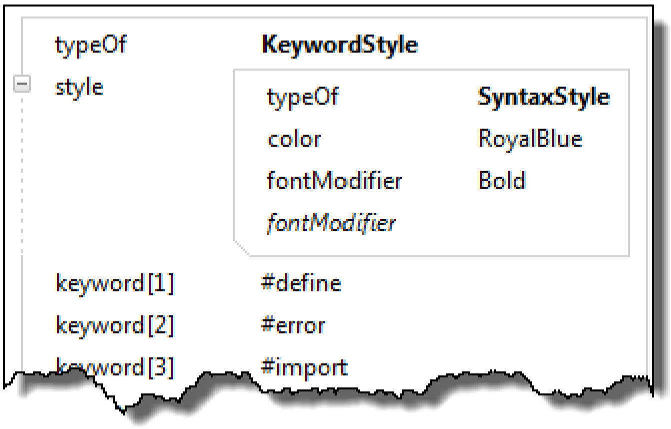
typeOf	**TemplateLanguage**
name	C++
fileNameExtension[1]	cpp
fileNameExtension[2]	hpp
fileNameExtension	
keywordStyle[1]	**: KeywordStyle**
keywordStyle[2]	**: KeywordStyle**
keywordStyle	
stringStyle[1]	**: StringStyle**
stringStyle[2]	**: StringStyle**
stringStyle	
singleLineComment	**: SingleLineComment**
singleLineComment	
multiLineComment	**: MultiLineComment**
multiLineComment	

fileNameExtension

The *file name extension* of the *template language* is used to automatically select the *language line* (see Chapter 8.3.3 Language Line) from the *file extension* in the *file line* (see Chapter 8.3.2 File Line).

keywordStyle

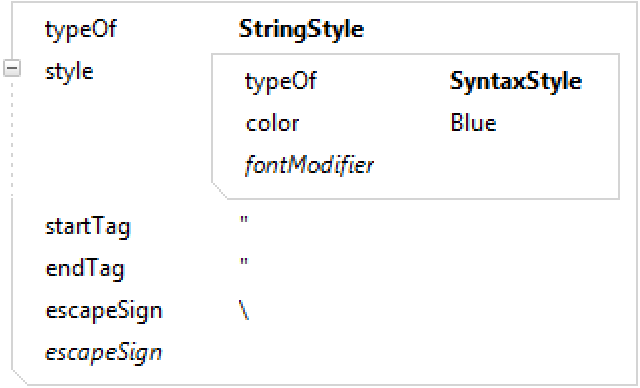
A list of keywords including the syntax style (color, font modifier)



typeOf	**KeywordStyle**
style	typeOf **SyntaxStyle** color RoyalBlue fontModifier Bold *fontModifier*
keyword[1]	#define
keyword[2]	#error
keyword[3]	#import

stringStyle

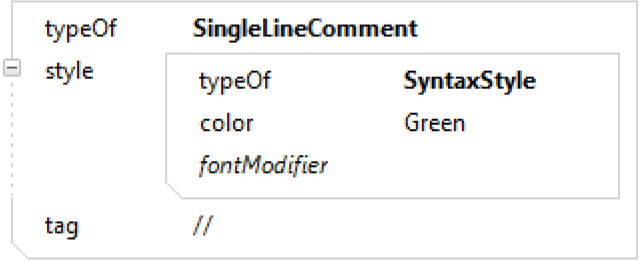
A declaration of start and end tag for strings including the syntax style (color, font modifier)



typeOf	**StringStyle**	
style	typeOf	**SyntaxStyle**
	color	Blue
	fontModifier	
startTag	"	
endTag	"	
escapeSign	\	
escapeSign		

singleLineComment

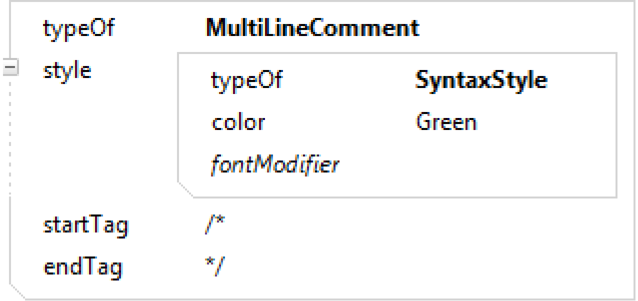
A declaration of the start tag for *single-line comments* including the syntax style (color, font modifier)



typeOf	**SingleLineComment**	
style	typeOf	**SyntaxStyle**
	color	Green
	fontModifier	
tag	//	

mutliLineComment

A declaration of start and end tag for *multi-line comments* including the syntax style (color, font modifier)



typeOf	**MultiLineComment**	
style	typeOf	**SyntaxStyle**
	color	Green
	fontModifier	
startTag	/*	
endTag	*/	

8.4.3 File Extension Priority Rules

You might define your own *template language* defining the same *file name extension* as a built-in language. Actifsource will handle user-defined *template languages* with higher priority so that you can overwrite the standard.

9 Functions

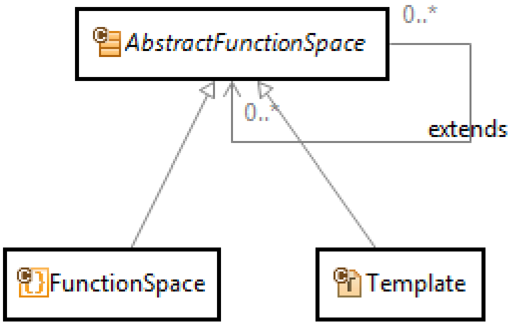
9.1 Overview

Actifsource functions might be called from templates (see Chapter 8 Template Editor), from other functions, or from selector relations (see Chapter 4.4.7 SelectorRelation). There are different supported types of functions (see Chapter 9.3 Function types).

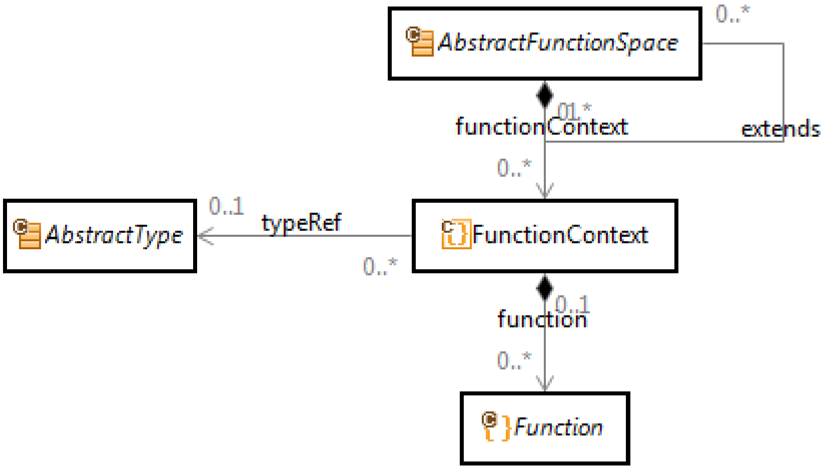
9.2 Function Space

The *function space* is the resource where functions are living. We know two different types of functions spaces, both derived from **AbstractFunctionSpace**.

FunctionSpace is the place where you can place any function. **Template** is a code template where you can place functions in the scope of the template.



Functions are grouped by a **FunctionContext**. By the **typeRef**, the **FunctionContext** is bound to a **Class** or an **Enum**. Functions in the function context are applicable on instances of types referenced by **typeRef**.

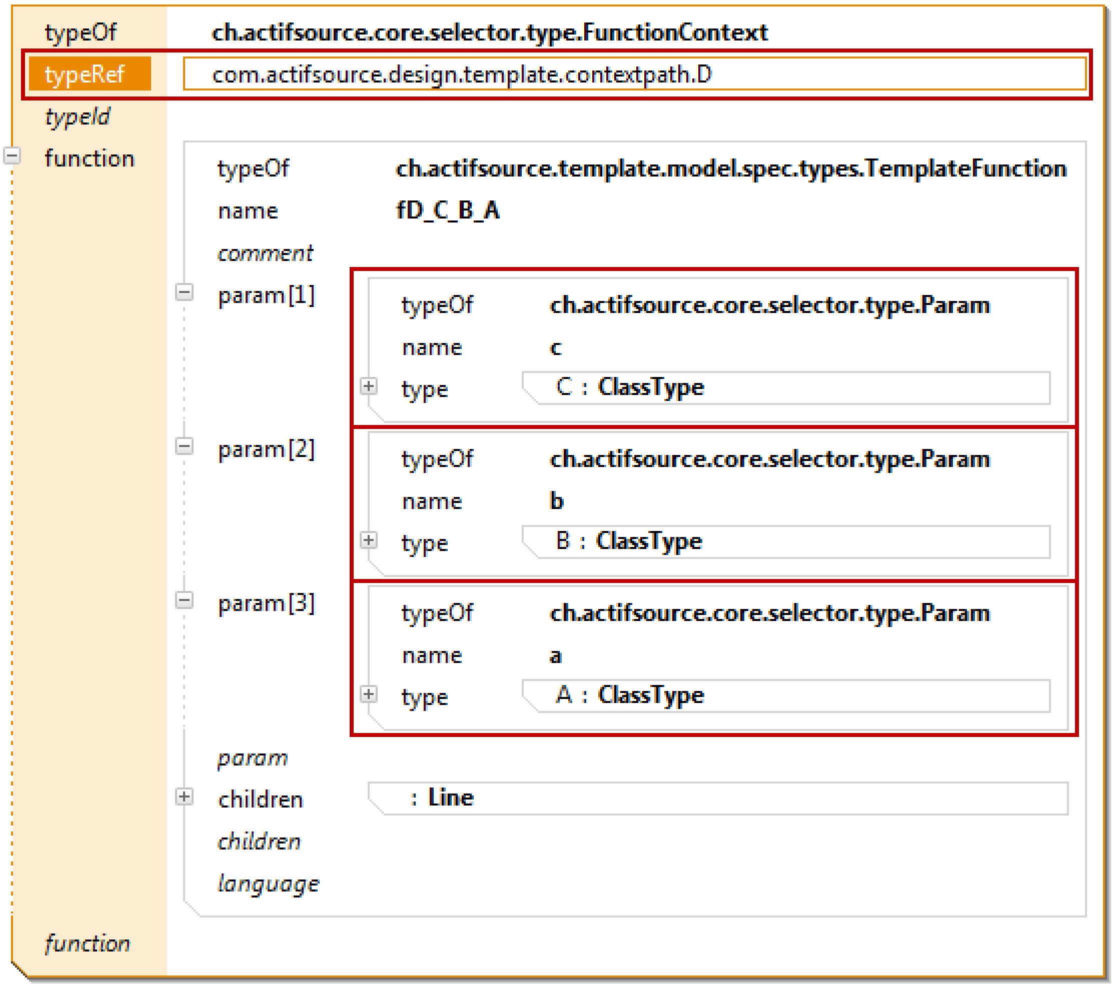


Note that Actifsource prohibits more than one function context with the same **typeRef** in the same function space.

9.2.1 Function Parameters

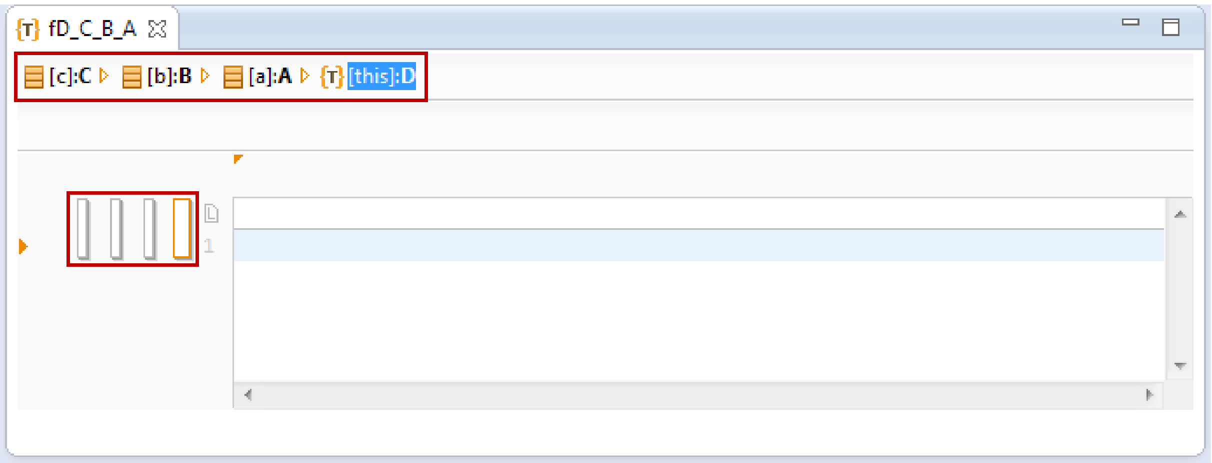
A function might define a set of parameters. If calling a function from a template, Actifsource automatically tries to match with the context path (see Chapter 8.3.17).

Function parameters are defined in the model of the specific function type (see Chapter 9.3 Function types).



typeOf	ch.actifsource.core.selector.type.FunctionContext
typeRef	com.actifsource.design.template.contextpath.D
typeId	
function	typeOf ch.actifsource.template.model.spec.types.TemplateFunction name fD_C_B_A *comment* param[1]: typeOf ch.actifsource.core.selector.type.Param; name c; type C : ClassType param[2]: typeOf ch.actifsource.core.selector.type.Param; name b; type B : ClassType param[3]: typeOf ch.actifsource.core.selector.type.Param; name a; type A : ClassType *param* children : Line *children* *language*
function	

Using function parameters in a template function (see Chapter 9.3.6 TemplateFunction) leads to the corresponding super contexts (see Chapter 8.3.5 SuperContext).



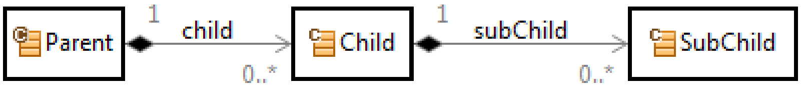
9.2.2 Using function parameters in a java function (see Chapter 9.3.3 SelectorFunction

Selector functions allow you to navigate the model by using the selector syntax. Starting from a Class defined by FunctionContext.typeRef you may navigate via the resource properties.

Selectors might be used in Templates to select a context. But selectors might also be called from within selectors – even recursively.

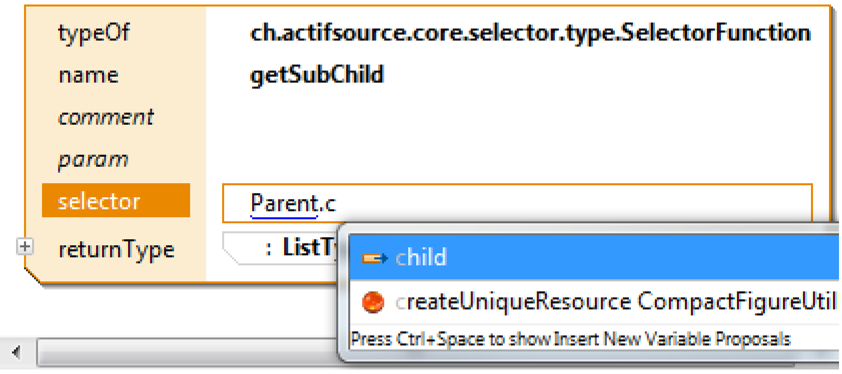
Forward navigation

Consider the following meta-model.



We like to define a selector function named getSubChild on Parent which returns all sub-children in all children of the parent. The return type when navigating along a property is given by the range of the property, i.e, in our example the expected return type is a list of elements of type SubChild.

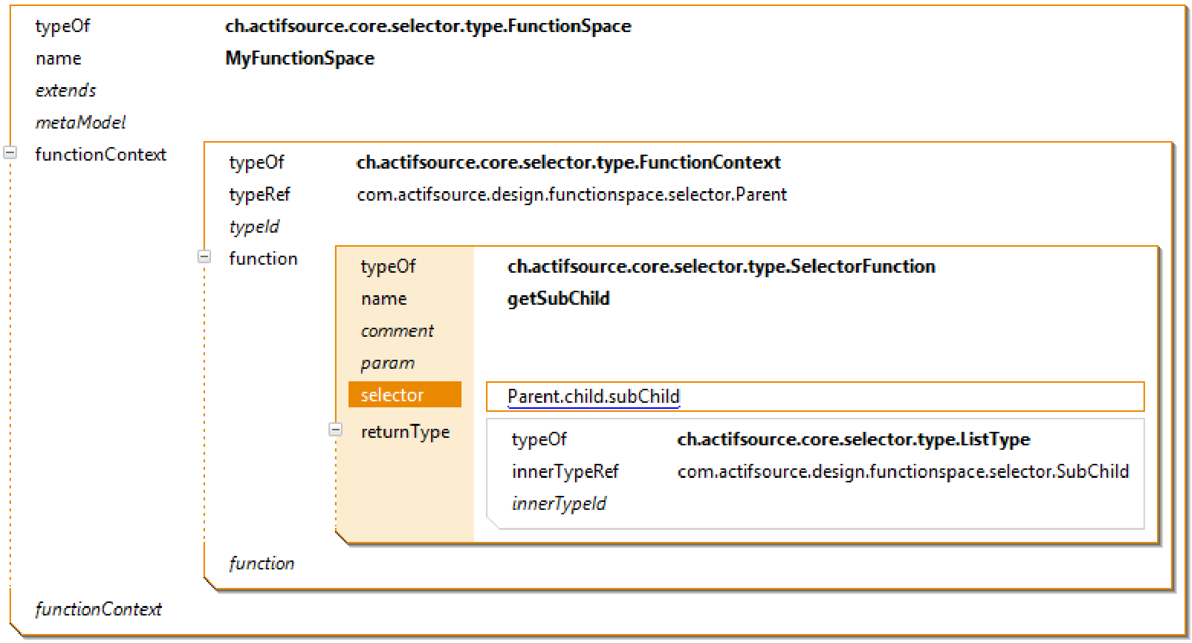
To navigate from Parent via child to subChild just assemble a selector Parent.child.subChild. Make sure to use content assist (Ctrl+Space) when writing selector functions.



typeOf ch.actifsource.core.selector.type.SelectorFunction
name getSubChild
comment
param
selector Parent.c
returnType : ListT

child
createUniqueResource CompactFigureUtil
Press Ctrl+Space to show Insert New Variable Proposals

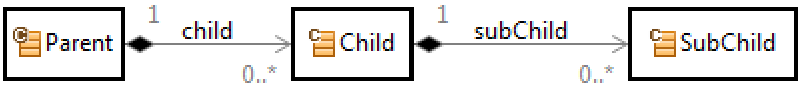
This is what a selector function could look like.



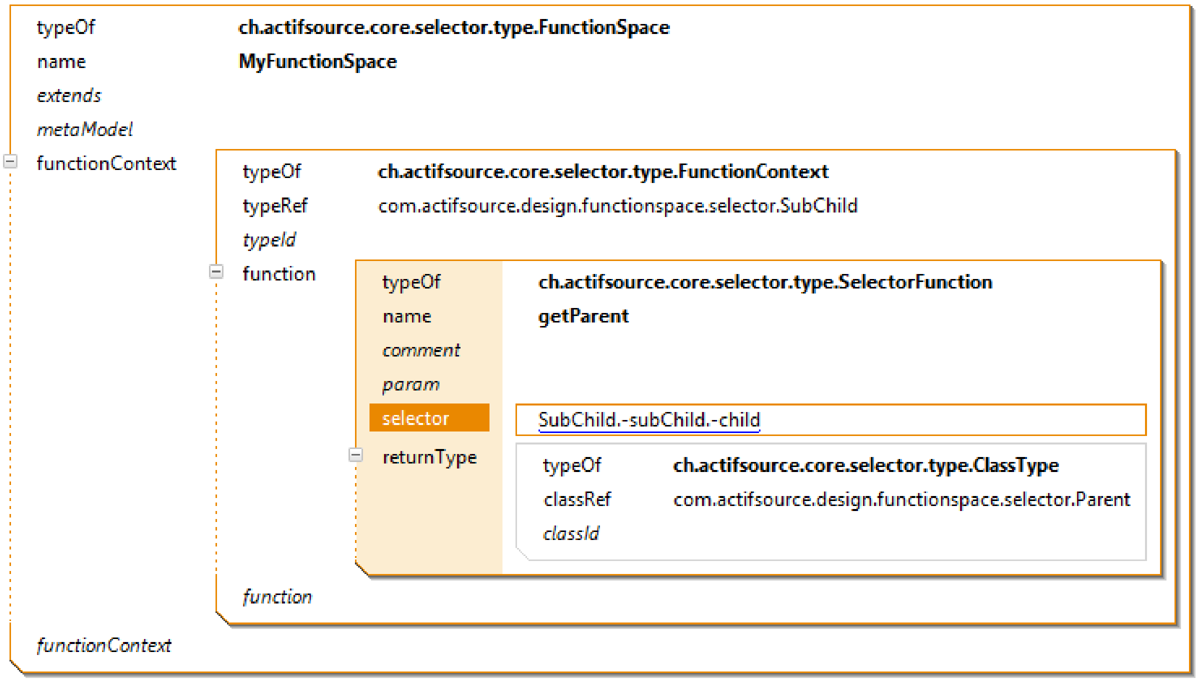
typeOf ch.actifsource.core.selector.type.FunctionSpace
name MyFunctionSpace
extends
metaModel
functionContext
  typeOf ch.actifsource.core.selector.type.FunctionContext
  typeRef com.actifsource.design.functionspace.selector.Parent
  typeId
  function
    typeOf ch.actifsource.core.selector.type.SelectorFunction
    name getSubChild
    comment
    param
    selector Parent.child.subChild
    returnType
      typeOf ch.actifsource.core.selector.type.ListType
      innerTypeRef com.actifsource.design.functionspace.selector.SubChild
      innerTypeId
  function
functionContext

Backward navigation

Consider the following meta-model.



To get the Parent instance for a SubChild instance we have to navigate backwards via the subChild and child relation. The selector allows backward navigation via the *minus relation*.


typeOf ch.actifsource.core.selector.type.FunctionSpace
name MyFunctionSpace
extends
metaModel
functionContext
typeOf ch.actifsource.core.selector.type.FunctionContext
typeRef com.actifsource.design.functionspace.selector.SubChild
typeId
function
typeOf ch.actifsource.core.selector.type.SelectorFunction
name getParent
comment
param
selector SubChild.-subChild.-child
returnType
typeOf ch.actifsource.core.selector.type.ClassType
classRef com.actifsource.design.functionspace.selector.Parent
classId
function
functionContext

List operators

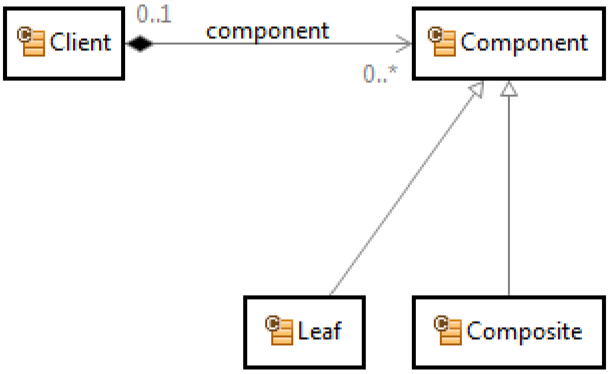
Actifsource provides you with the following operators which are defined on lists (where the lists are given by Selector expressions):

Operator	Description
A union B	The result is the concatenation of the two lists A and B. For example, [a1,a2] union [a3,a1] is equal to [a1,a2,a3,a1].
A intersect B	Only elements found in A and in B where duplicates are preserved and the resulting order is given by A. For example, [a1,a1,a2,a2] intersect [a2,a1,a1] is equal to [a1,a1,a2].
A except B	For all elements b in B, the first occurrence of b in A is removed from A. For example, [a2,a1,a3,a2,a1] except [a1,a2,a1] is equal to [a3,a2]
A else B	All elements in A if a is not empty, otherwise all elements in B. For example, [a1, a2] else [b1,b2] is equal to [a1,a2] and [] else [b1,b2] is equal to [b1,b2].

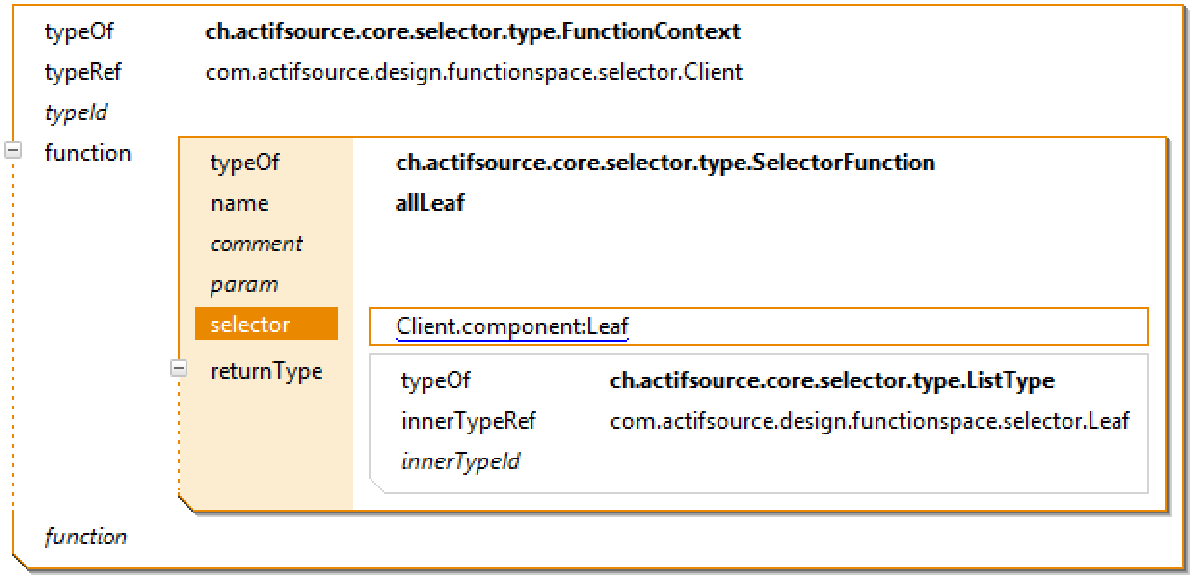
Note that you can use brackets to control precedence (i.e. (*A.x* union *A.y*) intersect *A.z*). The result type of the union, intersect and else operator is the most concrete supertype of the type of the two operands (e.g. if A is of type **NamedResource** and B is of type **Resource**, then A intersect B, A union B and A else B are all of type **Resource**).

Down Cast

Consider the following meta-model.



If you only like to get Leaf components from the Client, just use the type cast operator (colon).

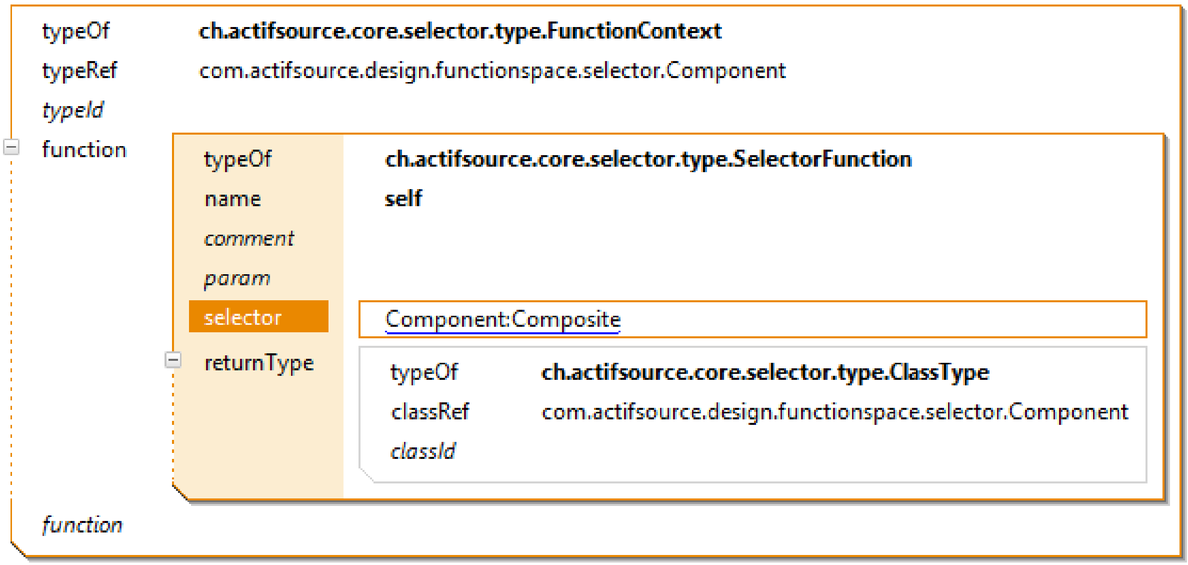

typeOf ch.actifsource.core.selector.type.FunctionContext
typeRef com.actifsource.design.functionspace.selector.Client
typeId
function
typeOf ch.actifsource.core.selector.type.SelectorFunction
name allLeaf
comment
param
selector Client.component:Leaf
returnType
typeOf ch.actifsource.core.selector.type.ListType
innerTypeRef com.actifsource.design.functionspace.selector.Leaf
innerTypeId
function

Up Cast

It is always possible to use an upcast to a base class (i.e. **Resource** or **NamedResource**) if needed.

Self Cast

If your selector has to return the typeRef instance itself, use the self-cast. Consider the following selector function for Component returning the component instance itself.



typeOf	**ch.actifsource.core.selector.type.FunctionContext**
typeRef	com.actifsource.design.functionspace.selector.Component
typeId	
function	

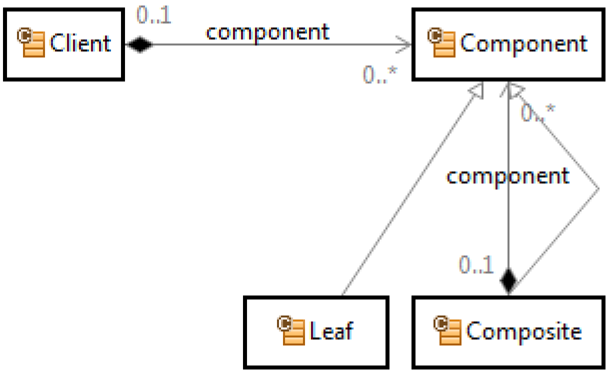
typeOf	**ch.actifsource.core.selector.type.SelectorFunction**
name	**self**
comment	
param	
selector	Component:Composite
returnType	

typeOf	**ch.actifsource.core.selector.type.ClassType**
classRef	com.actifsource.design.functionspace.selector.Component
classId	

function

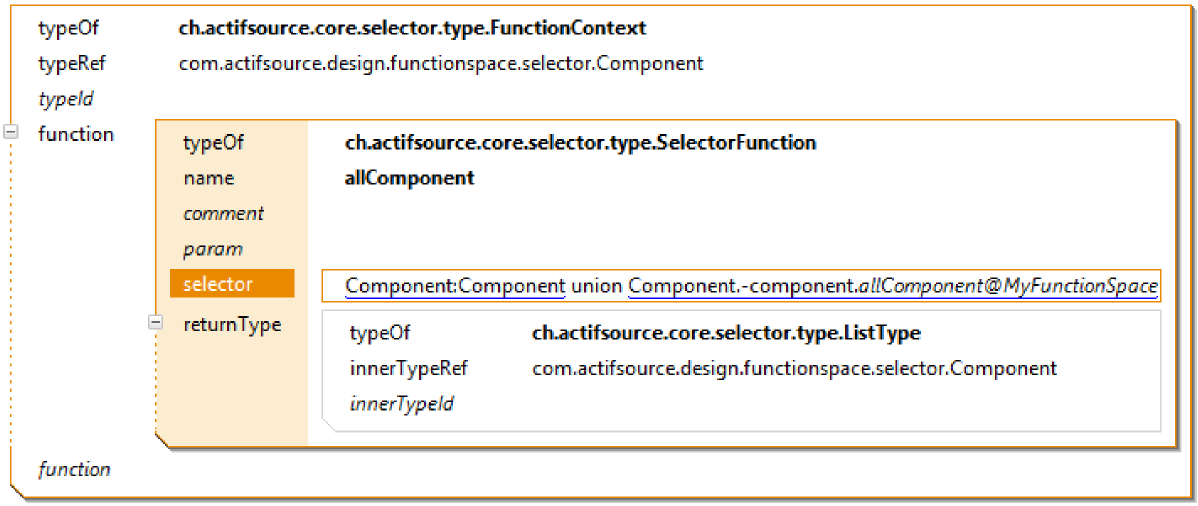
Recursive navigation

The diagram below shows a composite pattern as presented in the book Design Patterns from Erich Gamma et al. The composite pattern allows you to recursively instantiate Composite instances, which might aggregate other components of type Leaf or again – of type Composite.



There is an easy way to find the Client of this recursive model using selectors. First, we collect all components including the own component and all parent components.

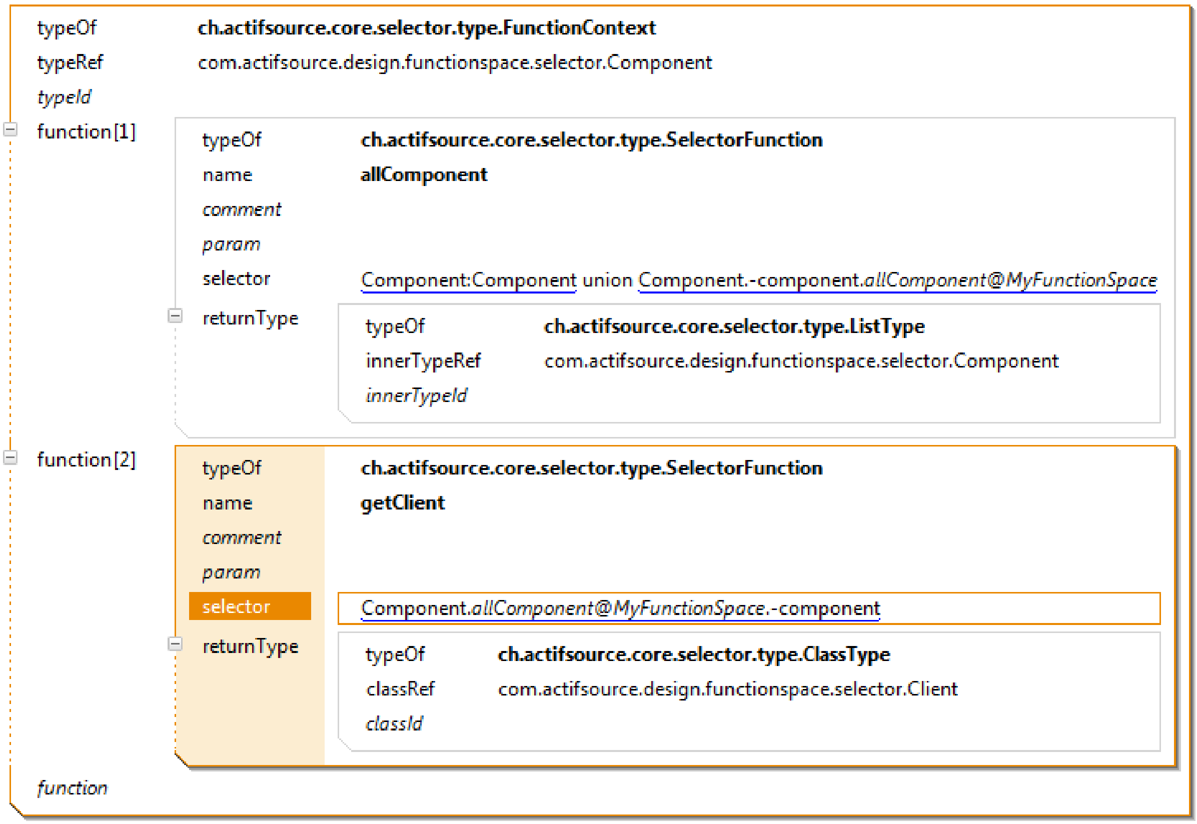
For that reason we write a selector function for Component which returns the component itself and also all parent components which are reachable by going backwards via the component relation.



typeOf ch.actifsource.core.selector.type.FunctionContext
typeRef com.actifsource.design.functionspace.selector.Component
typeId
function
typeOf ch.actifsource.core.selector.type.SelectorFunction
name allComponent
comment
param
selector Component:Component union Component.-component.allComponent@MyFunctionSpace
returnType
typeOf ch.actifsource.core.selector.type.ListType
innerTypeRef com.actifsource.design.functionspace.selector.Component
innerTypeId
function

To get the Client which is parent of all component just collect all component first by using the above selector functions. From all this components in the collected set there is only one instance aggregated by Client. Let's write a selector function for that.

Please take extra care because there are two relations named component. One is Composite.component; the other one is Client.component. Make sure to use Composite.component in allComponent and Client.component in getClient.

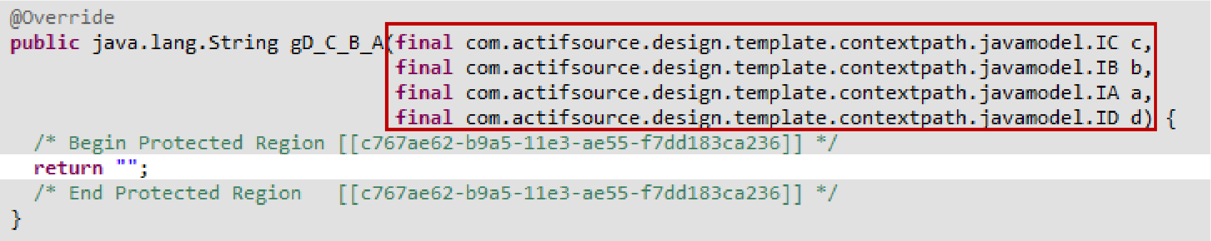


typeOf ch.actifsource.core.selector.type.FunctionContext
typeRef com.actifsource.design.functionspace.selector.Component
typeId
function[1]
typeOf ch.actifsource.core.selector.type.SelectorFunction
name allComponent
comment
param
selector Component:Component union Component.-component.allComponent@MyFunctionSpace
returnType
typeOf ch.actifsource.core.selector.type.ListType
innerTypeRef com.actifsource.design.functionspace.selector.Component
innerTypeId
function[2]
typeOf ch.actifsource.core.selector.type.SelectorFunction
name getClient
comment
param
selector Component.allComponent@MyFunctionSpace.-component
returnType
typeOf ch.actifsource.core.selector.type.ClassType
classRef com.actifsource.design.functionspace.selector.Client
classId
function

Calling selectors with parameters

[TBD]

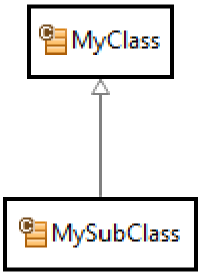
JavaFunction) leads to the corresponding java function arguments.



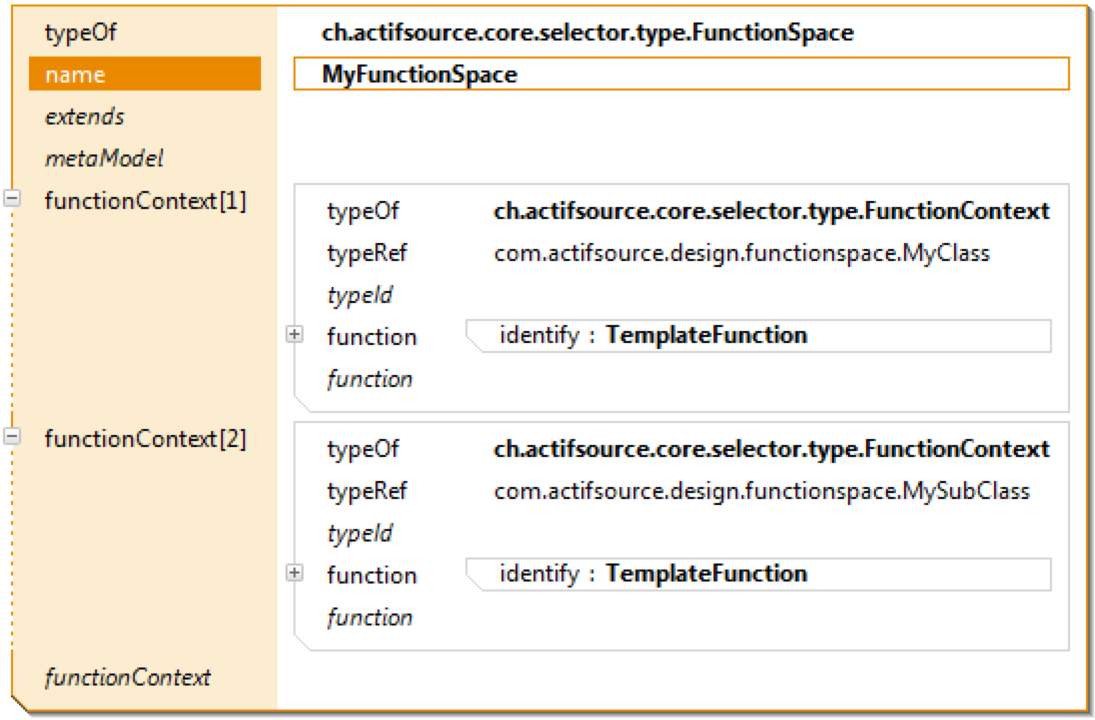
```
@Override
public java.lang.String gD_C_B_A(final com.actifsource.design.template.contextpath.javamodel.IC c,
                                 final com.actifsource.design.template.contextpath.javamodel.IB b,
                                 final com.actifsource.design.template.contextpath.javamodel.IA a,
                                 final com.actifsource.design.template.contextpath.javamodel.ID d) {
  /* Begin Protected Region [[c767ae62-b9a5-11e3-ae55-f7dd183ca236]] */
  return "";
  /* End Protected Region   [[c767ae62-b9a5-11e3-ae55-f7dd183ca236]] */
}
```

9.2.3 Polymorphic calls

Function calls are polymorphic if a function has the same name, the same parameters and a typeRef to a sub class.



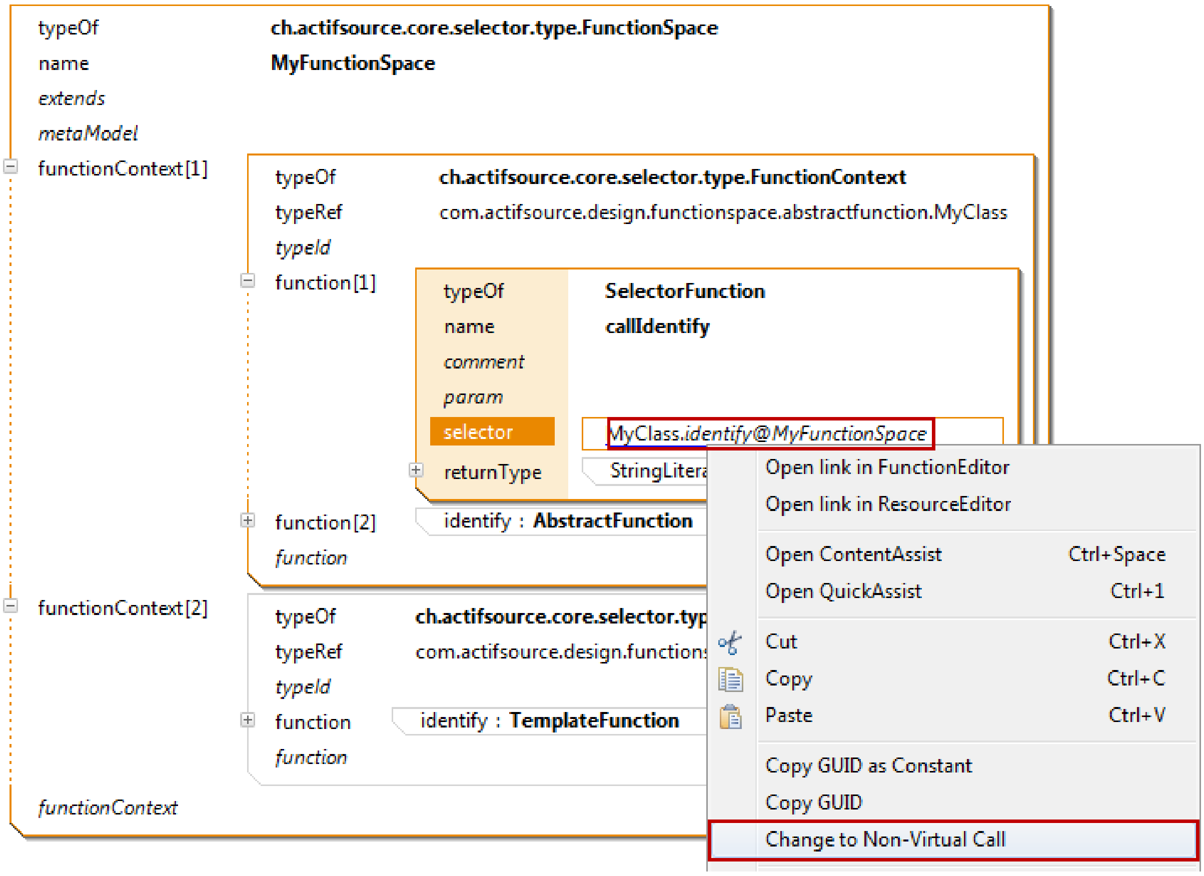
In the following example, the function identify is defined for *MyClass* and *MySubClass*. There will be a polymorphic call to MyClass.*identify*, dependent on the type of the instance.



typeOf ch.actifsource.core.selector.type.FunctionSpace
name MyFunctionSpace
extends
metaModel
functionContext[1]
  typeOf ch.actifsource.core.selector.type.FunctionContext
  typeRef com.actifsource.design.functionspace.MyClass
  typeId
  function identify : **TemplateFunction**
  function
functionContext[2]
  typeOf ch.actifsource.core.selector.type.FunctionContext
  typeRef com.actifsource.design.functionspace.MySubClass
  typeId
  function identify : **TemplateFunction**
  function
functionContext

9.2.4 Non-Polymorphic calls

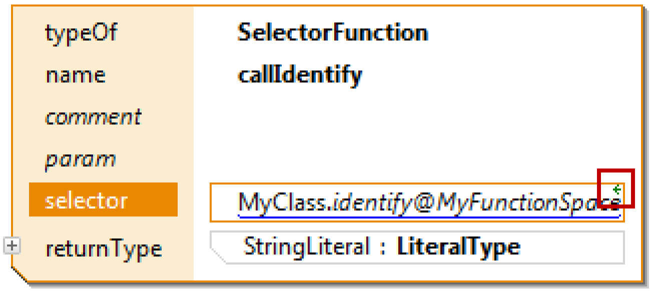
There are situations where polymorphic calls are not desired. You have to disable polymorphic calls on every caller. Use the context menu *Change to non-virtual call* on the function.



typeOf ch.actifsource.core.selector.type.FunctionSpace
name MyFunctionSpace
extends
metaModel
functionContext[1]
typeOf ch.actifsource.core.selector.type.FunctionContext
typeRef com.actifsource.design.functionspace.abstractfunction.MyClass
typeId
function[1]
typeOf SelectorFunction
name callIdentify
comment
param
selector MyClass.*identify*@*MyFunctionSpace*
returnType StringLitera
function[2] identify : AbstractFunction
function
functionContext[2]
typeOf ch.actifsource.core.selector.typ
typeRef com.actifsource.design.function
typeId
function identify : TemplateFunction
function
functionContext

Open link in FunctionEditor
Open link in ResourceEditor
Open ContentAssist Ctrl+Space
Open QuickAssist Ctrl+1
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Copy GUID as Constant
Copy GUID
Change to Non-Virtual Call

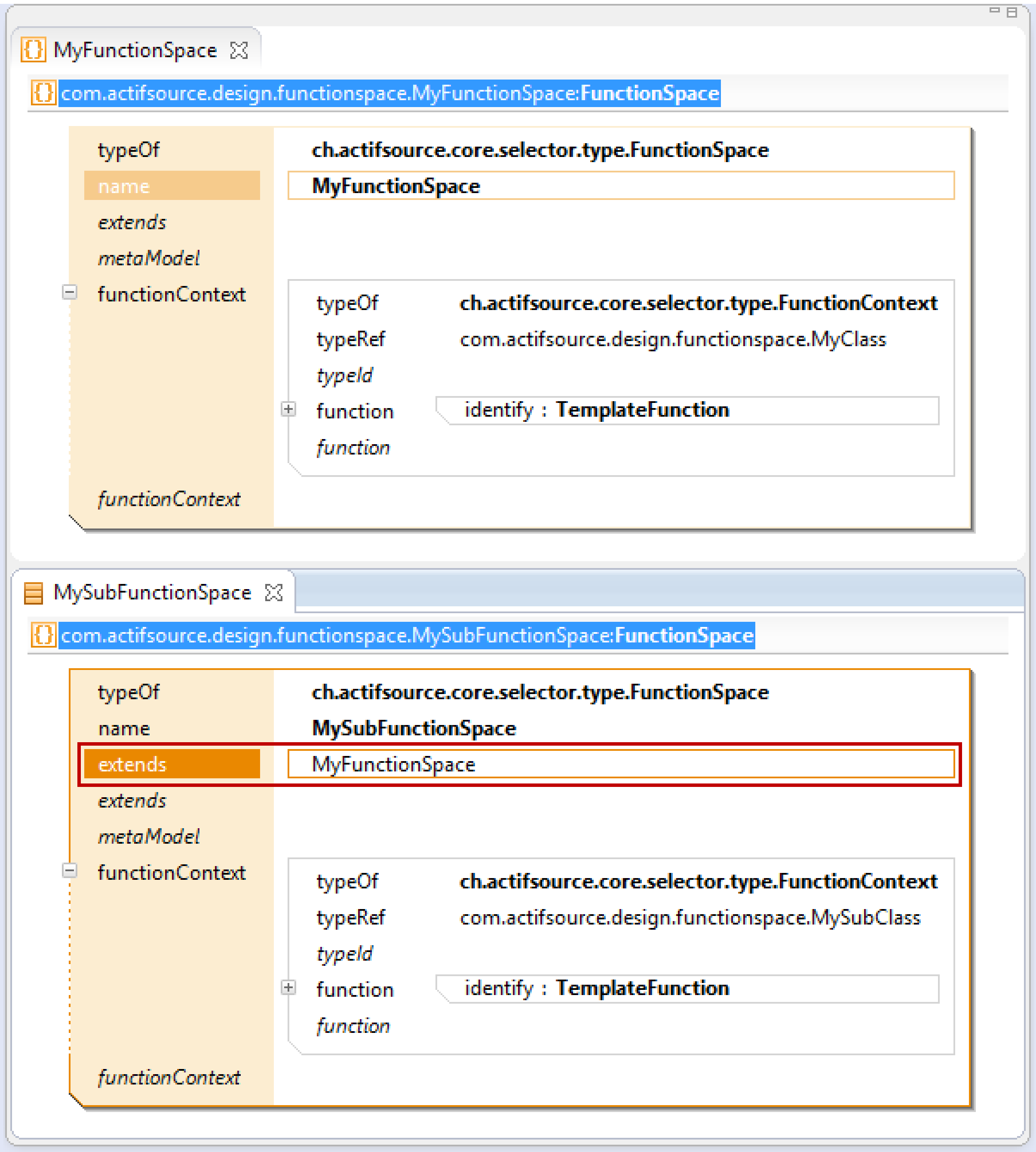
A small arrow indicates the non-polymorphic call.



typeOf SelectorFunction
name callIdentify
comment
param
selector MyClass.*identify*@*MyFunctionSpace*
returnType StringLiteral : LiteralType

9.2.5 Extends

Polymorphic calls are supported in the same function space by default. Extending another function space enables polymorphic calls over functions spaces.



MyFunctionSpace

com.actifsource.design.functionspace.MyFunctionSpace:**FunctionSpace**

typeOf	**ch.actifsource.core.selector.type.FunctionSpace**
name	**MyFunctionSpace**
extends	
metaModel	
functionContext	
	typeOf **ch.actifsource.core.selector.type.FunctionContext**
	typeRef com.actifsource.design.functionspace.MyClass
	typeId
	function identify : **TemplateFunction**
	function
functionContext	

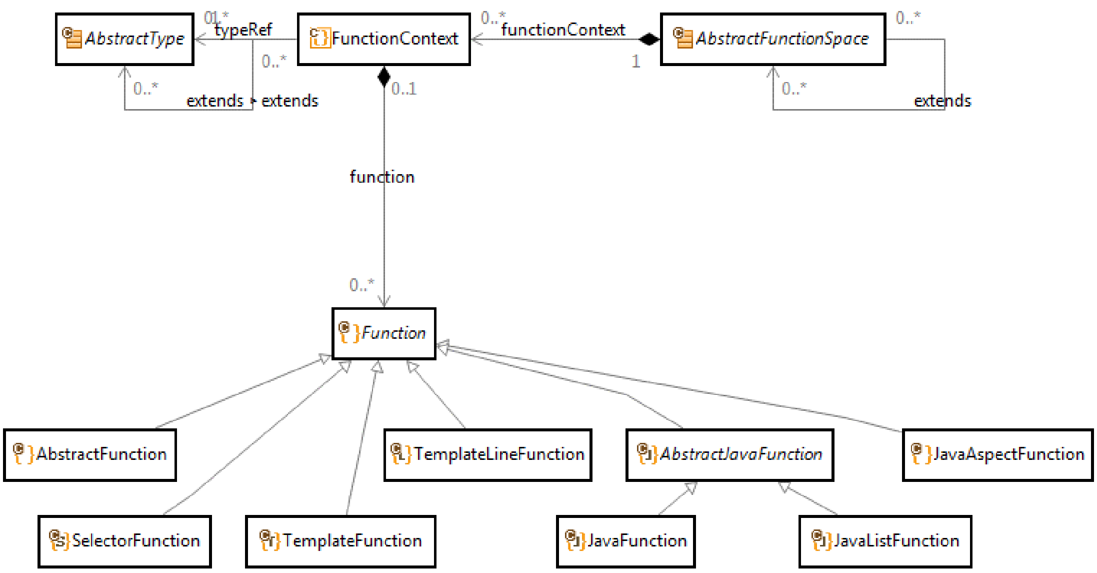
MySubFunctionSpace

com.actifsource.design.functionspace.MySubFunctionSpace:**FunctionSpace**

typeOf	**ch.actifsource.core.selector.type.FunctionSpace**
name	**MySubFunctionSpace**
extends	MyFunctionSpace
extends	
metaModel	
functionContext	
	typeOf **ch.actifsource.core.selector.type.FunctionContext**
	typeRef com.actifsource.design.functionspace.MySubClass
	typeId
	function identify : **TemplateFunction**
	function
functionContext	

9.3 Function types

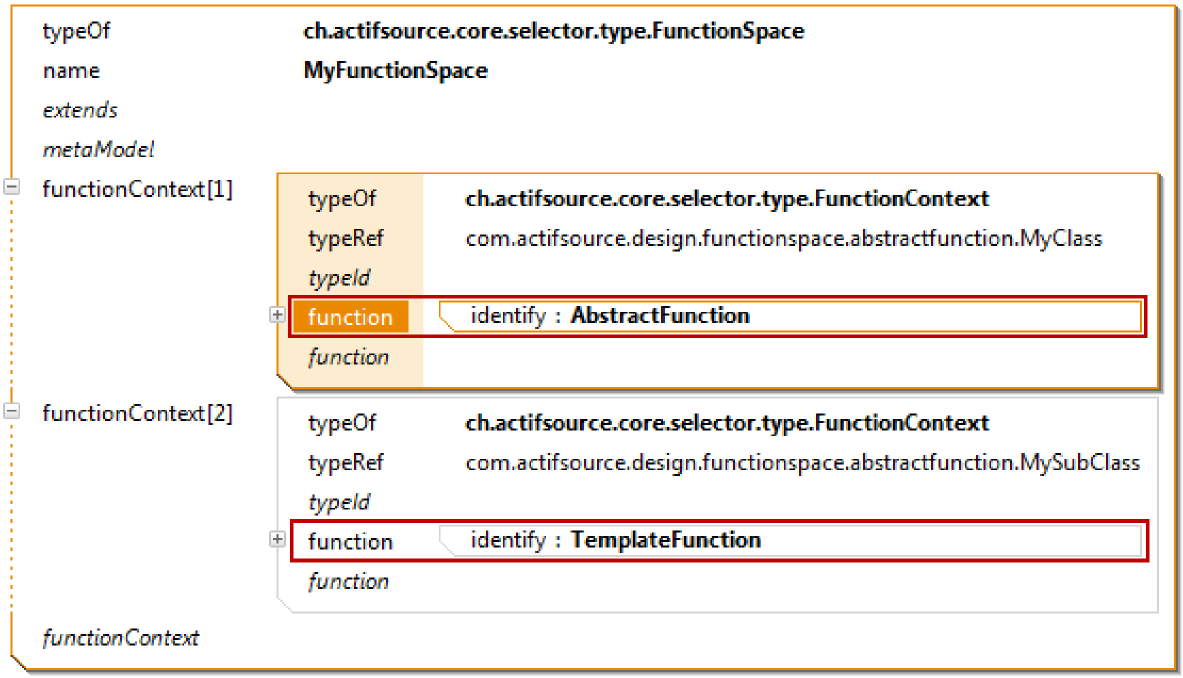
Actifsource supports different types of functions.



9.3.1 Abstract Function

Abstract functions shall only be defined on types with an abstract modifier (see Chapter 4.6.1 ch.actifsource.core.Class).

For an abstract function, there must be non-abstract function for any non-abstract subclass in the same function space or in a function space that extends it.



typeOf ch.actifsource.core.selector.type.FunctionSpace
name MyFunctionSpace
extends
metaModel
functionContext[1]
typeOf ch.actifsource.core.selector.type.FunctionContext
typeRef com.actifsource.design.functionspace.abstractfunction.MyClass
typeId
function identify : AbstractFunction
function
functionContext[2]
typeOf ch.actifsource.core.selector.type.FunctionContext
typeRef com.actifsource.design.functionspace.abstractfunction.MySubClass
typeId
function identify : TemplateFunction
function
functionContext

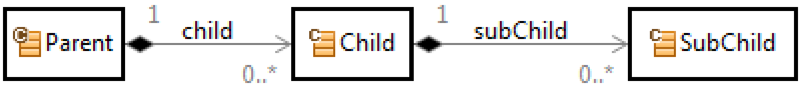
9.3.2 SelectorFunction

Selector functions allow you to navigate the model by using the selector syntax. Starting from a Class defined by FunctionContext.typeRef you may navigate via the resource properties.

Selectors might be used in Templates to select a context. But selectors might also be called from within selectors – even recursively.

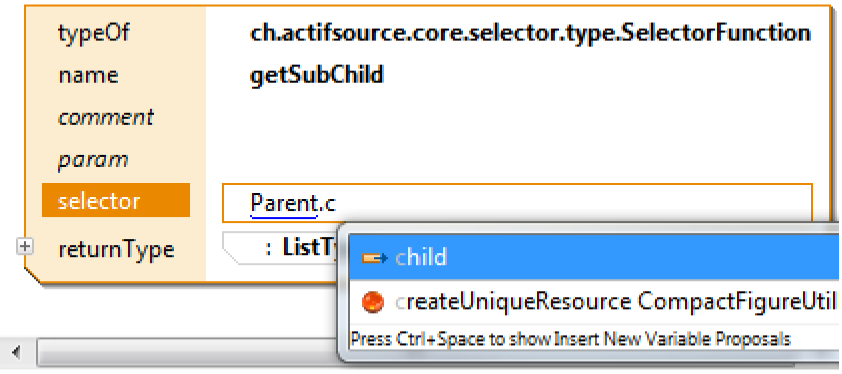
Forward navigation

Consider the following meta-model.



We like to define a selector function named getSubChild on Parent which returns all sub-children in all children of the parent. The return type when navigating along a property is given by the range of the property, i.e, in our example the expected return type is a list of elements of type SubChild.

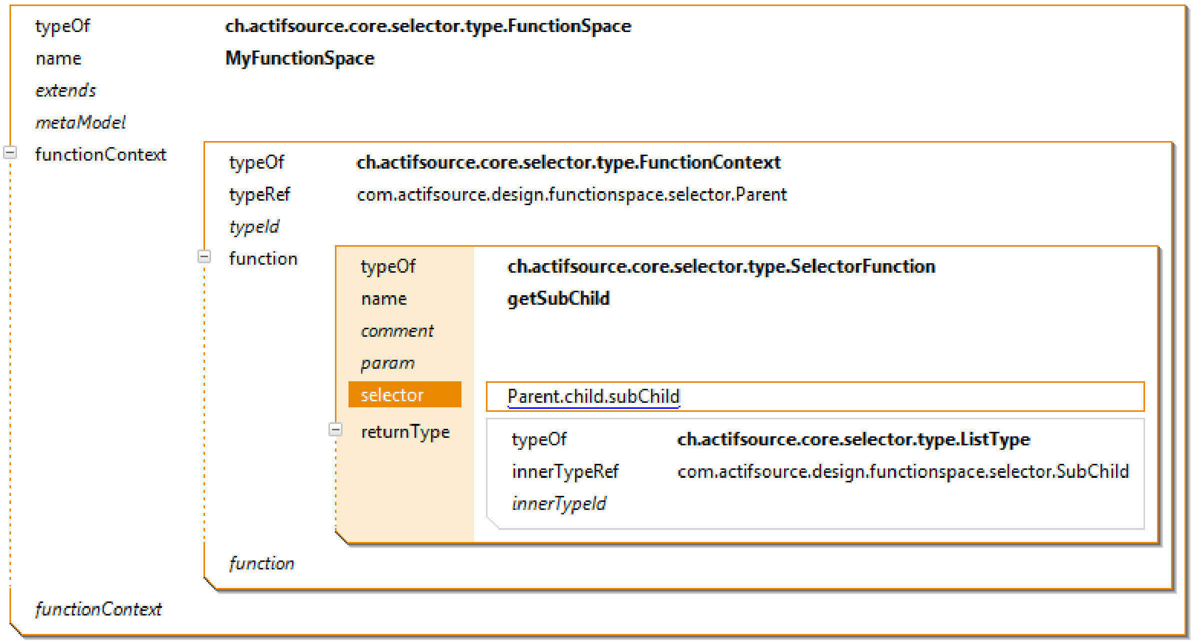
To navigate from Parent via child to subChild just assemble a selector Parent.child.subChild. Make sure to use content assist (Ctrl+Space) when writing selector functions.



typeOf ch.actifsource.core.selector.type.SelectorFunction
name getSubChild
comment
param
selector Parent.c
returnType : ListT

child
createUniqueResource CompactFigureUtil
Press Ctrl+Space to show Insert New Variable Proposals

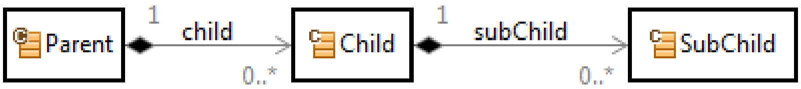
This is what a selector function could look like.



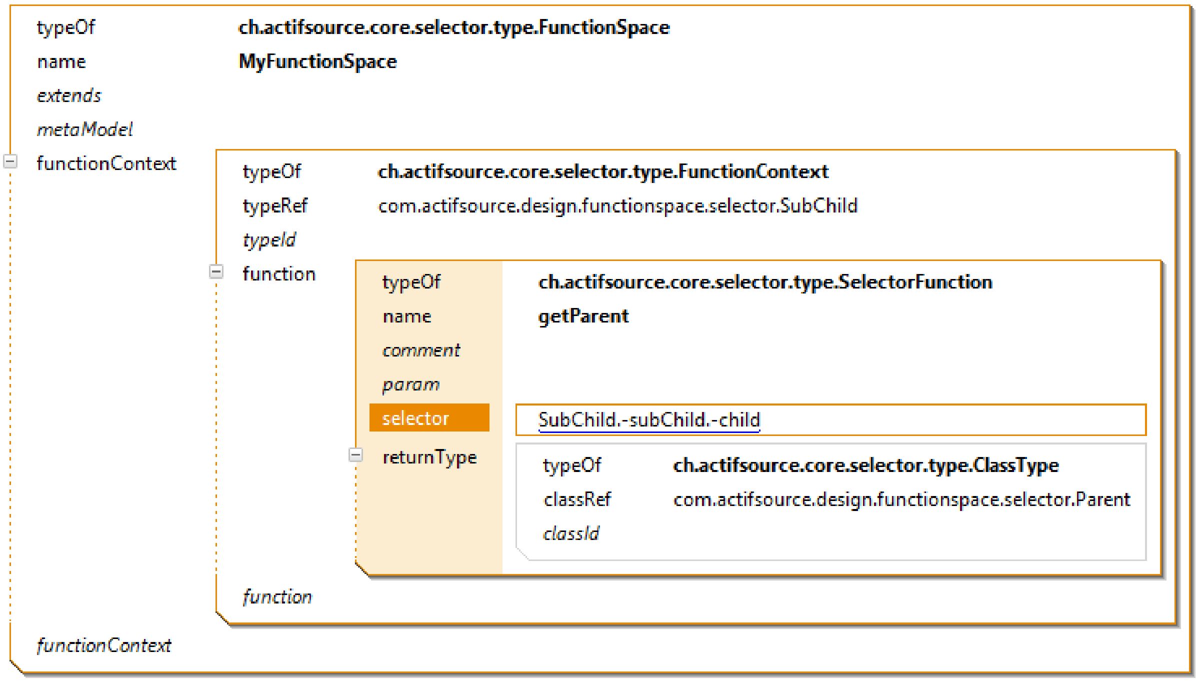
typeOf ch.actifsource.core.selector.type.FunctionSpace
name MyFunctionSpace
extends
metaModel
functionContext
  typeOf ch.actifsource.core.selector.type.FunctionContext
  typeRef com.actifsource.design.functionspace.selector.Parent
  typeId
  function
    typeOf ch.actifsource.core.selector.type.SelectorFunction
    name getSubChild
    comment
    param
    selector Parent.child.subChild
    returnType
      typeOf ch.actifsource.core.selector.type.ListType
      innerTypeRef com.actifsource.design.functionspace.selector.SubChild
      innerTypeId
  function
functionContext

Backward navigation

Consider the following meta-model.



To get the Parent instance for a SubChild instance we have to navigate backwards via the subChild and child relation. The selector allows backward navigation via the *minus relation*.


typeOf ch.actifsource.core.selector.type.FunctionSpace
name MyFunctionSpace
extends
metaModel
functionContext
typeOf ch.actifsource.core.selector.type.FunctionContext
typeRef com.actifsource.design.functionspace.selector.SubChild
typeId
function
typeOf ch.actifsource.core.selector.type.SelectorFunction
name getParent
comment
param
selector SubChild.-subChild.-child
returnType
typeOf ch.actifsource.core.selector.type.ClassType
classRef com.actifsource.design.functionspace.selector.Parent
classId
function
functionContext

List operators

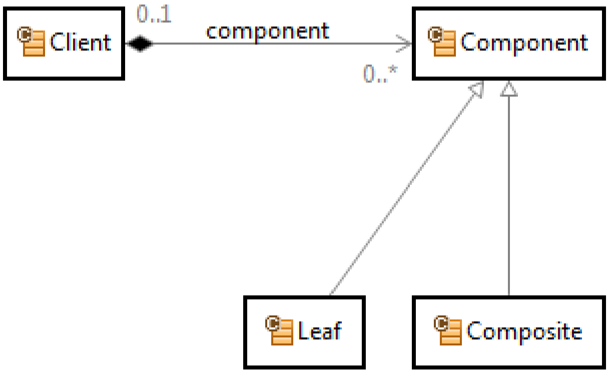
Actifsource provides you with the following operators which are defined on lists (where the lists are given by Selector expressions):

Operator	Description
A union B	The result is the concatenation of the two lists A and B. For example, [a1,a2] union [a3,a1] is equal to [a1,a2,a3,a1].
A intersect B	Only elements found in A and in B where duplicates are preserved and the resulting order is given by A. For example, [a1,a1,a2,a2] intersect [a2,a1,a1] is equal to [a1,a1,a2].
A except B	For all elements b in B, the first occurrence of b in A is removed from A. For example, [a2,a1,a3,a2,a1] except [a1,a2,a1] is equal to [a3,a2]
A else B	All elements in A if a is not empty, otherwise all elements in B. For example, [a1, a2] else [b1,b2] is equal to [a1,a2] and [] else [b1,b2] is equal to [b1,b2].

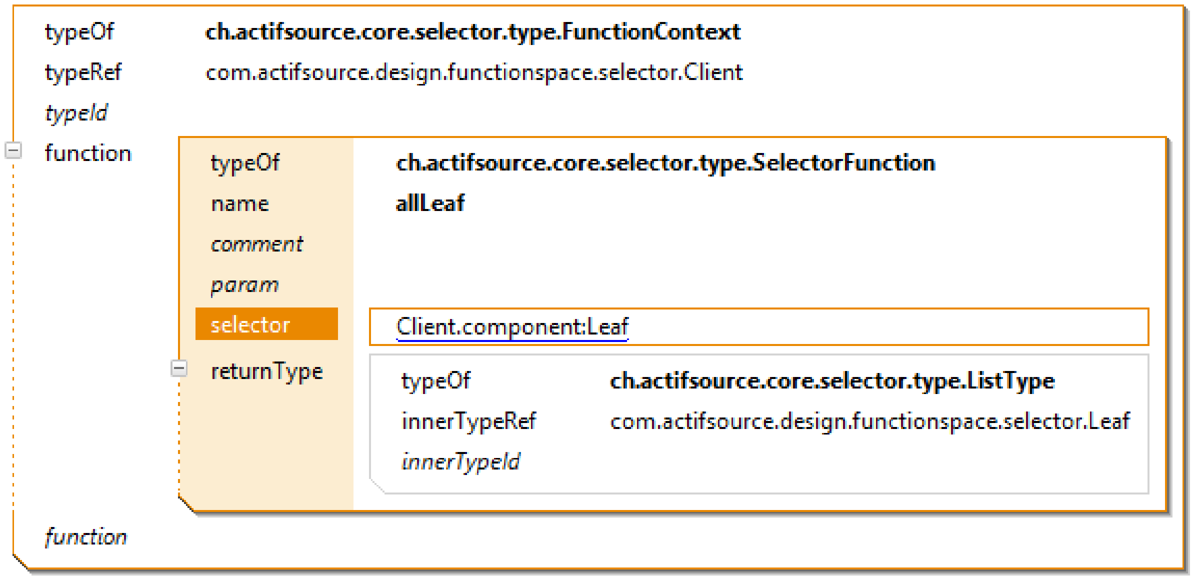
Note that you can use brackets to control precedence (i.e. (*A.x* union *A.y*) intersect *A.z*). The result type of the union, intersect and else operator is the most concrete supertype of the type of the two operands (e.g. if A is of type **NamedResource** and B is of type **Resource**, then A intersect B, A union B and A else B are all of type **Resource**).

Down Cast

Consider the following meta-model.



If you only like to get Leaf components from the Client, just use the type cast operator (colon).



typeOf	**ch.actifsource.core.selector.type.FunctionContext**
typeRef	com.actifsource.design.functionspace.selector.Client
typeId	
function	

typeOf	**ch.actifsource.core.selector.type.SelectorFunction**
name	**allLeaf**
comment	
param	
selector	Client.component:Leaf
returnType	

typeOf	**ch.actifsource.core.selector.type.ListType**
innerTypeRef	com.actifsource.design.functionspace.selector.Leaf
innerTypeId	

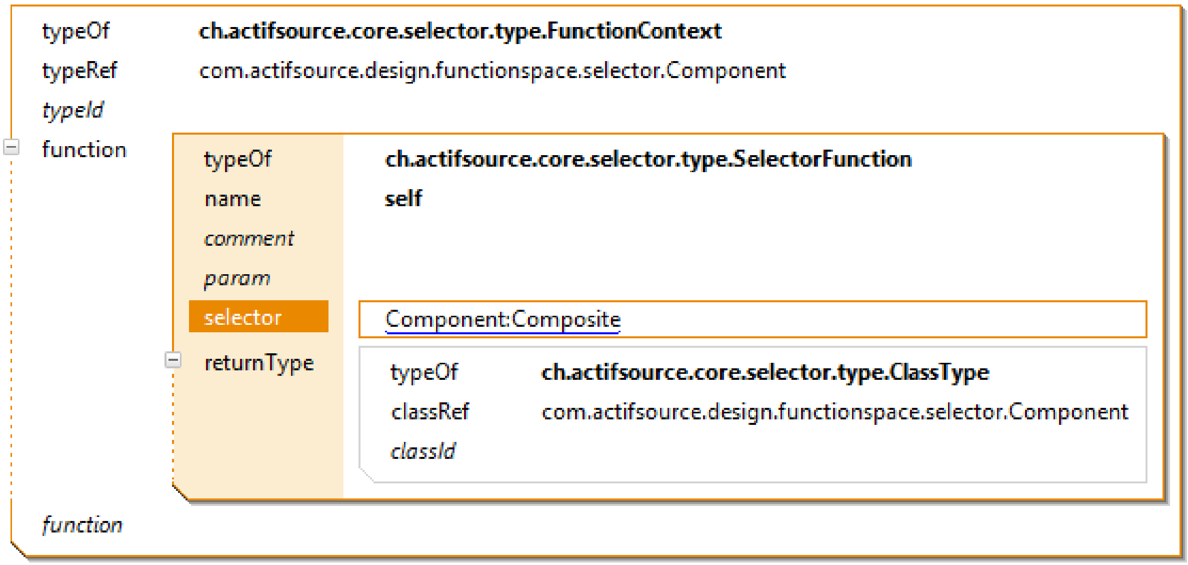
function

Up Cast

It is always possible to use an upcast to a base class (i.e. **Resource** or **NamedResource**) if needed.

Self Cast

If your selector has to return the typeRef instance itself, use the self-cast. Consider the following selector function for Component returning the component instance itself.



typeOf	**ch.actifsource.core.selector.type.FunctionContext**
typeRef	com.actifsource.design.functionspace.selector.Component
typeId	
function	

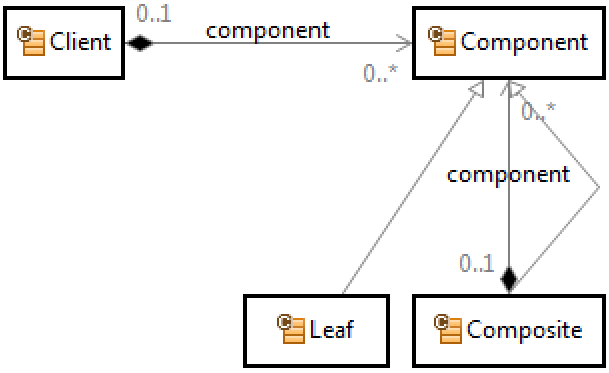
typeOf	**ch.actifsource.core.selector.type.SelectorFunction**
name	**self**
comment	
param	
selector	Component:Composite
returnType	

typeOf	**ch.actifsource.core.selector.type.ClassType**
classRef	com.actifsource.design.functionspace.selector.Component
classId	

function

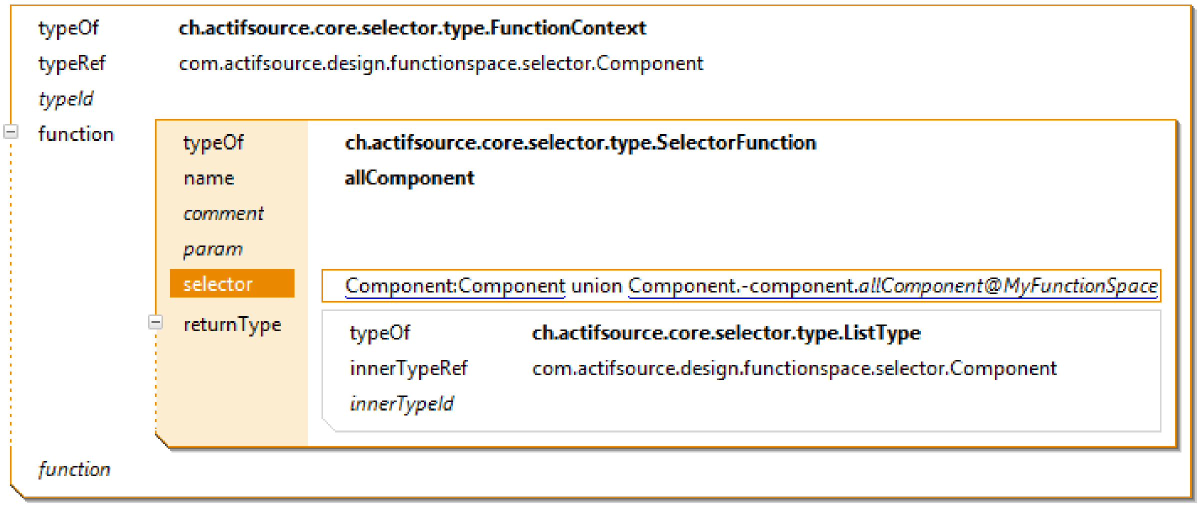
Recursive navigation

The diagram below shows a composite pattern as presented in the book Design Patterns from Erich Gamma et al. The composite pattern allows you to recursively instantiate Composite instances, which might aggregate other components of type Leaf or again – of type Composite.



There is an easy way to find the Client of this recursive model using selectors. First, we collect all components including the own component and all parent components.

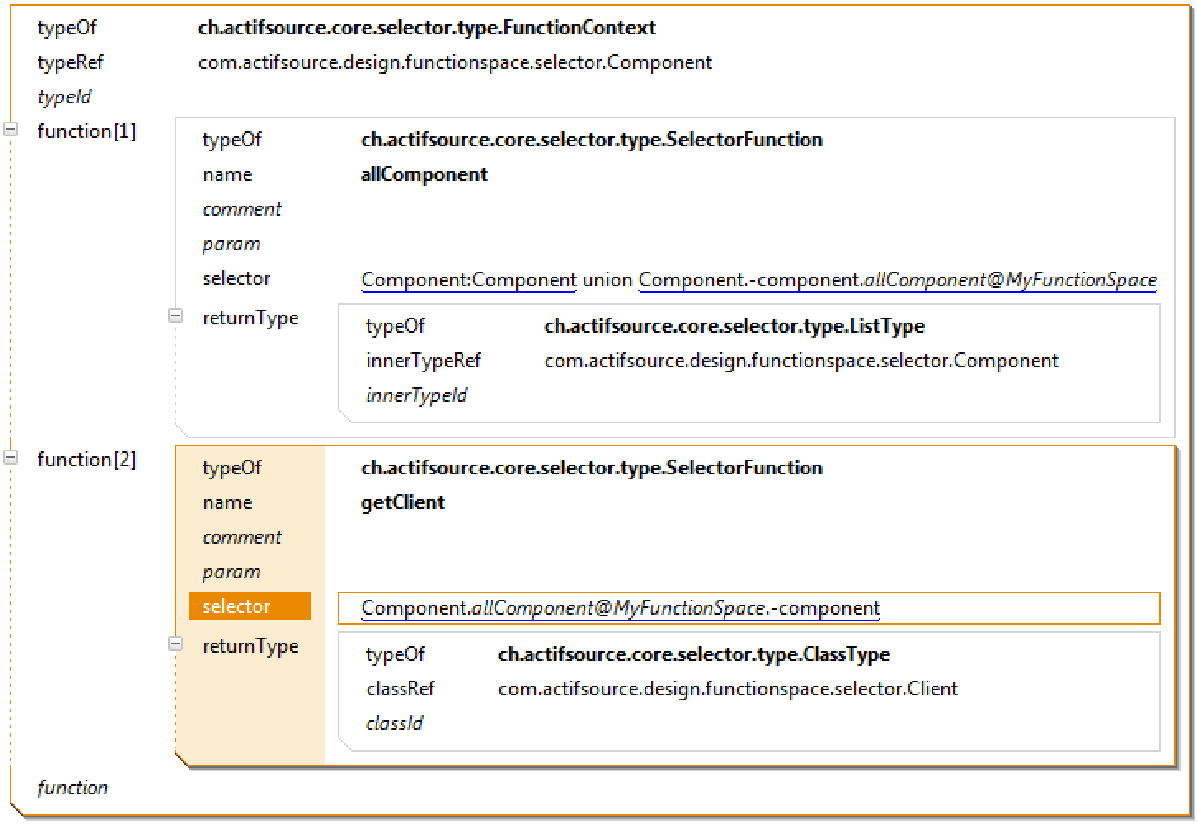
For that reason we write a selector function for Component which returns the component itself and also all parent components which are reachable by going backwards via the component relation.



typeOf ch.actifsource.core.selector.type.FunctionContext
typeRef com.actifsource.design.functionspace.selector.Component
typeId
function
typeOf ch.actifsource.core.selector.type.SelectorFunction
name allComponent
comment
param
selector Component:Component union Component.-component.*allComponent@MyFunctionSpace*
returnType
typeOf ch.actifsource.core.selector.type.ListType
innerTypeRef com.actifsource.design.functionspace.selector.Component
innerTypeId
function

To get the Client which is parent of all component just collect all component first by using the above selector functions. From all this components in the collected set there is only one instance aggregated by Client. Let's write a selector function for that.

Please take extra care because there are two relations named component. One is Composite.component; the other one is Client.component. Make sure to use Composite.component in allComponent and Client.component in getClient.



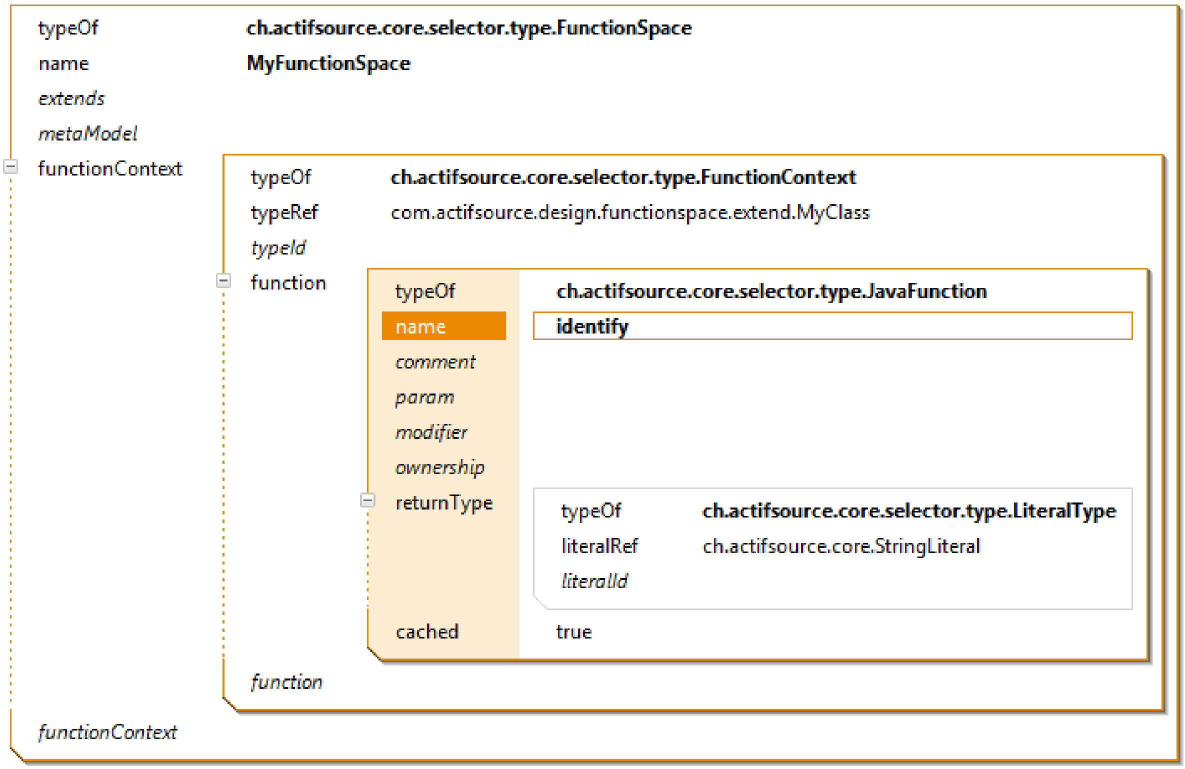
typeOf ch.actifsource.core.selector.type.FunctionContext
typeRef com.actifsource.design.functionspace.selector.Component
typeId
function[1]
typeOf ch.actifsource.core.selector.type.SelectorFunction
name allComponent
comment
param
selector Component:Component union Component.-component.*allComponent@MyFunctionSpace*
returnType
typeOf ch.actifsource.core.selector.type.ListType
innerTypeRef com.actifsource.design.functionspace.selector.Component
innerTypeId
function[2]
typeOf ch.actifsource.core.selector.type.SelectorFunction
name getClient
comment
param
selector Component.*allComponent@MyFunctionSpace*.-component
returnType
typeOf ch.actifsource.core.selector.type.ClassType
classRef com.actifsource.design.functionspace.selector.Client
classId
function

Calling selectors with parameters

[TBD]

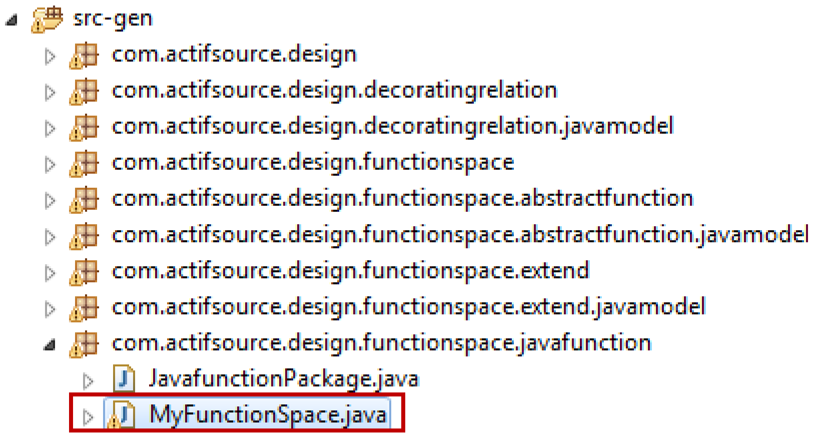
9.3.3 JavaFunction

Actifsource supports user-implemented Java functions that make use of the very powerful Javamodel to access the Actifsource models from Java code (see also Section 9.5).



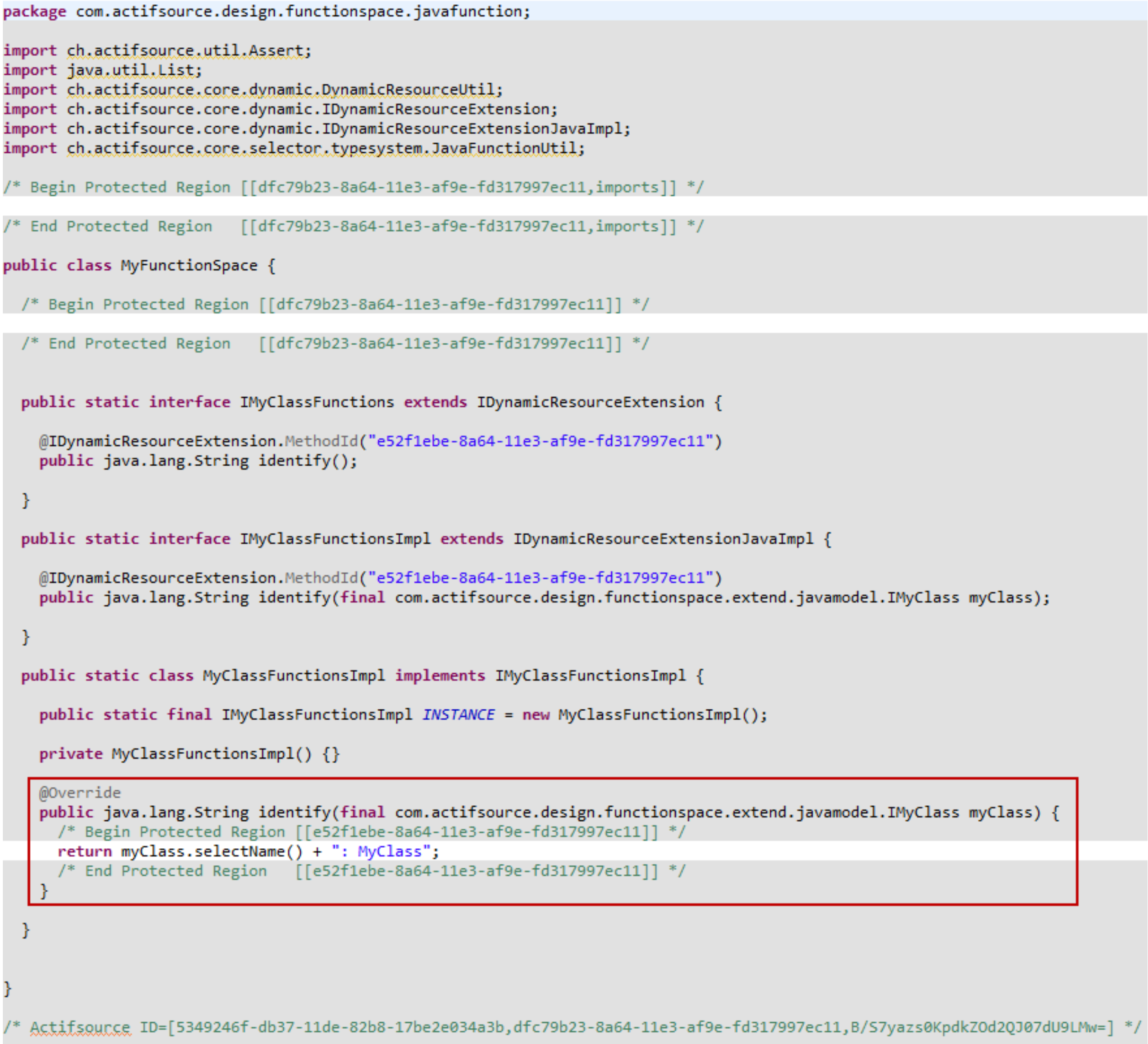
typeOf ch.actifsource.core.selector.type.FunctionSpace
name MyFunctionSpace
extends
metaModel
functionContext
  typeOf ch.actifsource.core.selector.type.FunctionContext
  typeRef com.actifsource.design.functionspace.extend.MyClass
  typeId
  function
    typeOf ch.actifsource.core.selector.type.JavaFunction
    name identify
    comment
    param
    modifier
    ownership
    returnType
      typeOf ch.actifsource.core.selector.type.LiteralType
      literalRef ch.actifsource.core.StringLiteral
      literalId
    cached true
  function
functionContext

When you declare a Java function, Actifsource automatically generates a function skeleton in a file with the same name as your function space found in the folder src-gen.



src-gen
com.actifsource.design
com.actifsource.design.decoratingrelation
com.actifsource.design.decoratingrelation.javamodel
com.actifsource.design.functionspace
com.actifsource.design.functionspace.abstractfunction
com.actifsource.design.functionspace.abstractfunction.javamodel
com.actifsource.design.functionspace.extend
com.actifsource.design.functionspace.extend.javamodel
com.actifsource.design.functionspace.javafunction
JavafunctionPackage.java
MyFunctionSpace.java

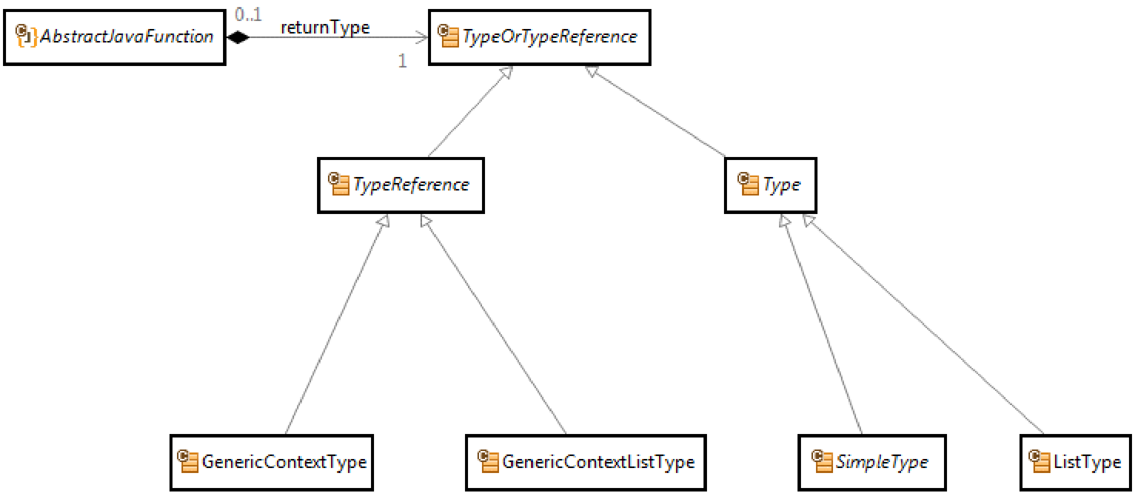
You should only modify generated files within protected regions (see Chapter 2.12.5 Protected Regions). Take extra care that import statements are placed within the protected regions – especially if inserted automatically by the Java content assist.



```java
package com.actifsource.design.functionspace.javafunction;

import ch.actifsource.util.Assert;
import java.util.List;
import ch.actifsource.core.dynamic.DynamicResourceUtil;
import ch.actifsource.core.dynamic.IDynamicResourceExtension;
import ch.actifsource.core.dynamic.IDynamicResourceExtensionJavaImpl;
import ch.actifsource.core.selector.typesystem.JavaFunctionUtil;

/* Begin Protected Region [[dfc79b23-8a64-11e3-af9e-fd317997ec11,imports]] */

/* End Protected Region   [[dfc79b23-8a64-11e3-af9e-fd317997ec11,imports]] */

public class MyFunctionSpace {

  /* Begin Protected Region [[dfc79b23-8a64-11e3-af9e-fd317997ec11]] */

  /* End Protected Region   [[dfc79b23-8a64-11e3-af9e-fd317997ec11]] */


  public static interface IMyClassFunctions extends IDynamicResourceExtension {

    @IDynamicResourceExtension.MethodId("e52f1ebe-8a64-11e3-af9e-fd317997ec11")
    public java.lang.String identify();

  }

  public static interface IMyClassFunctionsImpl extends IDynamicResourceExtensionJavaImpl {

    @IDynamicResourceExtension.MethodId("e52f1ebe-8a64-11e3-af9e-fd317997ec11")
    public java.lang.String identify(final com.actifsource.design.functionspace.extend.javamodel.IMyClass myClass);

  }

  public static class MyClassFunctionsImpl implements IMyClassFunctionsImpl {

    public static final IMyClassFunctionsImpl INSTANCE = new MyClassFunctionsImpl();

    private MyClassFunctionsImpl() {}

    @Override
    public java.lang.String identify(final com.actifsource.design.functionspace.extend.javamodel.IMyClass myClass) {
      /* Begin Protected Region [[e52f1ebe-8a64-11e3-af9e-fd317997ec11]] */
      return myClass.selectName() + ": MyClass";
      /* End Protected Region   [[e52f1ebe-8a64-11e3-af9e-fd317997ec11]] */
    }

  }


}
/* Actifsource ID=[5349246f-db37-11de-82b8-17be2e034a3b,dfc79b23-8a64-11e3-af9e-fd317997ec11,B/S7yazs0KpdkZOd2QJ07dU9LMw=] */
```

As function arguments, an instance of type FunctionContext.typeRef and all parameters are passed. Use the Java content assist (Ctrl+Space) to display available functions. To access properties choose myClass.selectMyProperty(). For more information on how to access the Javamodel see Chapter 9.5.

Return Types

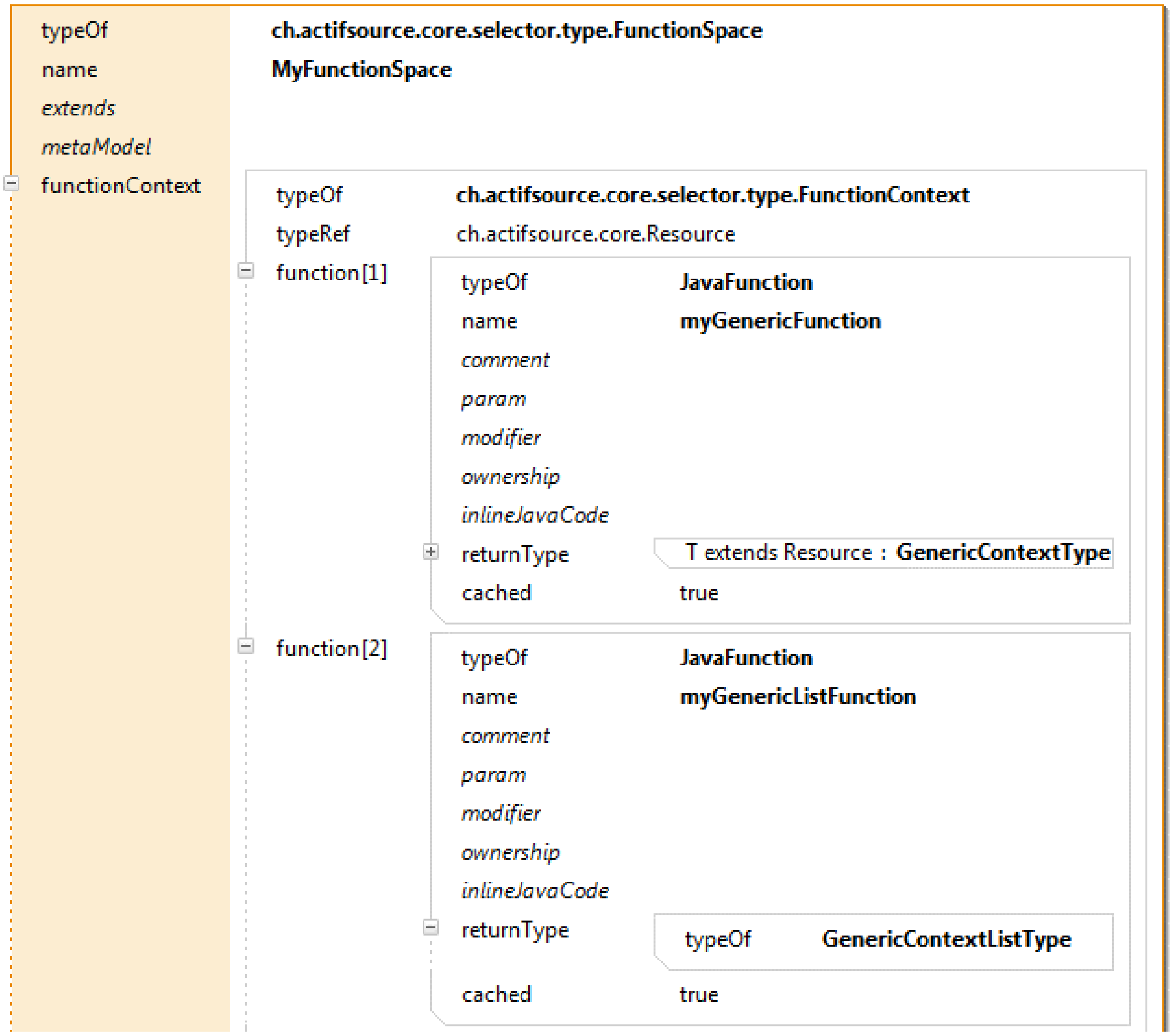


A Java function has a return type which is either a **Type** or a **TypeReference**.

A **Type** is either a **SimpleType** or a **ListType**. A **SimpleType** can, in particular, be a **ClassType**, which references any Class, or a **LiteralType**, which references any **Literal**. For a **LiteralType** the return type of the generated Java function is the Java class given by the return value of the method getValueType() of the **ILiteralAspect** (e.g. in the example above the LiteralAspect of StringLiterals (ch.actifsource.core.model.aspects.impl.String.StringLiteralAspect) defines that java.lang.String represents StringLiterals and, therefore, the the return type of *identify* is java.lang.String). A **ListType** references either a **Class** or a **Literal** (more precisely, it actually references an **AbstractType**). The return type of the generated Java function is then a java.util.List<Class>, where Class is the Java class that corresponds to the **Literal** or the **Class**. Note that in the latter case the Java class is the wrapper Java class that corresponds to the Actifsource **Class** and is provided by the Javamodel (see also Chapter 2 and Section 9.5).

A TypeReference is either a **GenericContextType** or a **GenericContextListType**. In the first case, the return type of the generated Java function is *<T extends C> T* where C is the Java class corresponding to the type of the element the function is called on (the *this*-instance). In the second case, the return type of the generated Java function is a *<T extends C> java.util.List<T>* where T is defined as before (see examples below).

Note that function with a return type of **GenericContextType** or a **GenericContextListType** can be applied to elements of any sub-type of the type given by **typeRef** of the **FunctionContext**, i.e., the this-parameter of the generated Java function is <T extends Class> T where Class is the Java wrapper class corresponding to the **typeRef** of the **FunctionContext**.



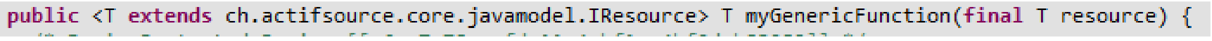
typeOf	**ch.actifsource.core.selector.type.FunctionSpace**
name	**MyFunctionSpace**
extends	
metaModel	
functionContext	

typeOf	**ch.actifsource.core.selector.type.FunctionContext**
typeRef	ch.actifsource.core.Resource
function[1]	

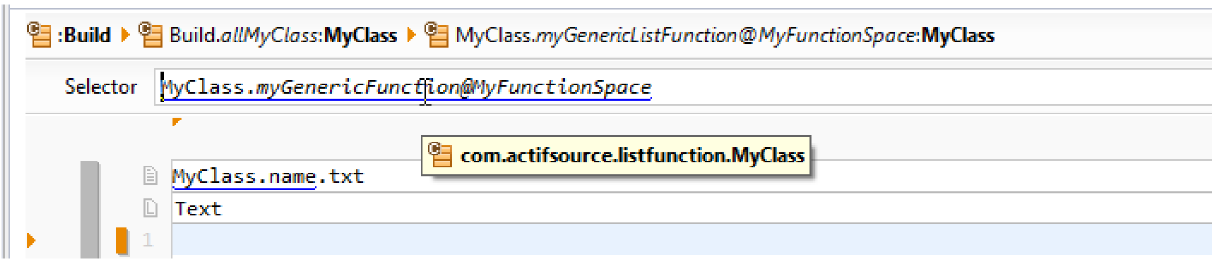
typeOf	**JavaFunction**
name	**myGenericFunction**
comment	
param	
modifier	
ownership	
inlineJavaCode	
returnType	T extends Resource : **GenericContextType**
cached	true

function[2]

typeOf	**JavaFunction**
name	**myGenericListFunction**
comment	
param	
modifier	
ownership	
inlineJavaCode	
returnType	typeOf **GenericContextListType**
cached	true



```
public <T extends ch.actifsource.core.javamodel.IResource> T myGenericFunction(final T resource) {
```

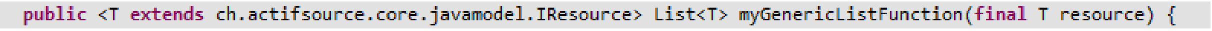


:Build ▸ Build.allMyClass:MyClass ▸ MyClass.myGenericListFunction@MyFunctionSpace:MyClass

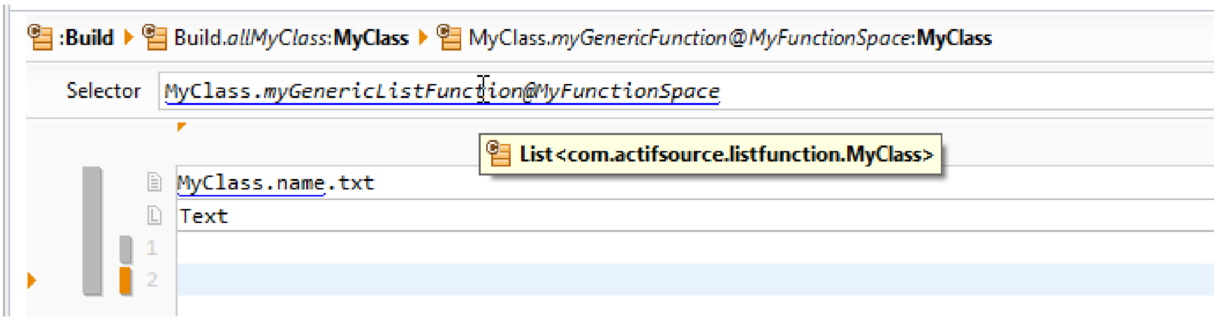
Selector MyClass.myGenericFunction@MyFunctionSpace

com.actifsource.listfunction.MyClass

MyClass.name.txt
Text
1



```
public <T extends ch.actifsource.core.javamodel.IResource> List<T> myGenericListFunction(final T resource) {
```



:Build ▸ Build.allMyClass:MyClass ▸ MyClass.myGenericFunction@MyFunctionSpace:MyClass

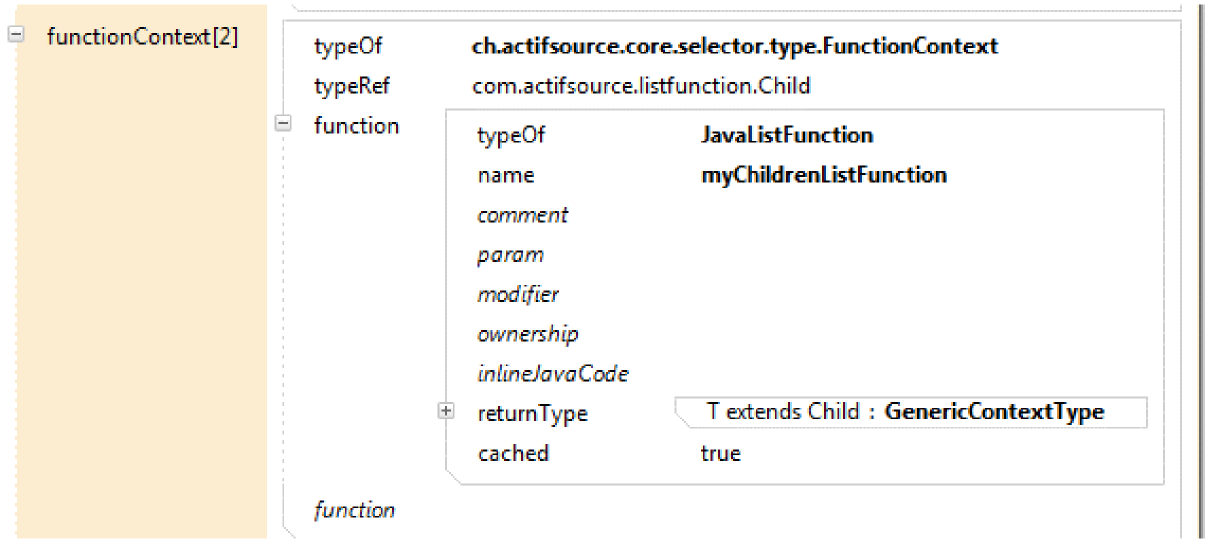
Selector MyClass.myGenericListFunction@MyFunctionSpace

List<com.actifsource.listfunction.MyClass>

MyClass.name.txt
Text
1
2

9.3.4 JavaListFunction

Java list functions can be applied to a *list of elements* defined by a Selector expression, e.g. in the Selector expression Parent.child.myFunction@ChildFunctionSpace the function myFunction is called on the list of all Children reachable from Parent via the relation child (see example in Section 9.3.2). The this-parameter of the generated Java function is then of type java.util.List<C> where C is the Java class corresponding to the **typeRef** of the **FunctionContext** (respectively java.util.List<T extends C> if the **returnType** of the Java list function is **GenericContext(List)Type**). Consider the following example that is based on the meta-model from Section 9.3.5:



functionContext[2]

typeOf	**ch.actifsource.core.selector.type.FunctionContext**
typeRef	com.actifsource.listfunction.Child
function	

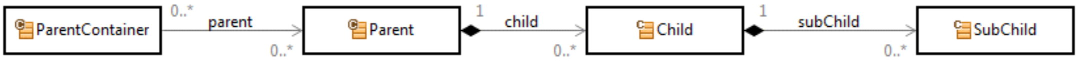
typeOf	**JavaListFunction**
name	**myChildrenListFunction**
comment	
param	
modifier	
ownership	
inlineJavaCode	
returnType	T extends Child : **GenericContextType**
cached	true

function

```
    public <T extends com.actifsource.listfunction.javamodel.IChild> T myChildrenListFunction(final List<T> childList) {
```

The return types of **JavaListFunctions** are determined in the same way as for **JavaFunctions** (see Section 9.3.3). A list of built-in (Java) list functions is presented in Section **Fehler! Verweisquelle konnte nicht gefunden werden.**.

Consider an extended meta-model where Parent can be referenced by a ParentContainer via a relation parent:



In this case, the selector ParentContainer.parent.child.myFunction@MyFunctionSpace constructs for each Parent the list of Children reachable from this Parent and then applies the function myFunction to each of these lists. If the function should be applied to the list of Children reachable indirectly via parent->child, we can write a (Selector)Function that returns a list of all these Children, e.g. ParentContainer.getAllChildren@MyFunctionSpace where getAllChildren is a SelectorFunction with the selector ParentContainer.parent.child. In the selector ParentContainer.getAllChildren@MyFunctionSpace.myFunction@MyFunctionSpace, the function myFunction is only called once on the list of all Children reachable from ParentContainer.

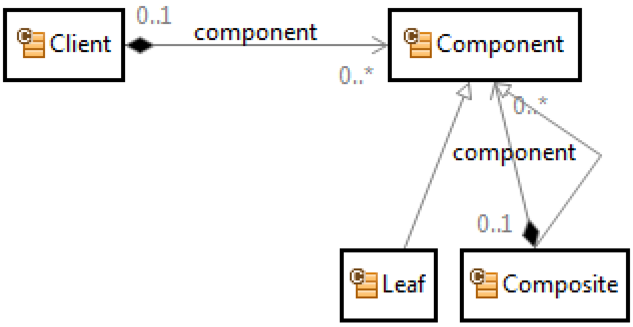
9.3.5 JavaAspectFunction

[TBD]

9.3.6 TemplateFunction

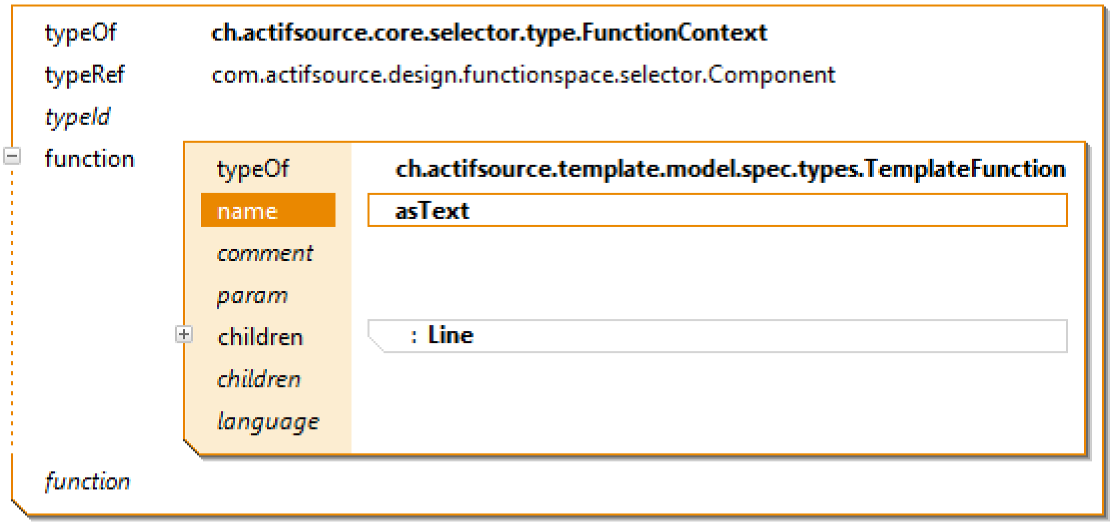
A template function behaves in the same way as a template, but there are no files generated from a template function. Just think of a template function as a sub template which can be expanded in a template or in another template function (also recursively).

Consider the following meta-model:



Let us now write a template function for a component which writes the name and type of the component and, if the given component is a composite, also does the same recursively for all subcomponents.

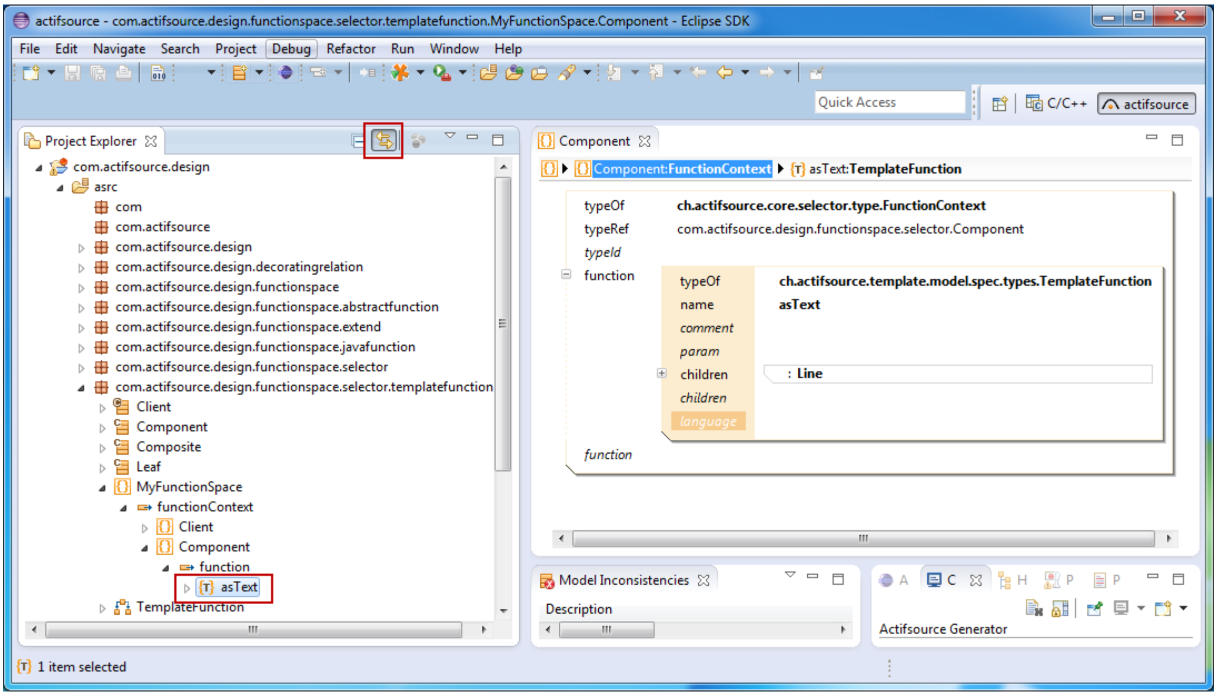
First of all we have to define the template function in the model.



typeOf	**ch.actifsource.core.selector.type.FunctionContext**
typeRef	com.actifsource.design.functionspace.selector.Component
typeId	
function	
function	

typeOf	**ch.actifsource.template.model.spec.types.TemplateFunction**
name	**asText**
comment	
param	
children	**: Line**
children	
language	

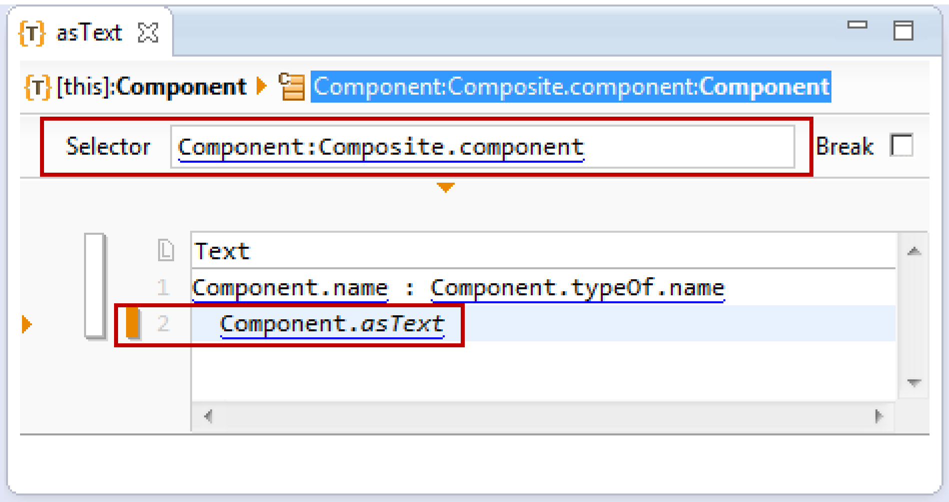
To open the template function with the template editor, just double click in the Project Explorer. Use the *Link with Editor* tool (see Chapter 2.10.1 Link with Editor) to easily locate the template function in the project explorer.


actifsource - com.actifsource.design.functionspace.selector.templatefunction.MyFunctionSpace.Component - Eclipse SDK
File Edit Navigate Search Project Debug Refactor Run Window Help
Quick Access
C/C++
actifsource
Project Explorer
com.actifsource.design
asrc
com
com.actifsource
com.actifsource.design
com.actifsource.design.decoratingrelation
com.actifsource.design.functionspace
com.actifsource.design.functionspace.abstractfunction
com.actifsource.design.functionspace.extend
com.actifsource.design.functionspace.javafunction
com.actifsource.design.functionspace.selector
com.actifsource.design.functionspace.selector.templatefunction
Client
Component
Composite
Leaf
MyFunctionSpace
functionContext
Client
Component
function
asText
TemplateFunction
Component
Component:FunctionContext
asText:TemplateFunction
Model Inconsistencies
Description
Actifsource Generator
1 item selected

typeOf ch.actifsource.core.selector.type.FunctionContext
typeRef com.actifsource.design.functionspace.selector.Component
typeId
function
typeOf ch.actifsource.template.model.spec.types.TemplateFunction
name asText
comment
param
children : Line
children
language
function

In line 1 we write out the component name and its type name. In line 2 we iterate over all aggregated components, but only if the component is of type Composite (type cast). In this context we call the template function *asText* recursively for all aggregated component.

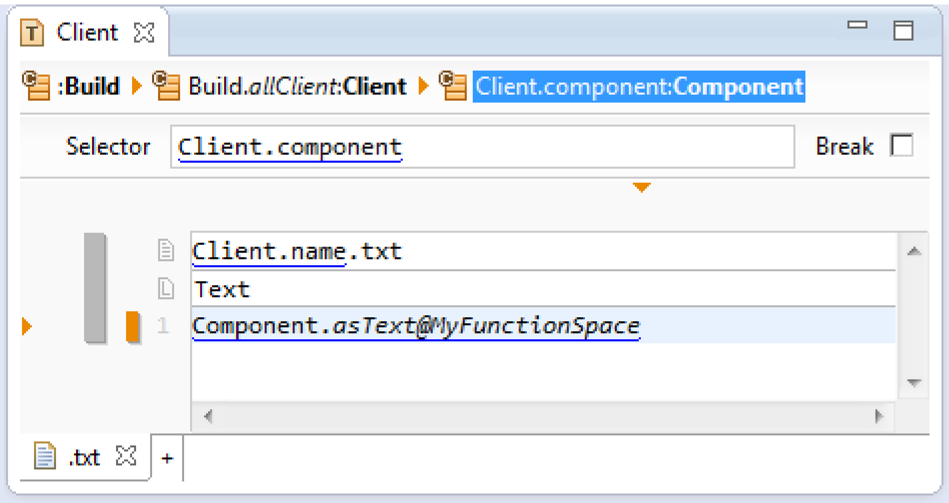
Note the indention of two spaces on line 2. Actifsource takes care of the indentions so that the whole content of the template function is indented.



Selector Component:Composite.component

```
1 Component.name : Component.typeOf.name
2   Component.asText
```

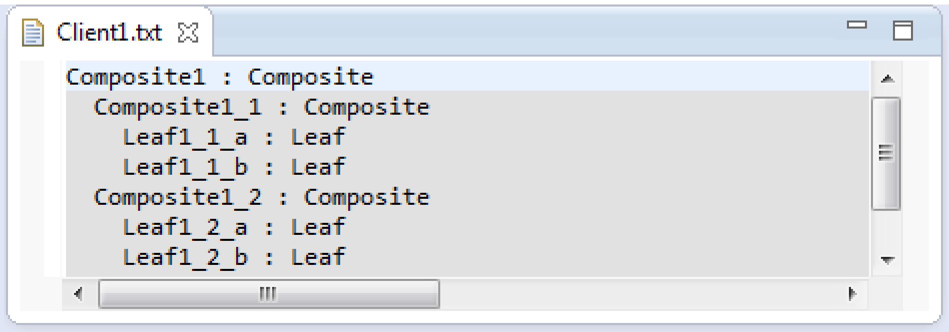
We can now call our template function from a template. If there is a call to a function from another function space, the function space is explicitly stated using the notation *myFunction@MyFunctionSpace*.



```
Client.component
```

```
Client.name.txt
Text
Component.asText@MyFunctionSpace
```

Next, we create an instance of type Client containing composites and leaves. The output from the above template might look as follows. Note that the indention is applied recursively.

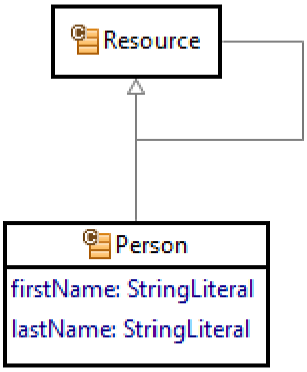


```
Composite1 : Composite
  Composite1_1 : Composite
    Leaf1_1_a : Leaf
    Leaf1_1_b : Leaf
  Composite1_2 : Composite
    Leaf1_2_a : Leaf
    Leaf1_2_b : Leaf
```

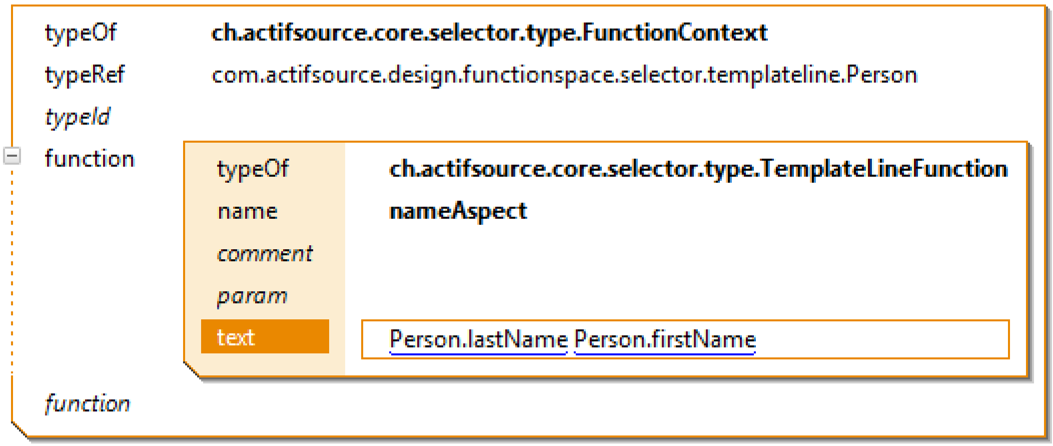
9.3.7 TemplateLineFunction

The *template line function* behaves like a template but without the possibility to set contexts (see Chapter 8.3.9 Line Context, Column Context, Protected Context). The template line allows you to create simple single line texts as for name aspects (see Chapter 4.6.1 ch.actifsource.core.Class).

Consider a resource Person with two string literals firstName and LastName.

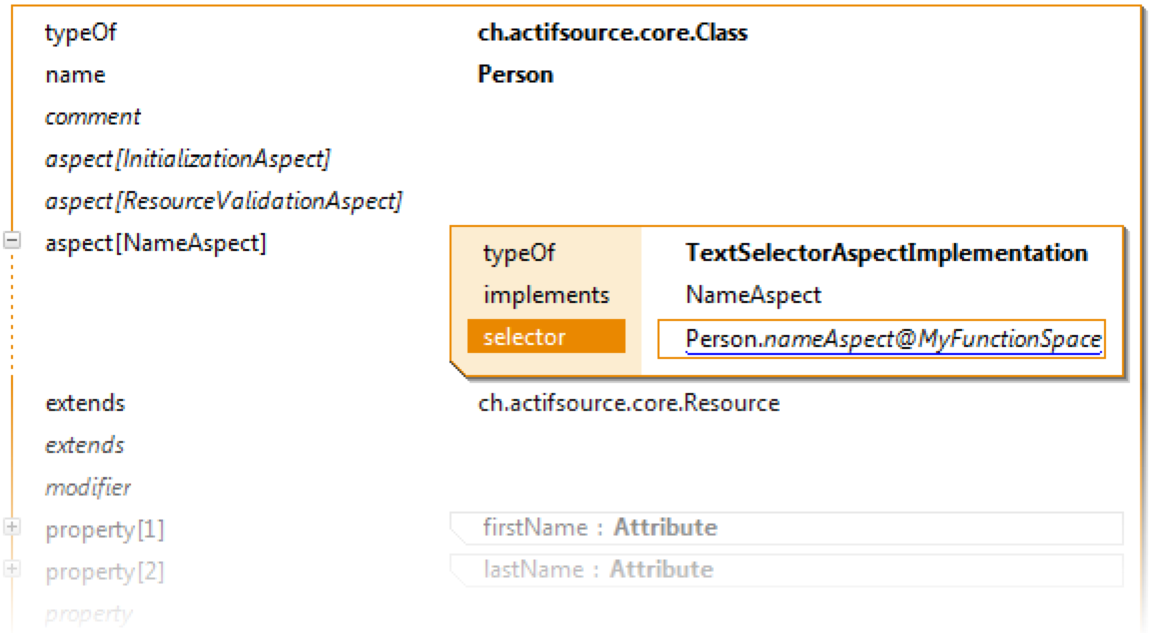


Write a template function for person, which prints out the person's last name and first name.



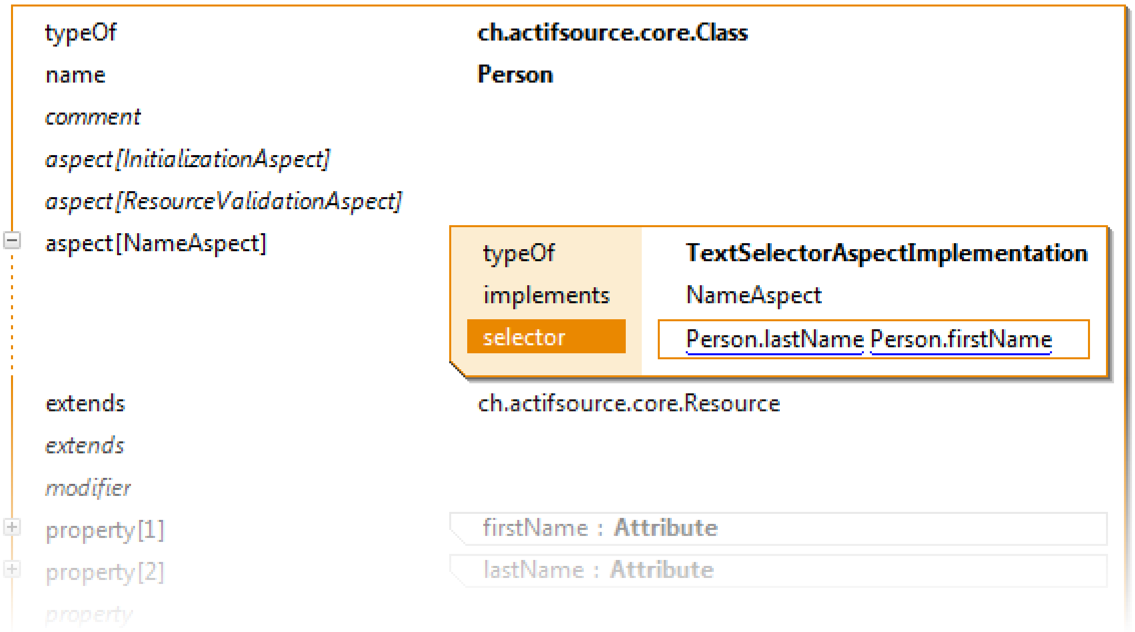
typeOf ch.actifsource.core.selector.type.FunctionContext
typeRef com.actifsource.design.functionspace.selector.templateline.Person
typeId
function
  typeOf ch.actifsource.core.selector.type.TemplateLineFunction
  name nameAspect
  comment
  param
  text Person.lastName Person.firstName
function

Simply use the template line function as name aspect in the class Person.



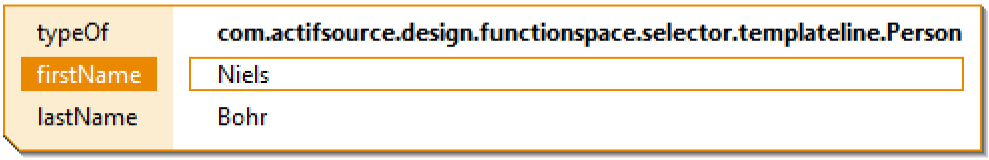
typeOf ch.actifsource.core.Class
name Person
comment
aspect[InitializationAspect]
aspect[ResourceValidationAspect]
aspect[NameAspect]
  typeOf TextSelectorAspectImplementation
  implements NameAspect
  selector Person.*nameAspect@MyFunctionSpace*
extends ch.actifsource.core.Resource
extends
modifier
property[1] firstName : Attribute
property[2] lastName : Attribute
property

Please not that it is also possible that the selector of the name aspect can be used directly as template line.



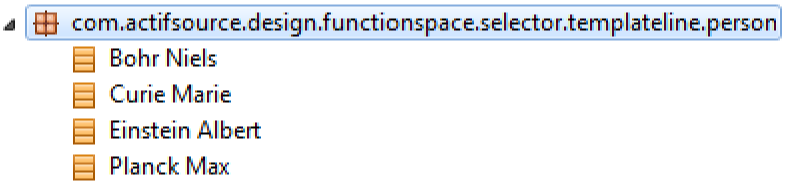
typeOf	**ch.actifsource.core.Class**
name	**Person**
comment	
aspect[InitializationAspect]	
aspect[ResourceValidationAspect]	
aspect[NameAspect]	typeOf **TextSelectorAspectImplementation** implements NameAspect selector Person.lastName Person.firstName
extends	ch.actifsource.core.Resource
extends	
modifier	
property[1]	firstName : **Attribute**
property[2]	lastName : **Attribute**
property	

Note that Person is only a **Resource** but not a **NamedResource**. The attributes firstName and lastName are therefore just normal properties.



typeOf	**com.actifsource.design.functionspace.selector.templateline.Person**
firstName	Niels
lastName	Bohr

Defining the name aspect as seen above synthesizes the name.



com.actifsource.design.functionspace.selector.templateline.person
- Bohr Niels
- Curie Marie
- Einstein Albert
- Planck Max

9.4 Built-in functions

Actifsource provides lots of useful built-in functions.

9.4.1 Built-in functions on Any

Actifsource provides the following built-in functions on **Any**.

Function	Return type	Description
guid	Literal	Gets the unique identifier of any Resource or Literal. (For Resources it is a GUID, for Literals it is the Literal itself.)

9.4.2 Built-in functions on Any List

Actifsource provides the following built-in functions on List of **Any**.

count	IntegerLiteral	Counts the number of elements in the list.
isEmpty	BooleanLiteral	Returns true if and only if the list is empty.
isSet	BooleanLiteral	Returns true if and only if the list contains no duplicates.
first	T	Returns the first element in the list.
last	T	Returns the last element in the list.

count	IntegerLiteral	Counts the number of elements in the list.
reverse	List of T	Reverses the elements in the list.
distinct	List of T	Remove duplicates from a list, first to last.

9.4.3 Built-in functions on Resource

Actifsource provides the following built-in functions on Resource.

Function	Return type	Description
package	String	Returns the package of the resource as string.
guid	String	Returns the GUID of the resource as string.
simpleName	String	Returns the Resource's name as defined by its NameAspect. If the resource extends NamedResource, the NameAspect returns the value of the name attribute. If no NameAspect is defined, the GUID of the resource is returned.

9.4.4 Built-in functions on List of Resource

Actifsource provides the following built-in functions on List of Resource.

Function	Return type	Description
sortByGuid	List of T	Sorts the list of resources by their GUIDs.
sortBySimpleName	List of T	Sorts the list of resources by their names.

9.4.5 Built-in functions on Literal

Actifsource provides the following built-in functions on Literal.

Function	Return type	Description
guid	T	Gets the identifier of the Literal value. This is the Literal itself.

9.4.6 Built-in functions on IntegerLiteral.

Actifsource provides the following built-in functions on IntegerLiteral.

Function	Return type	Description
increment	Integer	Increments an integer number.
decrement	Integer	Decrements an integer number.
notZero	Integer	Returns the number unless it is zero.

9.4.7 Built-in functions on IntegerLiteral.

Actifsource provides the following built-in functions on List of IntegerLiteral.

Function	Return type	Description
sum	Integer	Calculates the sum of a list of integer numbers.
minimum	Integer	Returns the minimum integer in a list.
maximum	Integer	Returns the maximum integer in a list.

9.4.8 Built-in functions on BooleanLiteral

Actifsource provides the following built-in functions on BooleanLiteral.

Function	Return type	Description
isFalse	Boolean	Returns true if the Boolean value is false.

9.4.9 Built-in functions on List of Character

Actifsource provides the following built-in functions on List of Character.

Function	Return type	Description
string	Boolean	Builds a string from characters.

9.4.10 Built-in functions on List of Letter

Actifsource provides the following built-in functions on List of Letter.

Function	Return type	Description
string	Word	Builds a word from letters.

9.4.11 Built-in functions on TextLiteral

Actifsource provides the following built-in functions on TextLiteral.

Function	Return type	Description
suppressIndent	Text	Sets the current intent mode to 'suppress indent': All lines after the first line start at the very beginning of the line. If applied in a template function, the setting of the outer template is not affected. The return value is the text itself.
indent	Text	Sets the indent mode to 'indent' (=default). All lines will start at the same position as the first line. (The preceding characters in the first line are copied, non-whitespace characters replaced by whitespaces.) If applied in a template function, the setting of the outer template is not affected. The return value is the text itself.
prefix	Text	Sets the indent mode to 'prefix'. All lines will repeat the preceding characters in of the first line. If applied in a template function, the setting of the outer template is not affected. The return value is the text itself.
splitLines	List<String>	Splits text at line breaks into a list of strings.
split80	List<String>	Splits text into a list of strings of maximum 80 characters. Words are considered atomic, if possible.
split100	List<String>	Splits text into a list of strings of maximum 100 characters. Words are considered atomic, if possible.
escapedString	String	Escapes the text such that it can be embedded into C, C++ or Java source code. Escaping for C/C++ only works for ASCII characters.
notEmpty	Text	Returns the text unless it is empty.

9.4.12 Built-in functions on StringLiteral

Actifsource provides the following built-in functions on StringLiteral.

Function	Return type	Description
character	List of Character	Gets the characters in the string.
length	Integer	Gets the number of characters in the string.
toFirstUpper	String	Gets the same string with capital first letter.
toFirstLower	String	Gets the same string with small first letter.
toAllUpper	String	Gets the string in all capital letters.
toAllLower	String	Gets the string in all small letters.
camelcapToUnderscore	String	Inserts an underscore before every uppercase letter unless it is the first letter in the string.
whitespaceToCamelcap	String	Replaces letters behind one or many whitespace characters by their uppercase counterparts, replacing those whitespace characters.
whitespaceToUnderscore	String	Replaces all whitespace characters by underscore characters.
split80	List of String	Splits the string into a list of strings of maximally 80 characters.
split100	List of String	Splits the string into a list of strings of maximally 100 characters.
packageToDirectory	String	Replaces '.' by '/'.
isNotEmpty	Boolean	Returns true if and only if the string is not an empty string.
escapedString	String	Escapes the string such that it can be embedded into C, C++ or Java source code. Escaping for C/C++ only works for ASCII

		characters.
part	List of Literal	Parses the string into words, natural numbers and special characters, removing whitespaces.

9.4.13 Built-in functions on Word

Actifsource provides the following built-in functions on Build.

Function	Return type	Description
character	Letter	Gets the letters in the word.

9.4.14 Built-in functions on Guid

Actifsource provides the following built-in functions on Build.

Function	Return type	Description
timestamp	Long	Returns the GUID's timestamp in 100 nanoseconds starting from Oct 15, 1582.
time	Time	Returns the GUID's time.
identify	Resource	Returns the Resource identified by the GUID.

9.4.15 Built-in functions on Build

Actifsource provides the following built-in functions on Build.

Function	Return type	Description
once	Build	Used in the selector of the template. Build.once means a template is not based on a resource but only built once.

9.4.16 Built-in functions on LinkSelector

Actifsource provides the following built-in functions on LinkSelector.

Function	Return type	Description
selectorText	String	Converts a selector to simple text string.
selectorResultType	AbstractType	Calculates a selectors result type.

9.4.17 Built-in functions on File

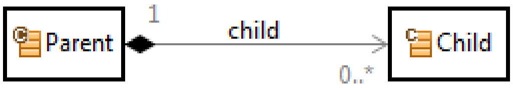
Actifsource provides the following built-in functions on File.

Function	Return type	Description
contents	Text	Returns a file's contents.

9.5 Accessing the model from within Java function

It is possible to access the model, other functions (see chapter 9.3 Function types), or even built-in functions (see chapter **Fehler! Verweisquelle konnte nicht gefunden werden. Fehler! Verweisquelle konnte nicht gefunden werden.**) from within Java functions.

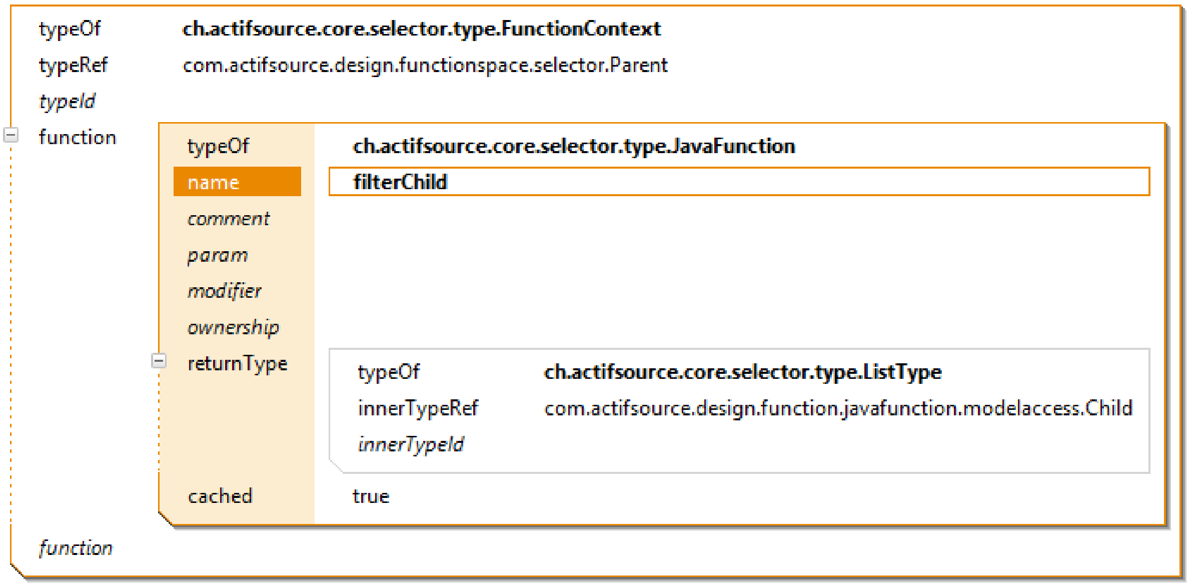
Consider the following meta-model for the subsequent examples:



9.5.1 Model forward access

Let’s write a Java Function for Parent which returns only instances of type Child with names beginning with “A”.

Start by declaring a Java Function named *filterChild* as seen in Chapter 9.3.3.



typeOf	**ch.actifsource.core.selector.type.FunctionContext**
typeRef	com.actifsource.design.functionspace.selector.Parent
typeId	
function	

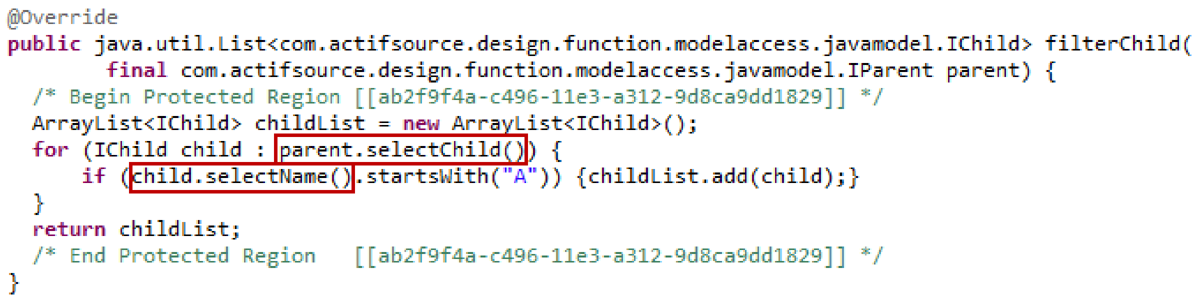
typeOf	**ch.actifsource.core.selector.type.JavaFunction**
name	**filterChild**
comment	
param	
modifier	
ownership	
returnType	
cached	true

typeOf	**ch.actifsource.core.selector.type.ListType**
innerTypeRef	com.actifsource.design.function.javafunction.modelaccess.Child
innerTypeId	

function

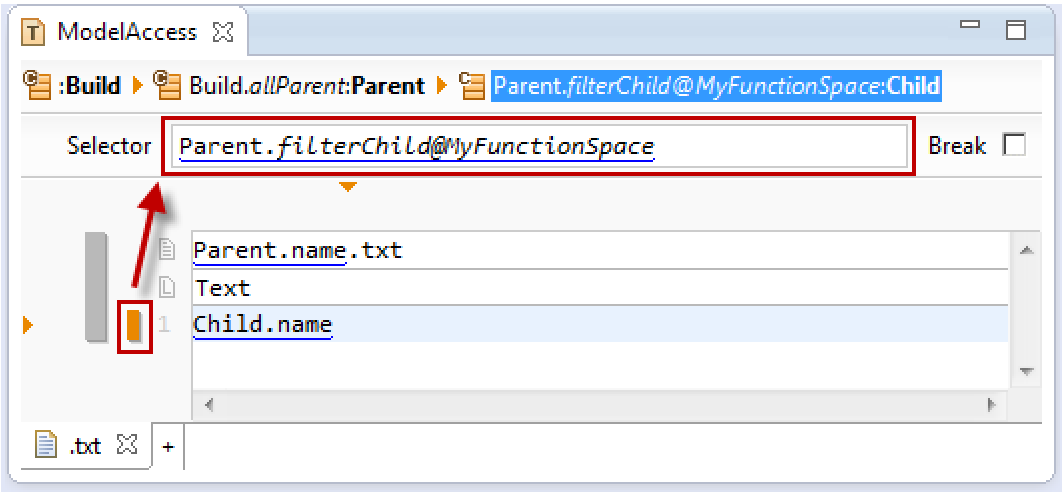
For the *model forward access* use the select*Property*() function on the given resource where *property* is the property to select.

The subsequent filter function iterates over the relation Parent.child in the *for*-Statement via parent.selectChild(). Then we check if Child.name starts with “A” via child.selectName(). If the condition is fulfilled we add the filtered child to the child list. At the end we return the child list with the filtered children.



```
@Override
public java.util.List<com.actifsource.design.function.modelaccess.javamodel.IChild> filterChild(
        final com.actifsource.design.function.modelaccess.javamodel.IParent parent) {
  /* Begin Protected Region [[ab2f9f4a-c496-11e3-a312-9d8ca9dd1829]] */
  ArrayList<IChild> childList = new ArrayList<IChild>();
  for (IChild child : parent.selectChild()) {
      if (child.selectName().startsWith("A")) {childList.add(child);}
  }
  return childList;
  /* End Protected Region   [[ab2f9f4a-c496-11e3-a312-9d8ca9dd1829]] */
}
```

You might use the filter function in the selector of a template or in any other function. The following template only prints children with names starting with “A”.



Selector Parent.filterChild@MyFunctionSpace Break □

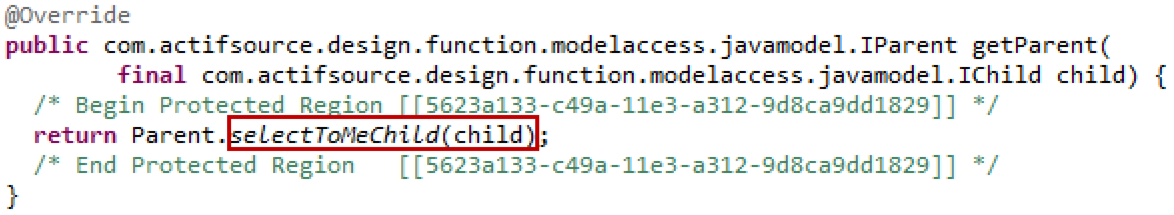
```
Parent.name.txt
Text
Child.name
```

9.5.2 Model backward access

Using the selector syntax accessing the model backwards is quite easy (see chapter 9.3.2 SelectorFunction) by the minus sign. Accessing the model backwards is also possible in the Java code.

For the *model backward access* use the *static function* selectToMe*Property*() on the class which defines the relation.

Since we want to access the relation Parent.child backwards we have to choose the static method Parent.selectToMeChild() providing the actual child as parameter. As a result we get the parent of the given child.



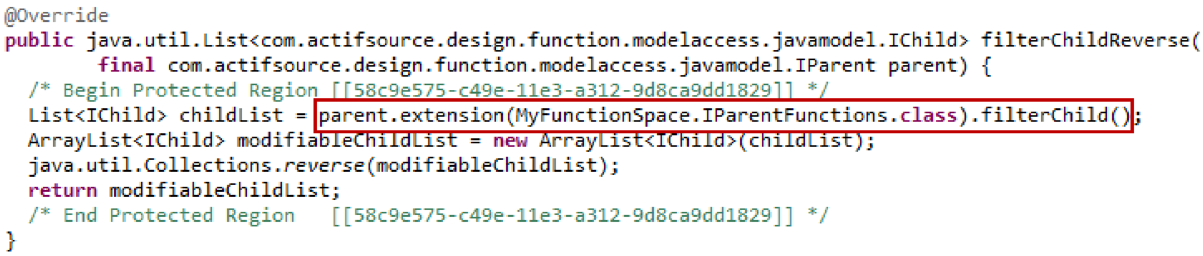
```
@Override
public com.actifsource.design.function.modelaccess.javamodel.IParent getParent(
        final com.actifsource.design.function.modelaccess.javamodel.IChild child) {
  /* Begin Protected Region [[5623a133-c49a-11e3-a312-9d8ca9dd1829]] */
  return Parent.selectToMeChild(child);
  /* End Protected Region   [[5623a133-c49a-11e3-a312-9d8ca9dd1829]] */
}
```

9.5.3 Function access

Use the *extension mechanism* to access any of your functions from within Java Code. Note that the *extension mechanism* also supports polymorphic calls (see chapter **Fehler! Verweisquelle konnte nicht gefunden werden. Fehler! Verweisquelle konnte nicht gefunden werden.**).

Let's assume that we have a function *filterChild* as shown in chapter 9.5.1 Model forward access. Let's write a Java function *filterChildReverse* which returns a reverse list of the filtered children based on *filterChild*.

For the *function access* use the extension() function on the given resource with ***FunctionSpace***.***ITypeRef****Functions.class* as parameter. FunctionSpace is the function space where your function is defined. TypeRef is FunctionContext.typeRef. The static property *.class* is given from Java and represents a class as an object.



```
@Override
public java.util.List<com.actifsource.design.function.modelaccess.javamodel.IChild> filterChildReverse(
        final com.actifsource.design.function.modelaccess.javamodel.IParent parent) {
  /* Begin Protected Region [[58c9e575-c49e-11e3-a312-9d8ca9dd1829]] */
  List<IChild> childList = parent.extension(MyFunctionSpace.IParentFunctions.class).filterChild();
  ArrayList<IChild> modifiableChildList = new ArrayList<IChild>(childList);
  java.util.Collections.reverse(modifiableChildList);
  return modifiableChildList;
  /* End Protected Region   [[58c9e575-c49e-11e3-a312-9d8ca9dd1829]] */
}
```

9.5.4 Built-in function access

9.5.5 Use the *extension mechanism* to access built-in functions on resources (see chapter 9.4.1 Built-in functions on Any

Actifsource provides the following built-in functions on **Any**.

Function	Return type	Description
guid	Literal	Gets the unique identifier of any Resource or Literal. (For Resources it is a GUID, for Literals it is the Literal itself.)

9.5.6 Built-in functions on Any List

Actifsource provides the following built-in functions on List of **Any**.

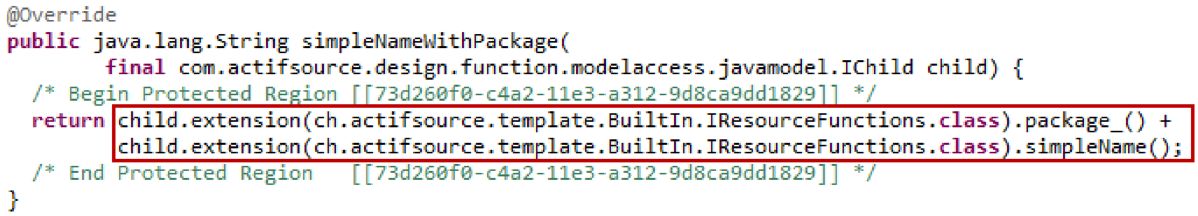
count	**IntegerLiteral**	**Counts the number of elements in the list.**
isEmpty	BooleanLiteral	Returns true if and only if the list is empty.
isSet	BooleanLiteral	Returns true if and only if the list contains no duplicates.

first	T	Returns the first element in the list.
last	T	Returns the last element in the list.
count	IntegerLiteral	Counts the number of elements in the list.
reverse	List of T	Reverses the elements in the list.
distinct	List of T	Remove duplicates from a list, first to last.

Built-in functions on Resource) from within Java Code. Accessing built-in functions is done the same way as seen in the above chapter 9.5.3 Function access.

To write a function which returns the package and the simpleName of a child we can reuse the built-in functions *package()* and *simpleName()*.

For the *built-in function access* use the extension() function on the given resource with ***Built-in**.I**Resource**Functions.class* as parameter.



```
@Override
public java.lang.String simpleNameWithPackage(
        final com.actifsource.design.function.modelaccess.javamodel.IChild child) {
  /* Begin Protected Region [[73d260f0-c4a2-11e3-a312-9d8ca9dd1829]] */
  return child.extension(ch.actifsource.template.BuiltIn.IResourceFunctions.class).package_() +
         child.extension(ch.actifsource.template.BuiltIn.IResourceFunctions.class).simpleName();
  /* End Protected Region   [[73d260f0-c4a2-11e3-a312-9d8ca9dd1829]] */
}
```

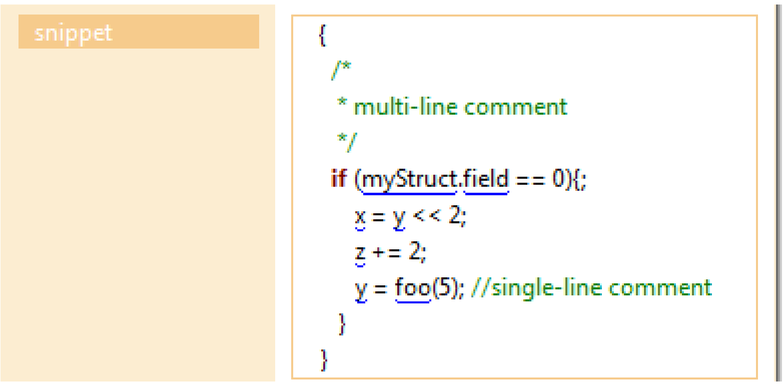
Note that you cannot access built-in functions for literals via the *extension mechanism*.

10 Code Snippets

10.1 Overview

Actifsource supports a special editor, the so-called Code Snippet editor, which allows the user to insert Actifsource resources as variables or functions into snippets of source code. Such a snippet of code is essentially a list of statements written in C-, where C- is a subset of the programming language ANSI C. A more precise definition of C- will be given below. Actifsource then provides the possibility to generate code in an arbitrary target language from these code snippets. This is achieved by parsing the input code according to the grammar of C- and applying either built-in or user-provided templates to the resulting parse tree. This parse tree is actually a model composed of temporary (i.e. non-persistent) resources.

The following example shows a code snippet where the underlined identifiers are resources used as variables and functions:

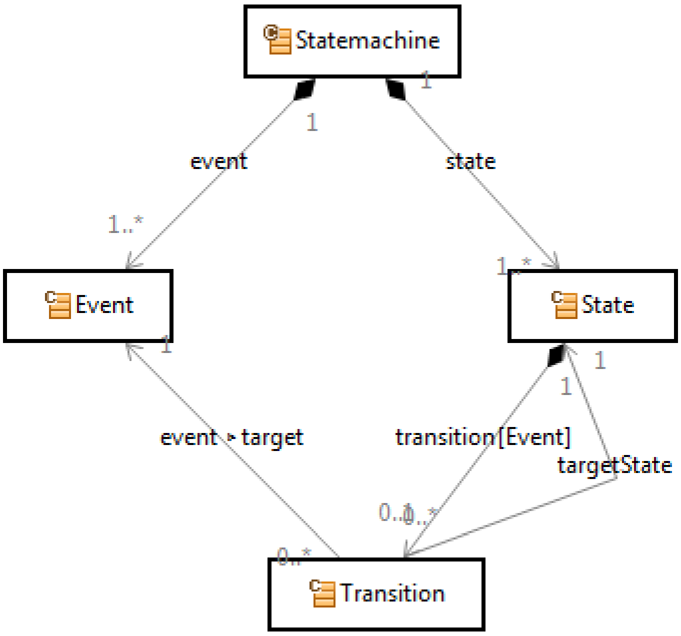


```
{
  /*
   * multi-line comment
   */
  if (myStruct.field == 0){;
     x = y << 2;
     z += 2;
     y = foo(5); //single-line comment
  }
}
```

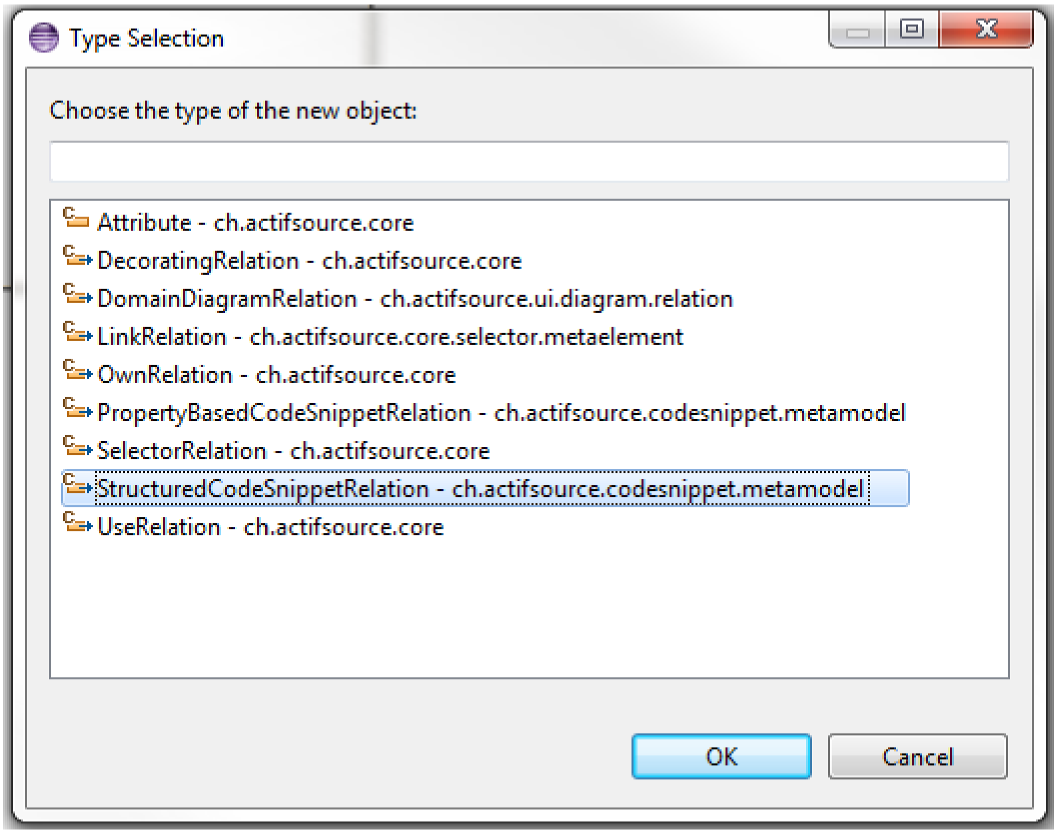
10.2 Defining Code Snippet Relations

First, we will show how to add a code snippet relation to a class and define all the necessary properties of this relation. Such a relation enables the code snippet editor on instances of this class and allows a user to add code written in the chosen input language to the resource.

Most of the examples and screenshots in the following are taken from the Code Snippet tutorial available at http://www.actifsource.com/tutorials/index.html. This tutorial is based on the following meta-model for statemachines and shows how to add code snippets for conditions on transitions and for actions taken when a condition is executed.



To add code snippets to resources of type MyClass, you have to edit MyClass in the resource editor, add a new property and choose the type StructuredCodeSnippetRelation for the relation in the Type Selection dialog:


Type Selection

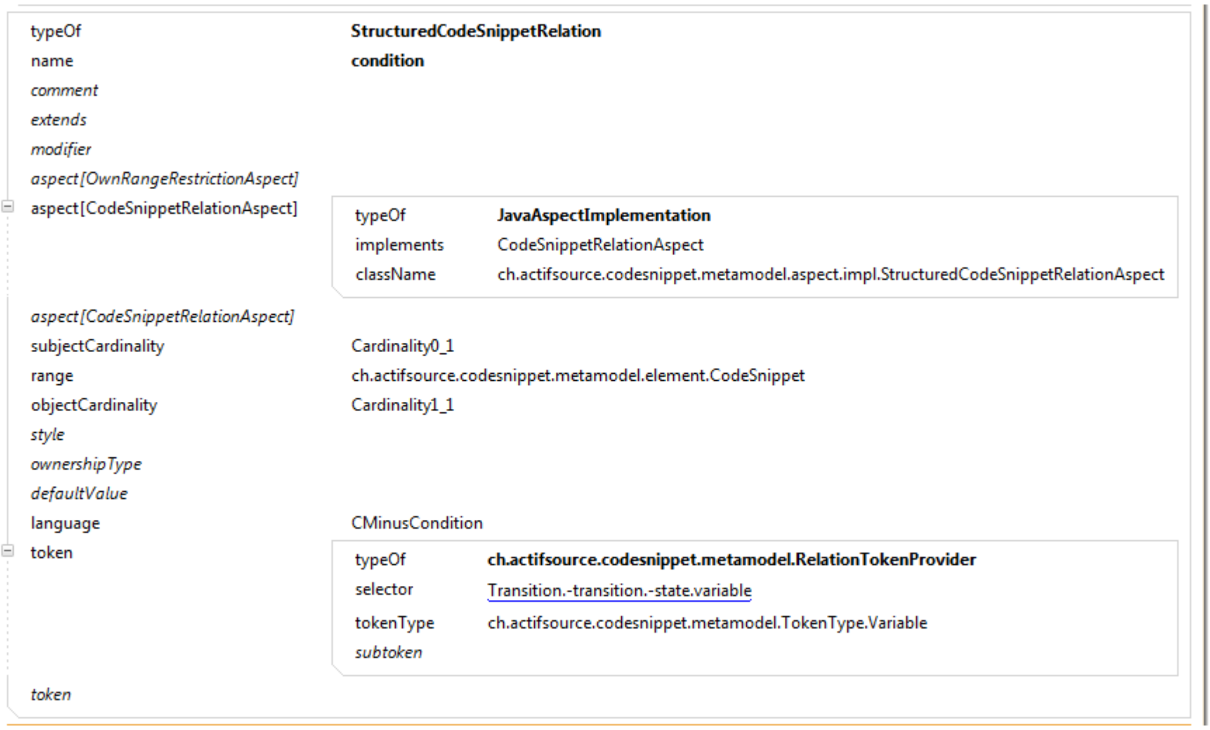
Choose the type of the new object:

- Attribute - ch.actifsource.core
- DecoratingRelation - ch.actifsource.core
- DomainDiagramRelation - ch.actifsource.ui.diagram.relation
- LinkRelation - ch.actifsource.core.selector.metaelement
- OwnRelation - ch.actifsource.core
- PropertyBasedCodeSnippetRelation - ch.actifsource.codesnippet.metamodel
- SelectorRelation - ch.actifsource.core
- StructuredCodeSnippetRelation - ch.actifsource.codesnippet.metamodel
- UseRelation - ch.actifsource.core

In the resulting property, you have to create the following statements: subjectObjectCardinality, objectCardinality and name as for Own- or UseRelations (see Section 4.3).

Additionally, we define the CodeSnippetRelationAspect as shown below with the class

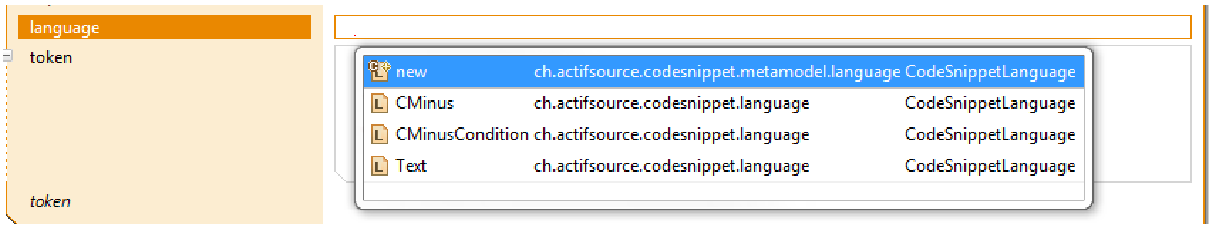
ch.actifsource.codesnippet.metamodel.aspect.impl.StructuredCodeSnippetRelationAspect.



typeOf	**StructuredCodeSnippetRelation**
name	**condition**
comment	
extends	
modifier	
aspect[OwnRangeRestrictionAspect]	
aspect[CodeSnippetRelationAspect]	typeOf **JavaAspectImplementation** implements CodeSnippetRelationAspect className ch.actifsource.codesnippet.metamodel.aspect.impl.StructuredCodeSnippetRelationAspect
aspect[CodeSnippetRelationAspect]	
subjectCardinality	Cardinality0_1
range	ch.actifsource.codesnippet.metamodel.element.CodeSnippet
objectCardinality	Cardinality1_1
style	
ownershipType	
defaultValue	
language	CMinusCondition
token	typeOf **ch.actifsource.codesnippet.metamodel.RelationTokenProvider** selector Transition.-transition.-state.variable tokenType ch.actifsource.codesnippet.metamodel.TokenType.Variable *subtoken*
token	

10.2.1 Language

Next, we choose an input language for the code snippet.



language

token

token

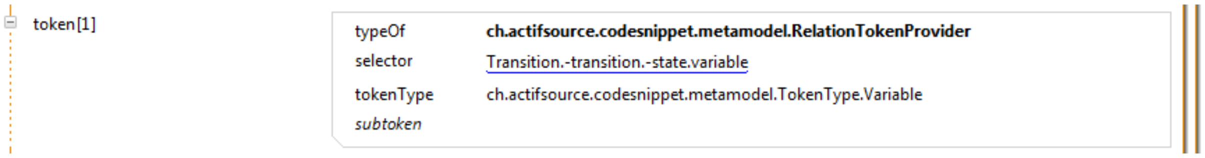
new	ch.actifsource.codesnippet.metamodel.language	CodeSnippetLanguage
CMinus	ch.actifsource.codesnippet.language	CodeSnippetLanguage
CMinusCondition	ch.actifsource.codesnippet.language	CodeSnippetLanguage
Text	ch.actifsource.codesnippet.language	CodeSnippetLanguage

The input language is used to check the input code syntactically and to highlight keywords of the language. Furthermore, the input language defines which parser will be applied to the input code when generating output code from the code snippet. See Section 10.3 for a description of the available input languages.

10.2.2 Tokens

Finally, we need to define which resources will be available as functions and variables in the code snippet editor. This is done by creating one or more token statements referring to a RelationTokenProvider. The RelationTokenProvider allows the user to define a selector (cf. Section 8.3.12) which defines a list of resources. Additionally, it allows you to choose a tokenType which is either

- ch.actifsource.codesnippet.metamodel.TokenType.Variable for variables or
- ch.actifsource.codesnippet.metamodel.TokenType.Function for functions.



token[1]

typeOf ch.actifsource.codesnippet.metamodel.RelationTokenProvider
selector Transition.-transition.-state.variable
tokenType ch.actifsource.codesnippet.metamodel.TokenType.Variable
subtoken

For variables you can define sub-tokens. The definition of sub-tokens instructs the Content Assist to propose all resources defined by the selector of the sub-token when you insert a '.' after a token. This means that sub-tokens can be used to insert resources as identifiers of fields in structs where the token corresponds to the struct and the sub-token to the field (for details see Section 0 below).

10.3 Input Languages

At the moment, the following languages are available: C-, CMinusCondition and Text.

10.3.1 C-

The language C- is a (proper) subset of the language ANSI C. It has the following restrictions:

1. C- supports no declarations (of variables, functions or types)
2. C- does not support the use of pointers and addresses
3. C- does not support type casts
4. C- does not support conditional statements (Expression ? Expression : Expression). However, they can easily be replaced by equivalent if-statements.
5. The comma operators is not supported, i.e., expressions such as
    a. X = 2, z = 42;
    b. Foo(x,(y=2,y));

    are not valid.
6. Postfix and prefix increment and decrement operators ('++','--') are not supported.

C- supports access to fields of structs. As identifiers for fields either variables or arbitrary identifiers are valid. The Content Assist provides proposals for fields of structs if the variable definitions are created accordingly (see Section 10.4.1).

C- knows the following list of keywords: **break, else, switch, return, continue, for, default, do, if, while, until, case.**

Furthermore, the language knows the following operators:

Infix-Operators: { ||, &&, |, ^, &, ==, !=, <, >, <=, >=, <<, >>, +, -, *, /, %}

Prefix-Operators: { !, ~, +, -}

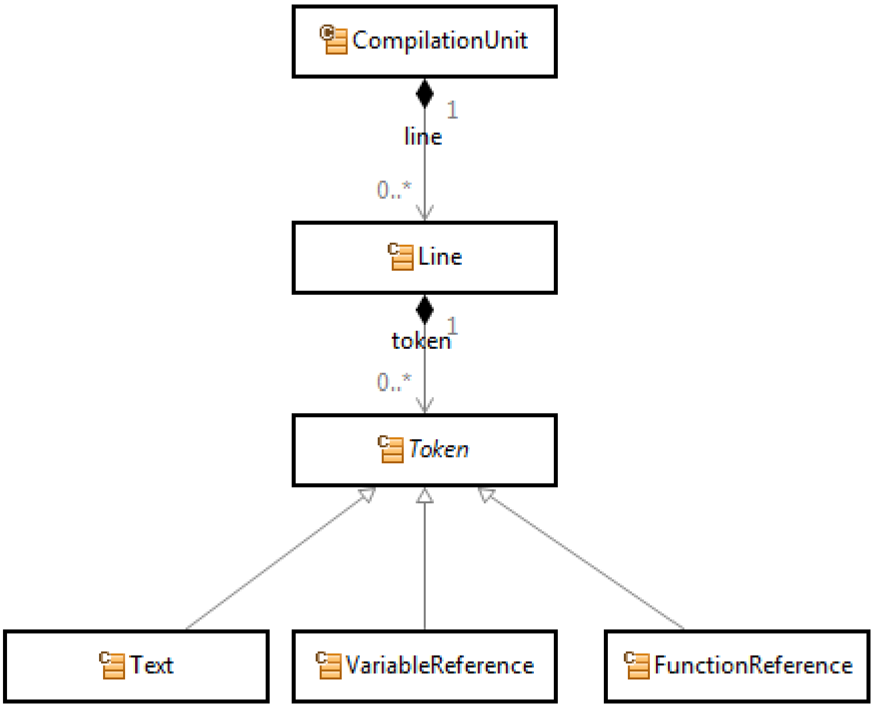
Assignment-Operators: {= =, *=, /= , %=, +=, -= , <<= , >>= , &=, ^= , |=}

10.3.2 CminusCondition

A code snippet with CMinusCondition allows the user to input a conditional expression as in ANSI C (e.g. a relational or equality expression) while the same restrictions apply as for C- (details see Section 10.3.1). Such an expression can then be used for example as the condition in an if-statement.

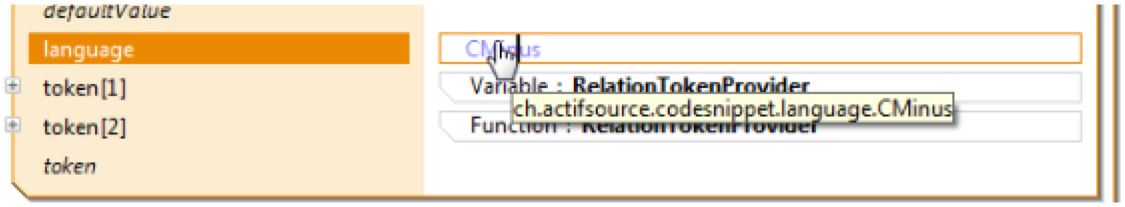
10.3.3 Text

Text allows the user to insert arbitrary text with resources added as either variables or functions. This language should only be used if C- is too restrictive and validation of the input code is not required. The input code is syntactically not validated[1] and the resulting parse tree is very simple:



10.4 Code Snippet Editor

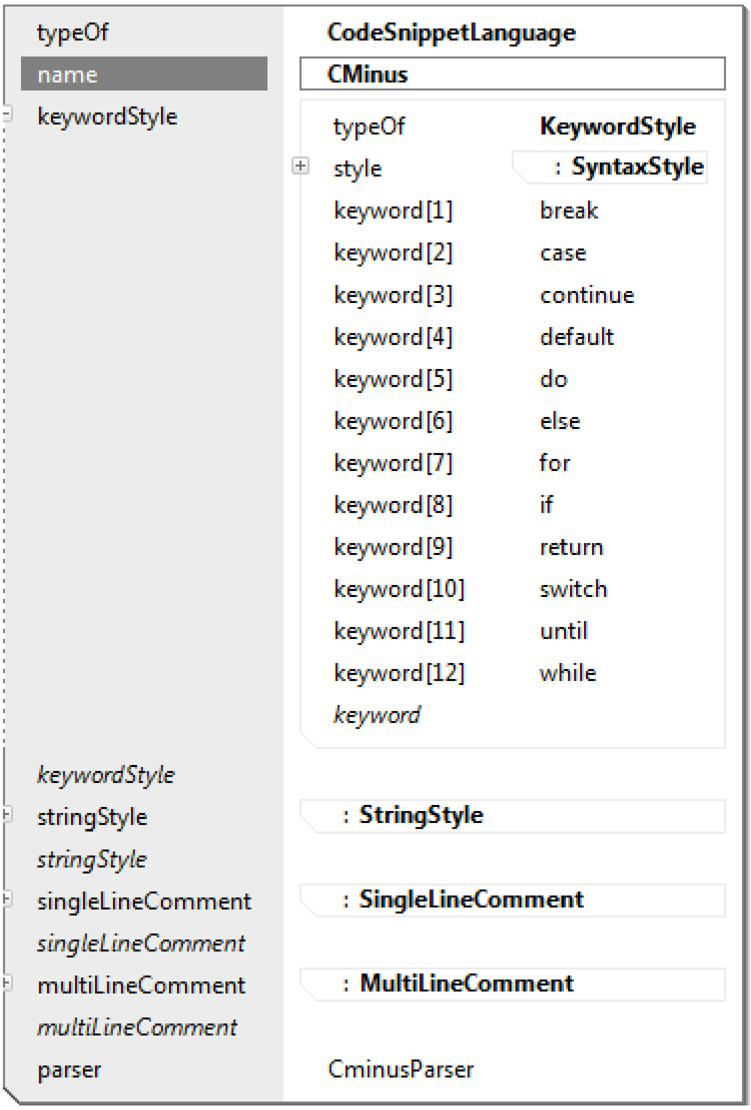
The code snippet editor is available in the resource editor for any property of type StructuredCodeSnippetRelation. The editor supports multi-line input. It highlights keywords and comments according to the language property of the StructureCodeSnippetRelation. You can inspect the definition of the input language by CTRL+Left-Click on the chosen language:


defaultValue
language
token[1]
token[2]
token

CMinus
Variable : RelationTokenProvider
ch.actifsource.codesnippet.language.CMinus
Function : RelationTokenProvider

This opens the chosen language in the resource editor and allows you to browse the properties such as keywords and style of comments of the language:

[1] Actually, it is validated, but it has a very simple and non-restrictive grammar.



typeOf	CodeSnippetLanguage	
name	CMinus	
keywordStyle	typeOf	KeywordStyle
	style	: SyntaxStyle
	keyword[1]	break
	keyword[2]	case
	keyword[3]	continue
	keyword[4]	default
	keyword[5]	do
	keyword[6]	else
	keyword[7]	for
	keyword[8]	if
	keyword[9]	return
	keyword[10]	switch
	keyword[11]	until
	keyword[12]	while
	keyword	
keywordStyle		
stringStyle	: StringStyle	
stringStyle		
singleLineComment	: SingleLineComment	
singleLineComment		
multiLineComment	: MultiLineComment	
multiLineComment		
parser	CminusParser	

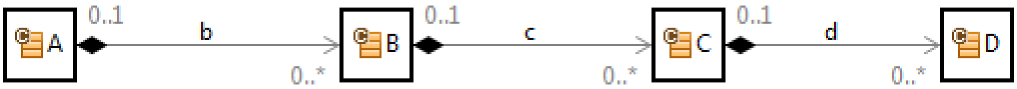
10.4.1 Content Assist

By using CTRL+Space in the code snippet editor you can as usual call the Content Assist. The Content Assist will show you all available resources to insert as functions and variables. The set of available resources is defined by the property token on the corresponding StructuredCodeSnippetRelation (see Section10.2). Inserted resources are underlined with blue color.

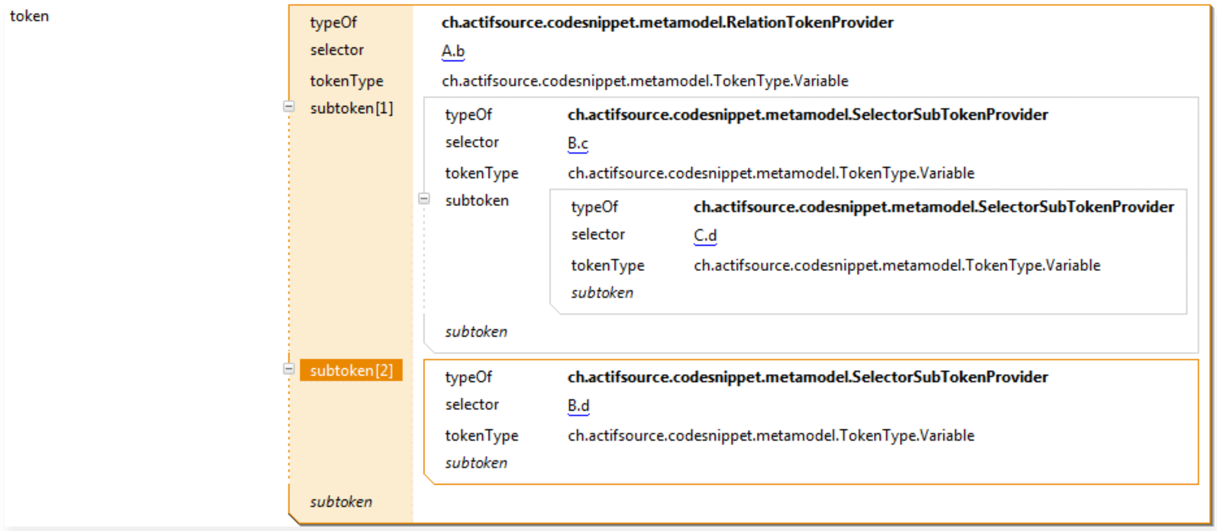
Structures

The StructuredCodeSnippetRelation supports the definition of complex and nested data types such as structs by defining a set of subtokens for a token. After inserting an instance of a token into the code snippet editor followed by a '.' (struct field access in C-), the Content Assist will propose the list of all resources defined by the selectors of its sub-tokens.

Consider for example the following meta-model:

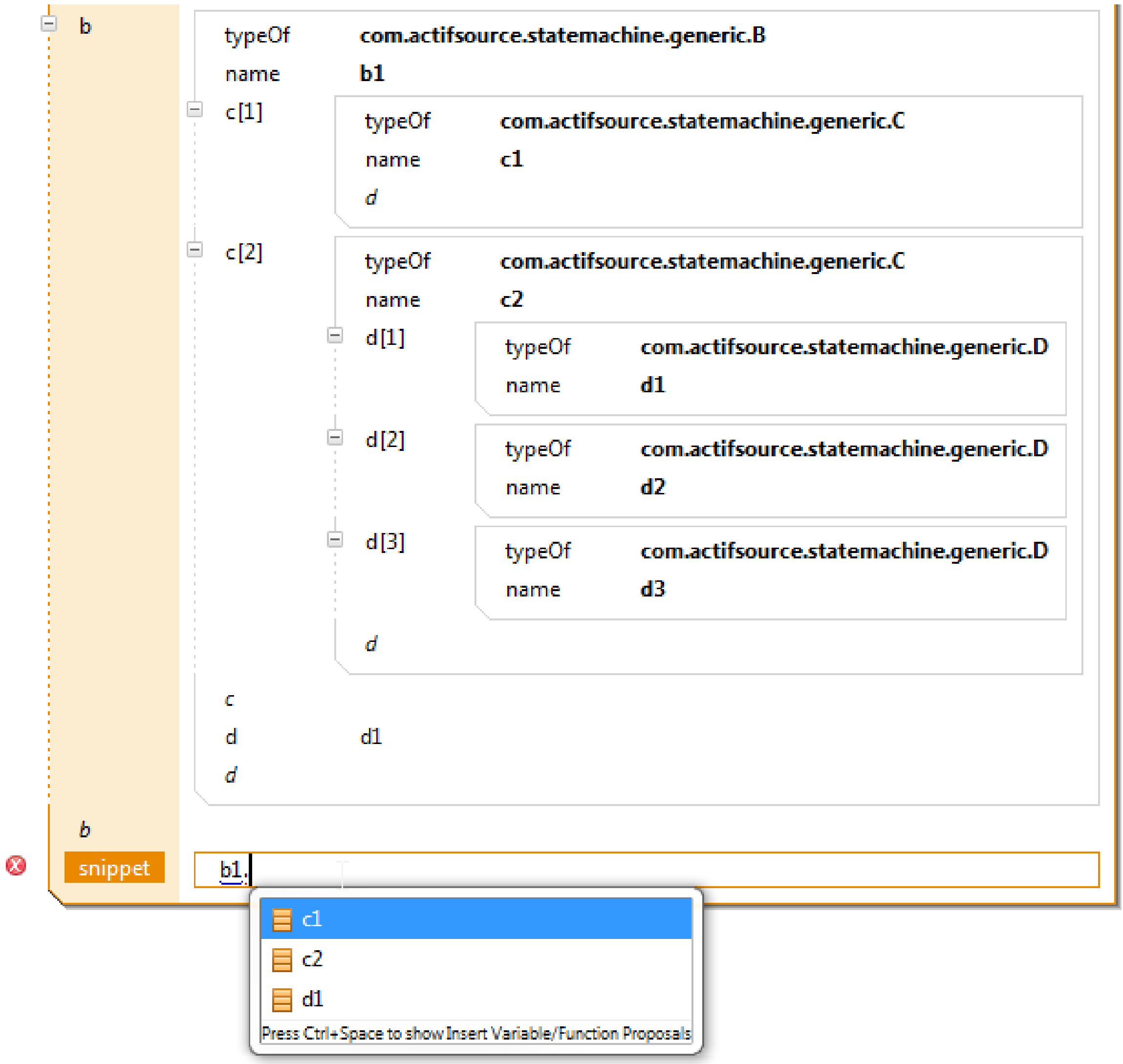


Furthermore, we consider a StructuredCodeSnippetRelation on class A with the following definition of tokens:



```
token   typeOf        ch.actifsource.codesnippet.metamodel.RelationTokenProvider
        selector      A.b
        tokenType     ch.actifsource.codesnippet.metamodel.TokenType.Variable
        subtoken[1]   typeOf      ch.actifsource.codesnippet.metamodel.SelectorSubTokenProvider
                      selector    B.c
                      tokenType   ch.actifsource.codesnippet.metamodel.TokenType.Variable
                      subtoken    typeOf      ch.actifsource.codesnippet.metamodel.SelectorSubTokenProvider
                                  selector    C.d
                                  tokenType   ch.actifsource.codesnippet.metamodel.TokenType.Variable
                                  subtoken
                      subtoken
        subtoken[2]   typeOf      ch.actifsource.codesnippet.metamodel.SelectorSubTokenProvider
                      selector    B.d
                      tokenType   ch.actifsource.codesnippet.metamodel.TokenType.Variable
                      subtoken
        subtoken
```

We can now create a resource a1 of type A and insert code into the code snippet editor. When calling the Content Assist after inserting b1., it proposes the list c1, c2 (available through the Selector B.c) and d1 (available through the Selector B.d). See first screenshot below. Note that it is also syntactically correct to insert arbitrary strings as identifiers for fields of structs. After inserting one or more such identifiers followed by a '.', the Content Assist will again propose the list of root elements (tokens) independently of possible resources before the string (see second screenshot below).
It is not possible to define structures recursively. Thus, you have to explicitly define the whole structure to the desired (finite) depth if it is self-referential.



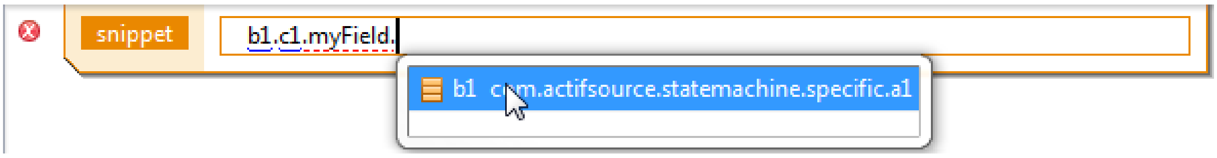
b

typeOf com.actifsource.statemachine.generic.B
name b1
c[1]
typeOf com.actifsource.statemachine.generic.C
name c1
d
c[2]
typeOf com.actifsource.statemachine.generic.C
name c2
d[1]
typeOf com.actifsource.statemachine.generic.D
name d1
d[2]
typeOf com.actifsource.statemachine.generic.D
name d2
d[3]
typeOf com.actifsource.statemachine.generic.D
name d3
d
c
d d1
d

b

snippet b1.

c1
c2
d1
Press Ctrl+Space to show Insert Variable/Function Proposals

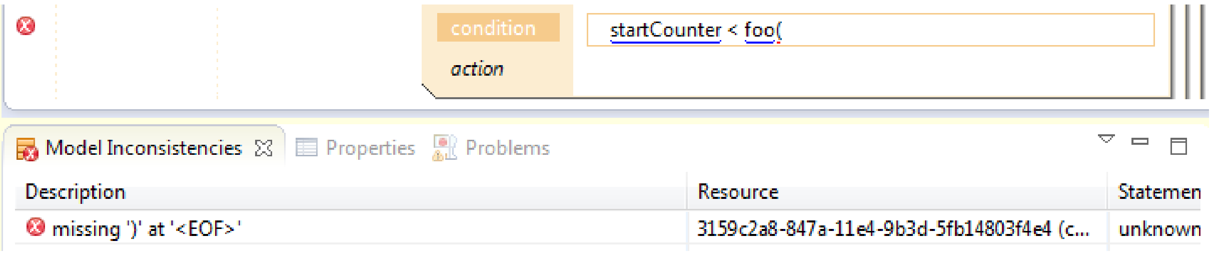


snippet b1.c1.myField.

b1 com.actifsource.statemachine.specific.a1

10.4.2 Validation and Errors

The syntax of the input code in the code snippet editor is continuously validated. Syntax errors are shown by underlining the errors in the code and adding an error description to the Model Inconsistencies view:



condition startCounter < foo(
action

Description	Resource	Statemen
missing ')' at '<EOF>'	3159c2a8-847a-11e4-9b3d-5fb14803f4e4 (c...	unknown

10.5 Code Generation

In this section, we will show how to generate code in an arbitrary target language from code snippets. As explained in the introduction of this chapter, the input code of a code snippet is parsed by a parser that depends on the chosen input language (e.g. C- or Text). From the resulting parse tree, Actifsource generates a model that is composed of temporary resources. The meta-model for parse trees of the language C- is available at http://www.actifsource.com/manuals/index.html. The parse tree for unvalidated input (Text) can be found in Section 10.3.3.

To generate output files from these temporary models, we can apply code templates or template functions to these temporary resources, i.e., the temporary resources behave in exactly the same way as persistent resources except that they are not shown in the resource browser (Project Explorer) and are deleted when the session ends (e.g. the project or the workspace is closed). To use the content of code snippets in templates, you can either use the built-in TemplateFunctions (see Section 10.5.1) or write your own templates or template functions by modifying the built-in templates or writing them from scratch. The behavior of the built-in template functions can be customized by overwriting the way names of variables and functions are created (see Section Overwrite Variable and Function Names (Name Provider) below).

10.5.1 Built-in Template Functions

Actifsource provides the following built-in template functions which are defined on resources of type ch.actifsource.codesnippet.metamodel.element.CodeSnippet:

- codeSnippetToST: generates Structured Text from a CodeSnippet with input language CMinus or CMinusCondition.
- codeSnippetToC: generates C code from a CodeSnippet with input language CMinus or CMinusCondition.
- codeSnippetToText: generates code from a CodeSnippet with input language Text (unvalidated code).
- codeSnippetToFormattedC: generates formatted C code (HTML) from a CodeSnippet with input language CMinus or CMinusCondition (the actual code is the p)
- codeSnippetToVHDL: generates VHDL code from a generates C code from a CodeSnippet with input language CMinus or CMinusCondition

For all the above template functions, the names of variables and functions in the output code are generated by calling the function simpleName@BuiltIn on the corresponding resource.

Note that the above template functions which are written for input languages CMinus and CMinusCondition can also be applied to Text. The output is the same as when calling the function codeSnippetToText in this case.

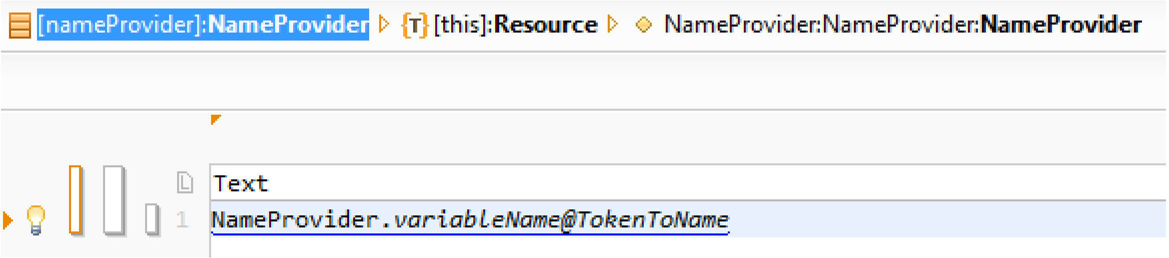
Overwrite Variable and Function Names (Name Provider)

Generating the names of variables and functions by calling simpleName@BuiltIn, is in practice not always sufficient to generate code that meets all the requirements (the requirements on the naming could depend on naming conventions of the target language or names of variables could depend on the context in which the corresponding resource is used). For these cases, Actifsource provides more flexible template functions which takes a Literal of type ch.actifsource.codesnippet.metamodel.parsetree.template.NameProvider as an additional parameter:

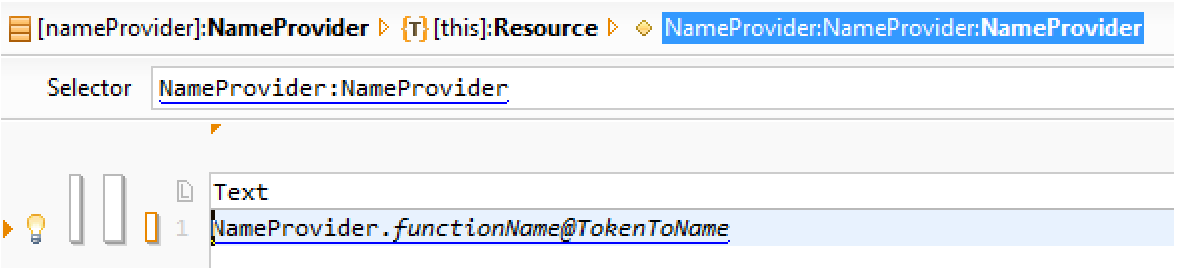
- codeSnippetToSTwithNameProvider
- codeSnippetToCwithNameProvider
- codeSnippetToText
- codeSnippetToFormattedC
- codeSnippetToVHDL

Apart from the generation of function and variable names, these templates have exactly the same behavior as the corresponding template functions from the section above.

The additional parameter to this functions can be used to store additional context information necessary to generate the names and to overwrite the functions used to generate the names, namely variableName@TokenToName and functionName@TokenToName. These two functions take a parameter of type Resource (the resource corresponding to the variable resp. function) and generate the name by calling NameProvider.variableName@TokenToName resp. NameProvider.functionName@TokenToName:


[nameProvider]:NameProvider
{T} [this]:Resource
NameProvider:NameProvider:NameProvider
Text

```
NameProvider.variableName@TokenToName
```


[nameProvider]:NameProvider
{T} [this]:Resource
NameProvider:NameProvider:NameProvider
Text

Selector NameProvider:NameProvider

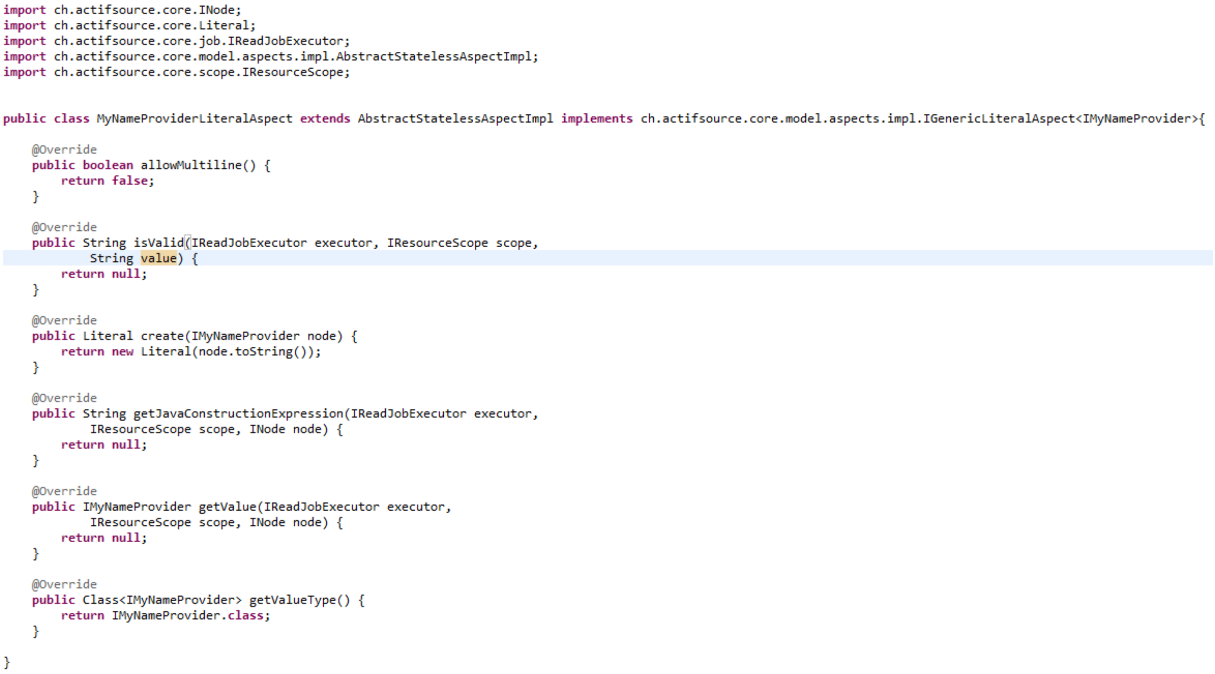
```
NameProvider.functionName@TokenToName
```

The Code Snippet tutorial available at http://www.actifsource.com/tutorials/index.html show how to implement a NameProvider by guiding you step-by-step through an example.

Implement a custom NameProvider

In general, one can implement and use a NameProvider as follows:

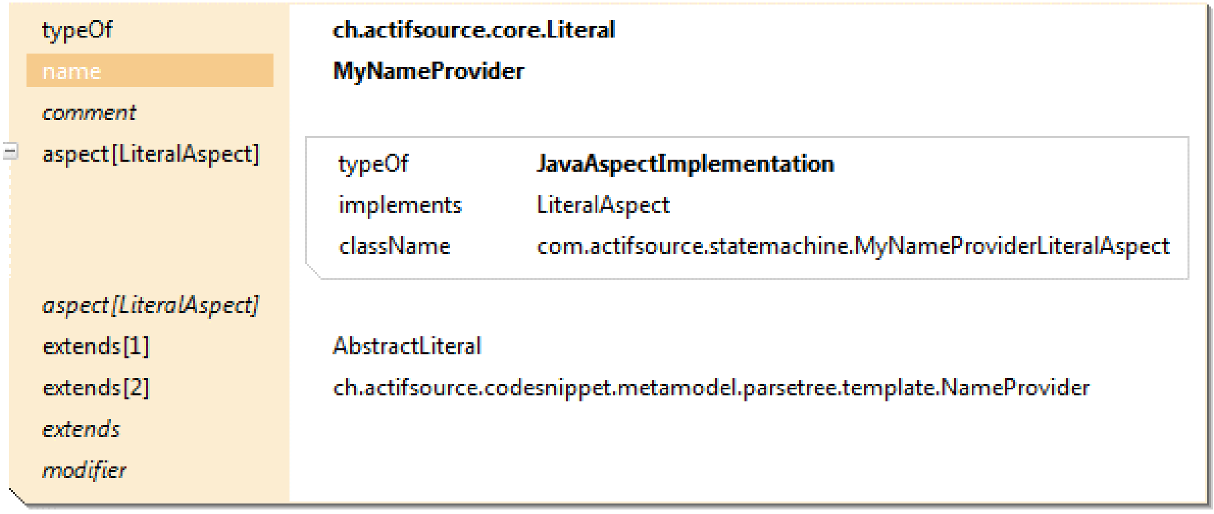
1. Write a LiteralAspect (e.g. MyNameProviderLiteralAspect) in Java which implements ch.actifsource.core.model.aspects.impl.IGenericLiteralAspect<MyNameProviderLiteralAspect>.



```
import ch.actifsource.core.INode;
import ch.actifsource.core.Literal;
import ch.actifsource.core.job.IReadJobExecutor;
import ch.actifsource.core.model.aspects.impl.AbstractStatelessAspectImpl;
import ch.actifsource.core.scope.IResourceScope;


public class MyNameProviderLiteralAspect extends AbstractStatelessAspectImpl implements ch.actifsource.core.model.aspects.impl.IGenericLiteralAspect<IMyNameProvider>{

    @Override
    public boolean allowMultiline() {
        return false;
    }

    @Override
    public String isValid(IReadJobExecutor executor, IResourceScope scope,
            String value) {
        return null;
    }

    @Override
    public Literal create(IMyNameProvider node) {
        return new Literal(node.toString());
    }

    @Override
    public String getJavaConstructionExpression(IReadJobExecutor executor,
            IResourceScope scope, INode node) {
        return null;
    }

    @Override
    public IMyNameProvider getValue(IReadJobExecutor executor,
            IResourceScope scope, INode node) {
        return null;
    }

    @Override
    public Class<IMyNameProvider> getValueType() {
        return IMyNameProvider.class;
    }

}
```

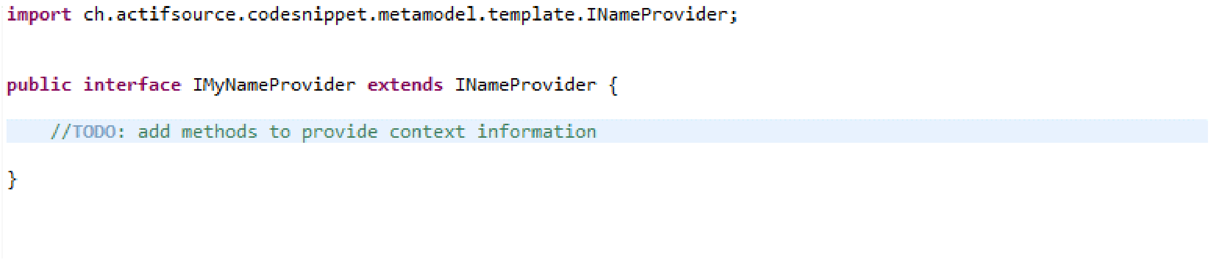
2. Create your own resource of type Literal (e.g. MyNameProvider) which extends

ch.actifsource.codesnippet.metamodel.parsetree.template.NameProvider:



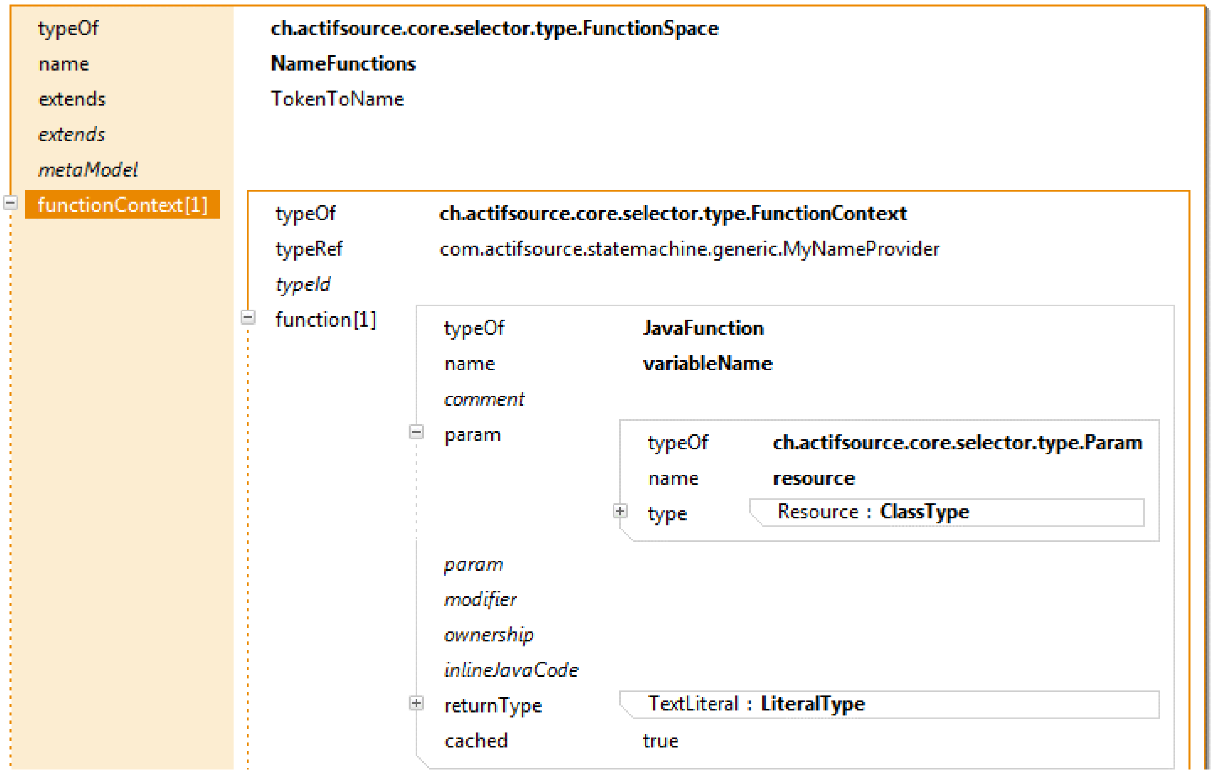
typeOf	ch.actifsource.core.Literal	
name	MyNameProvider	
comment		
aspect[LiteralAspect]	typeOf	JavaAspectImplementation
	implements	LiteralAspect
	className	com.actifsource.statemachine.MyNameProviderLiteralAspect
aspect[LiteralAspect]		
extends[1]	AbstractLiteral	
extends[2]	ch.actifsource.codesnippet.metamodel.parsetree.template.NameProvider	
extends		
modifier		

3. Create a Java interface (e.g. IMyNameProvider) which provides the members needed to manage and store the context information needed by the NameProvider.

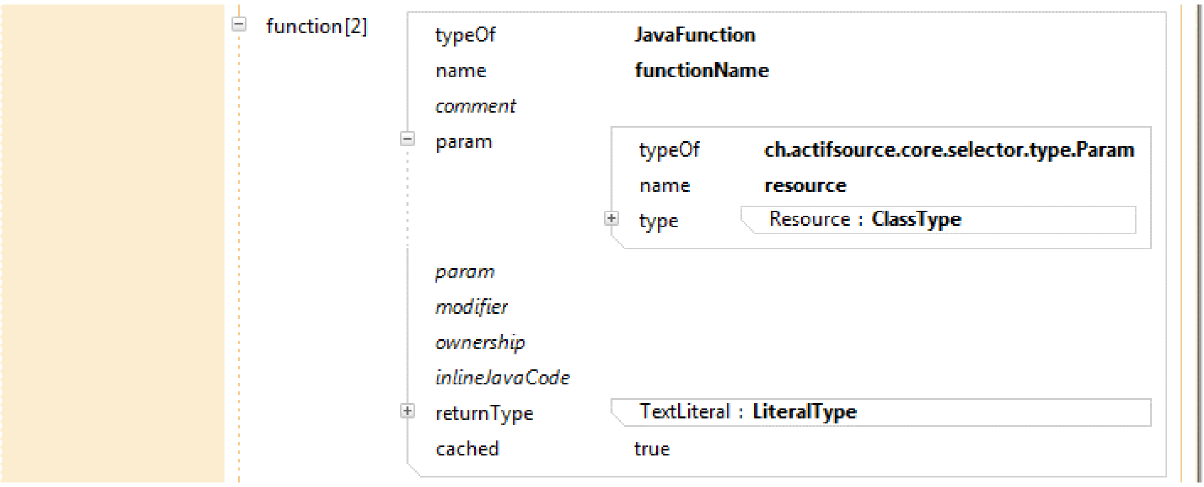


```
import ch.actifsource.codesnippet.metamodel.template.INameProvider;

public interface IMyNameProvider extends INameProvider {
    //TODO: add methods to provide context information
}
```

4. Create a FunctionSpace (e.g. MyNameFunctions) which extends TokenToName with a FunctionContext for the newly created Literal type (e.g. MyNameProvider). Create the two functions variableName and functionName with exactly the same signature as corresponding functions in the TokenToName.



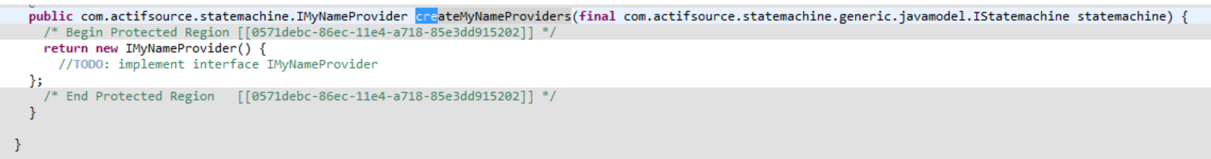
typeOf	ch.actifsource.core.selector.type.FunctionSpace				
name	NameFunctions				
extends	TokenToName				
extends					
metaModel					
functionContext[1]	typeOf	ch.actifsource.core.selector.type.FunctionContext			
	typeRef	com.actifsource.statemachine.generic.MyNameProvider			
	typeId				
	function[1]	typeOf	JavaFunction		
		name	variableName		
		comment			
		param	typeOf	ch.actifsource.core.selector.type.Param	
			name	resource	
			type	Resource : ClassType	
		param			
		modifier			
		ownership			
		inlineJavaCode			
		returnType	TextLiteral : LiteralType		
		cached	true		



function[2]

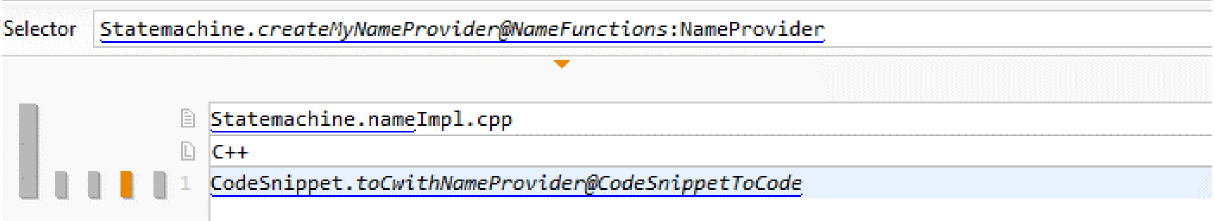
typeOf	JavaFunction		
name	functionName		
comment			
param	typeOf	ch.actifsource.core.selector.type.Param	
	name	resource	
	type	Resource : ClassType	
param			
modifier			
ownership			
inlineJavaCode			
returnType	TextLiteral : LiteralType		
cached	true		

5. Write a JavaFunctions (e.g. createMyNameProvider) that generates an instance of the newly created Java interface (e.g. IMyNameProvider).



```
  public com.actifsource.statemachine.IMyNameProvider createMyNameProviders(final com.actifsource.statemachine.generic.javamodel.IStatemachine statemachine) {
    /* Begin Protected Region [[0571debc-86ec-11e4-a718-85e3dd915202]] */
    return new IMyNameProvider() {
      //TODO: implement interface IMyNameProvider
  };
    /* End Protected Region   [[0571debc-86ec-11e4-a718-85e3dd915202]] */
  }

}
```

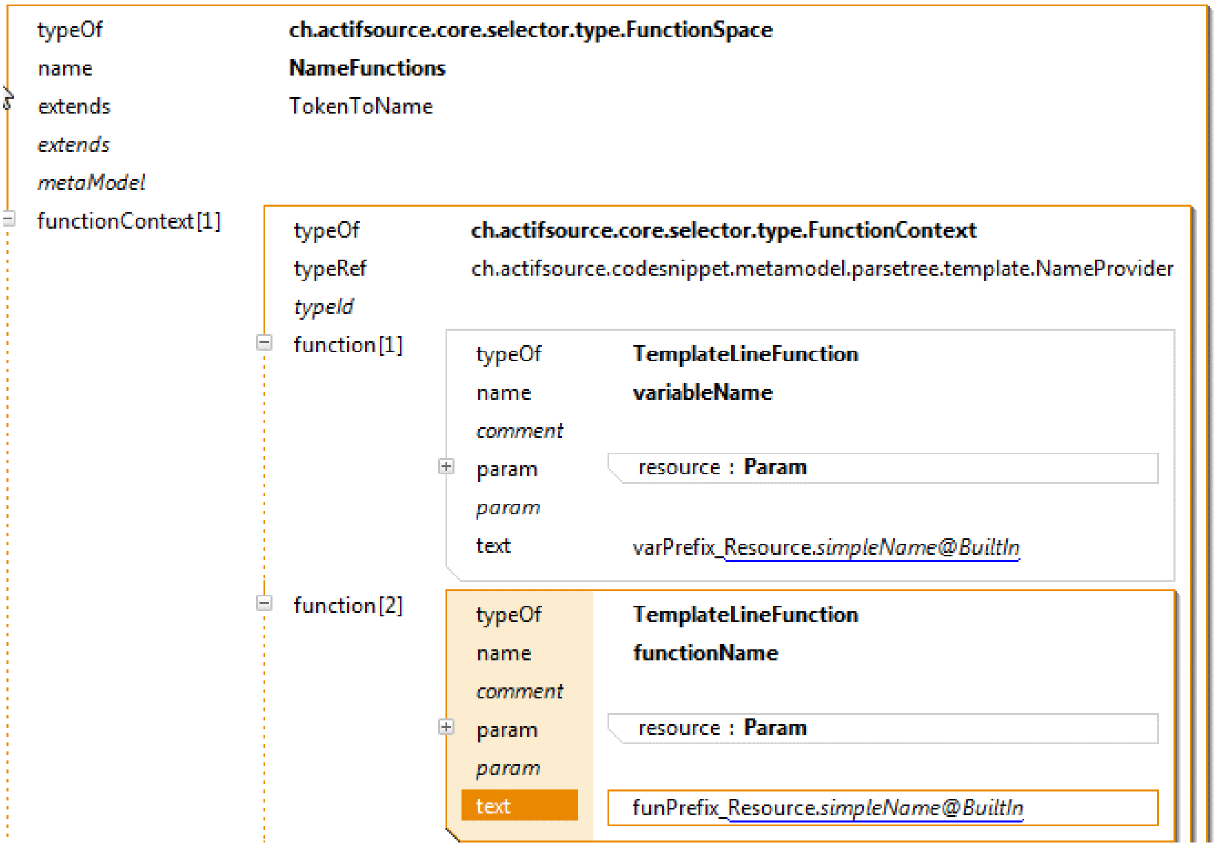
6. In the template where the function to<Language>withNameProvider is used, call the newly created JavaFunction (createMyNameProvider) and call to<Language>withNameProvider with the output of the JavaFunction.



Selector Statemachine.*createMyNameProvider@NameFunctions*:NameProvider

```
Statemachine.nameImpl.cpp
C++
CodeSnippet.toCwithNameProvider@CodeSnippetToCode
```

Note that the TemplateFunctions presented in Section 10.5.1 generate the names by internally calling the name functions variableName and functionName on ch.actifsource.codesnippet.metamodel.parsetree.template.NameProvider (the default NameProvider). Thus, overwriting these functions in a FunctionSpace that extends TokenToName for the type NameProvider (see example below) also changes the behavior of this TemplateFunctions without NameProvider (e.g. toC@CodeSnippetToCode). This can, in particular, change the behavior of already existing templates and template functions. Therefore, this approach should normally be avoided and a customized NameProvider implemented instead.



typeOf ch.actifsource.core.selector.type.FunctionSpace
name NameFunctions
extends TokenToName
extends
metaModel
functionContext[1]
typeOf ch.actifsource.core.selector.type.FunctionContext
typeRef ch.actifsource.codesnippet.metamodel.parsetree.template.NameProvider
typeId
function[1]
typeOf TemplateLineFunction
name variableName
comment
param resource : Param
param
text varPrefix_Resource.*simpleName@BuiltIn*
function[2]
typeOf TemplateLineFunction
name functionName
comment
param resource : Param
param
text funPrefix_Resource.*simpleName@BuiltIn*

Figure 1 FunctionSpace that extends TokenToName with functions overwriting the name functions on the built-in default NameProvider.

10.5.2 Display Code Snippets in Diagrams

Since the parse trees created from the input code of a code snippet are temporary resources, they are only visible for the code generator. To use the content of code snippets in diagrams, Actifsource provides template functions which allow you to display the code in diagrams:

- displayCodeSnippet
- displayCodeSnippetSingleLine

These TemplateFunctions show the unprocessed code as it is entered by the user in the code snippet editor. Both functions are defined on resources of type ch.actifsource.codesnippet.metamodel.element.CodeSnippet. An example application of these functions can be found in the Actifsource Tutorial – Code Snippets at http://www.actifsource.com/tutorials/index.html .

11 Java API

11.1 Select-Fassade

11.1.1 Select Functions for Property

Function	Return type	Description
rangeOrNull	Class	Returns a Property's range.
→ownerOrNull	Class	Returns a Property's domain.
isOwnRelation	boolean	Check if a Property is an OwnRelation.
isDecoratingRelation	boolean	Check if a Property is a DecoratingRelation.
isSubRelation	boolean	Checks if a Relation extends another Relation.
isComposition	boolean	Checks if a Relation is a Composition.
superRelations	Set of Relation	Returns the Relations a Relation extends.
subRelations	Set of Relation	Returns the Relations that extend a given Relation.
rootProperty	Relation	Returns the first Properties of the Set of Properties a Property extends, including itself.
rootRelation	Relation	Returns the first Relations of the Set of Relations a Relation extends, including itself.
possibleDecoratingTypes	Map ?	?

11.1.2 Select Functions for Statement

Function	Return type	Description
existsStatement	boolean	?
findNext	Property Set	Returns the successor Statement of a given Statement.
nameOf	Attribute Set	Returns the composed simpleName in the form (Subject, Predicate, Object).
statementPath	List of Statement	?
decoratedNode	Resource	Returns the Resource that is decorated via this decorating Statement.

11.1.3 Select Functions for Class

Function	Return type	Description
instances	Set of Resource	Returns the direct or indirect instances of a Class.
directInstances	Set of Resource	Returns the direct instances of a Class.
instancesWithPackage	Set of Resource	Returns the direct or indirect instances of a Class in all Packages.
instancesWithMainPackage	Set of Resource	Returns the direct or indirect instances of a Class with their main Package.
isTypeOfFilter	Filter on Resource	Returns a Filter which includes Resources of a given Class.
resourceBySimpleNameOrNull	Resource with Package	Returns the Instance having a given simple Name.
resourcesBySimpleName	Set of Resource with Package	Returns the Instances having a given simple Name.
resourceByFullNameOrNull	Resource with Package	Returns the Instance having a given simple Name.
resourcesByFullName	Set of Resource with Package	Returns the Instances having a given simple Name.
allowedPropertiesOfType	Set of Property	Returns the (inherited or defined) Properties of a Class.
allowedPropertiesOfTypeForRead	Set of Property	Returns the (inherited or defined) Properties of a Class, including the overridden Properties.
allowedAttributesOfType	Set of Attribute	Returns the (inherited or defined) Attributes of a

		Class.
allowedAttributesOfTypeForRead	Set of Attribute	Returns the (inherited or defined) Attributes of a Class, including the overridden Attributes.
allowedRelationsOfType	Set of Attribute	Returns the (inherited or defined) Relations of a Class.
allowedRelationsOfTypeForRead	Set of Attribute	Returns the (inherited or defined) Relations of a Class, including overridden Relations.
isAbstractClass	boolean	Checks if a Class is abstract.
isFinalClass	boolean	Checks if a Class is final.
isSubclass	Set of Type	Checks if a Class extends another Class.
subclasses	Set of Resource with Package	Returns the Sub-Classes of a given Class.
superclasses	Set of Resource with Package	Returns the Super-Classes of a given Class.
matchingInstances	Set of Type	?
isMatchingObjectFilter	Filter on Resource	?
matchingSuperTypes	Set of Type	?
matchingSubTypes	Set of Type	?
isMatchingSuperType	boolean	?
rangeToType	Set of Type	?

11.1.4 Select Functions for Resource

Function	Return type	Description
exists	boolean	?
packages	Relation Set	?
mainPackage	Relation Set	?
asPackagedResource	Resource with Package	Decorate a Resource by its main Package.
asPackagedResource	Set of Resource with Package	Decorate a Set of Resources by their main Packages. ?
namespace	string	Returns the name of the Packages and owning Resources, concatenated with dot.
simpleName	string	Returns the Name of a Resource defined by the NameAspect, or else the GUID.
hasName	boolean	Checks if a Resource has a Name, that is it has a NameAspect defined.
hasModifiableName	boolean	ERROR
fullName	boolean	Returns the Name, predeeded by Package Names and Name of the owner Resources.
isOwned	boolean	Checks if a Resource is directly or indirectly owned by another Resource, or it is the same Resource.
isRootResource	boolean	Checks if a Resource is not owned by any other Resource.
isAllowedPredicate	boolean	?
rootResource	Resource	Returns the owner Resource that directly resides in a Package.
rootStatements	Statement	?
directlyOwnedResources	Set of Resource	?
ownStatementOrNull	Statement	Returns the Statement stating that a given Resource is owned by another Resource.
decoratedNode	Resource	Returns the Resource being decorated by this Decorator.
shallowType	Type	Returns the direct Type of a Resource.
isTypeOf	boolean	Check if a Resource is instance of a given Class.
toMeRelationsForType	Relation	?
toMeTypes	Set of Type	?
isMatching	boolean	?

matchingTypes	Set of Type	?
allowedProperties	Property Set	Returns the (inherited or defined) Properties of a Resource's Class.
allowedPropertiesForRead	Property Set	Returns the (inherited or defined) Properties of a Resource's Class, including overridden Properties.
allowedAttributes	Attribute Set	Returns the (inherited or defined) Attributes of a Resource's Class.
allowedAttributesForRead	Attribute Set	Returns the (inherited or defined) Attributes of a Resource's Class, including overridden Attributes.
allowedRelations	Relation Set	Returns the (inherited or defined) Relations of a Resource's Class.
allowedRelationsForRead	Relation Set	Returns the (inherited or defined) Relations of a Resource's Class, including overridden Relations.
allowedToMeRelations	Relation Set	Returns the Relations having a Resource's Class as range.

11.1.5 Select Functions for Extendable

Function	Return type	Description
isAbstractExtendable	boolean	Checks if an Extendable is abstract.
isFinalExtendable	boolean	Checks if an Extendable is final.
extensions	Set of Extendable	Returns the extending Resources.
extendedResource	Set of Extendable	Returns the extended Resources.
isExtension	boolean	Check if a Resource extends another Resource.

11.1.6 Select Functions for (Resource, Property)

Function	Return type	Description
attributeStatementOrNull	Statement	?
allowedPropertiesForRead	Property Set	Returns the (inherited or defined) Properties of a Resource's Class, including overridden Properties.
allowedAttributes	Attribute Set	Returns the (inherited or defined) Attributes of a Resource's Class.
allowedAttributesForRead	Attribute Set	Returns the (inherited or defined) Attributes of a Resource's Class, including overridden Attributes.
allowedRelations	Relation Set	Returns the (inherited or defined) Relations of a Resource's Class.
allowedRelationsForRead	Relation Set	Returns the (inherited or defined) Relations of a Resource's Class, including overridden Relations.
allowedToMeRelations	Relation Set	Returns the Relations having a Resource's Class as range.
objectsForAttribute	List of Any	Returns the Objects as result of the Evaluation of a given Attribute on a given Resource
objectForAttribute	List of Any	Returns the first Objects as result of the Evaluation of a given Attribute on a given Resource, or else a default.
objectForAttributeOrNull	List of Any	Returns the first Objects as result of the Evaluation of a given Attribute on a given Resource.
statementForAttributeOrNull	List of Statements	Returns the first Statement on a given Resource for a given Attribute.
objectForPropertyOrNull	List of Statements	Returns the first Objects as result of the Evaluation of a given Property on a given Resource.
objectMaxCard	int	Returns the maximum allowed count of Statement of a given Relation for a Resource in the Relation's range.
objectMinCard	int	Returns the minimum allowed count of Statement of a given Relation for a Resource in the Relation's range.
attributeMinCard	int	Returns the minimum allowed count of Statement of a given Attribute for a Resource in the Attribute's domain.
attributeMaxCard	int	Returns the minimum allowed count of Statement of a given Attribute for a Resource in the Attribute's domain.

subjectMinCard	int	Returns the minimum allowed count of Statement of a given Relation for a Resource in the Property’s domain.
subjectMaxCard	int	Returns the minimum allowed count of Statement of a given Relation for a Resource in the Property’s domain.
objectForAttributeOrNull	List of Literal	Returns the first Objects as result of the Evaluation of a given Attribute on a given Resource.
valueForAttributeOrNull	string	Returns the string value of the first Object as result of the Evaluation of a given Attribute on a given Resource.
valueForBooleanAttribute	boolean	Returns the string value of the first Object as result of the Evaluation of a given Attribute on a given Resource, or a default value.
objectForRelationOrNull	List of Resource	Returns the first Objects as result of the Evaluation of a given Relation on a given Resource.
objectsForRelation	List of Resource	Returns the Objects as result of the Evaluation of a given Relation on a given Resource.
objectsForRelationOfType	List of Resource	Returns the Objects as result of the Evaluation of a given Relation on a given Resource, having a given Type.
ownerOrNull	Resource	Returns a Resource’s owner Resource.
relationStatementOrNull	Statement	?
statementOrNull	Statement	Returns the Statement with given Subject, Predicate and Object.
statement	Set of Statement	Returns the Statements with given Subject.
statementsForAttribute	Set of Statement	Returns the Statements with Subject and Predicate given by Resource and Attribute.
statementsForRelation	Set of Statement	Returns the Statements with Subject and Predicate given by Resource and Relation.
statementForRelationOrNull	Set of Statement	Returns the first Statement with Subject and Predicate given by Resource and Relation.
subjectForRelationOrNull	Set of Statement	Returns the first Subject with Object and Predicate given by Resource and Relation.
subjectsForRelation	Set of Statement	Returns the Subjects with Object and Predicate given by Resource and Relation.
toMeStatementForRelationOrNull	Set of Statement	Returns the first Statement with Object and Predicate given by Resource and Relation.
toMeStatementsForAttribute	Set of Statement	Returns the Statements with Object and Predicate given by Resource and Attribute.
toMeStatementsForRelation	Set of Statement	Returns the Statements with Object and Predicate given by Resource and Relation.
toMeStatements	Set of Statement	Returns the first Statement with given Object.
treeSelectObjects	Set of Object	Calculates the closure for given Resource and Relation. ??
topoSelectObjects	Set of Object	?
treeSelectSubjects	Set of Object	?
decoratableNodes	Set of Resource	?

11.1.7 Select Functions for Package

Function	Return type	Description
allReferencedPackages	Package	?
allStatements	Set of Statement	Returns all the Statements in a Package.
allRequiredScopes	Set of Statement	?
allAvailableRequiredScopes	Set of Resource Scope	?
resourceByNameOrNull	Resource	Returns the Resource in the Package having a given simple Name.

11.1.8 Select Functions for Resource Scope

getScope	Resource Scope	Returns the Scope.
isRequired	Set of Resource Scope	?
allResourcesInScope	Unordered Set of Resource	Returns all Resources reachable from a given Resource Scope.
allReferencedResourcesInScope	Unordered Set of Resource	?
classByName	Java class	Loads a Java class of given name.
classByNameOrNull	Java class	Loads a Java class of given name.
packagesInScope	Set of Package	?
unreferencedResource	Set of Resource with Package	?
types	Set of Class	Returns the direct and indirect Types of a Resource.

11.1.9 Additional Select Functions

Function	Return type	Description
getScope	Resource Scope	Returns the Scope.
getRequiredScopes	Set of Resource Scope	?
allRequiredScopes	Set of Resource Scope	?
allAvailableRequiredScopes	Set of Resource Scope	?
isRequired	Set of Resource Scope	?
allResources	Unordered Set of Resource	Returns all Resources in Scope.
allResourcesInPackage	Unordered Set of Resource	Unordered Set of Resource
allResourcesInPackage2	Unordered Set of Resource with Package	Returns all Resources in a Package.
allResourcesInPackages2	Unordered Set of Resource with Package	Returns all Resources in a List of Packages.
allRootResourcesInPackage	Unordered Set of Resource	Returns all Resources residing directly in a Package.
allStatements	Set of Statement	Returns all Statements.
resourceBySimpleNameOrNull	Resource with Package	Returns the Resource having a given simple Name.
resourcesBySimpleName	Set of Resource with Package	Returns the Resource having a given simple Name.
resourceByFullNameOrNull	Resource with Package	Returns the Resource having a given full Name.
resourcesByFullName	Set of Resource with Package	Returns the Resource having a given full Name.
packagesByName	Set of Package	Returns the Packages having a given Name.
subPackagesByName	Set of Package	Returns the Packages having a given Name, including the Sub-Packages.
packagesByExpressions	Iterable of Package	?
allPackages	Iterable of Package	?

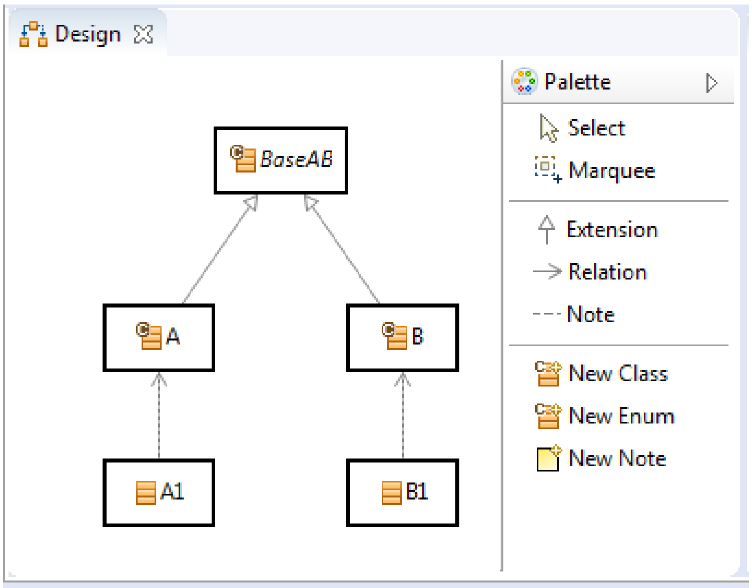
11.2 Update-Fassade

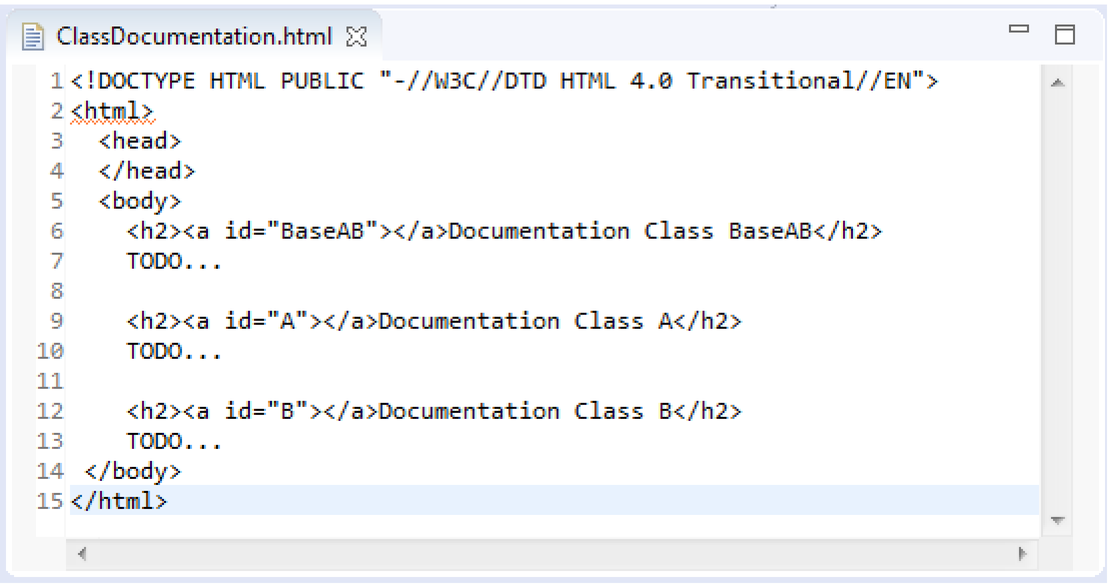
12 Context Sensitive Help

In this section we will show how to set up the context sensitive help system. The Actifsource context sensitive help is integrated into the Eclipse Platform help system.

As a very simple example, let us define a help system for the following meta-model. This example allows to show help information about the Class A, B and BaseAB, and the respective instances A1 and B1.

First, we create the help file “ClassDocument.html” which contains all the help information and which defines the link targets for the context sensitive help.





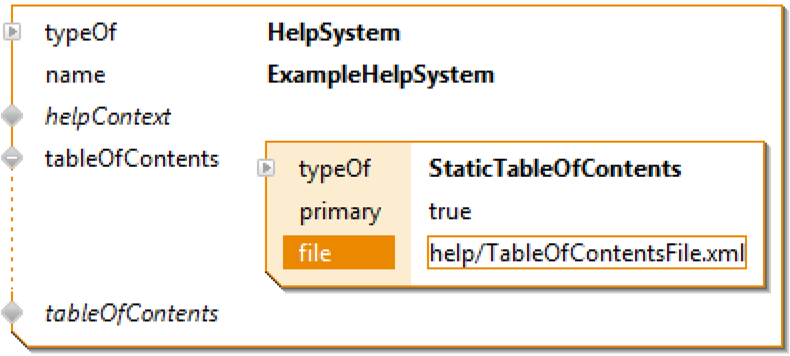
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
  <head>
  </head>
  <body>
    <h2><a id="BaseAB"></a>Documentation Class BaseAB</h2>
    TODO...

    <h2><a id="A"></a>Documentation Class A</h2>
    TODO...

    <h2><a id="B"></a>Documentation Class B</h2>
    TODO...
 </body>
</html>
```

12.1 Table of Contents

The table of contents (toc) contains a collection of topics for display inside the Eclipse help system. The topics here unlike in the context sensitive help are not bound to a context. For the detailed information on the supported features (Topics, Link and Anchor), see the Eclipse documentation

Now we create a HelpSystem resource in Actifsource: “ExampleHelpSystem”. Let us add a table of contents which has its contents statically defined as xml file: “StaticTableOfContents”. It has two properties:

- **primary**: If true: The table of contents is displayed as a book within the Eclipse help.
- **file**: The location of the toc xml file.



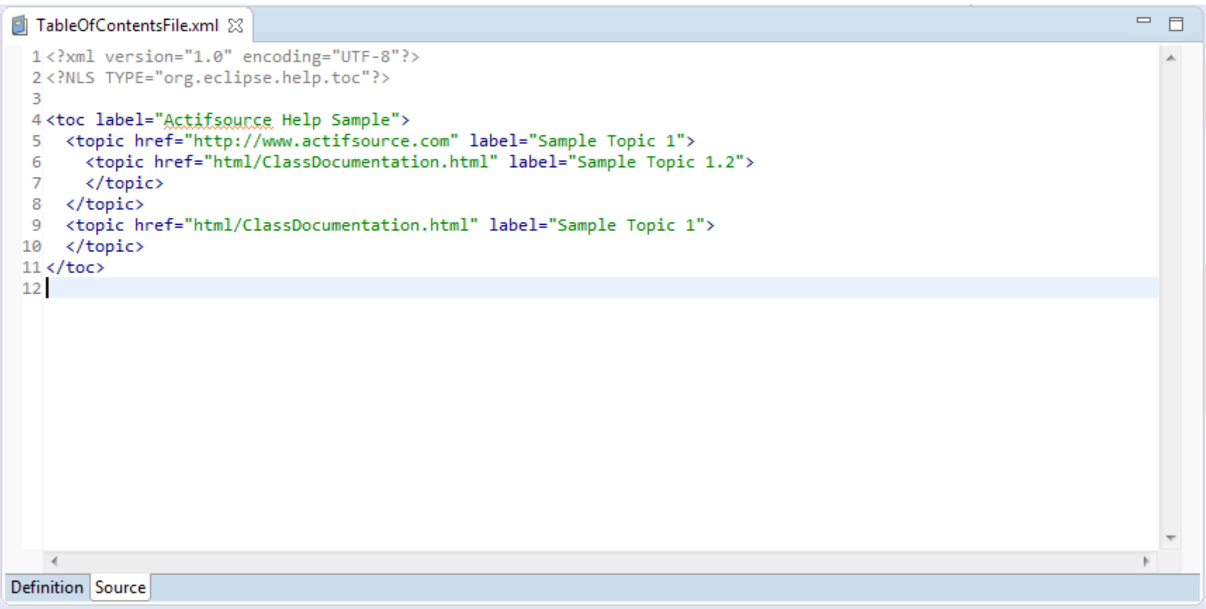
typeOf	**HelpSystem**
name	**ExampleHelpSystem**
helpContext	
tableOfContents	typeOf **StaticTableOfContents** primary true file help/TableOfContentsFile.xml
tableOfContents	

For editing the toc xml file, you can use the Eclipse specific editor, or edit it directly in an xml editor.

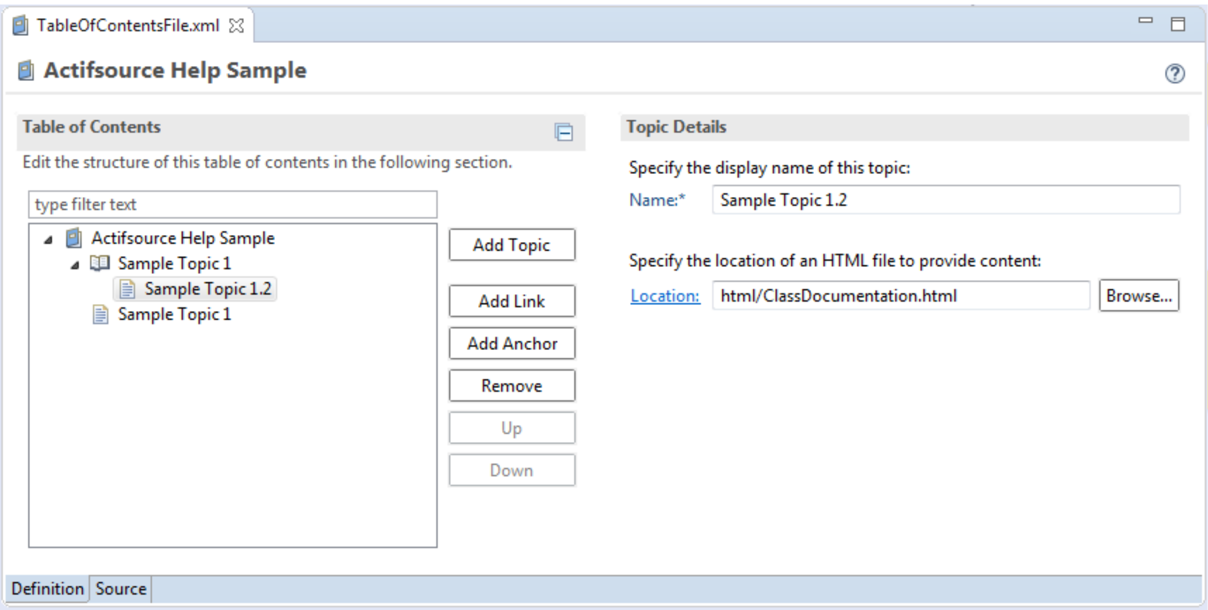
The toc xml file consist of a root xml element “toc” having an attribute “label” and containing nested “topic” elements. A “topic” element has the two xml attributes:

- **label**: The label to display for the topic.
- **href**: The location of the html File. The link providing anchors.

Now we create a toc file (TableOfContentsFile.xml) with the following topics and nested topics.



```xml
<?xml version="1.0" encoding="UTF-8"?>
<?NLS TYPE="org.eclipse.help.toc"?>

<toc label="Actifsource Help Sample">
  <topic href="http://www.actifsource.com" label="Sample Topic 1">
    <topic href="html/ClassDocumentation.html" label="Sample Topic 1.2">
    </topic>
  </topic>
  <topic href="html/ClassDocumentation.html" label="Sample Topic 1">
  </topic>
</toc>
```



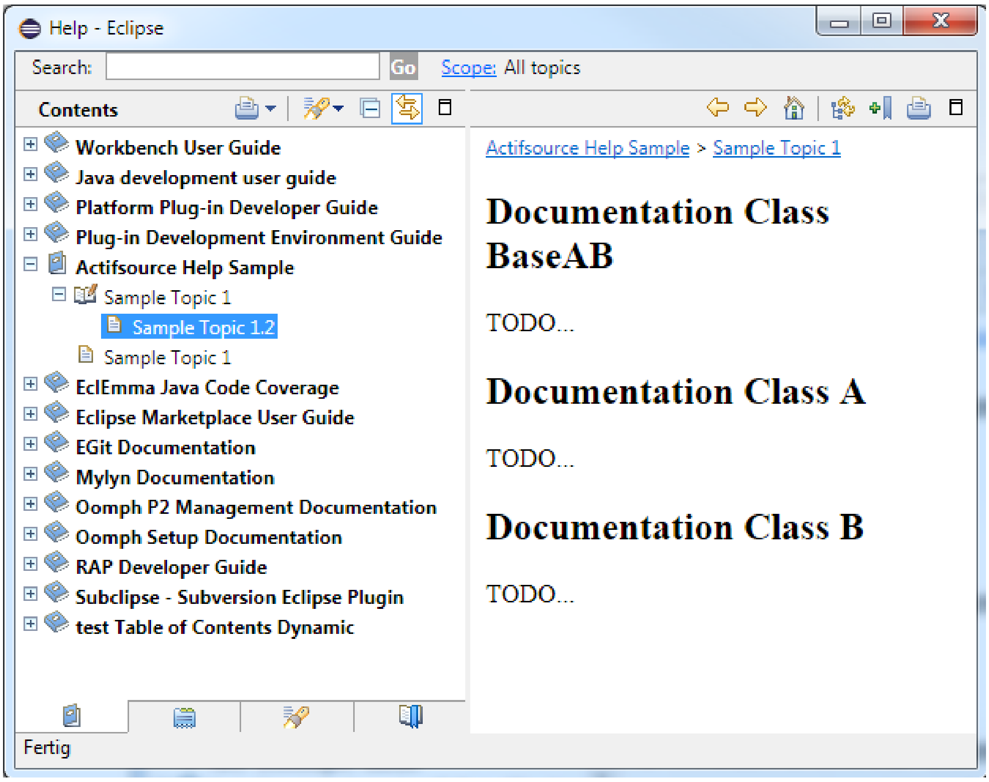
Specify the display name of this topic:

Name:* Sample Topic 1.2

Specify the location of an HTML file to provide content:

Location: html/ClassDocumentation.html Browse...

To open the Eclipse help system choose Help -> Help Contents. Select the ‘Actifsource Help Sample’ and expand the topics and you will see the following.



Help - Eclipse
Search:
Go
Scope: All topics
Contents
Workbench User Guide
Java development user guide
Platform Plug-in Developer Guide
Plug-in Development Environment Guide
Actifsource Help Sample
Sample Topic 1
Sample Topic 1.2
Sample Topic 1
EclEmma Java Code Coverage
Eclipse Marketplace User Guide
EGit Documentation
Mylyn Documentation
Oomph P2 Management Documentation
Oomph Setup Documentation
RAP Developer Guide
Subclipse - Subversion Eclipse Plugin
test Table of Contents Dynamic
Fertig

Actifsource Help Sample > Sample Topic 1

Documentation Class BaseAB

TODO...

Documentation Class A

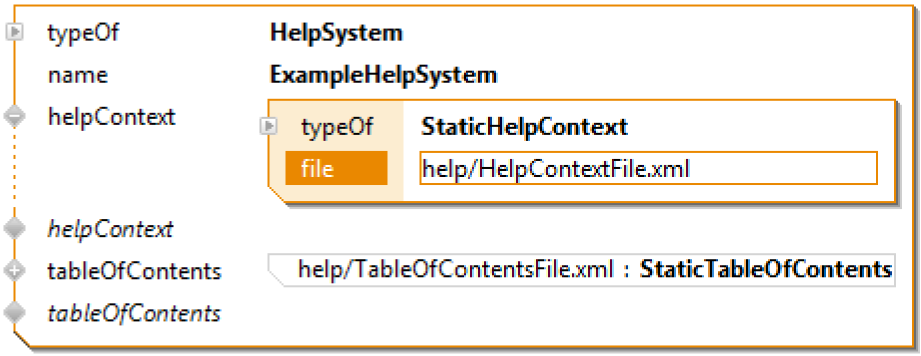
TODO...

Documentation Class B

TODO...

12.2 Help Context

Context sensitive help is a mechanism for linking a Resource to a specific help topic. For the detailed information on the supported features (Context, Topics and Commands), see the Eclipse documentation.



typeOf HelpSystem
name ExampleHelpSystem
helpContext
typeOf StaticHelpContext
file help/HelpContextFile.xml
helpContext
tableOfContents help/TableOfContentsFile.xml : StaticTableOfContents
tableOfContents

The information, which defines the different contexts and how they are linked to a topic, is stored in a help context xml file. When triggering the context sensitive help, the defined contexts are matched against the currently selected resource in the following order (most specific to least specific):

- Matching of the instance by its GUID (instance match)
- Matching of a resource by the GUID of its class (direct class match), and it super-classes (indirect class match).

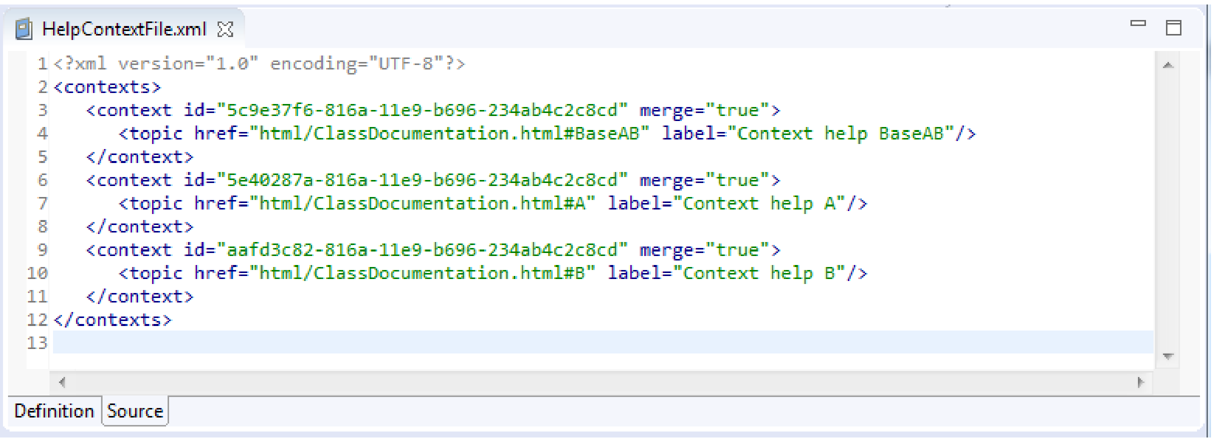
You can edit the help context xml file directly or you using the Eclipse specific editor. A context xml entry has the two attributes:

- **id**: The GUID of the Resource to associate the help.
- **merge**: If true, also the topics of less specific contexts than this context are shown. If false, the topics of less specific context matches are omitted.

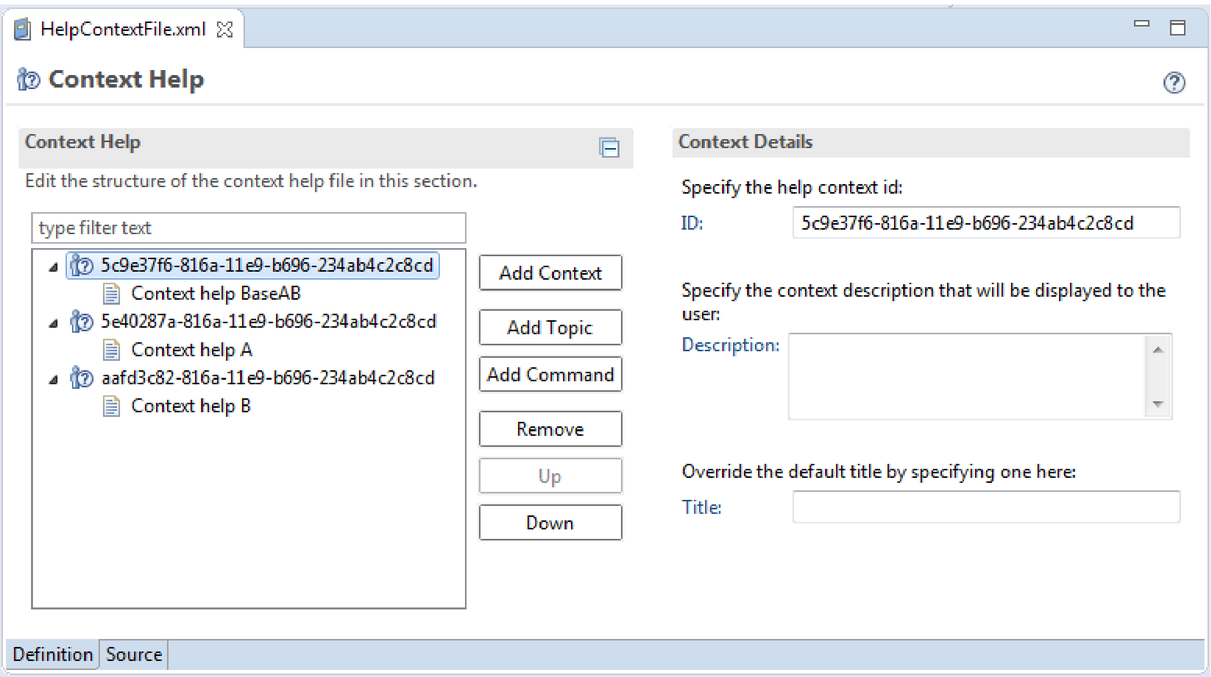
A topic entry in the xml file has the attributes:

- **label**: The label displayed for the topic.
- **href**: The location of the html file. The link providing anchors.

Now we create a help context file (HelpContextfile.xml) with the following contexts and topics.



```
<?xml version="1.0" encoding="UTF-8"?>
<contexts>
   <context id="5c9e37f6-816a-11e9-b696-234ab4c2c8cd" merge="true">
      <topic href="html/ClassDocumentation.html#BaseAB" label="Context help BaseAB"/>
   </context>
   <context id="5e40287a-816a-11e9-b696-234ab4c2c8cd" merge="true">
      <topic href="html/ClassDocumentation.html#A" label="Context help A"/>
   </context>
   <context id="aafd3c82-816a-11e9-b696-234ab4c2c8cd" merge="true">
      <topic href="html/ClassDocumentation.html#B" label="Context help B"/>
   </context>
</contexts>
```



Context Help

Context Help

Edit the structure of the context help file in this section.

type filter text

- 5c9e37f6-816a-11e9-b696-234ab4c2c8cd
  - Context help BaseAB
- 5e40287a-816a-11e9-b696-234ab4c2c8cd
  - Context help A
- aafd3c82-816a-11e9-b696-234ab4c2c8cd
  - Context help B

Context Details

Specify the help context id:

ID: 5c9e37f6-816a-11e9-b696-234ab4c2c8cd

Specify the context description that will be displayed to the user:

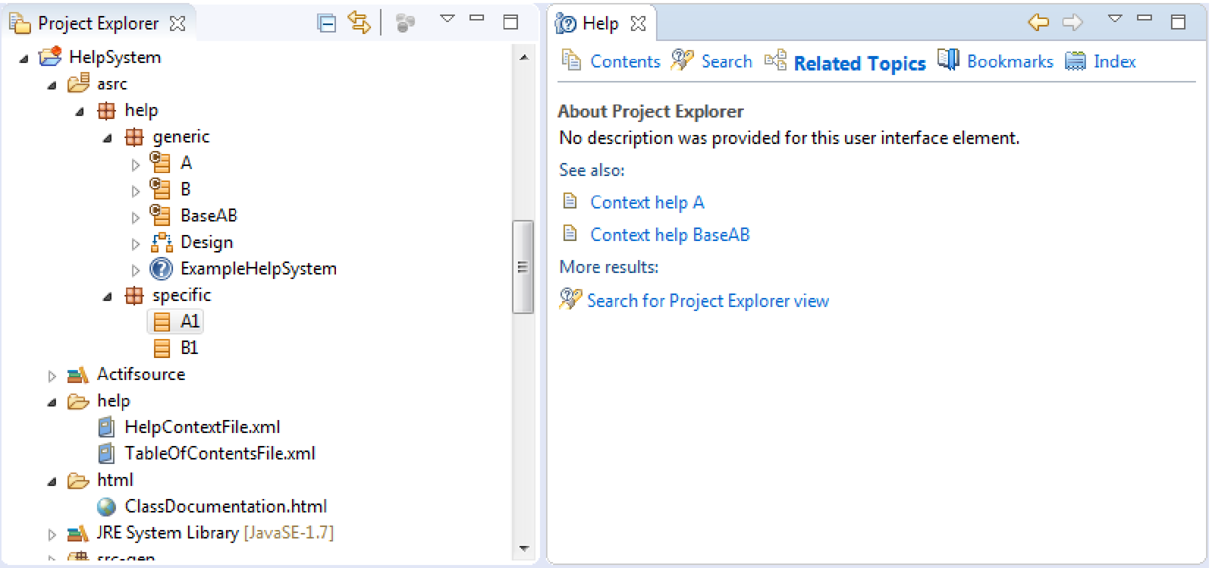
Description:

Override the default title by specifying one here:

Title:

The content sensitive help can be opened by selecting a Resource via any Actifsource editor and pressing ‘F1’.

All related topics started from the selected Resource (A1) via his types (from specific to generic type) are merged together. If only a single context matches the currently selected resource (setting the “merge” flag to false), then the html content is immediately shown inside the Eclipse help view.



About Project Explorer

No description was provided for this user interface element.

See also:

Context help A

Context help BaseAB

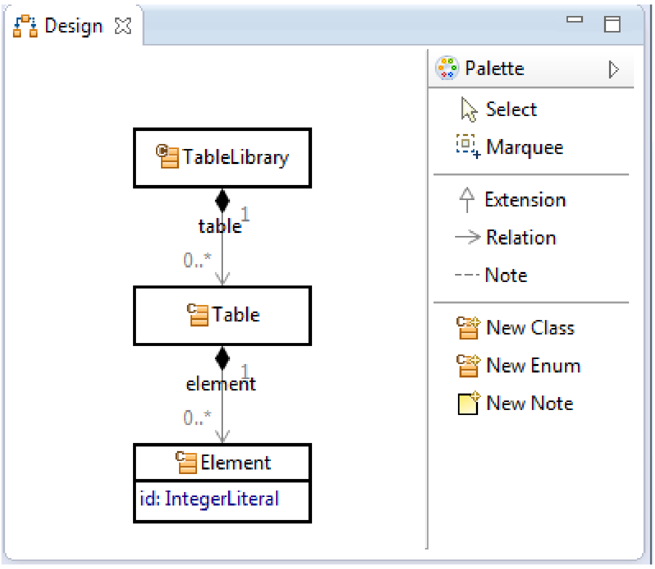
More results:

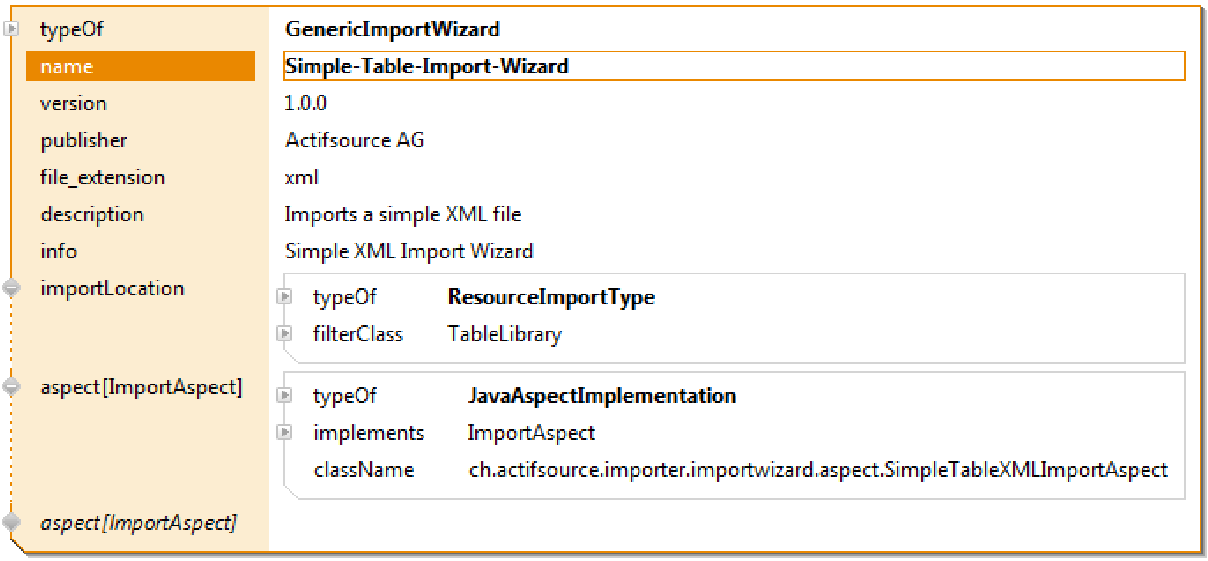
Search for Project Explorer view

13 Generic Import Wizard

In this section we will show how to import any file into a model.

As a very simple example, let us define a generic import wizard for the table-library meta-model. This example allows importing tables directly into the table-library. Every table and element has a mandatory name and the element also defines a mandatory id.





typeOf	GenericImportWizard
name	Simple-Table-Import-Wizard
version	1.0.0
publisher	Actifsource AG
file_extension	xml
description	Imports a simple XML file
info	Simple XML Import Wizard
importLocation	typeOf ResourceImportType filterClass TableLibrary
aspect[ImportAspect]	typeOf JavaAspectImplementation implements ImportAspect className ch.actifsource.importer.importwizard.aspect.SimpleTableXMLImportAspect
aspect[ImportAspect]	

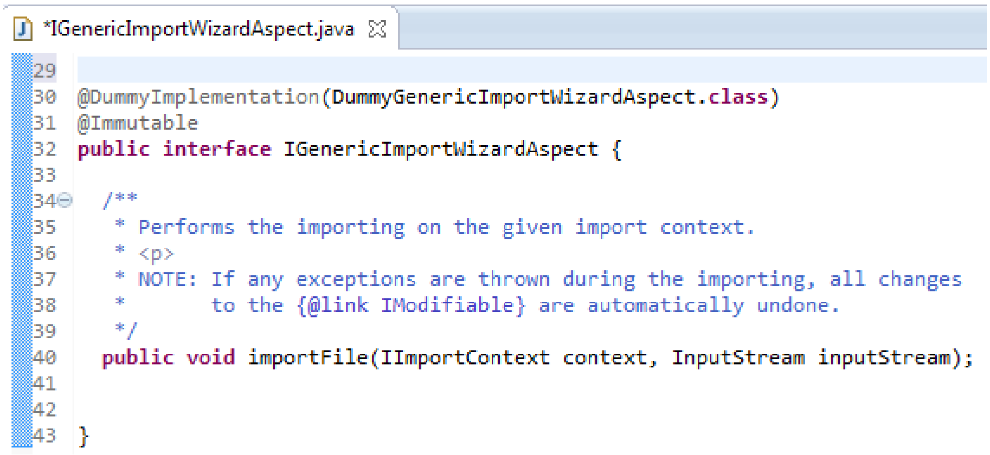
Generic Import Wizard

Property	Description
name	Defines the import wizard name.
version	Defines the version from the import wizard.
publisher	Defines the publisher from the import wizard.
File_extension	Defines the extension of the source file.
description	Defines the description inside the eclipse info page. Html tags are allowed.
info	Defines the info inside the import wizard.
importLocation	Defines the import location inside the project: • **GlobalImportType:** Import location is inside the workspace. • **PackageImportType:**

	Import location is inside a project package. • **ResourceImportType:** Import location is inside a resource. The property ‘filteClass’ can be used to restrict the resource type.
aspect[ImportAspect]	The import aspect defines the behavior of the import. The aspect interface is defined in the class ‘IGenericImportWizardAspect.java’.

GenericImportWizardAspect.java

The generic import wizard aspect, defines only the function ‘importFile’ with the parameter ‘context’ and ‘imputStream’ from the source file. The functionality of the parameter ‘context’ is described in the table below.



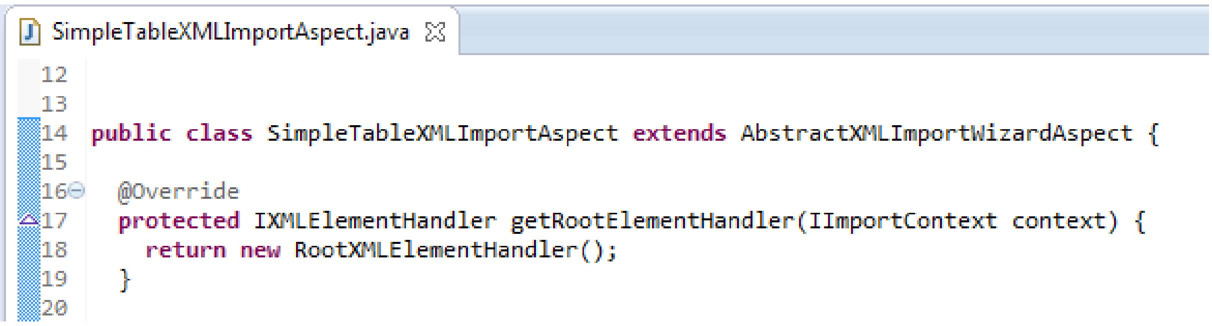
```java

@DummyImplementation(DummyGenericImportWizardAspect.class)
@Immutable
public interface IGenericImportWizardAspect {

	/**
	 * Performs the importing on the given import context.
	 * <p>
	 * NOTE: If any exceptions are thrown during the importing, all changes
	 *       to the {@link IModifiable} are automatically undone.
	 */
	public void importFile(IImportContext context, InputStream inputStream);


}
```

IImportContext.java

Interface	Return type	Description
getFileName	String	Returns the name of the source file.
getReadJobExecutor	IReadJobExecutor	The read-job is used to read from the model by Select-Facade.
getWriteJobExecutor	IWriteJobExecutor	The write-job is used to modify the model by Update-Facade.
getImportType	ImportType	Returns the import-type ‘GlobalImportType’, ‘PackageImportType’ and ‘ResourceImportType’.
getPackage	Package	Returns the package of the target resource, only if the import-type is ‘PackageImportType’ or ‘ResourceImportType’’ else null.
getResouce	INode	Returns the target resource, only if the import-type is ‘ResourceImportType’ else null.
putInfo		Puts any information string to the user.
hasInfo	Boolean	Check if any information is available.
putError		Puts any error string to the user. If any error occurred the modification during the import is undone.
hasErrors	Boolean	Checks if any error is available.
incrementModifiedCount		Increment statistic modification count.
incrementCreateCount		Increment statistic creates count.
incrementDisposeCount		Increment statistic disposes count.
incrementElementCount		Increment statistic element count.

This feature supports the import of any files but you have to parse the input-stream by yourself. If you import an xml-document, you have a prefabricated solution. Now we use this solution for the simple example.

Let’s define a simple table xml import aspect to import the xml into the table-library. To use the prefabricated xml solution, you have to extend the aspect by ‘AbstractXMLImportwizardAspect’.



```java

public class SimpleTableXMLImportAspect extends AbstractXMLImportWizardAspect {

  @Override
  protected IXMLElementHandler getRootElementHandler(IImportContext context) {
    return new RootXMLElementHandler();
  }

```

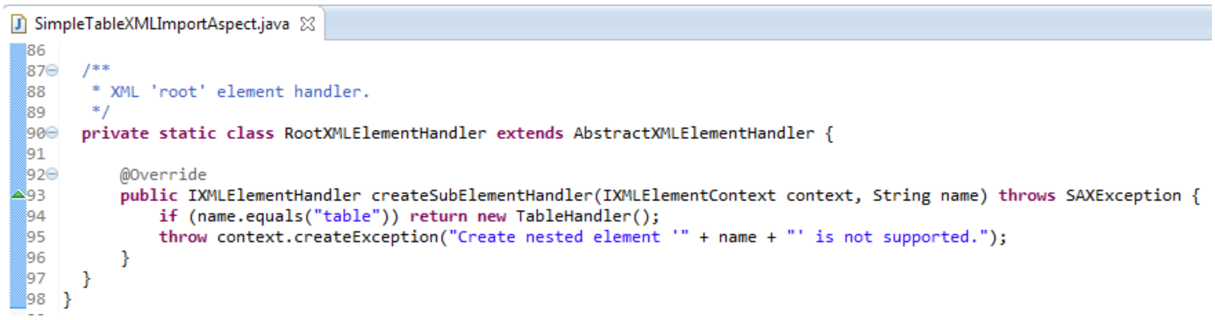
This solution expects only the root element handler. Any handlers have to implement the interface ‘IXMLElementHandler.java’. This interface is described in the table below.

IXMLElementHandler.java

Interface	Return type	Description
createElement	INode	Create and returns the resource correspond to the xml-element.
setAttribute		Sets any xml-attribute to the model.
closeElement		Handles any operation if the end of the xml-element has arrived.
parentElementClose		Handles any operation if the end of the parent xml-element has arrived.
setCharacters		Handler xml text (<tag> text</tag>)
createSubElementHandler	IXMLElementHandler	Creates and returns a sub handler for the xml-sub element.

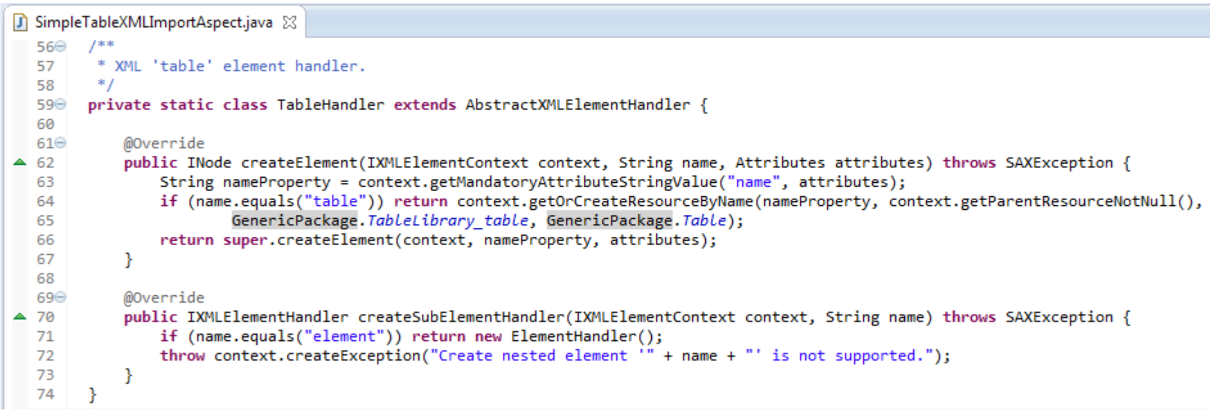
For the simple example, we have to define three xml element handlers “RootXMLElementHandler”, “TableHandler” and “ElementHandler”.

SimpleTableXMLImportAspect.RootXMLElementHandler



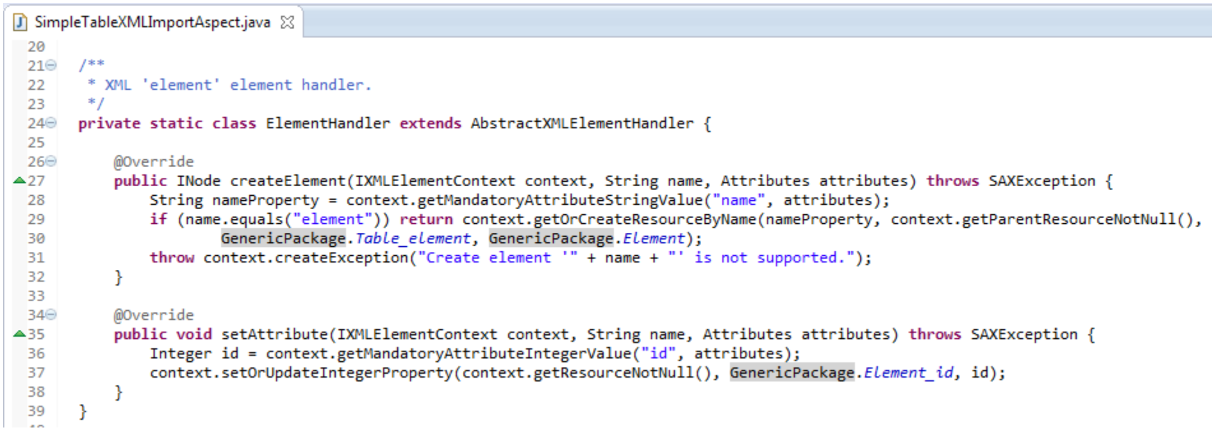
```java

  /**
   * XML 'root' element handler.
   */
  private static class RootXMLElementHandler extends AbstractXMLElementHandler {

      @Override
      public IXMLElementHandler createSubElementHandler(IXMLElementContext context, String name) throws SAXException {
          if (name.equals("table")) return new TableHandler();
          throw context.createException("Create nested element '" + name + "' is not supported.");
      }
  }
}
```

SimpleTableXMLImportAspect.TableHandler


SimpleTableXMLImportAspect.java

```
/**
 * XML 'table' element handler.
 */
private static class TableHandler extends AbstractXMLElementHandler {

    @Override
    public INode createElement(IXMLElementContext context, String name, Attributes attributes) throws SAXException {
        String nameProperty = context.getMandatoryAttributeStringValue("name", attributes);
        if (name.equals("table")) return context.getOrCreateResourceByName(nameProperty, context.getParentResourceNotNull(),
                GenericPackage.TableLibrary_table, GenericPackage.Table);
        return super.createElement(context, nameProperty, attributes);
    }

    @Override
    public IXMLElementHandler createSubElementHandler(IXMLElementContext context, String name) throws SAXException {
        if (name.equals("element")) return new ElementHandler();
        throw context.createException("Create nested element '" + name + "' is not supported.");
    }
}
```

SimpleTableXMLImportAspect. ElementHandler


SimpleTableXMLImportAspect.java

```

/**
 * XML 'element' element handler.
 */
private static class ElementHandler extends AbstractXMLElementHandler {

    @Override
    public INode createElement(IXMLElementContext context, String name, Attributes attributes) throws SAXException {
        String nameProperty = context.getMandatoryAttributeStringValue("name", attributes);
        if (name.equals("element")) return context.getOrCreateResourceByName(nameProperty, context.getParentResourceNotNull(),
                GenericPackage.Table_element, GenericPackage.Element);
        throw context.createException("Create element '" + name + "' is not supported.");
    }

    @Override
    public void setAttribute(IXMLElementContext context, String name, Attributes attributes) throws SAXException {
        Integer id = context.getMandatoryAttributeIntegerValue("id", attributes);
        context.setOrUpdateIntegerProperty(context.getResourceNotNull(), GenericPackage.Element_id, id);
    }
}
```

The parameter ‘context’ in these functions is used to read or write to the model and to handle the import process. This interface is described in the table below.

IXMLElementContext.java

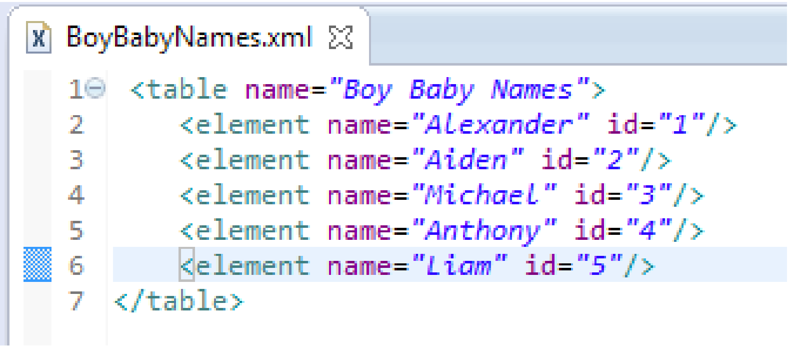
Interface get	Return type	Description
getImportContext	IImportContext	Returns the import context.
getName	String	Returns the name of the xml-tag
getLocator	Locator	Returns the xml document position locator.
getResource	INode	Returns the resource created from the xml element handler function ‘createElement’ or null if the resource doesn’t exist.
getResourceNotNull	INode	Returns the resource created from the xml element handler function ‘createElement’ or throw an exception if the resource doesn’t exist.
getParentResource	INode	Returns the parent resource created from the parent xml element handler function ‘createElement’ or null if the parent resource doesn’t exist.
getParentResourceNotNull	INode	Returns the parent resource created from the parent xml element handler function ‘createElement’ or throw an exception if the parent resource doesn’t exist.

createException	SAXException	Create a sax parsing exception with document location.
getOrCreateResourceByName	INode	Return a new or existing resource with the corresponding name.
getOrCreateResourceByNameAndType	INode	Return a new or existing resource with the corresponding name and type.
setOrUpdateReference		Set or update the resource reference.
setOrUpdateStringProperty		Set or update string property to the model.
setOrUpdateIntegerProperty		Set or update integer property to the model.
setOrUpdateLongProperty		Set or update long property to the model.
setOrUpdateBooleanProperty		Set or update Boolean property to the model.
getMandatoryResurceByName		Return an existing resource from the xml attribute with the corresponding name or an exception if the resource doesn't exists.
getOptionalResurceByName		Return an existing resource from the xml attribute with the corresponding name or a null if the resource doesn't exists.
getMandatoryAttributeStringValue		Return the value of the xml attribute as string or an exception if the value doesn't exists.
getOptionalAttributeStringValue		Return the value of the xml attribute as string or null if the value doesn't exists.
getMandatoryAttributeIntegerValue		Return the value of the xml attribute as integer or an exception if the value doesn't exists.
getOptionalAttributeIntegerValue		Return the value of the xml attribute as integer or null if the value doesn't exists.
getMandatoryAttributeLongValue		Return the value of the xml attribute as long or an exception if the value doesn't exists.
getOptionalAttributeLongValue		Return the value of the xml attribute as long or null if the value doesn't exists.
getMandatoryAttributeBooleanValue		Return the value of the xml attribute as Boolean or an exception if the value doesn't exists.
getOptionalAttributeBooleanValue		Return the value of the xml attribute as Boolean or null if the value doesn't exists.

13.1 Import Wizard

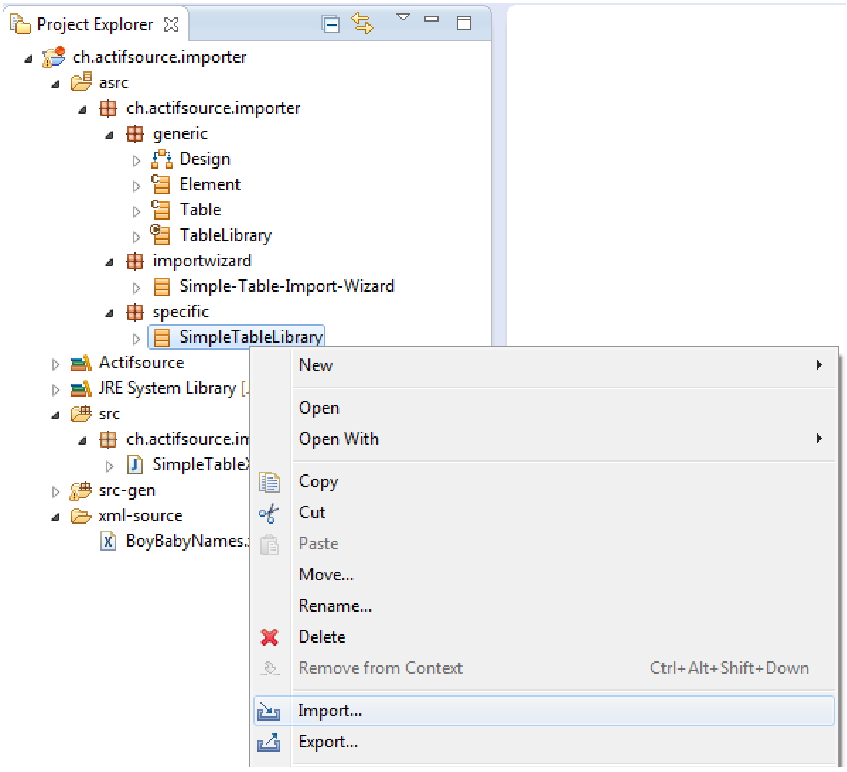
Let's import a xml-table file to verify the import. For example we use the xml-file 'BoyBabyNames.xml'

BoyBabyNames.xml

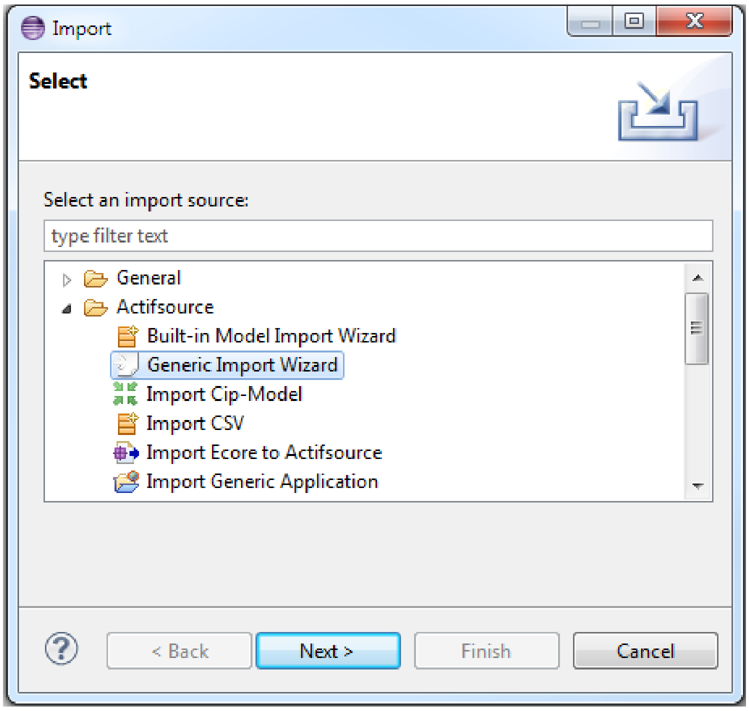


```
<table name="Boy Baby Names">
    <element name="Alexander" id="1"/>
    <element name="Aiden" id="2"/>
    <element name="Michael" id="3"/>
    <element name="Anthony" id="4"/>
    <element name="Liam" id="5"/>
</table>
```

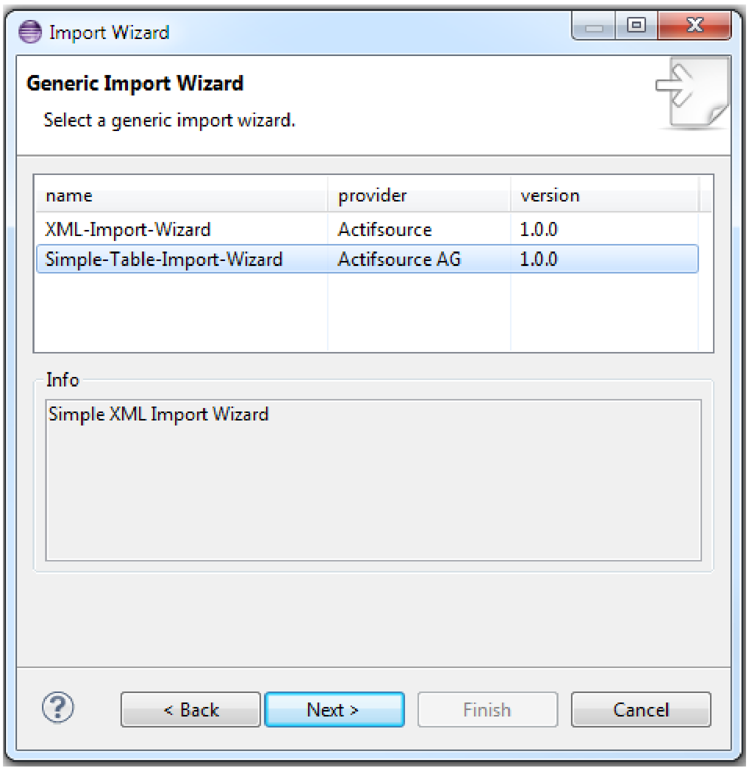
You can start the import sequence via the context menu in the project explorer.



Select the generic import wizard to continue the import.



Select the simple table import wizard to continue the import



Import Wizard

Generic Import Wizard

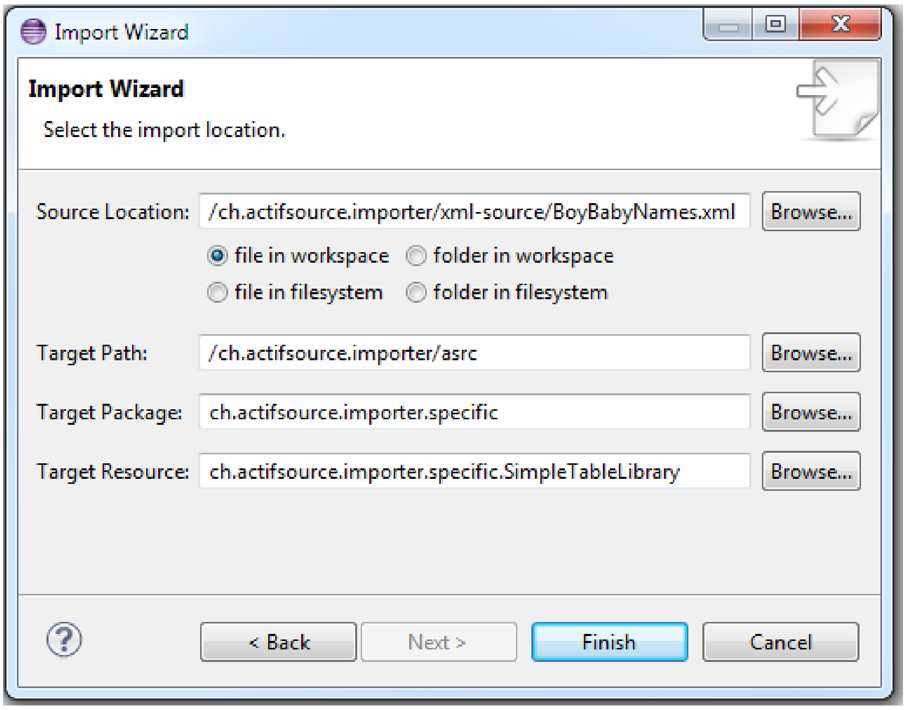
Select a generic import wizard.

name	provider	version
XML-Import-Wizard	Actifsource	1.0.0
Simple-Table-Import-Wizard	Actifsource AG	1.0.0

Info

Simple XML Import Wizard

Preselecting a package or resource in the project explorer will directly fill the target fields. Select the source location file to complete the importing sequence.



Import Wizard

Import Wizard

Select the import location.

Source Location: /ch.actifsource.importer/xml-source/BoyBabyNames.xml

● file in workspace ○ folder in workspace

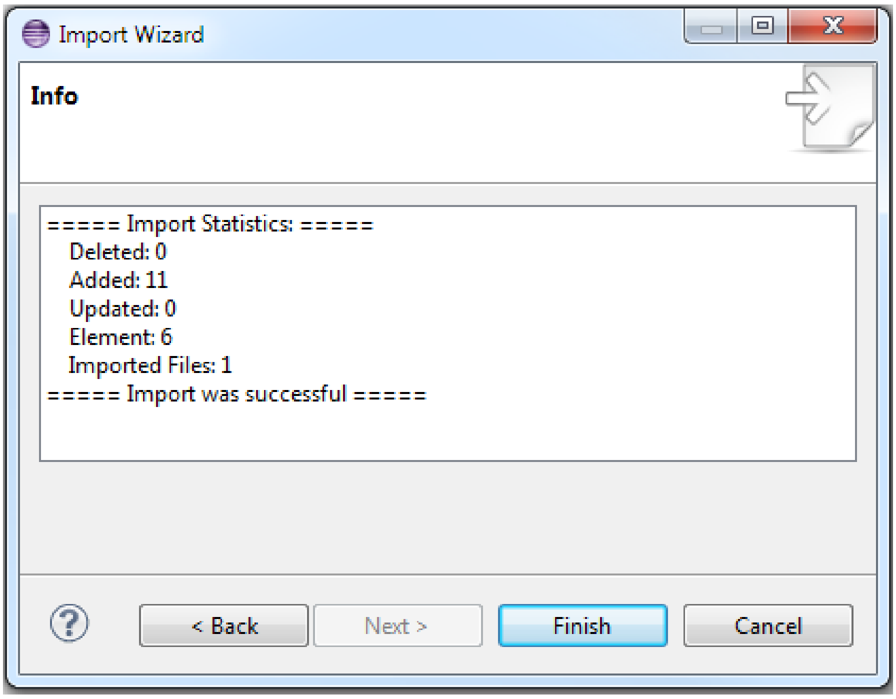
○ file in filesystem ○ folder in filesystem

Target Path: /ch.actifsource.importer/asrc

Target Package: ch.actifsource.importer.specific

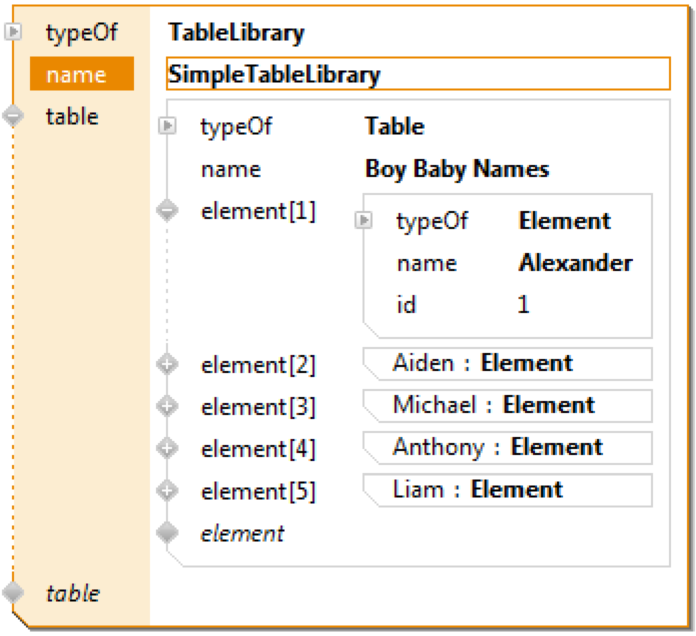
Target Resource: ch.actifsource.importer.specific.SimpleTableLibrary

After importing any errors and infos are shown in the info-Page



```
===== Import Statistics: =====
   Deleted: 0
   Added: 11
   Updated: 0
   Element: 6
   Imported Files: 1
===== Import was successful =====
```

If the importing is successful the table library is synchronized with the xml-file.



```
typeOf   TableLibrary
name     SimpleTableLibrary
table    typeOf       Table
         name         Boy Baby Names
         element[1]   typeOf  Element
                      name    Alexander
                      id      1
         element[2]   Aiden : Element
         element[3]   Michael : Element
         element[4]   Anthony : Element
         element[5]   Liam : Element
         element
table
```

14 Code Generator

14.1 Overview

Click on GUID

Eclipse Builder

Working with GCC

15 Plugin Project

15.1 Overview