Tigerseye User Manual
Version 0.0.1

Leo Roos
`leo_roos@rbg.informatik.tu-darmstadt.de`

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Department of Computer Science
Advisor: Tom Dinkelaker

Contents

Introduction

The *TigersEye* Eclipse Plug-in is an IDE that strives to support easy creation of EDSLs. This guide describes how to install *TigersEye* for development, it shows the currently implemented features and show cases how to create and install a new language. *TigersEye* has been previously known as *Popart*, which has been developed by Yevgen Fanshil, Leonid Melnyk, Thorsten Peter, David Marx, Kamil Erhard and Tom Dinkelaker. The used examples are adjusted versions of their prior work.

Installation

This section describes the installation process for multiple use cases. One subsection describes the installation for the *TigersEye* plug-in for development. A further subsection describes the installation process of a language designed with *TigersEye*.

Installation for Development

The plug-in itself consists of multiple separate plug-ins and has further dependencies to third party plug-ins and libraries. It uses classes and extensions from the Groovy Plug-in and Eclipse's JDT plug-ins, makes some use of different `apache.commons` libraries and uses further libraries to process code. It's core dependency is an *Earley* parser and the *TigersEye* builder. The builder is currently included in the core plug-in. The following table shows the necessary plug-ins and the required versions, if any.

The next table lists further Eclipse projects relevant for the *TigersEye* development.

To set up an Eclipse workspace in order to run the *TigersEye* IDE the following steps must be performed.

1. Install the Groovy Feature into your Eclipse.
2. Copy all other plug-ins into your Eclipse workspace (usually you will check them out from a repository).
3. Perform a complete build of your workspace. It might be necessary to clean all projects before an error free build will be performed.

Once all plug-ins have been set up, *TigersEye* can be started using the predefined `Tigerseye_IDE` launch configuration. The configuration is located in the TigerseyeConfiguration project. It is possible that depending on the used operating system this launch configuration has to be adjusted. Alternatively a new Eclipse Plug-in configuration can be started with all available plug-ins active.

Plug-in name	Version	Description
de.tud.stg.tigerseye.eclipse.core	0.0.1	Core functionality.
de.tud.stg.tigerseye.eclipse.ui	0.0.1	User Interface functionality.
de.tud.stg.tigerseye.eclipse	0.0.1	Contains libraries and plug-ins used by multiple *TigersEye* plug-ins.
Groovy Eclipse Feature	2.1.1	Groovy Eclipse Plug-ins.
org.apache.commons.collections	-	Apache utility classes for collection handling.
org.apache.commons.io	-	Apache IO utility classes.
org.apache.commons.lang	-	Apache general language utility classes.
org.apache.log4j	[1.2 - 1.3)	Employed logging framework.
parlex	-	Earley parser, performs transformations.
slf4j-log4j12	-	Employed logging facade with log4j binding.

Table 1: Necessary Plug-ins

Project name	Description
TigerseyeConfiguration	Contains configuration related data.
documentation	Contains documentation resources.

Table 2: Further *TigersEye* related Eclipse projects

Installing a Language

To install a newly created language it is currently necessary to export it as a plug-in. As in the Examples section described the project can be converted into a plug-in project using the Eclipse PDE tools. It is than necessary to add the `de.tud.stg.tigerseye.eclipse` and the `de.tud.stg.tigerseye.eclipse.core` plug-ins as dependencies. The core plug-in provides the `dslLanguages` extension point. This point has to be extended by declaring what the language class of the exported language is. Optionally a user friendly name and a default file extension can be defined.

Since the language definition will typically be a Groovy file, of which the PDE Builder is not aware of, the default behavior when exporting a plug-in has to be adjusted. Two different approaches can be taken.

Using class Files Compiled in the Workspace

Instead of a specific source folder, which would be compiled during an export using the PDE Export Wizard, the binaries created during the development have to be included, so that they are available on the classpath of the exported language. This can be done in different ways:

- Assuming that the output folder for the `class` files is `bin`, the `bin` folder can be added to the `build.properties` file as value for the `build.includes` property and the entry `bin` has to be added to the `MANIFEST.MF` file as `Bundle-Classpath` entry (the default classpath is the root of the created jar file).
- Alternatively the binary folder can be specified as source folder, in which case the `MANIFEST.MF` does not have to be modified. The resulting `build.properties` file will look something like this:

```
source.. = src/,\
           bin/
bin.includes = META-INF/,\
               .,\
               plugin.xml
output.. = bin/
```

Using Groovy-specific Build Properties

When the Groovy plug-in is installed, additional properties for the `build.properties` file can be used to tell the PDE to include Groovy files during an export. Simply add the following three lines to your build.properties file:

```
sourceFileExtensions=*.java, *.groovy
compilerAdapter=org.codehaus.groovy.eclipse.ant.GroovyCompilerAdapter
compilerAdapter.useLog=true
```

The resulting build.properties file will look similar to this:

```
source.. = src/
bin.includes = META-INF/,\
               .,\
               plugin.xml,\
output.. = bin/
sourceFileExtensions=*.java, *.groovy
compilerAdapter=org.codehaus.groovy.eclipse.ant.GroovyCompilerAdapter
compilerAdapter.useLog=true
```

The additional entries will cause a compilation of Java *and* Groovy files. For general information about valid values for the `build.properties` file check out the Eclipse Help following the path `Plug-in Development Environment Guide > Reference >` For more information about the Groovy specific build properties have a look at Andrew Eisenberg's Blog.

Additional Remarks

Currently the *TigersEye* implementation (Version 0.0.1) assumes that languages can be found in an accessible folder. So the exported language has to be unpacked into the `plugins` folder of the target Eclipse installation before it is usable.

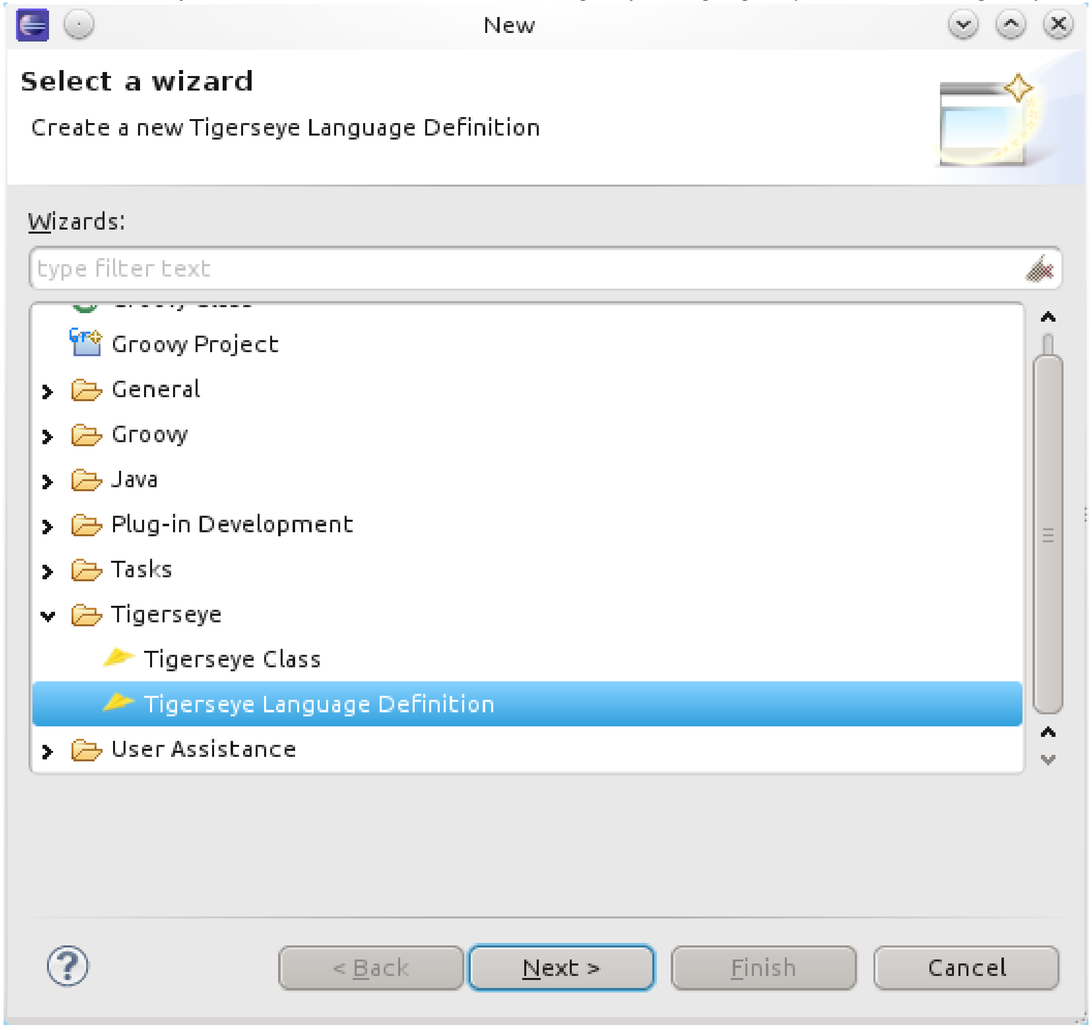
Examples

This section showcases typical use cases for the *TigersEye* IDE.

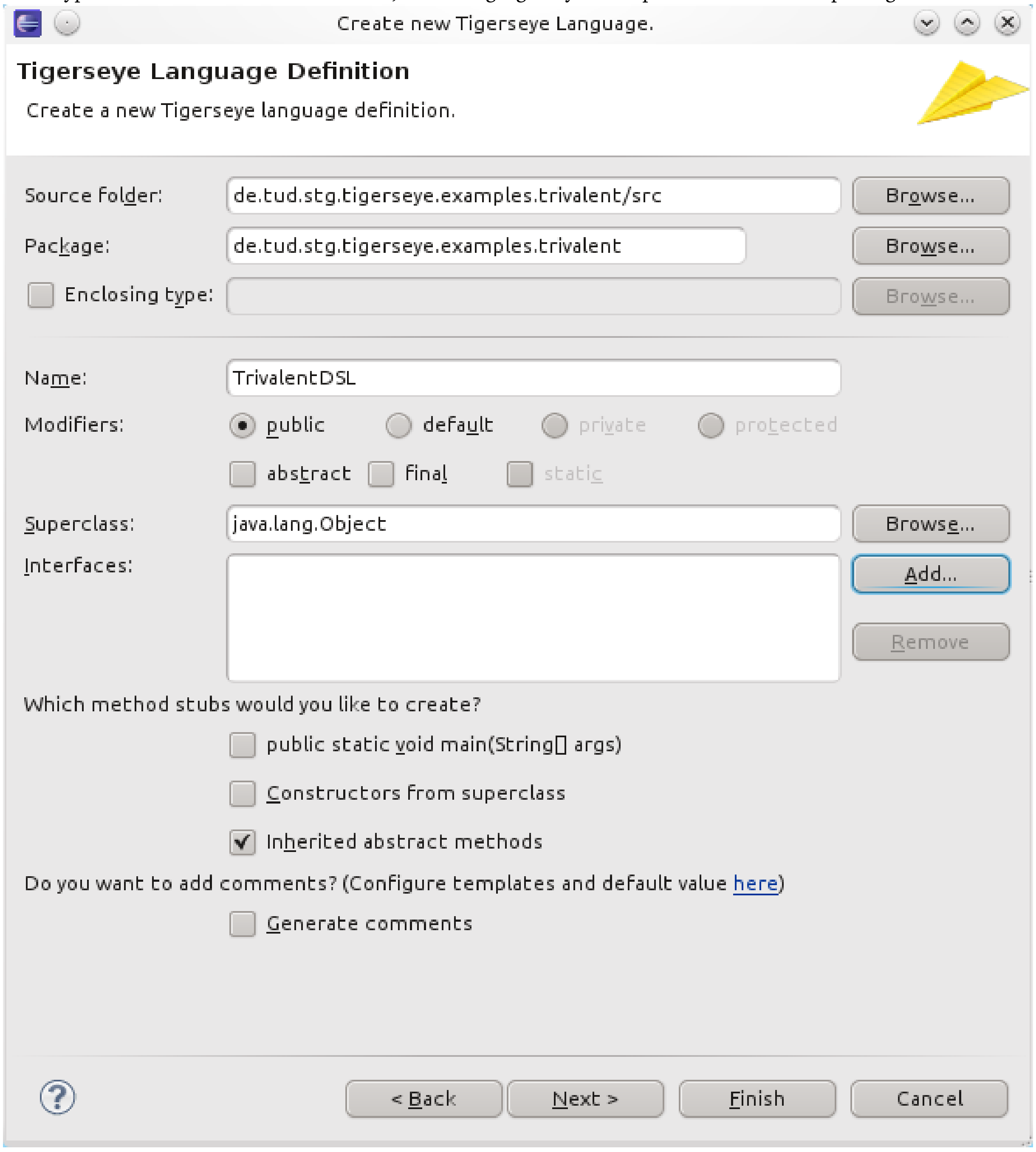
Creating a new Language Definition

The usage of the New Tigerseye Language Definition wizard will be explained by creating a Trivalent-DSL. A Trivalent logic DSL adds an unknown value `U` to the boolean values true `T` and false `F`. So that `T&U` is `U` and `T|U` is `T` and so on.

First create a new Java Project. This Project will contain the Trivalent language created by the *New Tigerseye Language* wizard. In this example a project called `de.tud.stg.tigerseye.examples.trivalent` will be created. In the context menu of the Project choose `New > Other`. Choose *Tigerseye Language Definition* in the Tigerseye folder.



Each language definition consists of a Groovy class that defines all operations, literals and structured elements of the DSL. Type TrivalentDSL as the class name, de.tud.stg.tigerseye.examples.trivalent as the package and select `Next`.



Create new Tigerseye Language.

Tigerseye Language Definition

Create a new Tigerseye language definition.

Source Folder:	de.tud.stg.tigerseye.examples.trivalent/src	Browse...
Package:	de.tud.stg.tigerseye.examples.trivalent	Browse...
☐ Enclosing type:		Browse...
Name:	TrivalentDSL	
Modifiers:	[● public] ○ default ○ private ○ protected	
	☐ abstract ☐ final ☐ static	
Superclass:	java.lang.Object	Browse...
Interfaces:		Add... Remove

Which method stubs would you like to create?

☐ public static void main(String[] args)

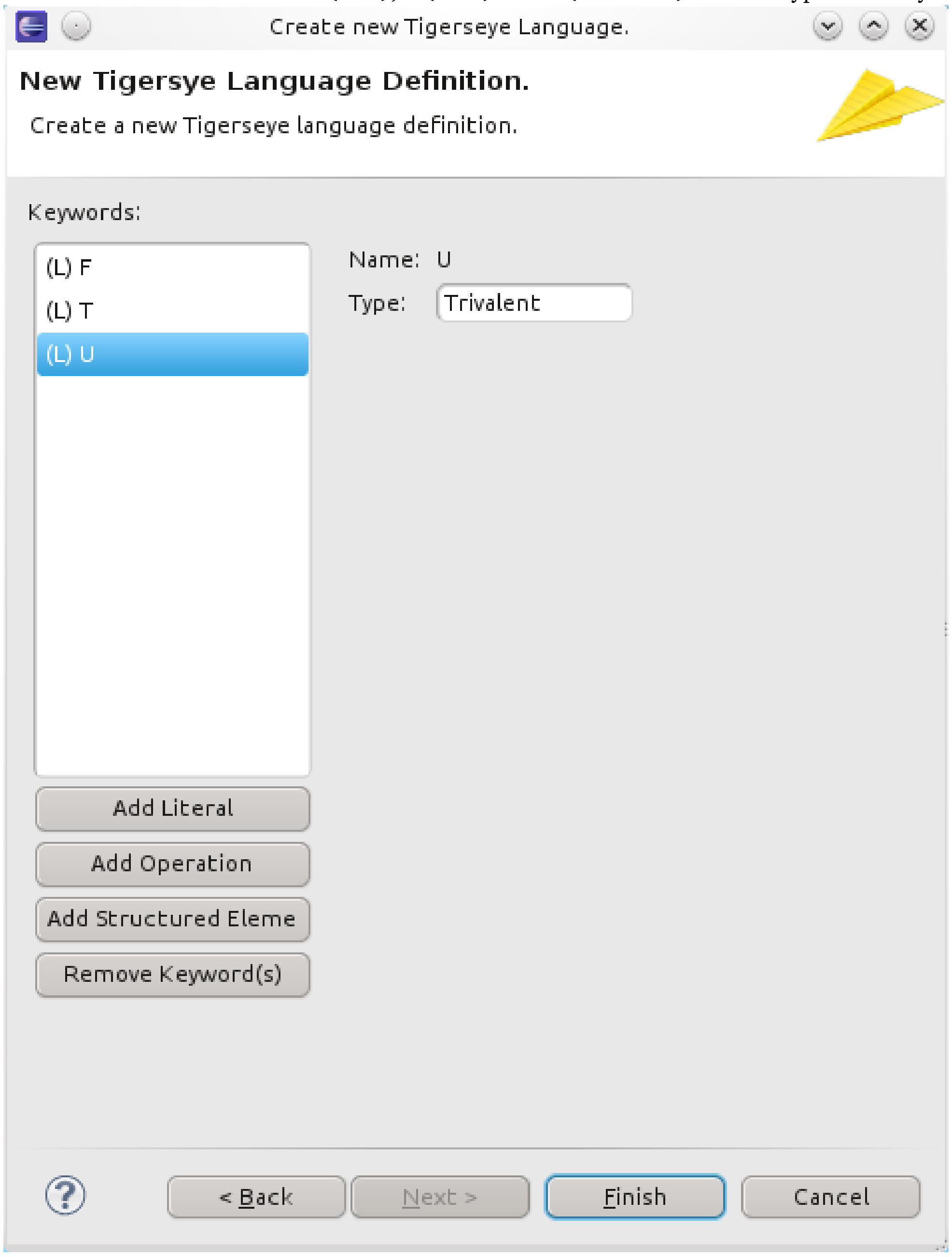
☐ Constructors from superclass

☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

☐ Generate comments

Now add three new literals: `T` (true), `F` (false) and `U` (unknown). Set the type for every literal to `Trivalent`.



Create new Tigerseye Language.

New Tigersye Language Definition.

Create a new Tigerseye language definition.

Keywords:

(L) F
(L) T
(L) U

Name: U

Type: Trivalent

When finishing the wizard these three classes and their super type Trivalent will be created. Notice that `T`, `F` and `U` extend Trivalent. The only operation will be an enhanced `println` which takes a String and a Trivalent expression and prints both. For example

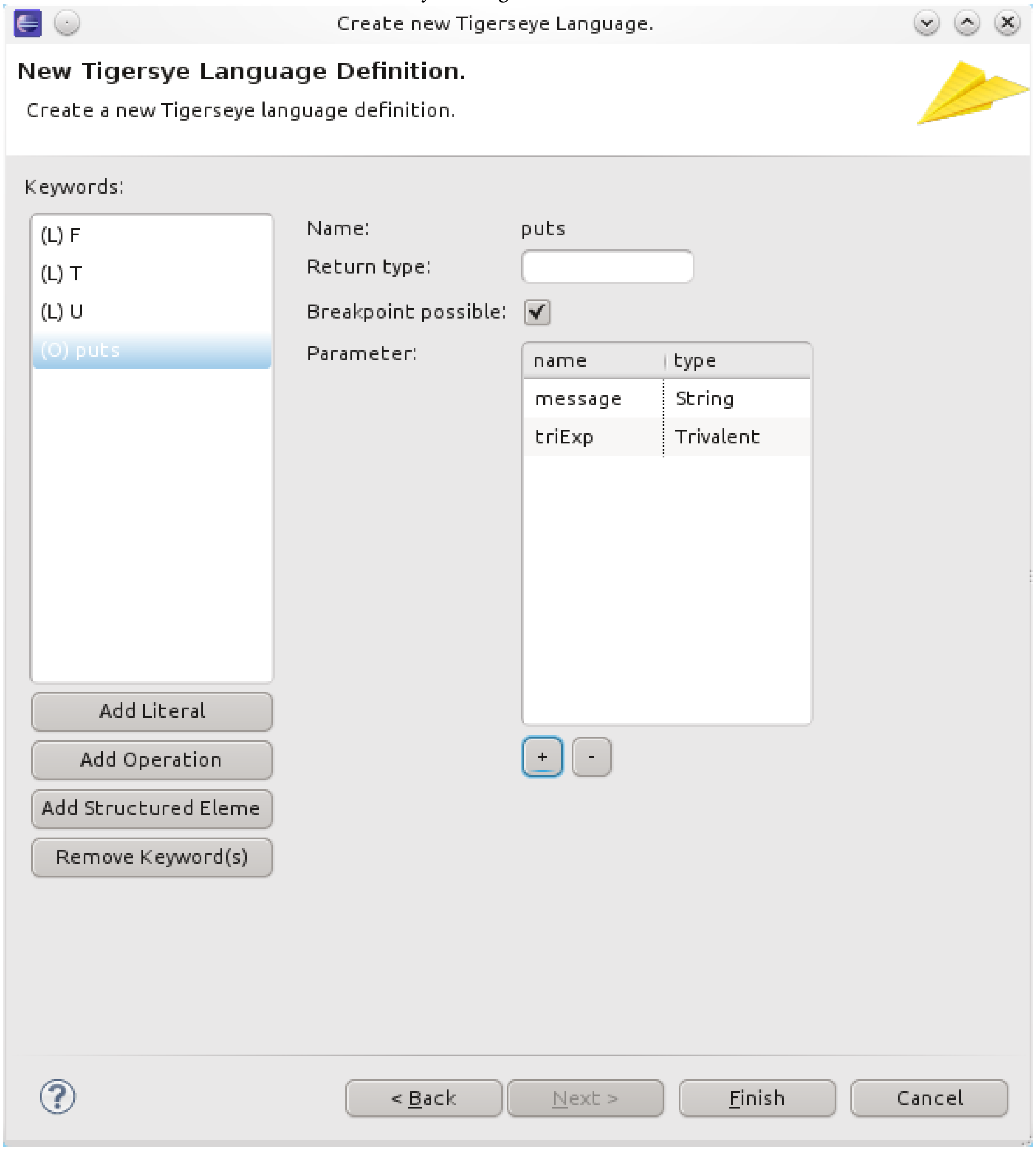
```
puts("T|U: ",T|U)
```

produces

```
T|U: T
```

The return type will be `void` so we leave the `Return type:` field empty. For each operation you can choose if setting a breakpoint on a line containing this keyword should be possible. You can define parameters and their types in the

`Parameter:` section. Parameters are added by clicking on the + button.

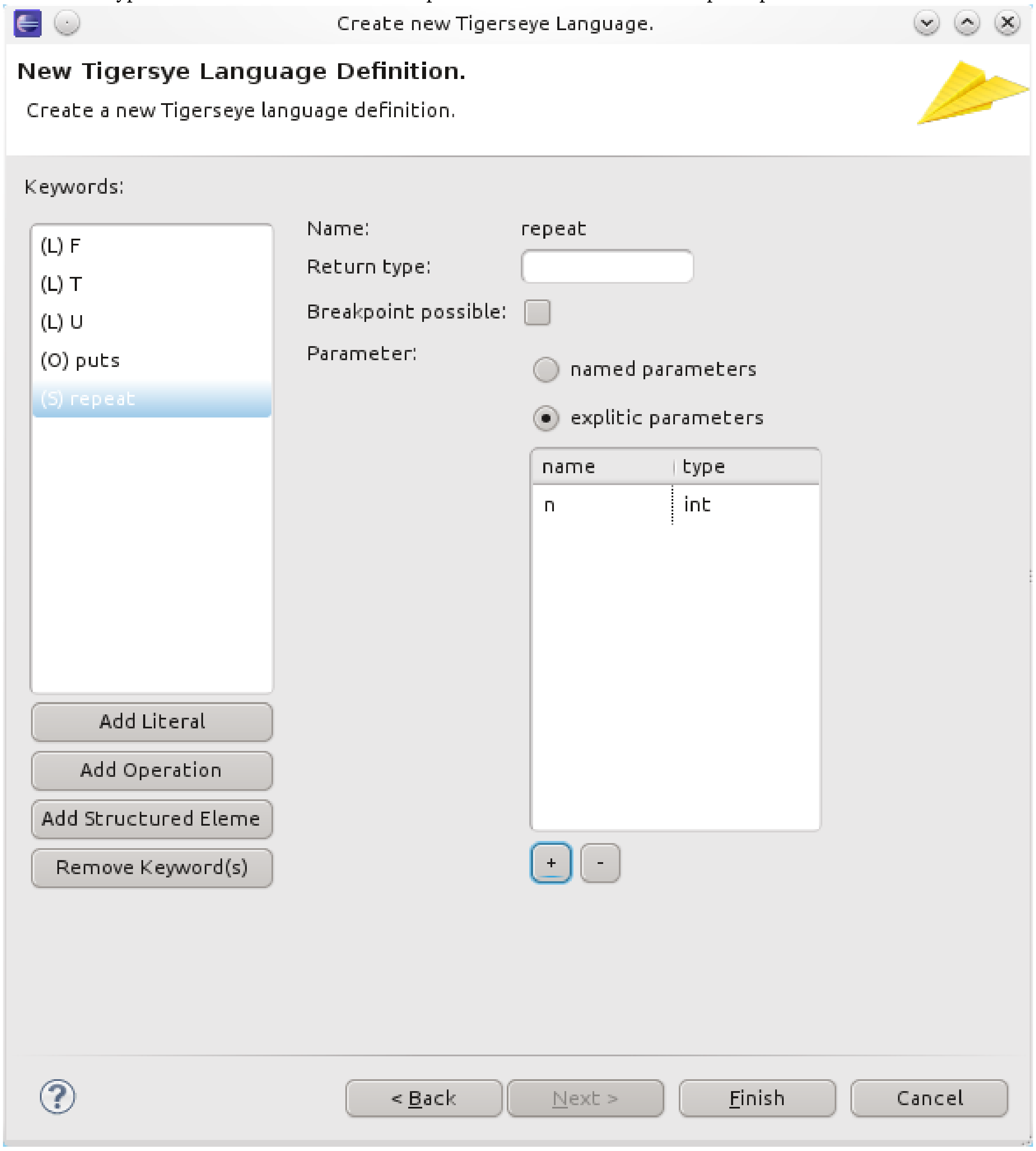


name	type
message	String
triExp	Trivalent

Now add a repeat-statement. The following will simply print `T: T` ten times to standard out.

```
repeat(10) {
   puts("T: ",T);
}
```

The return type will be void and there is one parameter named n. Select explicit parameters.



Create new Tigerseye Language.

New Tigersye Language Definition.

Create a new Tigerseye language definition.

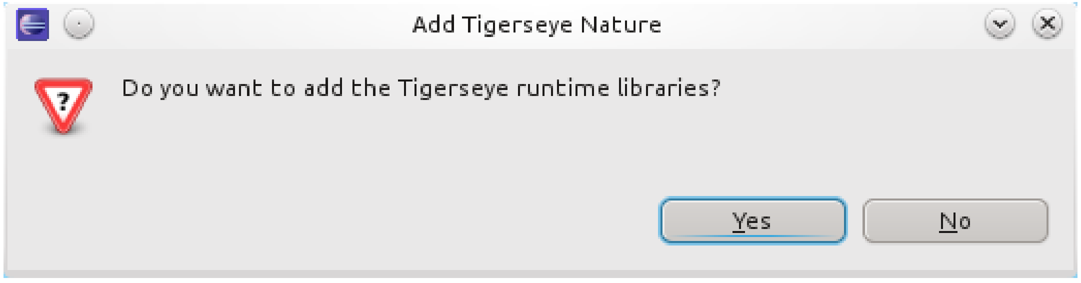
Keywords:

(L) F
(L) T
(L) U
(O) puts
(S) repeat

Name: repeat
Return type:
Breakpoint possible: ☐
Parameter:
( ) named parameters
(●) explitic parameters

name	type
n	int

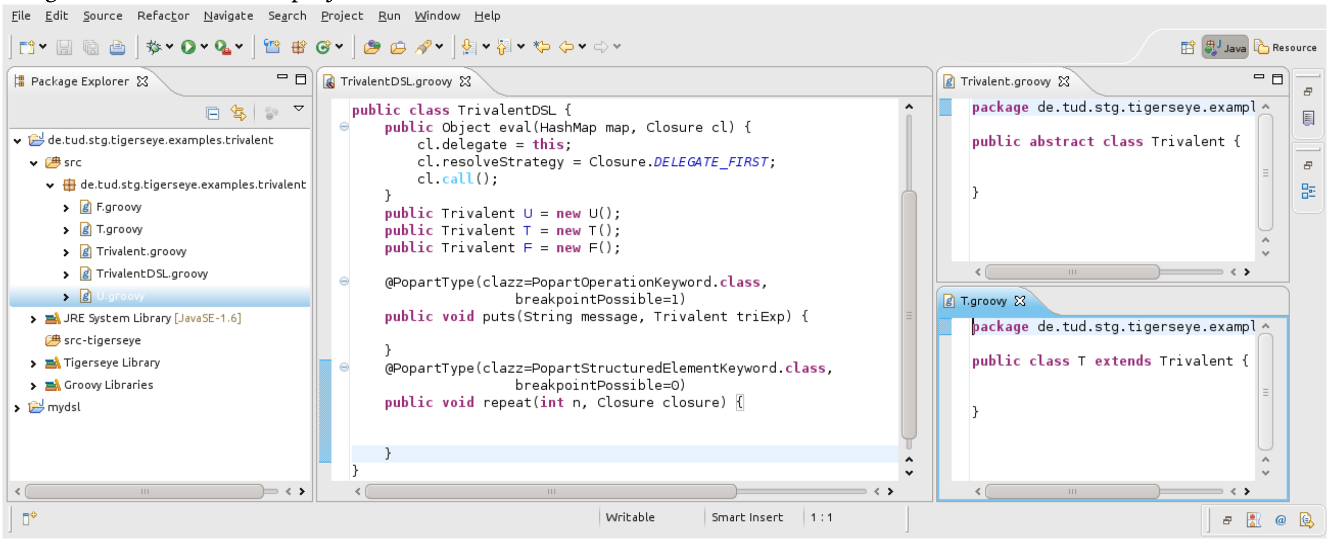
Again, you can choose if it should be possible to set a breakpoint on a line containing this keyword. After you select `Finish` a dialog will pop up asking you if you want to add the *TigersEye* runtime libraries.



Add Tigerseye Nature

Do you want to add the Tigerseye runtime libraries?

Usually you should say yes, since language definitions have dependencies to the runtime libraries. The next figure shows the generated classes and project structure.



```
public class TrivalentDSL {
    public Object eval(HashMap map, Closure cl) {
        cl.delegate = this;
        cl.resolveStrategy = Closure.DELEGATE_FIRST;
        cl.call();
    }
    public Trivalent U = new U();
    public Trivalent T = new T();
    public Trivalent F = new F();

    @PopartType(clazz=PopartOperationKeyword.class,
                breakpointPossible=1)
    public void puts(String message, Trivalent triExp) {

    }
    @PopartType(clazz=PopartStructuredElementKeyword.class,
                breakpointPossible=0)
    public void repeat(int n, Closure closure) {


    }
}
```

```
package de.tud.stg.tigerseye.exampl

public abstract class Trivalent {


}
```

```
package de.tud.stg.tigerseye.exampl

public class T extends Trivalent {


}
```

For the new type `Trivalent` as well as for the literals `T`,`U` and `F` a separate Groovy class has been created. The language configuration is defined in the `TrivalentDSL` class.

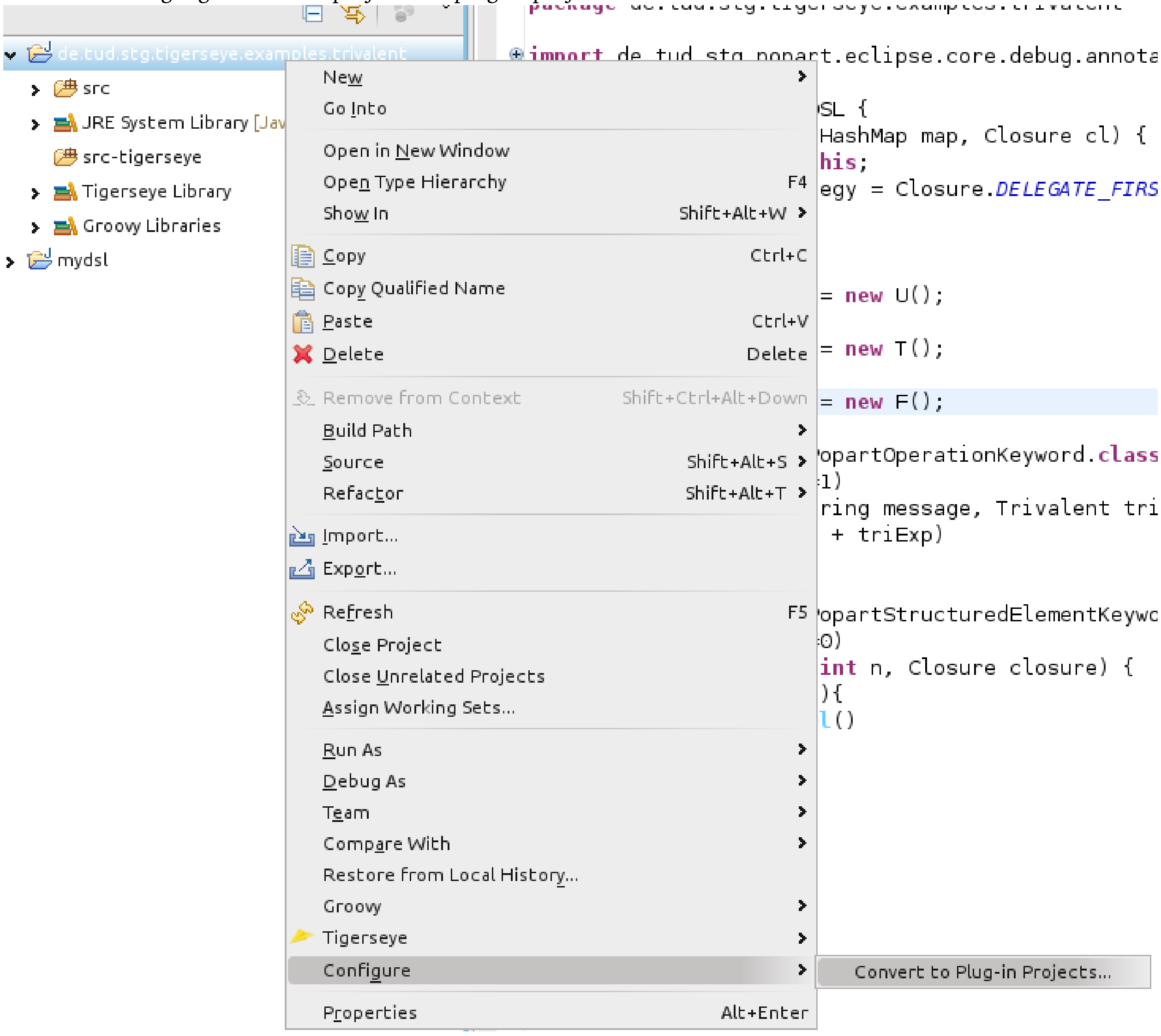
Deployment of a Language

This section shows how to deploy a *TigersEye* language to a plug-in project. This plug-in project will declare its dependencies to two plug-ins:

- de.tud.stg.tigerseye
- de.tud.stg.tigerseye.eclipse.core

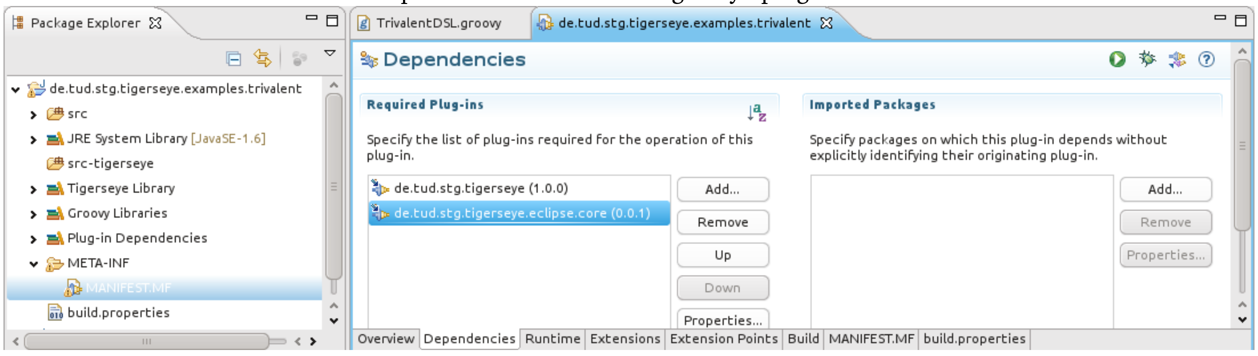
The `de.tud.stg.tigerseye` plug-in provides the dependent on runtime libraries and the `de.tud.stg.tigerseye.eclipse.core` plug-in the extension which declares that this plug-in project actually provides a new language. The following steps have to be performed:

1. Convert the language definition project to a plug-in project.



```
import de.tud.stg.popart.eclipse.core.debug.annota
SL {
HashMap map, Closure cl) {
his;
egy = Closure.DELEGATE_FIRS
= new U();
= new T();
= new F();
opartOperationKeyword.class
1)
ring message, Trivalent tri
 + triExp)
opartStructuredElementKeywo
0)
int n, Closure closure) {
){
l()
```

2. In the MANIFEST.MF file add the dependencies to the two Tigerseye plug-ins.

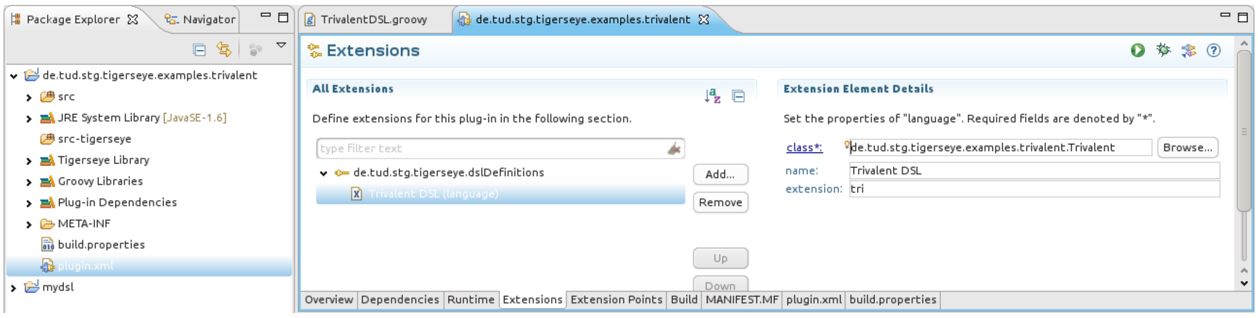


Specify the list of plug-ins required for the operation of this plug-in.

- de.tud.stg.tigerseye (1.0.0)
- de.tud.stg.tigerseye.eclipse.core (0.0.1)

Specify packages on which this plug-in depends without explicitly identifying their originating plug-in.

3. Now open the plugin.xml and go the Extensions tab. Add the `de.tud.stg.tigerseye.dslDefinitions` extension point. On the extension point select `New > language`. There you can define the language class to be used. In this example `de.tud.stg.tigerseye.examples.trivalent.Trivalent` is used. Additionally you should define a user friendly name of your new language, e.g. *Trivalent DSL*. Optionally you can define the default extension identifying your language, such as `tri`. The following figure shows the configuration.



- de.tud.stg.tigerseye.examples.trivalent
  - src
  - JRE System Library [JavaSE-1.6]
  - src-tigerseye
  - Tigerseye Library
  - Groovy Libraries
  - Plug-in Dependencies
  - META-INF
  - build.properties
  - plugin.xml
- mydsl

Define extensions for this plug-in in the following section.

- de.tud.stg.tigerseye.dslDefinitions
  - Trivalent DSL (language)

Set the properties of "language". Required fields are denoted by "*".

class*: de.tud.stg.tigerseye.examples.trivalent.Trivalent
name: Trivalent DSL
extension: tri

The `extension` field can later also be configured via the preference pages.

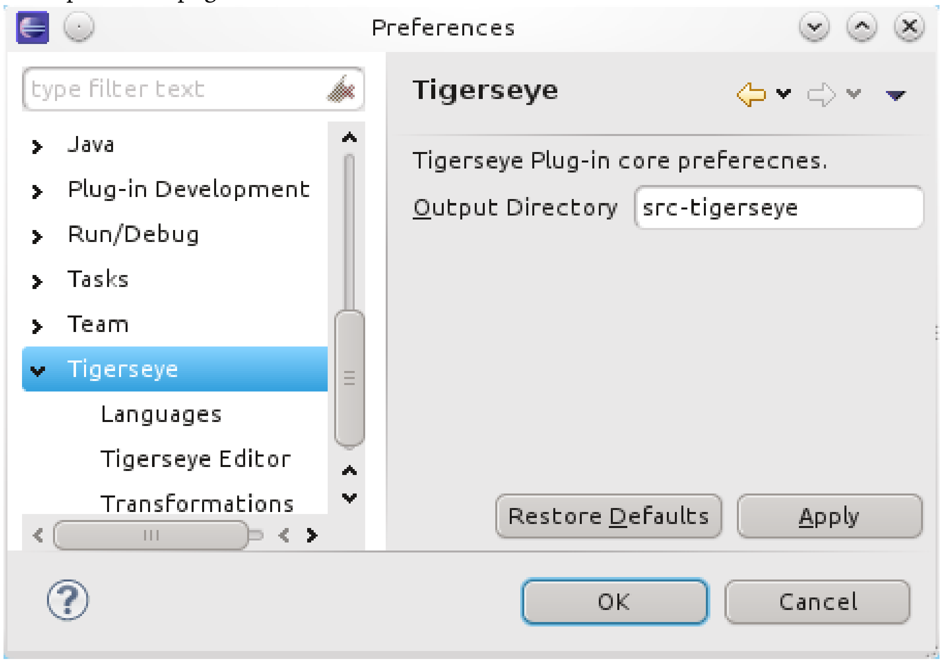
4. Currently only the deployment for development is supported. A language can either be copied or linked inside the eclipse instance in which the plug-in is developed. The next time the Eclipse instance is started the language will be visible in the preference pages and can be used.

Features

The current version of *TigersEye* provides the fundamental functionalities in order to be able to use it as a language workbench. New languages can easily be created and deployed. After a restart they can be used in projects that have the *TigersEye* nature. A *TigersEye* nature can easily be added and removed via the context menu. The installed DSLs can be configured through the preference pages, as well as the employed transformations. The editor is an extended version of the Groovy editor and provides keyword coloring for keywords of installed and activated DSLs.

Preferences

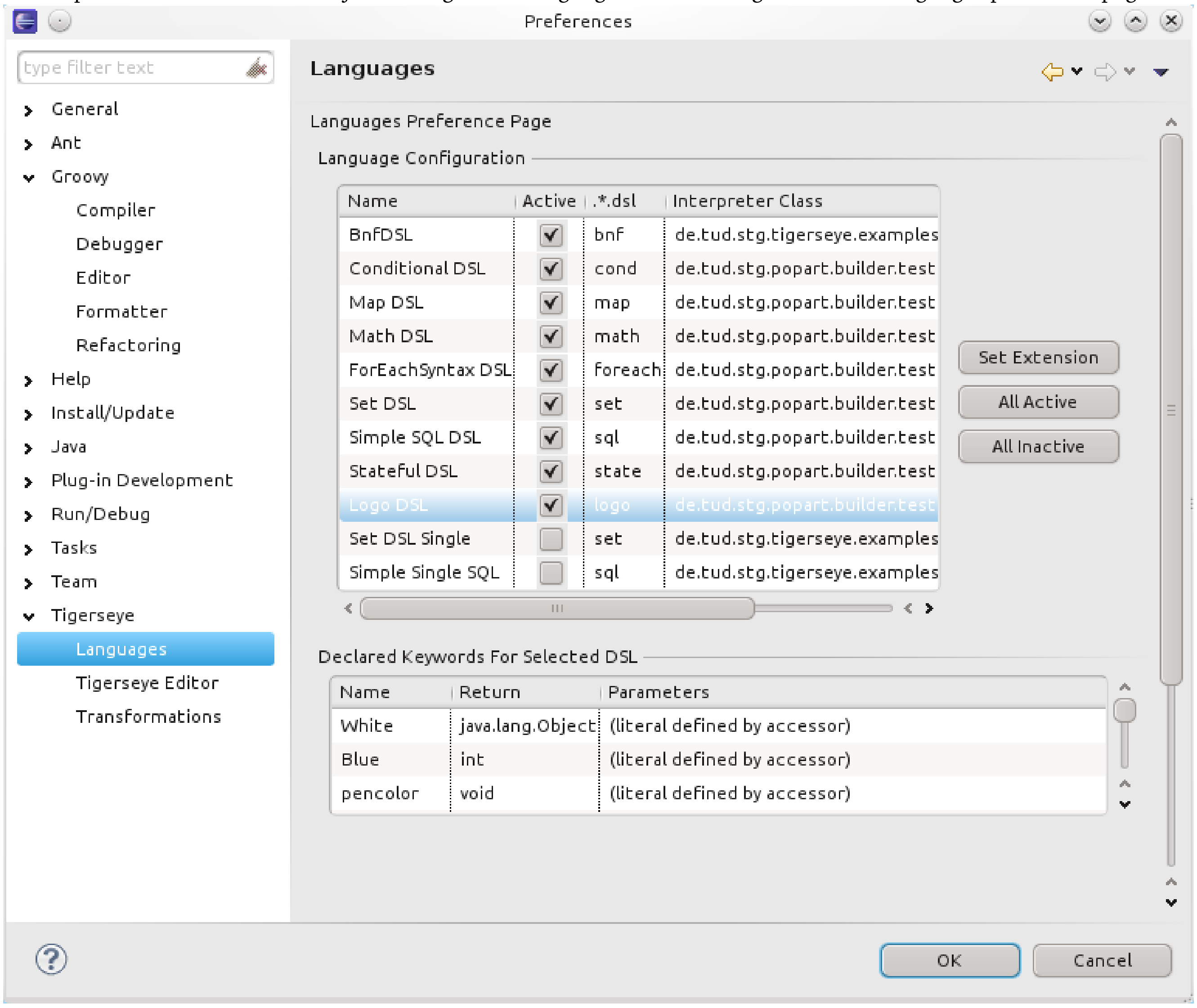
The preference pages are an important part of *TigersEye*, since they provide the configuration of registered DSLs. In the main prefrence page



Tigerseye Plug-in core preferecnes.

Output Directory: src-tigerseye

the output source folder can be adjusted. Registered languages can be configured on the languages preference page.



Languages

Languages Preference Page

Language Configuration

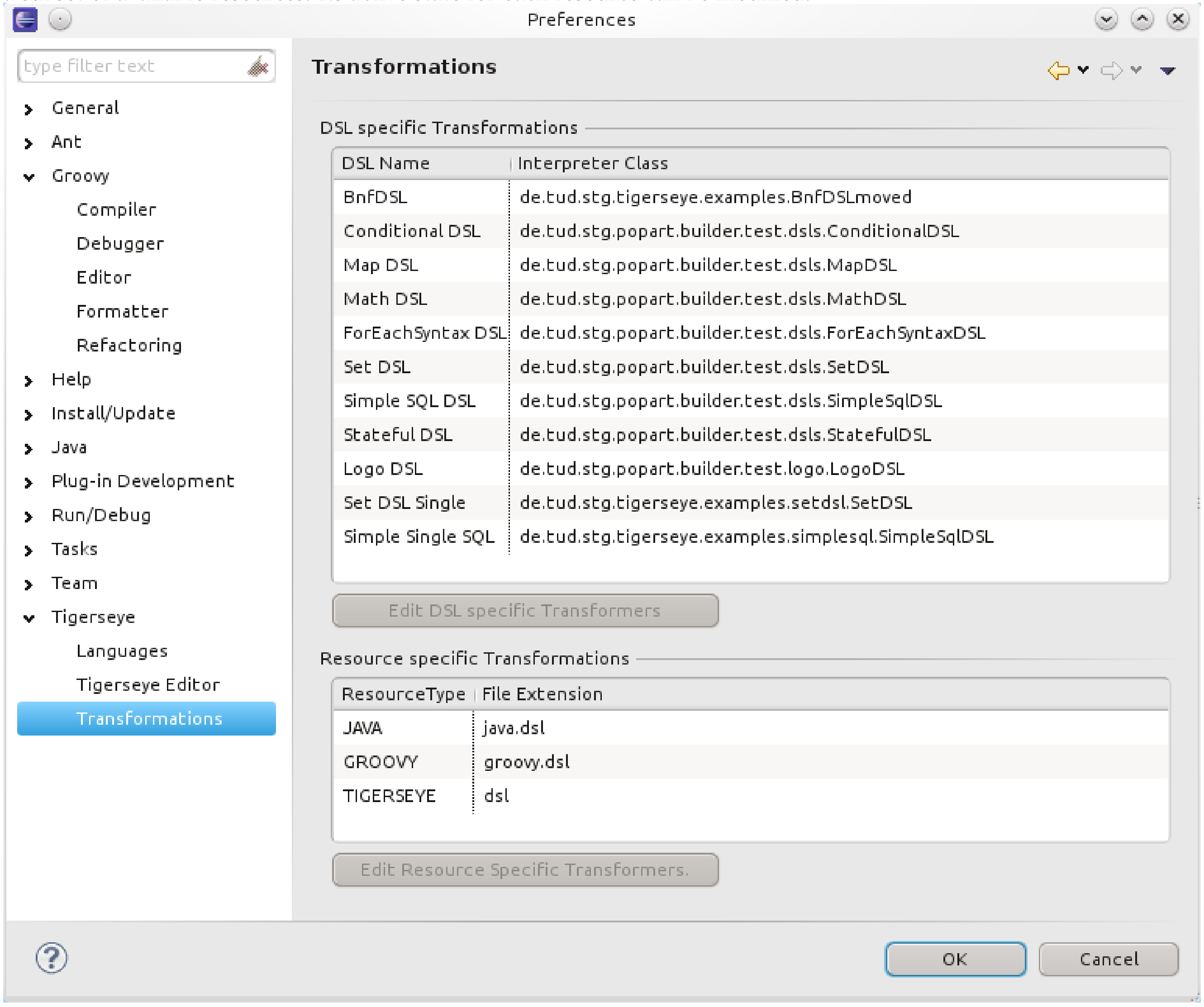
Name	Active	.*.dsl	Interpreter Class
BnfDSL	✓	bnf	de.tud.stg.tigerseye.examples
Conditional DSL	✓	cond	de.tud.stg.popart.builder.test
Map DSL	✓	map	de.tud.stg.popart.builder.test
Math DSL	✓	math	de.tud.stg.popart.builder.test
ForEachSyntax DSL	✓	foreach	de.tud.stg.popart.builder.test
Set DSL	✓	set	de.tud.stg.popart.builder.test
Simple SQL DSL	✓	sql	de.tud.stg.popart.builder.test
Stateful DSL	✓	state	de.tud.stg.popart.builder.test
Logo DSL	✓	logo	de.tud.stg.popart.builder.test
Set DSL Single		set	de.tud.stg.tigerseye.examples
Simple Single SQL		sql	de.tud.stg.tigerseye.examples

Declared Keywords For Selected DSL

Name	Return	Parameters
White	java.lang.Object	(literal defined by accessor)
Blue	int	(literal defined by accessor)
pencolor	void	(literal defined by accessor)

DSLs have to be set into activated state in order to use them. The extensions, that indicate which concrete DSL class is responsible for which files of the according extension can be adjusted in the Extensions column. When a DSL is selected its keywords are shown below in the *Declared Keywords of Selected DSL* table.
In the transformations preference page used transformations can be configured. A transformation usually only supports

a subset of available resources. Its active state for each resource can be modified.



Transformations

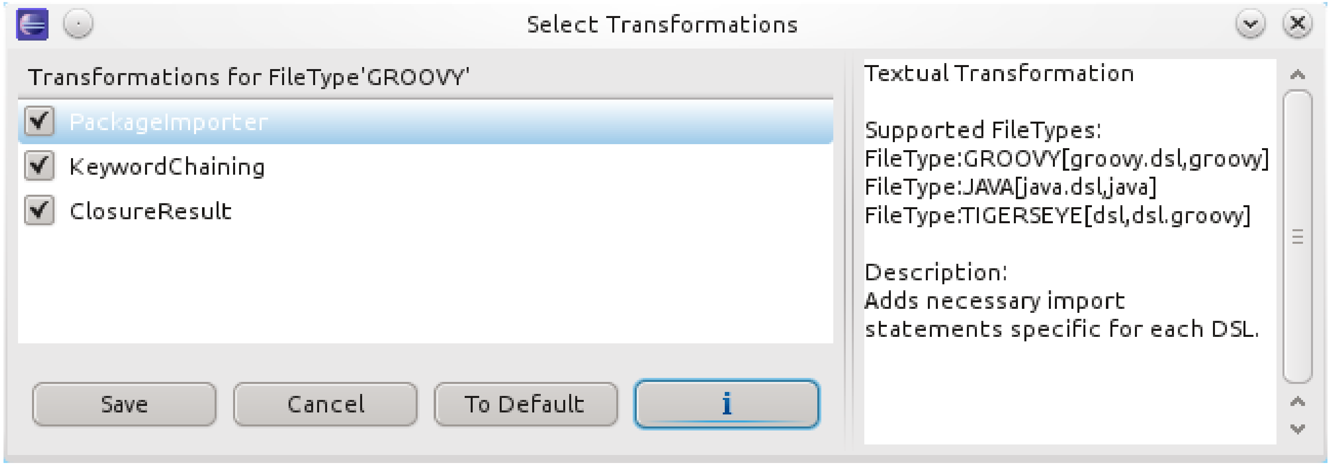
DSL specific Transformations

DSL Name	Interpreter Class
BnfDSL	de.tud.stg.tigerseye.examples.BnfDSLmoved
Conditional DSL	de.tud.stg.popart.builder.test.dsls.ConditionalDSL
Map DSL	de.tud.stg.popart.builder.test.dsls.MapDSL
Math DSL	de.tud.stg.popart.builder.test.dsls.MathDSL
ForEachSyntax DSL	de.tud.stg.popart.builder.test.dsls.ForEachSyntaxDSL
Set DSL	de.tud.stg.popart.builder.test.dsls.SetDSL
Simple SQL DSL	de.tud.stg.popart.builder.test.dsls.SimpleSqlDSL
Stateful DSL	de.tud.stg.popart.builder.test.dsls.StatefulDSL
Logo DSL	de.tud.stg.popart.builder.test.logo.LogoDSL
Set DSL Single	de.tud.stg.tigerseye.examples.setdsl.SetDSL
Simple Single SQL	de.tud.stg.tigerseye.examples.simplesql.SimpleSqlDSL

Resource specific Transformations

ResourceType	File Extension
JAVA	java.dsl
GROOVY	groovy.dsl
TIGERSEYE	dsl

For each resource and DSL different transformations might be available and active. These can be configured using the *Select Transformations* dialog.



Transformations for FileType'GROOVY'

- ✓ PackageImporter
- ✓ KeywordChaining
- ✓ ClosureResult

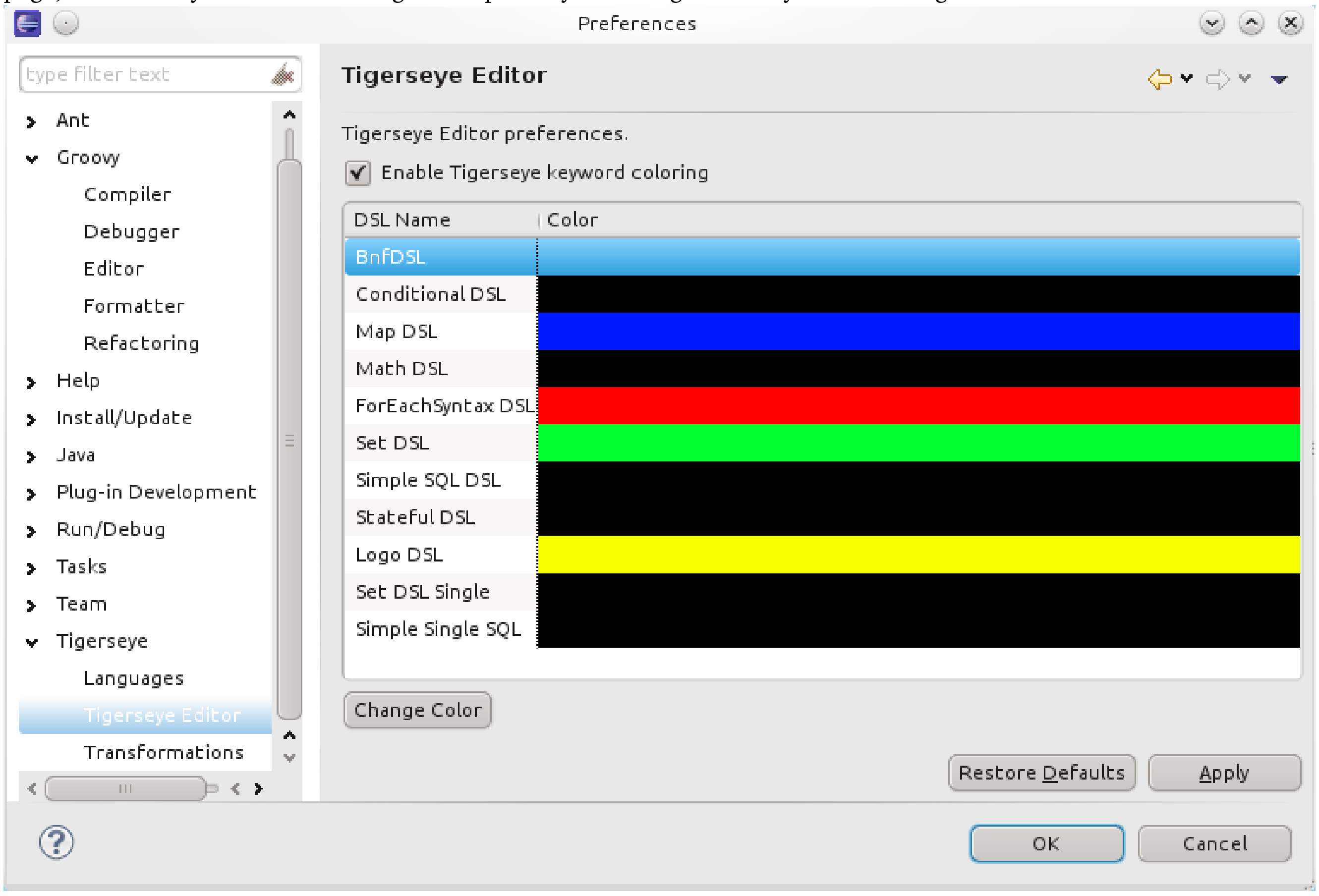
Textual Transformation

Supported FileTypes:
FileType:GROOVY[groovy.dsl,groovy]
FileType:JAVA[java.dsl,java]
FileType:TIGERSEYE[dsl,dsl.groovy]

Description:
Adds necessary import statements specific for each DSL.

The dialog also shows additional informations about the currently selected transformation in a tray window that can be opened clicking on the additinal information button. i

The editor provides keyword coloring for active DSLs. The colors can be configured using the *Tigerseye Editor* preference page, where every DSL can be configured separately and the general keyword coloring can be activated or deactivated.



Tigerseye Editor

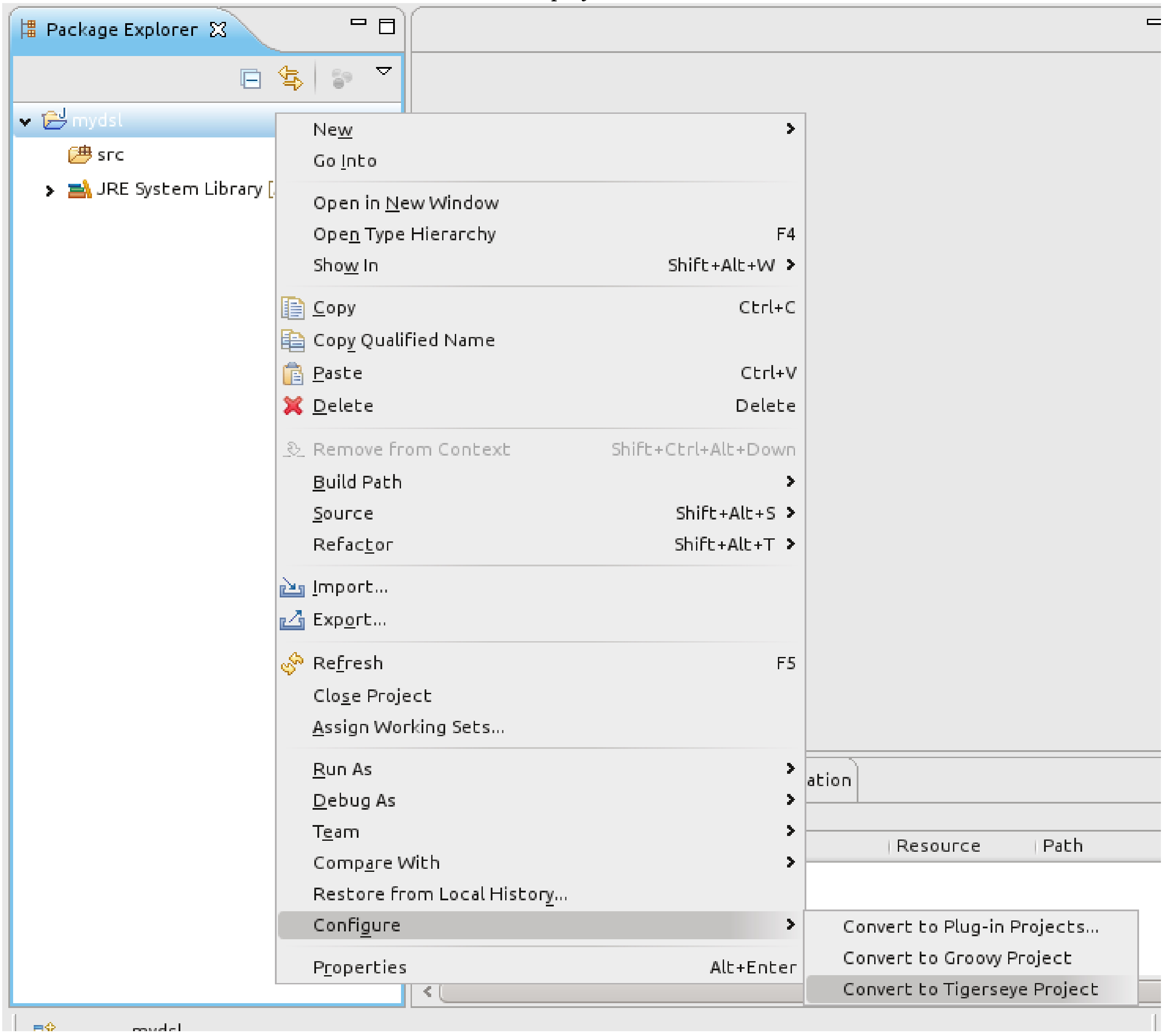
Tigerseye Editor preferences.

[✓] Enable Tigerseye keyword coloring

DSL Name	Color
BnfDSL	🟦
Conditional DSL	⬛
Map DSL	🟦
Math DSL	⬛
ForEachSyntax DSL	🟥
Set DSL	🟩
Simple SQL DSL	⬛
Stateful DSL	⬛
Logo DSL	🟨
Set DSL Single	⬛
Simple Single SQL	⬛

Add and Remove Nature

TigersEye has additional requirements which will be imported when adding the *TigersEye* nature to a project. The project must have at least the Java nature otherwise the transformation to a *TigersEye* project is not possible. Following figure shows the available context menu to add the nature to a project.

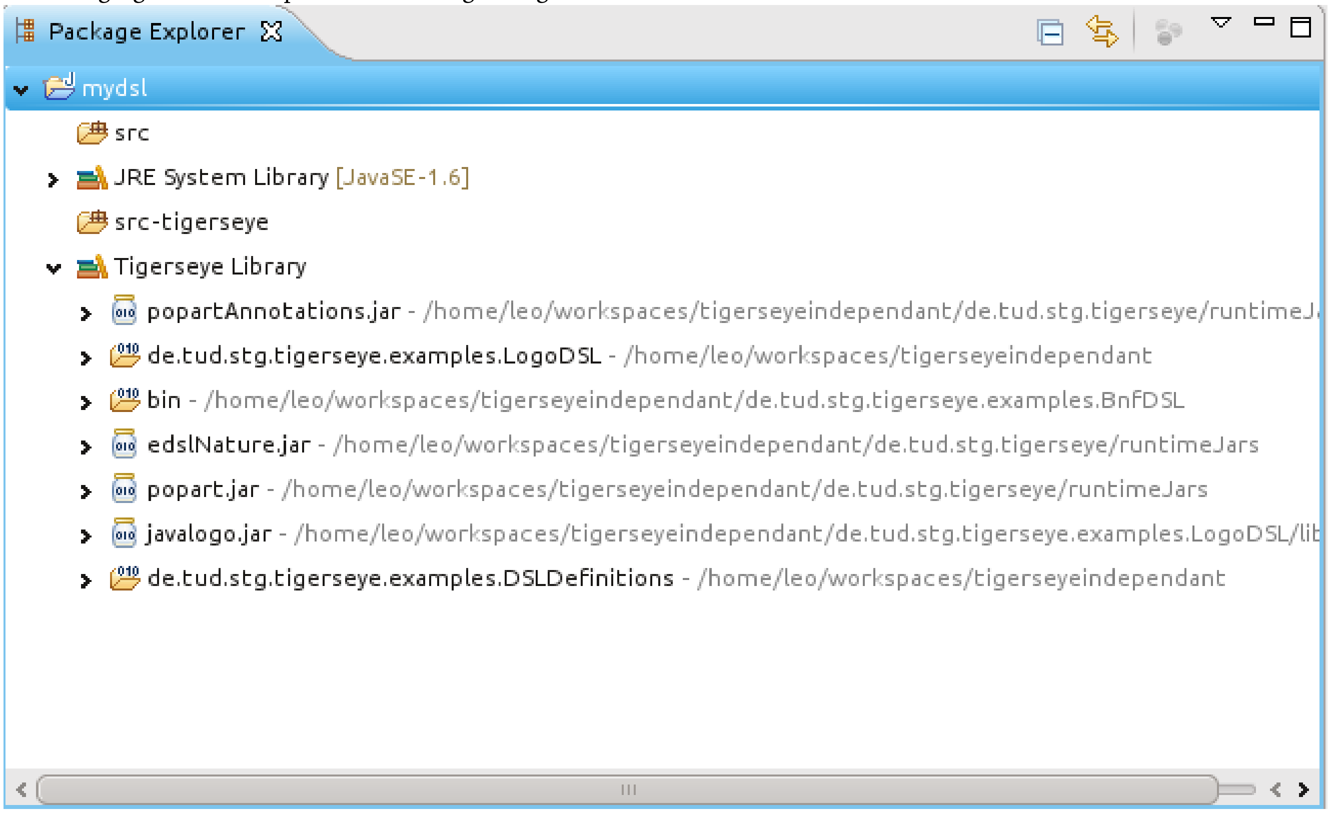


This will do two things:

1. A seperate source folder will be created into which the translated DSL files will be put (here: `src-tigerseye`).

2. A new class path container will be added which contains the runtime libraries (popartAnnotations.jar, popart.jar, edslNature.jar) and the libraries of registered DSLs as well as their dependencies. For example `de.tud.stg.tigerseye.examples.LogoDSL` and `de.tud.stg.tigerseye.examples.DSLDefinitions`. Additionally the GroovyNature will be added if not already configured.

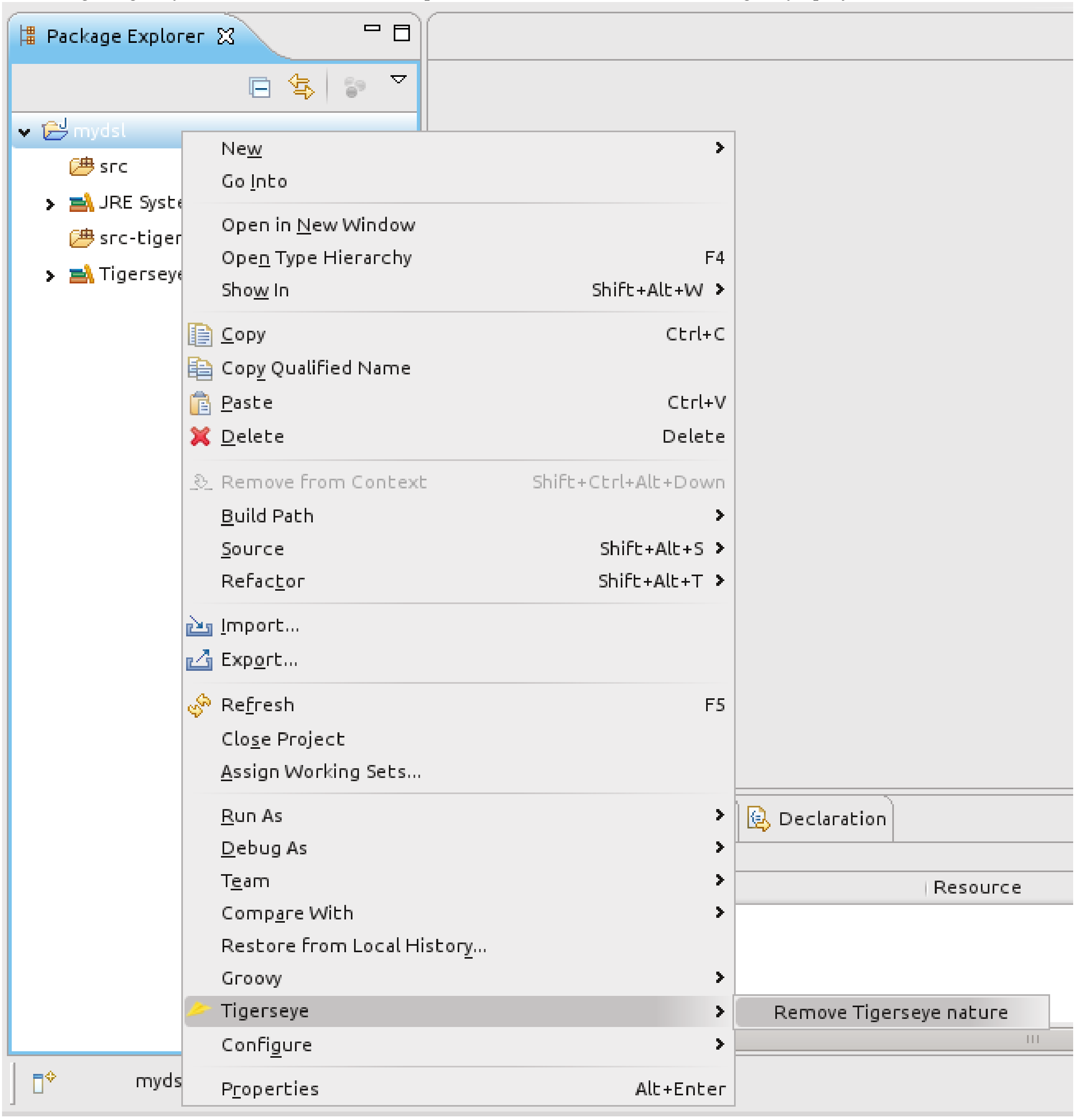
Following figure shows a possible resulting configuration.



Package Explorer

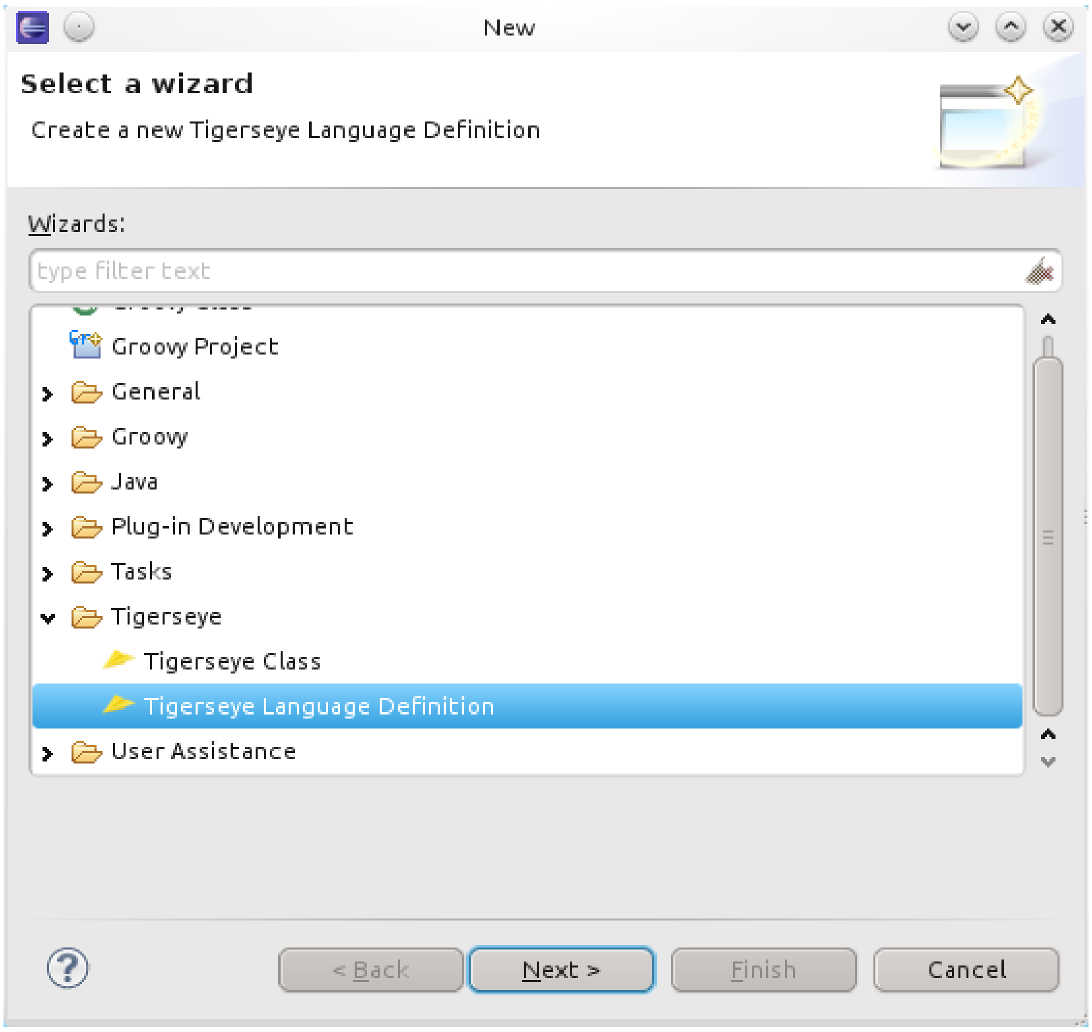
mydsl
src
JRE System Library [JavaSE-1.6]
src-tigerseye
Tigerseye Library
popartAnnotations.jar - /home/leo/workspaces/tigerseyeindependant/de.tud.stg.tigerseye/runtimeJ
de.tud.stg.tigerseye.examples.LogoDSL - /home/leo/workspaces/tigerseyeindependant
bin - /home/leo/workspaces/tigerseyeindependant/de.tud.stg.tigerseye.examples.BnfDSL
edslNature.jar - /home/leo/workspaces/tigerseyeindependant/de.tud.stg.tigerseye/runtimeJars
popart.jar - /home/leo/workspaces/tigerseyeindependant/de.tud.stg.tigerseye/runtimeJars
javalogo.jar - /home/leo/workspaces/tigerseyeindependant/de.tud.stg.tigerseye.examples.LogoDSL/lit
de.tud.stg.tigerseye.examples.DSLDefinitions - /home/leo/workspaces/tigerseyeindependant

Removing a *TigersEye* nature can also be accomplished via the context menu of a *TigersEye* project.



Language Definition Wizard

A new language can be created using the *Tigerseye Language Wizard*. The Wizard can be accessed via `File > New > Other`



New

Select a wizard

Create a new Tigerseye Language Definition

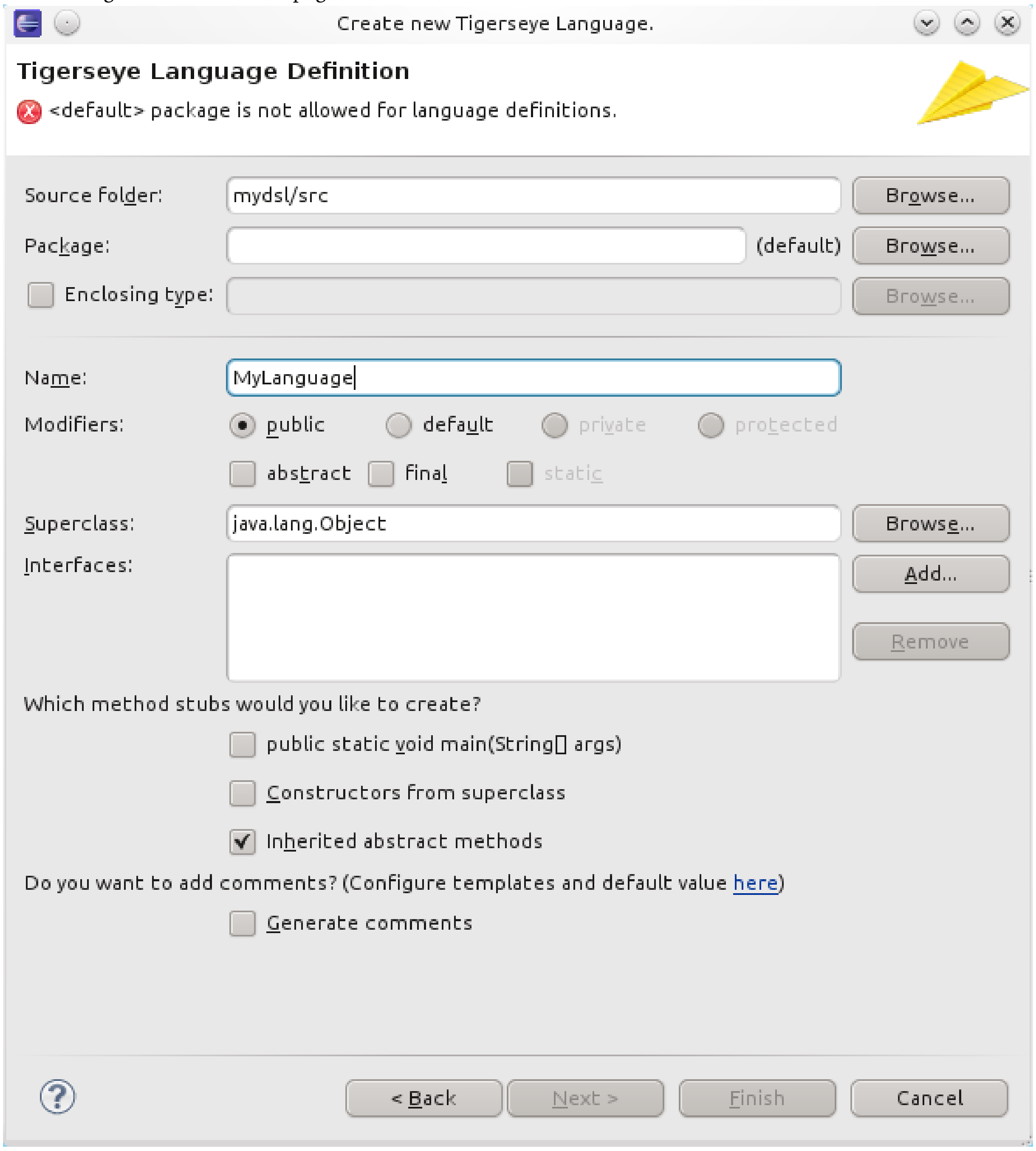
Wizards:

type filter text

- Groovy Project
- General
- Groovy
- Java
- Plug-in Development
- Tasks
- Tigerseye
  - Tigerseye Class
  - Tigerseye Language Definition
- User Assistance

< Back | Next > | Finish | Cancel

The next figure shows the first page of the Wizard.


Create new Tigerseye Language.

Tigerseye Language Definition

<default> package is not allowed for language definitions.

Source folder: mydsl/src [Browse...]

Package: (default) [Browse...]

☐ Enclosing type: [Browse...]

Name: MyLanguage

Modifiers: [● public] ○ default ○ private ○ protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object [Browse...]

Interfaces: [Add...] [Remove]

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

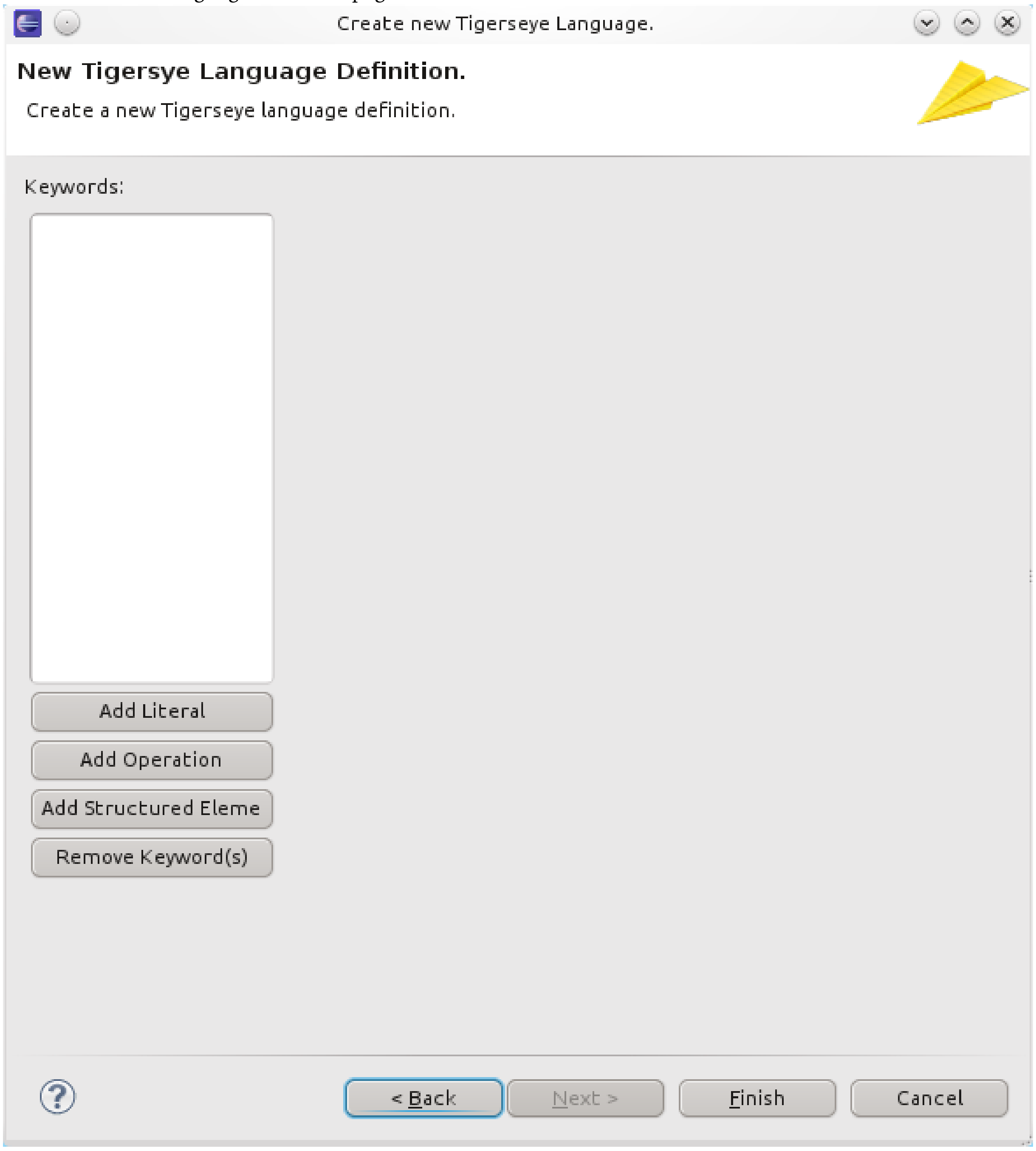
☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

☐ Generate comments

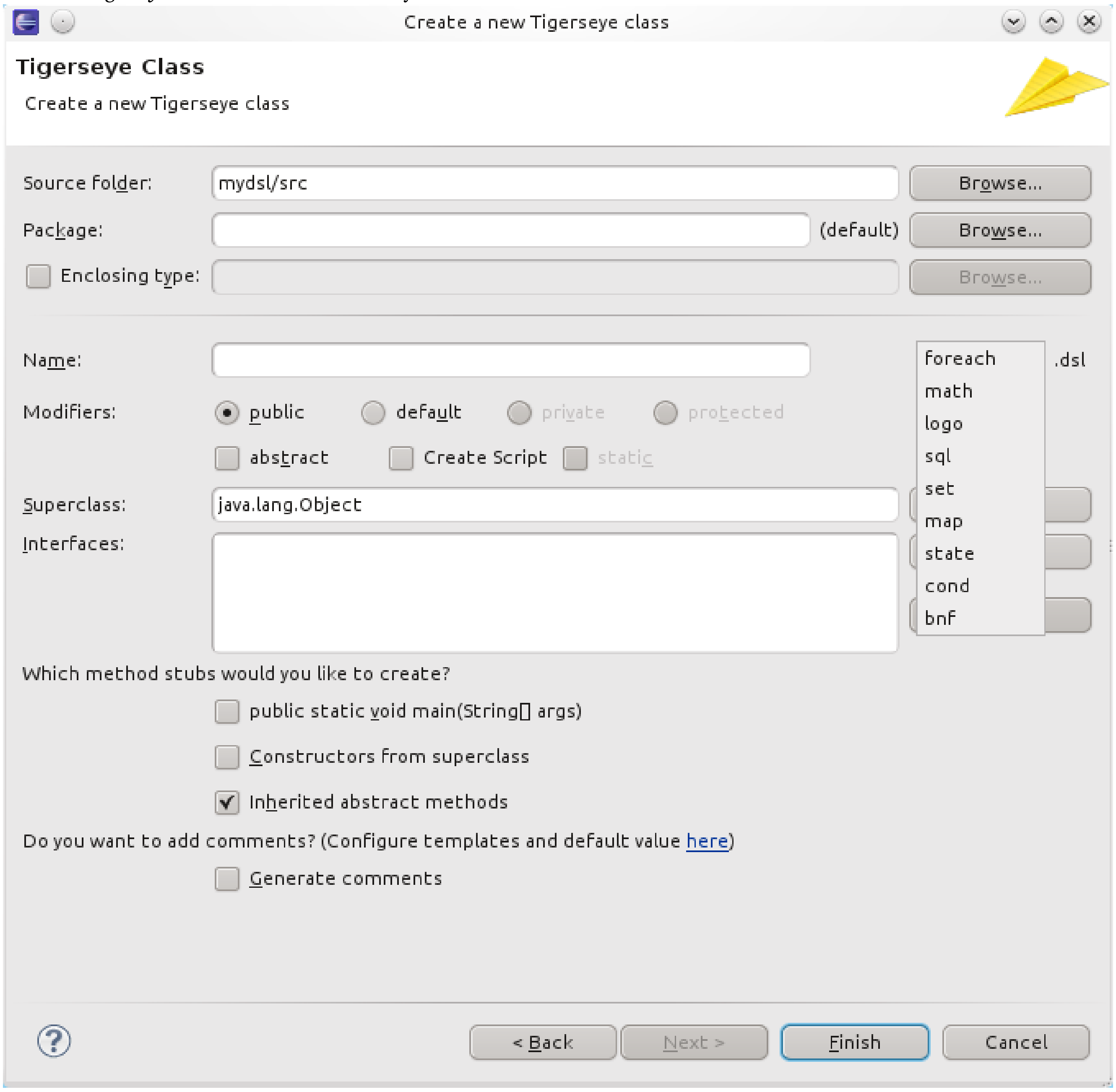
< Back | Next > | Finish | Cancel

There the name of the main language class can be defined. As shown the default package is not a valid package for a language definition, since this will cause problems when trying to use the language within a Java class. The next figure shows the actual language definition page.


Create new Tigerseye Language.
New Tigersye Language Definition.
Create a new Tigerseye language definition.
Keywords:
Add Literal
Add Operation
Add Structured Eleme
Remove Keyword(s)
< Back
Next >
Finish
Cancel

On the language definition page the different literals, operations and structured elements can be added. In Section Examples the usage of the wizard is showcased.

The *New Tigerseye Class* Wizard enables easy creation of new DSL classes.



Tigerseye Class

Create a new Tigerseye class

Source folder: mydsl/src

Package: (default)

Enclosing type:

Name: .dsl

Modifiers: [✓ public] default private protected

abstract Create Script static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

public static void main(String[] args)

Constructors from superclass

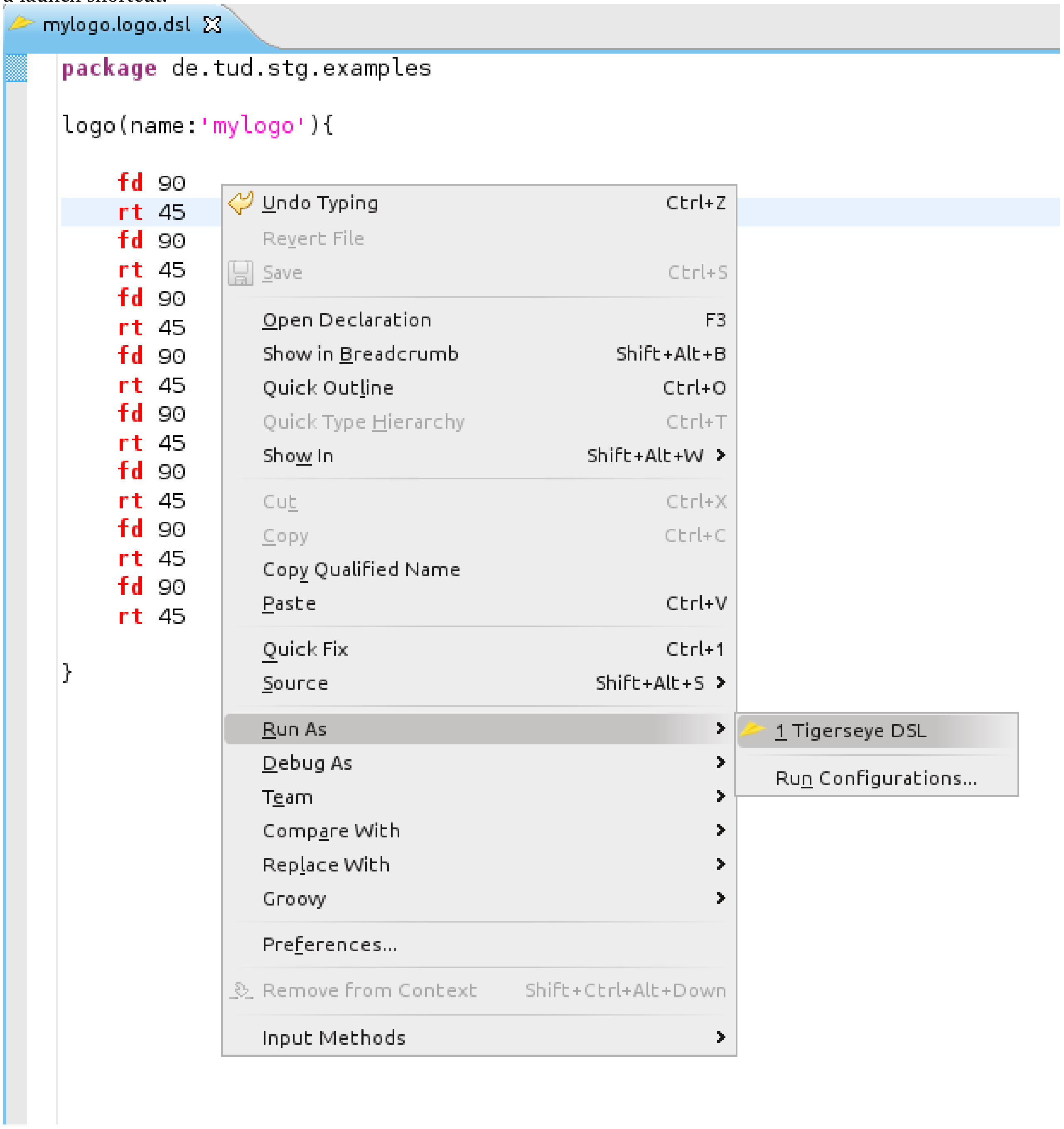
[✓ Inherited abstract methods]

Do you want to add comments? (Configure templates and default value here)

Generate comments

It is basically a version of the *Groovy Class* Wizard. Not all modifiable areas have influence on the generated code. The useful fields for generation are `Source folder`, `Package`, `Name` and the DSL extension.

A DSL can be launched using a *launch shortcut* or via the *Run Configurations* dialog. The next figure shows a launch using a launch shortcut.



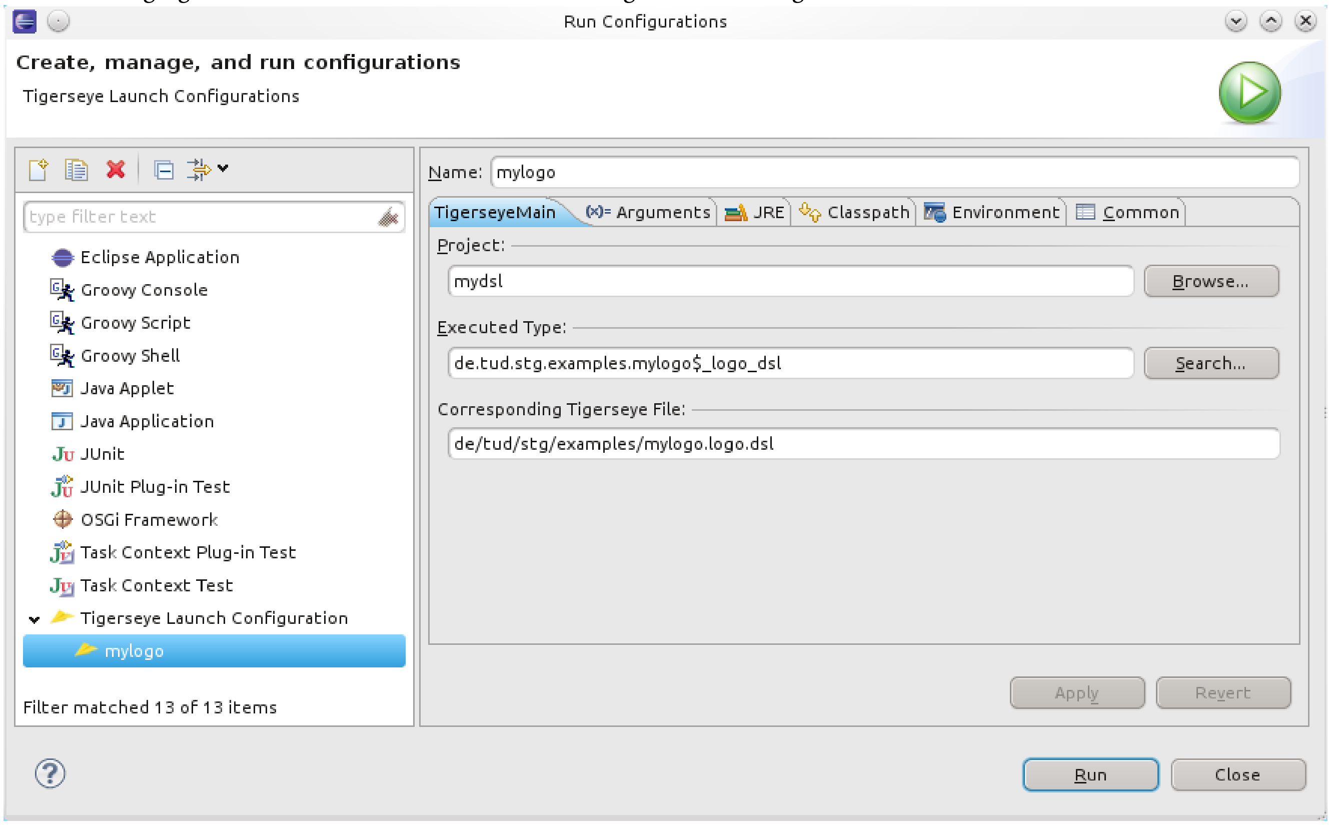
```
package de.tud.stg.examples

logo(name:'mylogo'){

      fd 90
      rt 45
      fd 90
      rt 45
      fd 90
      rt 45
      fd 90
      rt 45
      fd 90
      rt 45
      fd 90
      rt 45
      fd 90
      rt 45
      fd 90
      rt 45

}
```

The following figure shows the launch via the Run Configurations Dialog.



Run Configurations

Create, manage, and run configurations

Tigerseye Launch Configurations

Name: mylogo

Project: mydsl

Executed Type: de.tud.stg.examples.mylogo$_logo_dsl

Corresponding Tigerseye File: de/tud/stg/examples/mylogo.logo.dsl

Filter matched 13 of 13 items

In the Run Configurations Dialog a new launch can be configured or a previous launch adjusted. On the `Tigerseye` tab the project from which a DSL will be launched as well as the DSL file to launch can be chosen. When using the launch shortcut the Groovy default launch configuration is assumed, which will set additional classpath properties. Later these can be modified using this dialog.

List of Tables