jStar Eclipse tutorial

September 30, 2010

1 Installation

Prerequisites:

- JDK 6 (not JRE since it does not have a compiler needed for annotation processing). You can specify it in eclipse.ini:

  Windows Example
  -vm
  C:\Java\JDK\1.6\bin\javaw.exe

 Linux Example
  -vm
  /opt/sun-jdk-1.6.0.02/bin/java

  Mac Example
  -vm
  /System/Library/Frameworks/JavaVM.framework/Versions/1.6.0/Home/bin/java

 For more information go to http://wiki.eclipse.org/Eclipse.ini

Two ways:

1. Update site (Recommended). Go to **Help → Install New Software...**.
   Click **Add...** and type in the name and update site for the plugin.
   The update site for the jStar Eclipse Plug-in is
   http://github.com/da319/jStar-eclipse/raw/master/com.jstar.eclipse.update.site/
   Press **OK**.
   Uncheck **Group items by category**.
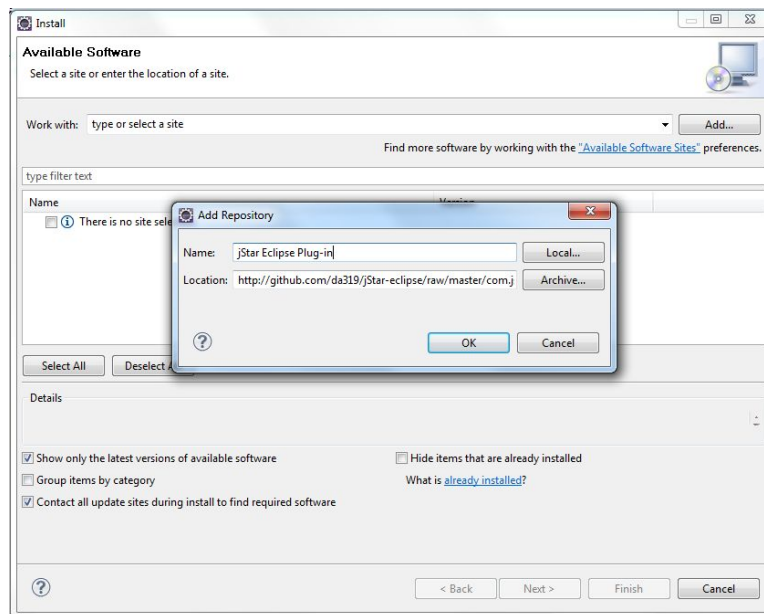
The jStar Eclipse Plug-in should appear, as the following figure shows.
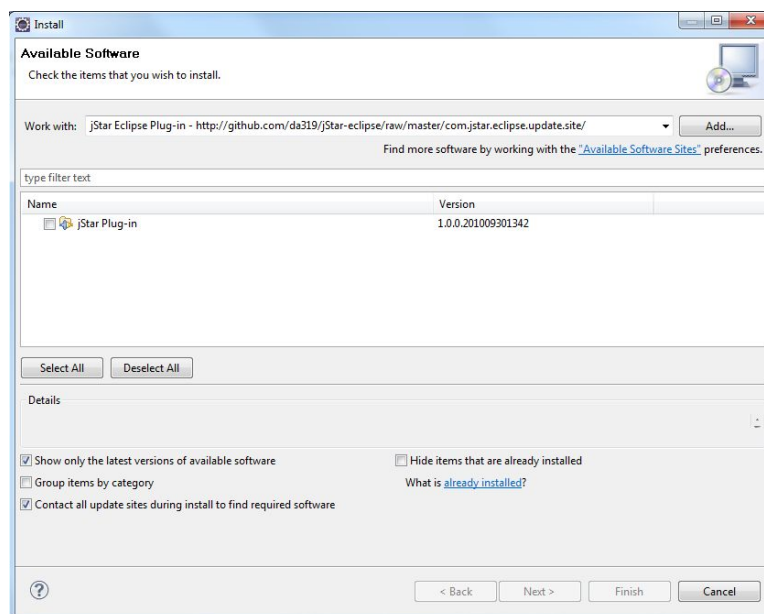
Select the plugin and click **Next**.

Click **Next** in the following dialog. Accept the terms of the license agreement and click **Finish**.

2. Manually. Add the latest plugin jar file
   com.jstar.eclipse.update.site/plugins/com.jstar.eclipse_1.0.0.x.jar
   to `eclipse/dropins/` folder and restart Eclipse.

2 Configuration

2.1 Windows with cygwin

Go to Windows → Preferences → jStar and set the required directories.
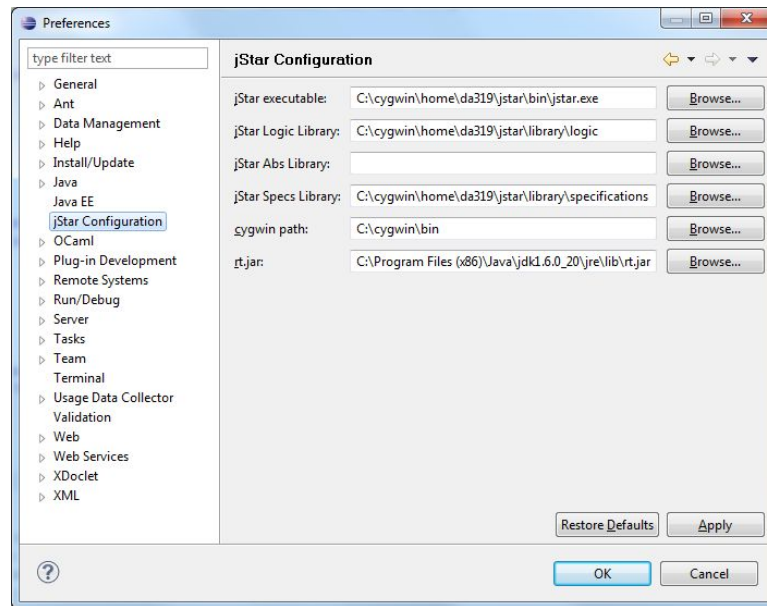
   rt.jar could be found in jdk1.6.0/jre/lib/. It is required library for soot. The plugin will try to find it automatically.

2.2 Linux

rt.jar could be found in jdk1.6.0/jre/lib/. It is required library for soot. The plugin will try to find it automatically.

2.3 Mac

classes.jar and ui.jar cound be found in /System/Library/Frameworks/JavaVM.framework/Versions/1.6.0/Classes/. They are required libraries for soot. The plugin will try to find them automatically.

3 Verification

You can verify source file by selecting `Verify with jStar` from the context menu or the main toolbar.

By selecting `Verify with jStar Configurations...`, you can indicate the location of specification, logic and abstraction rules.
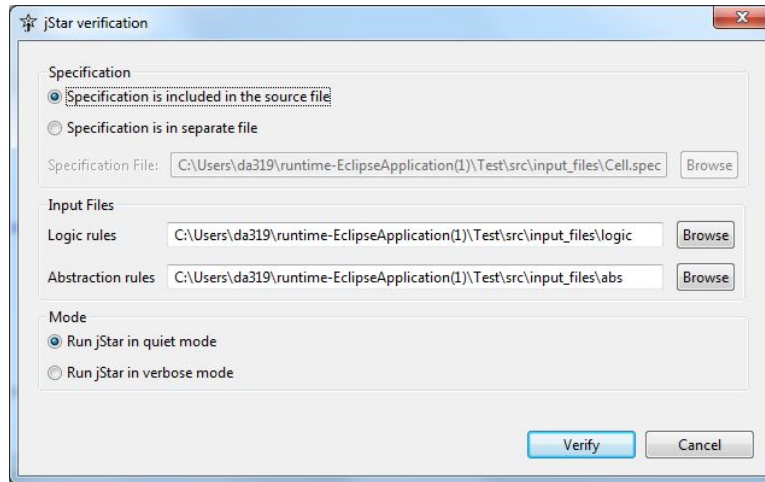

Specification in source file

If you want to write specification in the source file, you need to add annotations.jar (can be found in com.jstar.eclipse/jar files/annotations) to your project Java Build Path. More information about annotations could be found in 4 .

Verification errors

In case there are some verification errors, you can see error messages in console. The lines in source code where the problem appeared are annotated as squiggly marks.
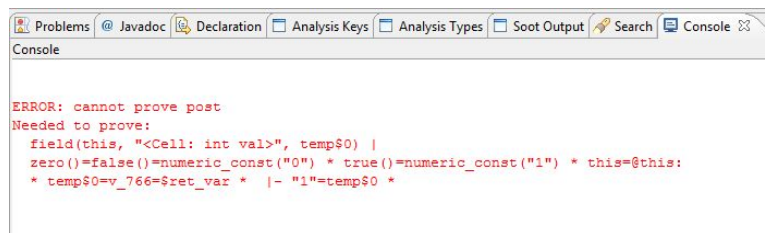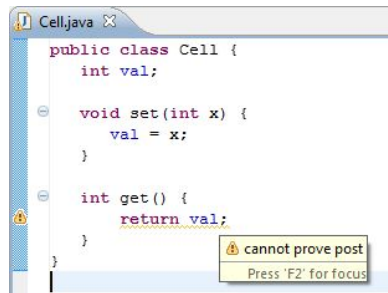
4 Annotation Processing

Annotations

- @Import has one element String[] value. This annotation can be used to annotate only type declarations.

- @Predicate is used for define and export statements. It has three elements: String predicate, String formula, DefinitionType type. DefinitionType is enum with two values Define and Export. The default value of type is DefinitionType.Define. This annotation can be used to annotate only type declarations.

- @Predicates is used if you want to have more than one define and/or export statements. It has one element Predicate[] value. This annotation can be used to annotate only type declarations.

- @InitSpec is a (dynamic / both static and dynamic) specification for constructor which is not explicitly defined in the source code. It has two elements: String pre, String post. This annotation can be used to annotate only type declarations.

- @InitSpecs is (dynamic / both static and dynamic) specifications which are conjuncted with also for constructor which is not explicitly defined in the source code. It has one element InitSpec[] value. This annotation can be used to annotate only type declarations.

- @InitSpecStatic is a static specification for constructor which is not explicitly defined in the source code. It has two elements: String pre, String post. This annotation can be used to annotate only type declarations.

- @InitSpecsStatic is static specifications which are conjuncted with also for constructor which is not explicitly defined in the source code. It has one element InitSpecStatic[] value. This annotation can be used to annotate only type declarations.

- `@Spec` is a (dynamic / both static and dynamic) specification for a method or a constructor. It has two elements: `String pre`, `String post`. This annotation can be used to annotate only method and constructor declarations.

- `@Specs` is (dynamic / both static and dynamic) specifications which are conjuncted with `also` for a method or a constructor. It has one element: `Spec[] value`. This annotation can be used to annotate only method and constructor declarations.

- `@SpecStatic` is a static specification for a method or a constructor. It has two elements: `String pre`, `String post`. This annotation can be used to annotate only method and constructor declarations.

- `@SpecsStatic` is static specifications which are conjuncted with `also` for a method or a constructor. It has one element: `SpecStatic[] value`. This annotation can be used to annotate only method and constructor declarations.

Examples of annotations in the source code:

Annotation in source file	Generated specification file
`@Import("Spec.spec")`	`import("Spec.spec");`
`@Import({"Spec1.spec", "Spec2.spec"})`	`import("Spec1.spec");` `import("Spec2.spec");`

`@Predicate(`
`   predicate = "P(x)",`
`   formula = "F(x)"`
`)`
	`define P(x) as F(x);`

`@Predicate(`
`   predicate = "P(x)",`
`   formula = "F(x)",`
`   type = DefinitionType.Export`
`)`
	`export P(x) as F(x);`

```
@Predicates({
    @Predicate(
        predicate = "P1(x)",
        formula = "F1(x)",
        type = DefinitionType.Export
    ),
    @Predicate(
        predicate = "P2(x)",
        formula = "F2(x)"
    )
})
```

```
export P1(x) as F1(x);
define P2(x) as F2(x);
```

```
@InitSpec(
    pre = "precondition",
    post = "postcondition"
)
```

```
void <init>() :
    { precondition }
    { postcondition }
```

```
@InitSpecs({
    @InitSpec(
        pre = "precondition 1",
        post = "postcondition 1"
    ),
    @InitSpec(
        pre = "precondition 2",
        post = "postcondition 2"
    )
})
```

```
void <init>() :
    { precondition 1 }
    { postcondition 1 }
    andalso
    { precondition 2 }
    { postcondition 2 }
```

```
@InitSpecStatic(
    pre = "precondition",
    post = "postcondition"
)
```

```
void <init>() static :
    { precondition }
    { postcondition }
```

```
@InitSpecsStatic({
   @InitSpecStatic(
      pre = "precondition 1",
      post = "postcondition 1"
   ),
   @InitSpecStatic(
      pre = "precondition 2",
      post = "postcondition 2"
   )
})
```

```
void <init>() static :
   { precondition 1 }
   { postcondition 1 }
   andalso
   { precondition 2 }
   { postcondition 2 }
```

```
@Spec(
   pre = "precondition",
   post = "postcondition"
)
```

method declaration

method declaration :

   `{ precondition }`
   `{ postcondition }`

```
@Specs({
   @Spec(
      pre = "precondition 1",
      post = "postcondition 1"
   ),
   @Spec(
      pre = "precondition 2",
      post = "postcondition 2"
   )
})
```

method declaration :

   `{ precondition 1 }`
   `{ postcondition 1 }`
   `andalso`
   `{ precondition 2 }`
   `{ postcondition 2 }`

method declaration

```
@SpecStatic(
   pre = "precondition",
   post = "postcondition"
)
```

method declaration `static` :

   `{ precondition }`
   `{ postcondition }`

method declaration

```
@SpecsStatic({
    @SpecStatic(
        pre = "precondition 1",
        post = "postcondition 1"
    ),
    @SpecStatic(
        pre = "precondition 2",
        post = "postcondition 2"
    )
})
```

method declaration

|

method declaration `static :`

```
{ precondition 1 }
{ postcondition 1 }
andalso
{ precondition 2 }
{ postcondition 2 }
```

Eclipse plugin

You can write specifications in your java source file. By selecting an option "Specification is included in the source file" in verification configuration, the specification file will be generated from your annotations. This specification file will be used to verify your program.

Command line

You can generate a specification file from your java source file with annotations using command line:

javac
    -proc:only
    -cp ".;jstar_processing.jar;commons-io-1.4.jar;annotations.jar"
    -d .
    -processor com.jstar.eclipse.processing.SpecAnnotationProcessor
    MyClass.java

-proc:only
    Only annotation processing is done, without any subsequent compilation.

-cp
    Specifies the classpath. As well as specifying the classpath to the classes MyClass is referencing to, you need to add the following jar files:

- jstar_processing.jar
  where the processor **com.jstar.eclipse.processing.SpecAnnotationProcessor** lives in.

- commons-io-1.4.jar
  required library for annotation processing.

- annotations.jar
  where the specification annotations live in.

-d
      The destination directory for the specification file.
-processor
      The name of annotation processor.
MyClass.java
       One or more source files with annotations.