

Eclipse plugin

Introduction

Since AspectWerkz 2.x an Eclipse plugin is provided. This one is largely a beta version, but in any way it will change your Eclipse project configuration, so it is safe to give it a try, come with feedback on the mailing lists and enrol for contribution.

Getting the AspectWerkz Eclipse plugin

Use the Eclipse manager to install from our update site.

Open Eclipse, use the **Help > Software Update > Find and Install** menu.

Select **Search for new features to install**.

Click **New remote site**

Name it **AspectWerkz**, and use the following URL: `*http://aspectwerkz.codehaus.org/downloads/eclipse*`

Follow the procedure to install it.

- ✓ The latest version is 2.0.4 for AW 2.0.RC1 and later, released on Déc 20, 2004.
- ✓ Tested on Eclipse 3.0 and 3.1M4

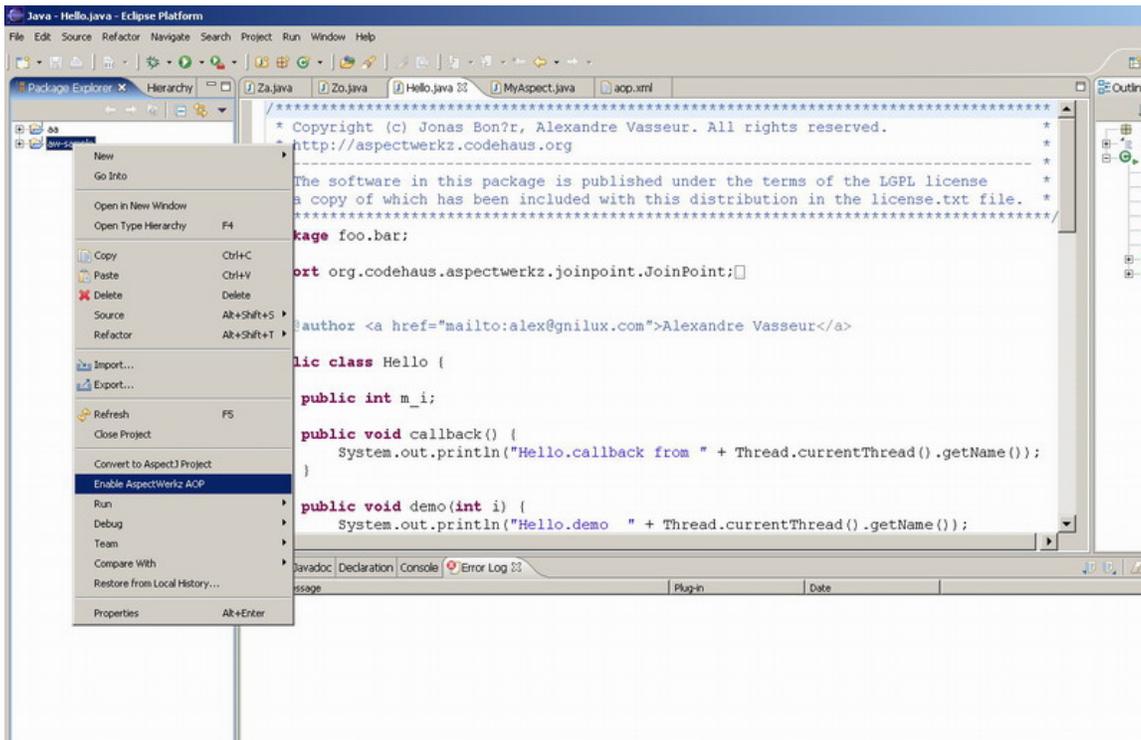
Enabling / Disabling AspectWerkz IDE support

In the *package explorer*, open the Java project for which you want to activate the AspectWerkz IDE support.

Right click on it. You should see a **Enable AspectWerkz AOP** menu. Select it.

AspectWerkz IDE support will be from now on activated for this project.

You can later on remove it by right-clicking again and using the **Disable AspectWerkz AOP** menu (if you do, tell us why !)



The plugin is generating traces in the **Error Log** view. You can display it thru the menu **Window > Show view >**

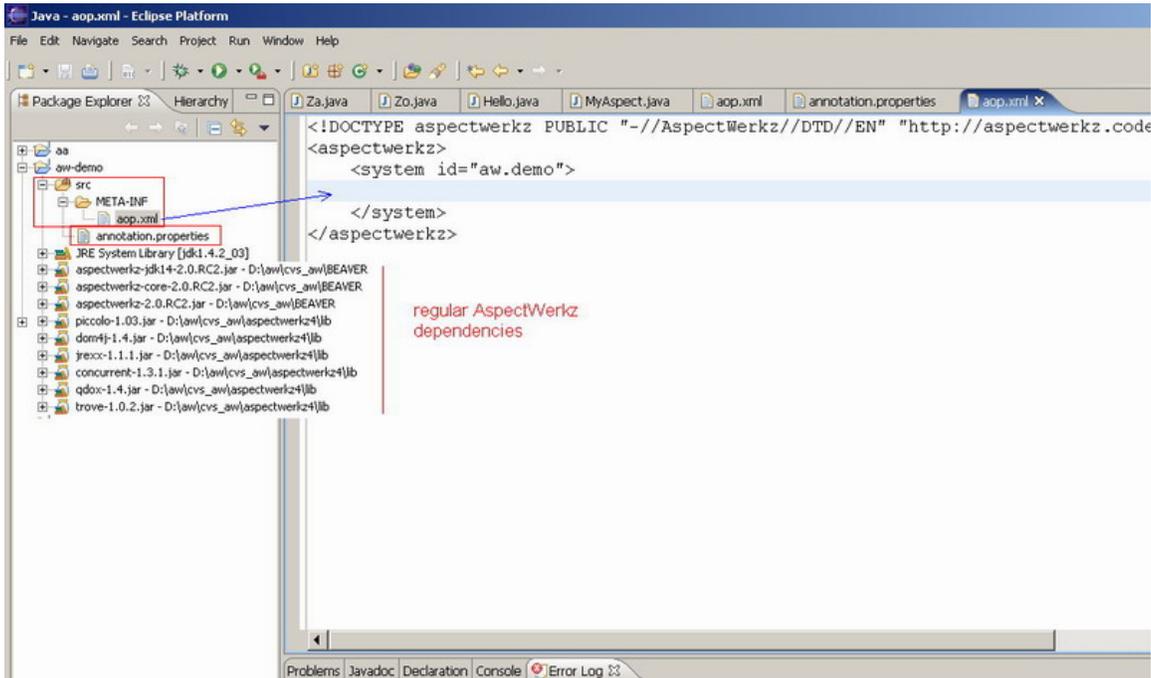
Error log.

Some more log might be in the .metadata/*.log file in your Eclipse Workspace root.

Project dependencies

Since we are about to use AspectWerkz, the project should contains the AspectWerkz 2.x (at least 2.0.RC2) in its classpath and the dependencies (concurrent, trove, qdox and alike - see the distribution), and the AspectWerkz Java 1.4 jar file to deal with Java 1.4 annotations.

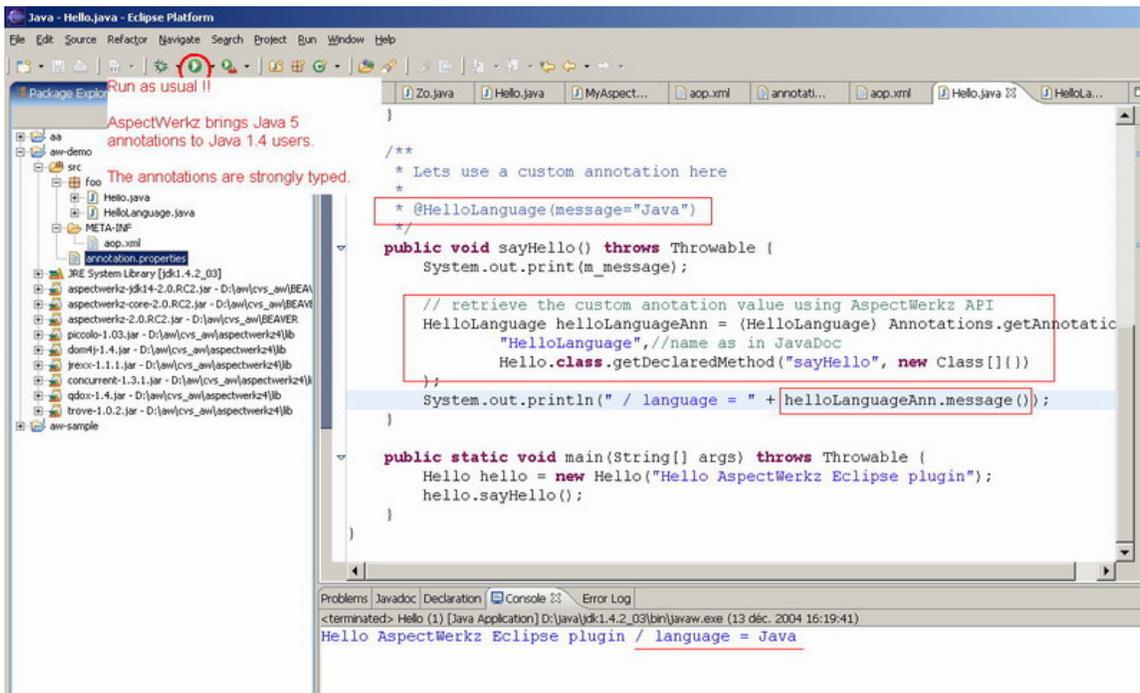
Activating AspectWerkz support will not interfere with your configuration so you have to do it manually just as usual.



Annotation support

Annotations

The plugin contains the AspectWerkz Java 1.4 annotation compiler. Annotations that you will write in Javadoc and that are known by the framework (like **@Before**, **@Expression**, **@Around**etc) will be compiled as the code gets compiled (clean build, or incremental build when you save your Java file).



Custom annotations

AspectWerkz provides custom annotation support. For this to work in the plugin, it is mandatory to declare the annotations interface implementations just as usual in a dedicated file (see the documentation - http://aspectwerkz.odehaus.org/annotations.html#Compiling_Annotations).

There are 2 requirements:

1. the file(s) must be in the **project classpath** (can be in the project itself or within a dependency jar file)
2. the file(s) must (all) be named **annotation.properties**

A common practice is to have the project annotation.properties file in the **<project>/src/annotation.properties** location.

AOP weaving

For weaving to occur, the system needs to know which class is an aspect. Just as usual, this is done thru the **META-INF/aop.xml** descriptor.

There are no requirements for this part. You just need to have one or more META-INF/aop.xml file in your **classpath** - be it the project itself or within a dependency jar file that may contain bundled aspects.

A common practice is to have the project aop.xml file in the **<project>/src/META-INF/aop.xml** location.

When you save a file, Eclipse will trigger a build that will trigger the weaving, based on the available META-INF/aop.xml file(s) and compiled annotations that define the aspects.

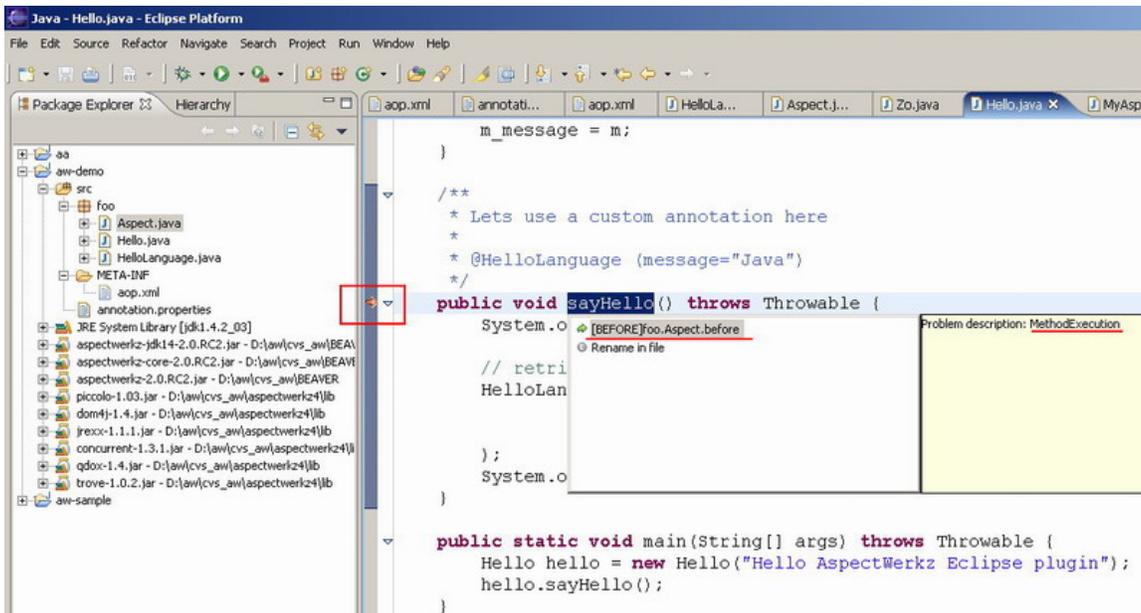
AOP cross cutting view support

When the weaving has occurred, markers will display where advices are/may be applied using a gutter marker.

The markers contain rich information that will inform you why there is a (or several) joinpoints on this specific line of source code.

Hit **CTRL + 1** to make the **Quick fix** tips appear. It will show you which advice is affecting the joinpoint, and the type of advice (before / around / after returning / after throwing / after finally). If the information starts with a question

mark "?" it means that the advice execution depends on a runtime check, like a "cflow" evaluation, or a "target is an instance of X" test.



When you select an advice and hit **RETURN** you will jump to the source code of that advice if available.

Running your application from within Eclipse

You can decide to run the application just as a regular Java application, since the classes have been compiled by Eclipse thus weaved by the plugin.

The META-INF/aop.xml files will be discovered at startup time, though you can point to a specific one using the "-Daspectwerkz.definition.file=.." JVM option as usual (see docs - <http://aspectwerkz.codehaus.org/deployment.html>)

If you do, the dependencies in third parties jar file will not be weaved since those are not part of your Java project source code.

To circumvent that, you should use AspectWerkz "online mode" ie load time weaving.

The plugin provides a trick to make this easier to deal with.

For it to work, you must add the Java **tools.jar** dependency to your project (it is in JAVA_HOME/lib/tools.jar)

In the package explorer, right click on your "main class" that you would like to run.

Select **Run > Run...**

Select the **AspectWerkz Application** configuration* style, right click and select **New**.

Run it from there as you would do for any regular application.

You should see a new line on system out: *AspectWerkz - INFO - Pre-processor org.codehaus.aspectwerkz.transform.AspectWerkzPreProcessor loaded and initialized* which confirms that the online mode is turned on.

Sample Eclipse project

To give it a try, you can use the sample project at <http://aspectwerkz.codehaus.org/downloads/eclipse/aspectwerkz-eclipse-demo.zip>

Known issues in this beta.

When you modify an aspect, the weaving of target classes is not triggered thus the markers are not updated. You need to wait the next time you change your target class, or you need to trigger a clean build manually.

When you trigger a clean build and the project contains both aspects and classes to be weaved, it may happen that the classes are not weaved. This happens when the aspect is defined with annotations, since the weaving of the target class may happen before the annotation compiler runs on the aspect. Trigger another clean build for now.