Dragon Developer Guide

This document explains how to configure the developer's environment for Dragon, by eBM WebSourcing.

Dragon Team
Olivier FABRE <olivier.fabre@ebmwebsourcing.com>

- May 2009 -

Table of Contents

List of Figures

Structure of the document

This document explains how to configure the developer's environment. It is designed for Dragon developers.

Chapter 1. Building Dragon from sources

1.1. Pre-Requisites

To be able to get the sources and build Dragon, you must install the following tools :

- **JDK 1.5.X**: Dragon is developed using java 5 features. You need to install a JDK 1.5.x to be abble to compile Dragon.
- **Maven 2.1.0+**: We use the Apache maven project to build Dragon. Maven can be downloaded here: http://maven.apache.org/download.html.
- **svn** : A subversion client is required to work with the project sources.

We suppose that all the required binaries (mvn, java, javac, svn) are defined in you path. All the following commands are Linux system targeted. Please adapt for a Windows system.

1.2. Checkout the sources

The Dragon sources are hosted by the OW2 forge. As developer you can checkout the sources from the repository with a svn client:

1. Create a base directory (for example `dragon`)
2. Checkout the Dragon sources in the `trunk` directory :
   - As anonymous user :

     svn checkout svn://svn.forge.objectweb.org/svnroot/dragon/trunk trunk
   - As developer (you must be registered in the OW2 forge and be activated as Dragon developer) :

     svn checkout svn+ssh://<developer-login>@svn.forge.objectweb.org/svnroot/dragon/trunk trunk

1.3. Compiling Dragon

Since the compilation uses lots of memory, you must define specific system variables for Maven2 :

MAVEN_OPTS=-Xmx512m

export MAVEN_OPTS

Maven2 also needs the JAVA_HOME system variable to be set :

JAVA_HOME=<your java path>

export JAVA_HOME

Note

You can define these variables in your `.bashrc` file.

In order to retrieve all the artifacts (Dragon dependencies), you must add specific configuration to Maven. To do so, add a settings.xml file in your local Maven repository (~/.m2/ under Linux, C:\Documents and Settings*username*\.m2 on Windows).

```
<settings>
 -<profiles>
 - -<profile>
```

```
 - - -<id>default-profile</id>
 - - -<activation>
 - - - -<activeByDefault>TRUE</activeByDefault>
 - - -</activation>
 - - -<repositories>
 - - - -<repository>
 - - - - -<id>ebmws-public.release</id>
 - - - - -<name>eBMWebsourcing -Public -Releases -Repository</name>
 - - - - -<url>http://maven.ebmwebsourcing.com/public</url>
 - - - - -<releases>
 - - - - - -<enabled>true</enabled>
 - - - - -</releases>
 - - - - -<snapshots>
 - - - - - -<enabled>false</enabled>
 - - - - -</snapshots>
 - - - -</repository>
 - - - -<repository>
 - - - - -<id>ebmws-public.snapshot</id>
 - - - - -<name>eBMWebsourcing -Public -Snapshot -Repository</name>
 - - - - -<url>http://maven.ebmwebsourcing.com/public-snapshot</url>
 - - - - -<releases>
 - - - - - -<enabled>false</enabled>
 - - - - -</releases>-
 - - - - -<snapshots>
 - - - - - -<enabled>true</enabled>
 - - - - -</snapshots>
 - - - -</repository>
 - - -</repositories>
 - -</profile>
 -</profiles>
</settings>
```

Now that all is ready, you can compile the sources:

1. Go into the sources directory

   cd dragon/trunk

2. Build the required dependencies first

   mvn -Denv=quality

3. Build Dragon

   mvn

   Note that you can skip all the unit tests like this (generation will be faster...)

   mvn -Dmaven.test.skip

1.4. Dragon generated binaries

Once the sources are succesfuly compiled, two binary distributon of Dragon are available:

1. A web application that exposes only Dragon Web Service API, available at:

   cd dragon/trunk/web-service/target/dragon-ws.war

2. A web application that exposes Dragon Web Service API and Web user interface, available at:

   cd dragon/trunk/ui/target/dragon-XXX.war (where XXX is the Dragon version)

Chapter 2. Dragon Coding Rules

All Dragon source code must respect some coding rules :

- formatting rules,
- naming conventions,
- SCM rules,
- coding rules.

These rules will be verified automatically by tools like: Checkstyle, PMD and Findbugs.

2.1. Formatting rules

Dragon formating rules are designed to format all the code according to the Java Sun code style recommandation with some little changes to match the Dragon configuration of Checkstyle, PMD and Findbugs. These rules are the following :

- class members sorting rule :
  - Inside a class, member must be sorted by **category** in the following order :
    1. types,
    2. static fields,
    3. fields,
    4. constructors
    5. static initializers,
    6. static methods
    7. initializers,
    8. methods.
  - Inside a category, members must be sorted by **visibility** in the following order :
    1. public,
    2. protected,
    3. default,
    4. private.
- import statements sorting rule: they are ordered as :
  1. java
  2. javax
  3. org
  4. org.ow2.dragon
  5. com

2.2. Naming conventions

2.2.1. Dragon maven modules

2.2.1.1. Folder Name

- Dragon modules are placed under trunk sub folder.
- The folder name don't need to be prefixed with "dragon". For exemple we don't use "dragon-uddi-ws" for the uddi web service module but "uddi-ws" instead.

2.2.1.2. Maven artifact name

- Maven artifact names need to use the "dragon" prefix. For exemple the uddi web service artifactId is "dragon-uddi-ws"
- The groupId need to be "org.ow2.dragon" or a sub group like "org.ow2.dragon.extensions"

2.3. SCM Rules

2.3.1. Rules on "commit"

To compare easily two consecutive revisions of the same file, some rules must be respected:

- Don't mix two changes in the same commit,
- The source code formating is allowed only on the updated source code,
- A specific commit MUST be use to format only source code.

2.4. Java Coding Rules

Coding rules are classified in several categories. Some rules can be put in several categories. In this case, they will be placed in the most importaant categories. The coding rules categories are, sorted in importance order:

1. Stability: The goal of these rules is to avoid such problems as: commons bugs, threads synchronization problems, concurrent acces problems, ...
2. Performance: The goal of these rules is to increase or optimize the performances,
3. Documentation: The goal of these rules is to have a source code correctly commented and documented

2.4.1. Rules about stability

2.4.1.1. About the pattern "Singleton"

To initialize a singleton, instead of using an implementation similar to the following one:

```
public -class -MySingleton -{
 - - -private -static -final -MySingleton -mySingleton -= -null;

 - - -private -MySingleton() -{
 - - -}

 - - -public -static -synchronized -MySingleton -getInstance() -{
 - - - - - -if -(myString -== -null) -{
 - - - - - - - - -mySingleton -= -new -MySingleton();
 - - - - - -}
```

```
 - - - - - -return -mySingleton;
 - - -}
}
```

prefer to use a **static initialization bloc**:

```
public -class -MySingleton -{
 - - -private -static -final -MySingleton -mySingleton -= -null;
 - - -static -{
 - - - - - - -mySingleton -= -new -MySingleton();
 - - -}

 - - -private -MySingleton() -{
 - - -}

 - - -public -static -MySingleton -getInstance() -{
 - - - - - -return -mySingleton;
 - - -}
}
```

2.4.1.2. Comparaison pattern

For performance and reliability reasons, use the following pattern to compare an object (ref) to a constant object (const):

```
 - - -if -(const.equals(ref)) -{
 - - - - - -...
 - - -}
```

instead of using the following one:

```
 - - -if -((ref!=null) -&& -(ref.equals(const))) -{
 - - - - - -...
 - - -}
```

In the last pattern, if the object is not compared to "null", a NullPointerException can occur.

2.4.2. Rules about performance

2.4.2.1. Usage of the method "equals"

As the returned value by the method "equals" is a boolean, it is unneeded to compare the result to "true" or "false". Use the following pattern:

```
 - - -if -(arg0.equals(arg1)) -{
 - - - - - -...
 - - -}
```

instead of using the following one that is forbidden:

```
 - - -if -(arg0.equals(arg1) -== -true) -{
 - - - - - -...
 - - -}
```

2.4.2.2. Usage of the StringBuilder vs StringBuffer

When a string buffer is used by a **single thread**, it is recommended that the classe StringBuilder be used in preference to StringBuffer as it will be faster under most implementations.

> **Warning**
>
> The class StringBuilder is only available since Java 1.5

Chapter 3. Developing under Eclipse

This chapter talks about how to configure and use the Eclipse IDE to develop Dragon.

3.1. Tools and plugins

You can download the eclipse IDE on the eclipse website (http://www.eclipse.org). Additionaly you have to install these plugins (Please refer to the plugins web sites for installation):

- Subclipse : Used for source code management under SVN
- Web Tools Platform : Used for launching Servlet container (like Tomcat) from Eclipse IDE. Useful to debug Dragon from Eclipse.

3.2. Preparing Eclipse

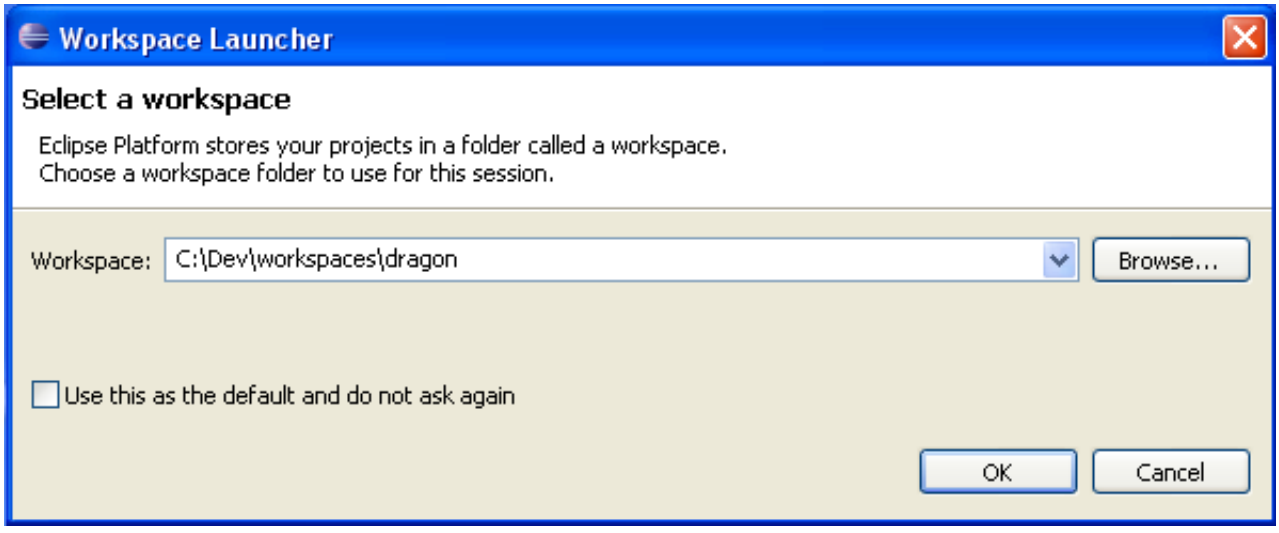
3.2.1. Multiple workspaces

The first thing you must know is getting work with multiple workspaces in Eclipse.

For developers a workspace is a collection of related projects. You can have more than one workspace for different sets of projects. In almost cases the best way to manage these multiple workspaces is configuring Eclipse to prompt which workspace to use every startup (otherwise, you should create various copies of Eclipse start icon/link on your desktop using the *-data <workspace_dir>* as an executable argument to specify your choice).

To configure this option in Eclipse you must access **Window -> Preferences...**, then select **General -> Startup and Shutdown** item in the navigable tree and check the box for Prompt for workspace on startup - if not set. You will be prompted to select a workspace in the next time you start Eclipse.

Figure 3.1. Eclipse prompt at startup



If the specified path does not exist Eclipse will try create the directory. You can change the workspace while Eclipse is already running (File - Switch Workspace... submenu).

Be aware that a workspace is more than a collection of projects. Eclipse internally uses the workspace directory to store your preferences for that set of projects - and other runtime files and views. In each workspace you can have specific configurations, for example: code formatting, code comments templates, JRE used, so on... Separating the workspace

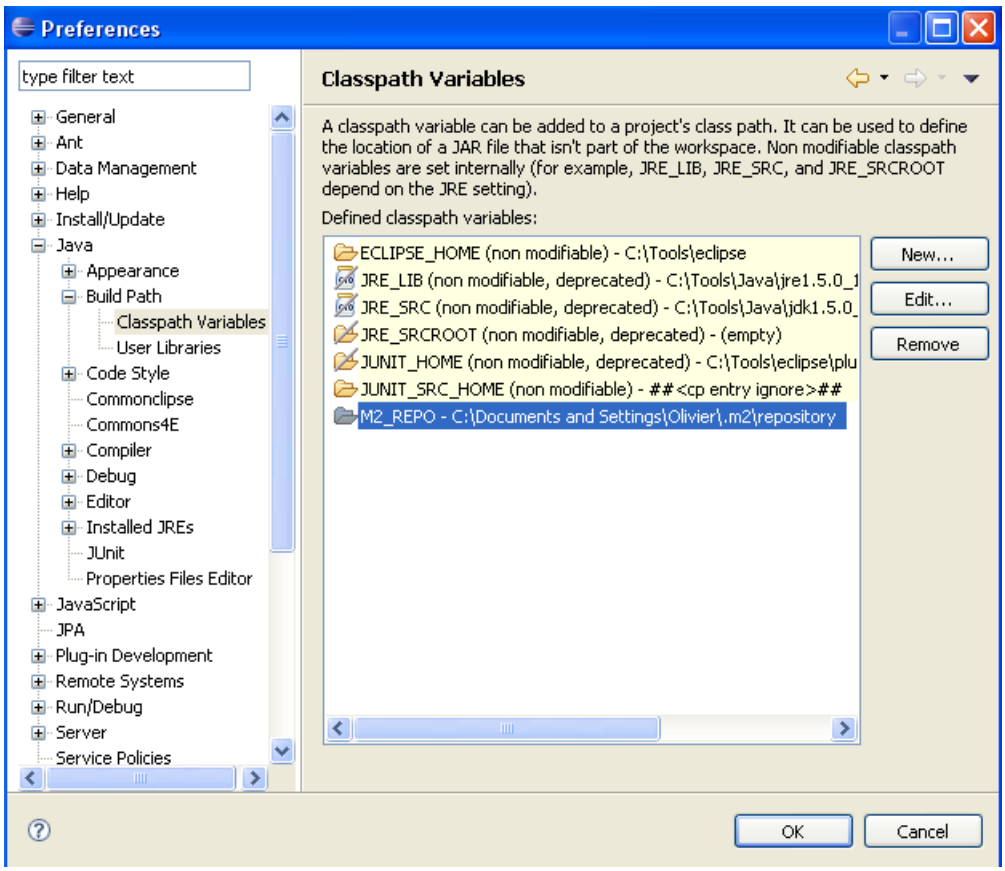
directory tree from project directory tree is a good approach for better management. That way you are able to share project among various workspaces. At least you can avoid unexpected projects removal when installing a new version of Eclipse.

3.2.2. Maven classpath variables

When a Maven project is imported into Eclipse it expects a **M2_REPO** classpath variable entry.

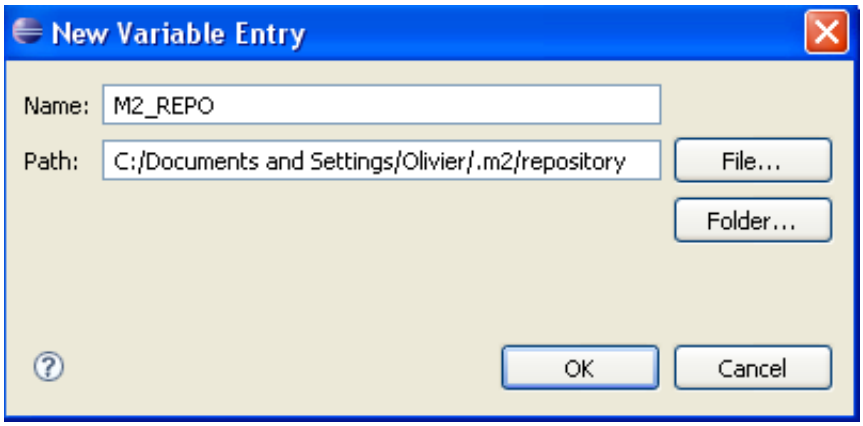
This variable is used to reference project libraries within your local Maven repository. To setup this variable go to **Window -> Preferences...** and select **Java -> Build Path -> Classpath Variables** on the navigable tree.

Figure 3.2. Eclipse classpath variables



Add a new **M2_REPO** entry pointing to the local Maven 2 repository (Under Linux, it is generally : M2_REPO=~/.m2/repository).

Figure 3.3. Define M2_REPO variable

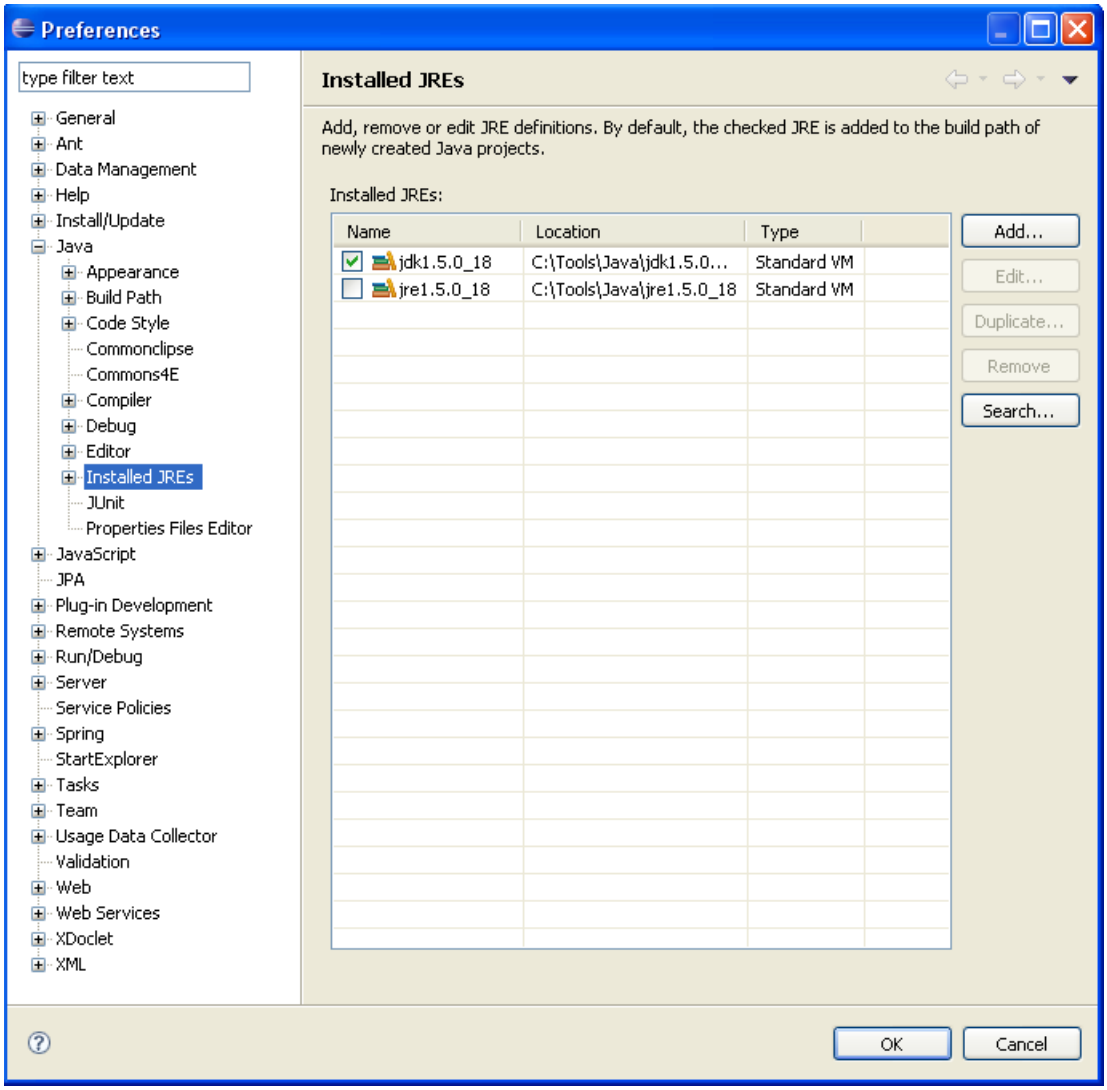


3.2.3. Selecting the JDK and its options

Eclipse allow a number of installed JRE/JDK configured in your environment. In this case, you can easily use other java runtime by accessing **Window -> Preferences** and then selecting the **Java -> Installed JREs** on the navigable tree.

For Dragon, choose a "Java(TM) 1.5" compatible JDK.

Figure 3.4. Choosing Java Development Kit



3.2.4. Multiple workspaces building

The Maven Eclipse plugin is used to create eclipse projects from the Dragon maven modules:

mvn eclipse:eclipse

This command will generate Eclipse project required files (`.project`,`.classpath`, `.checkstyle`...) wich will be used for project imports. It will also dowload all the required projects sources from Maven repositories.

Note

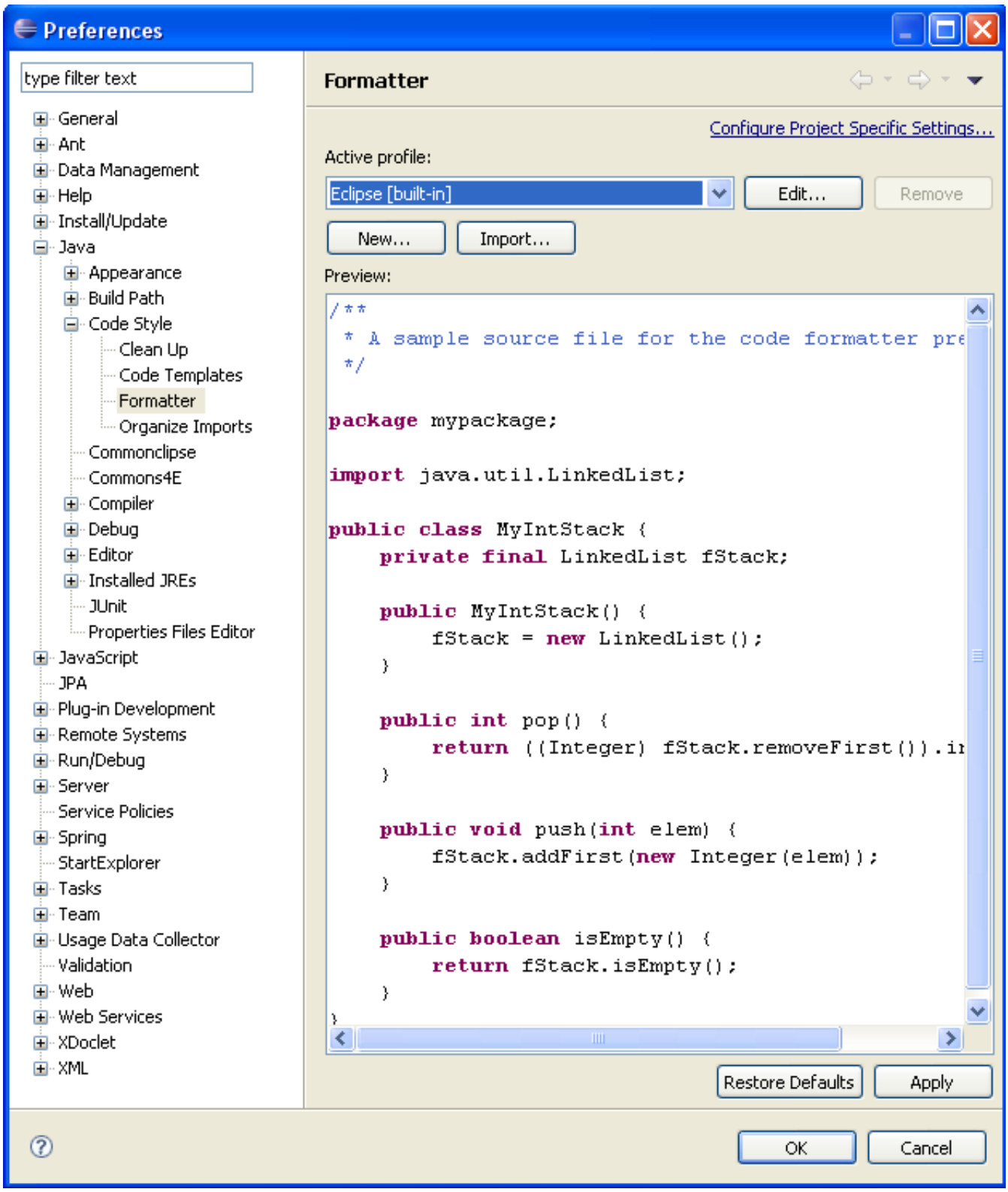
Generating all projects from the root path can takes a lot of time. Of course, you can only build the projects you need.

3.2.5. Use the good source code formatter

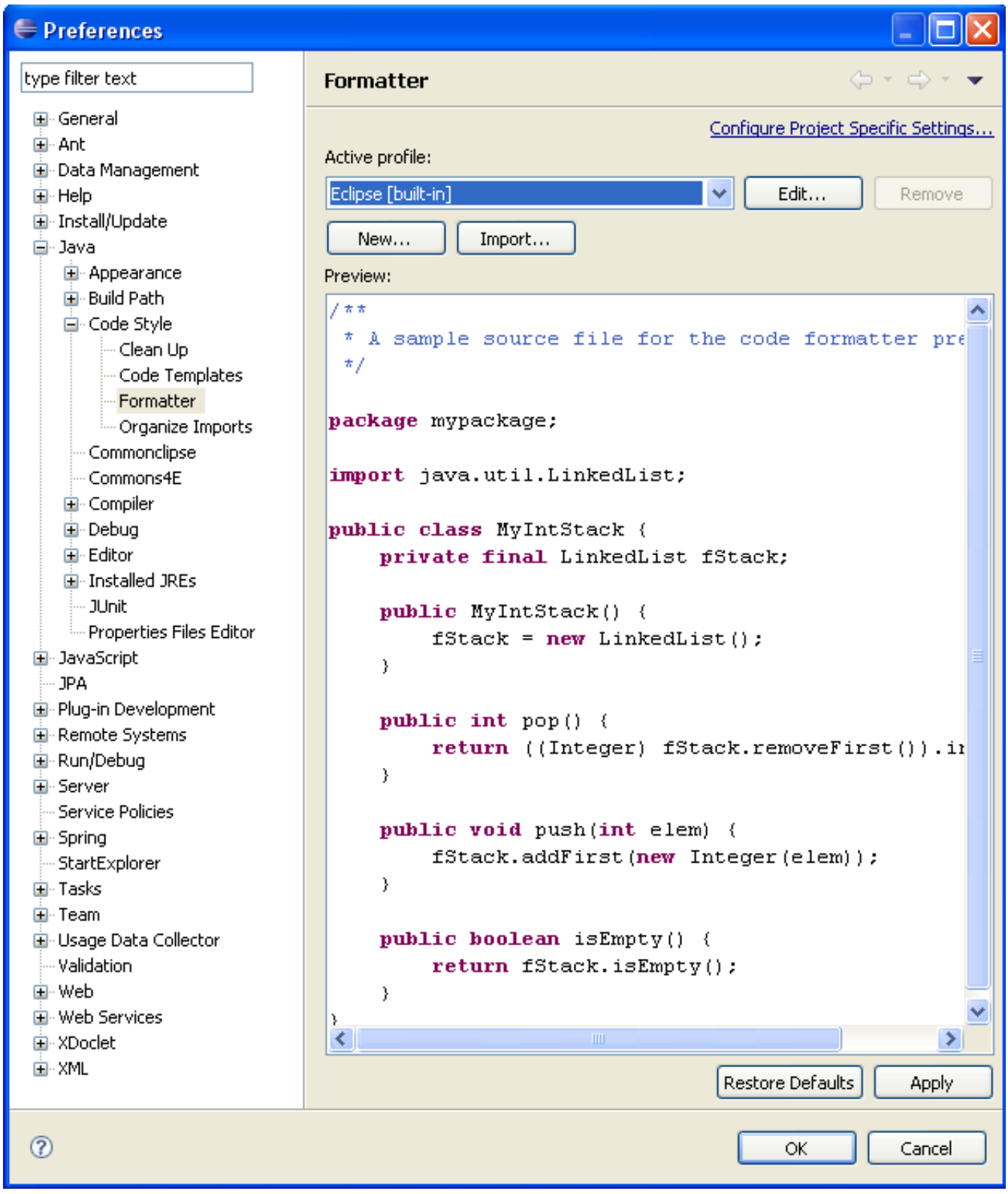
The Dragon formatter is now configured by the maven-eclipse-plugin. To check if the good formatter is activated, look at the properties of your Eclipse project. In the 'Java Code Style -> Formatter' section you must see the good Dragon formatter.

If the good formatter is not set, you can define it by following these steps :

- go to **Window -> Preferences** , then select **Java -> Code Style -> Formatter** :



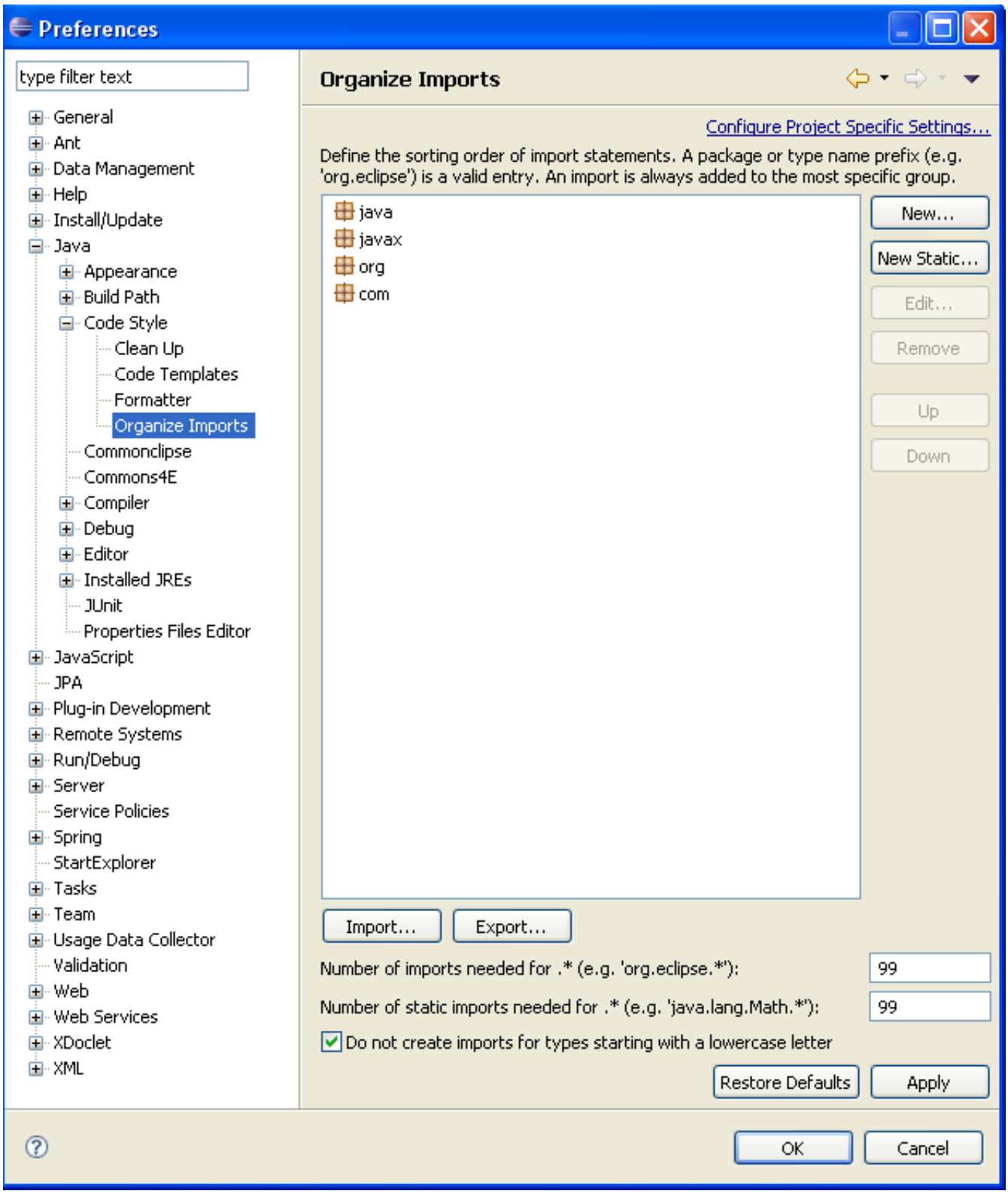
- click on "**Import**", and select the file "`dragon-formatter.xml`" from the directory `<DRAGON_SRC_HOME>/quality/src/main/resources/eclipse/global-configuration` of the Dragon project :



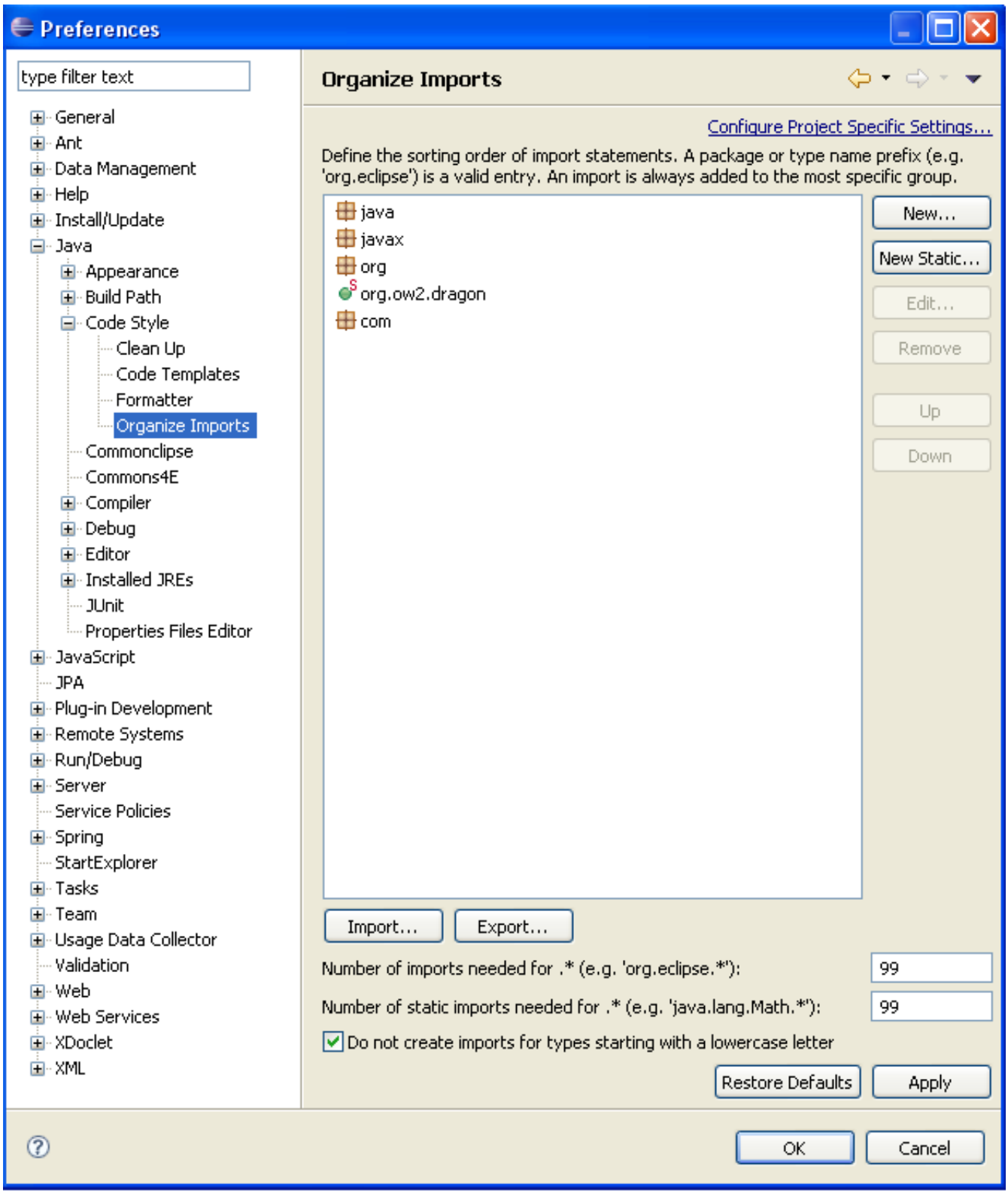
- click on "**Apply**" to save.

The import statements ordering rules are set from a file :

- go to **Window -> Preferences** , then select **Java -> Code Style -> Organize Imports** :



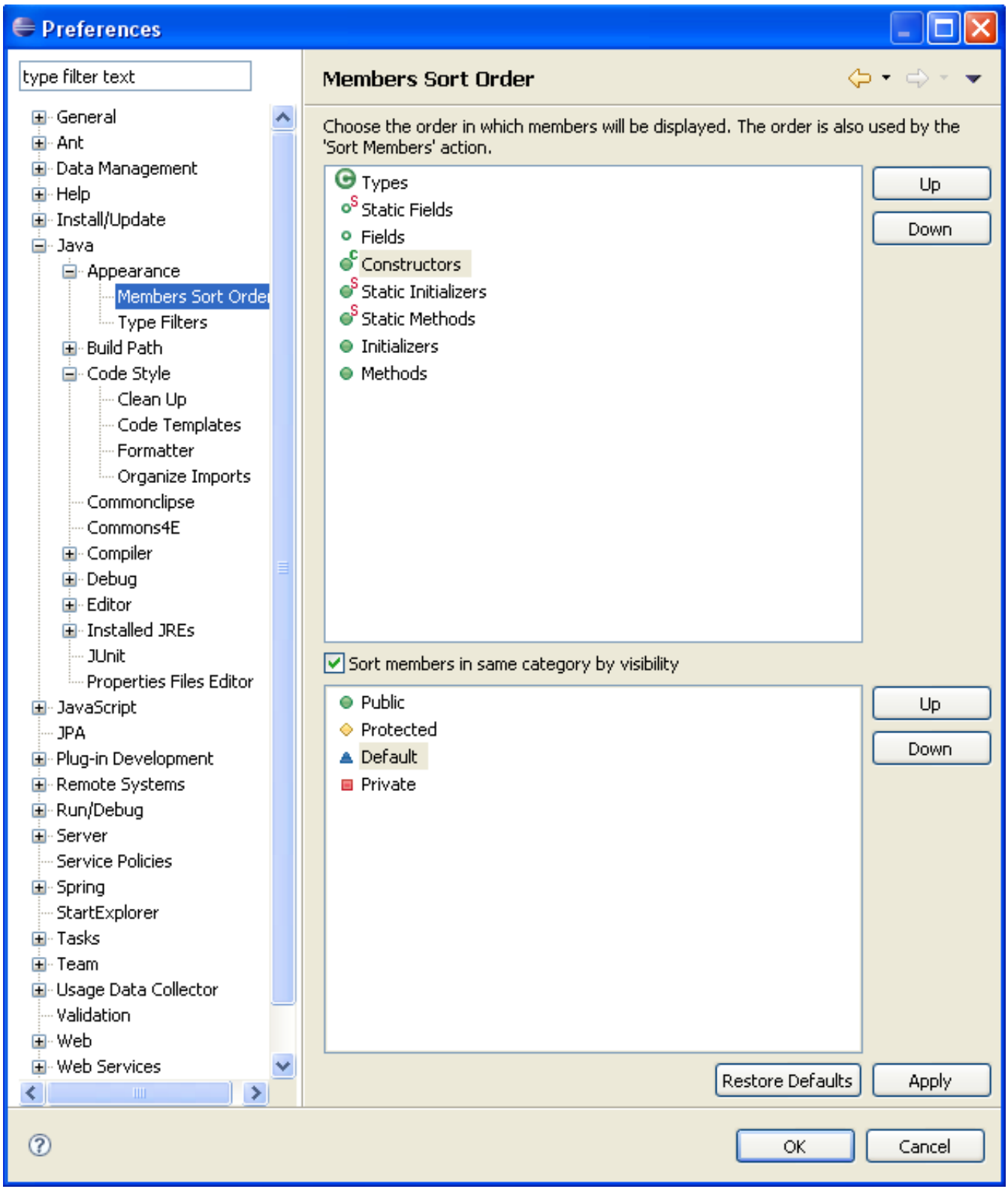
- click on "**Import**", and select the file "`eclipse.importorder`" in the directory `<DRAGON_SRC_HOME>/quality/src/main/resources`/eclipse/global-configuration of the Dragon project :



- click on "**Apply**" to save.

The class members ordering rules must be set manualy :

- go to **Window -> Preferences** , then select **Java -> Appearance -> Members Sort Order** :



- use buttons "**Up**" and "**Down**" to have the same ordering as the previous screenshot.
- click on "**OK**" to save.

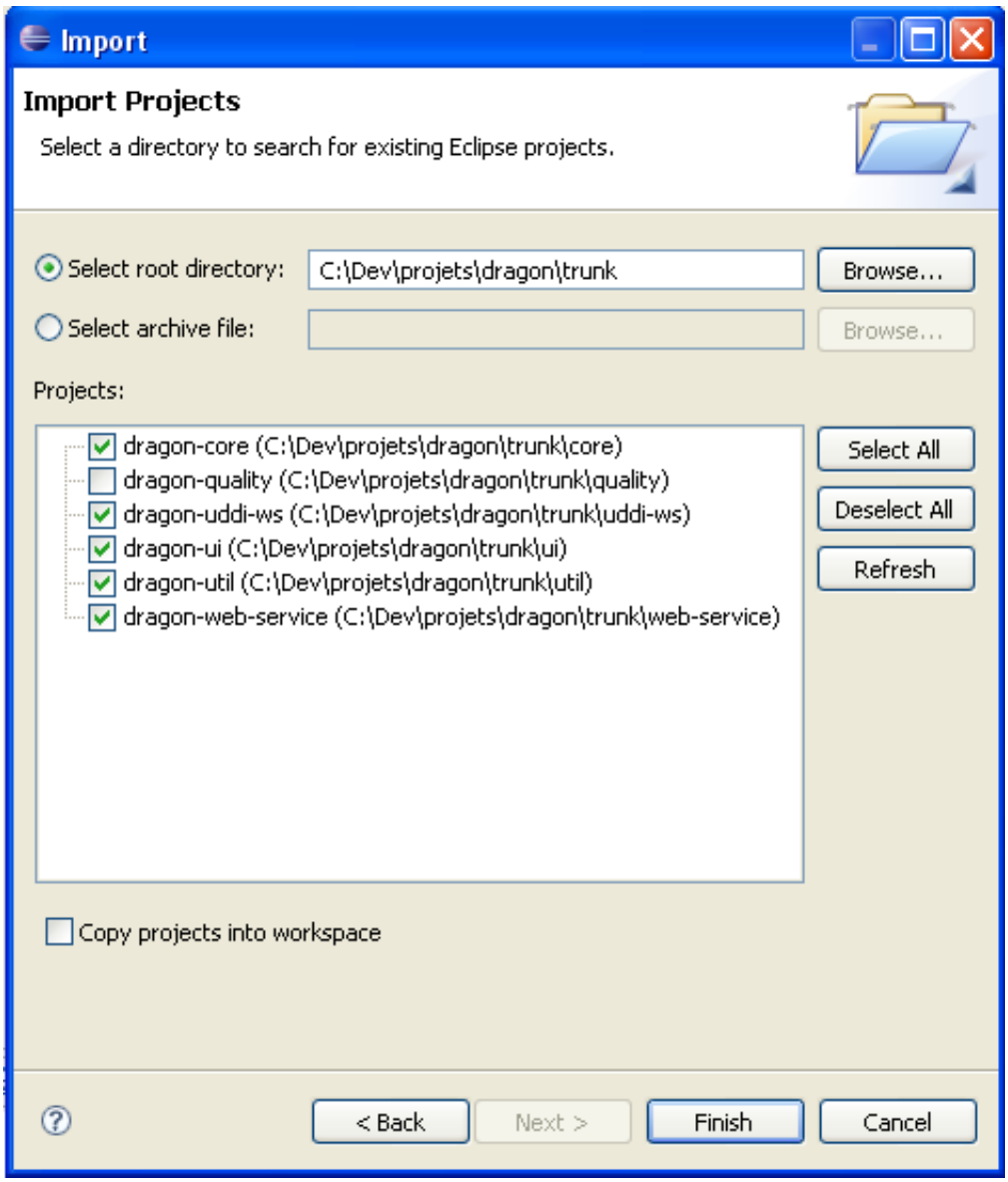
Now you can format all the code you write by doing a right click on the java file and selecting **Source -> Format**.

3.3. Working on Dragon projects

3.3.1. Importing project

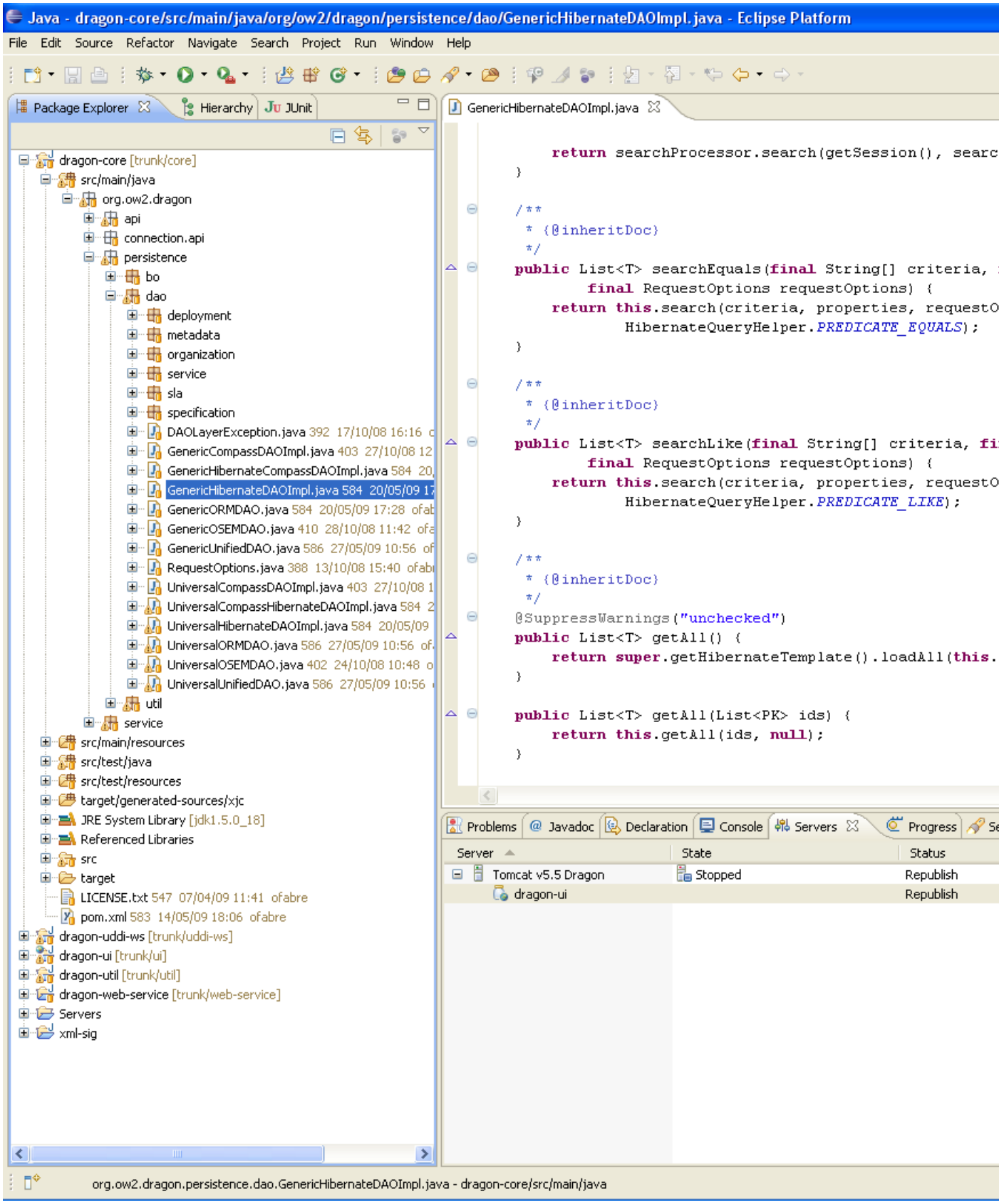
To import a generated Eclipse project in Eclipse, go to **File -> Import -> Existing Projects into Workspace** command. You can choose to import all Dragon projects by selecting the root folder of the Dragon project or just some of them by selecting a sub folder.

Figure 3.5. Import projects into Eclipse



Once required projects selected, you should see them in your workspace :

Figure 3.6. Dragon projects in Eclipse



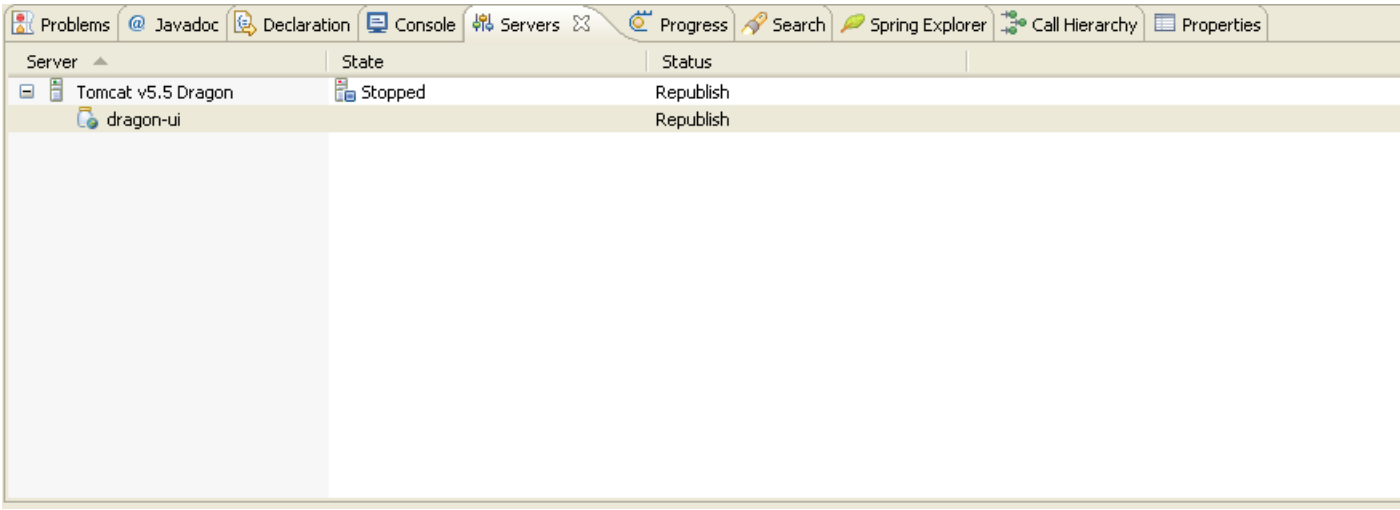
3.3.2. Create a Dragon launcher with WTP

The next configuration you need to create is a WTP server configuration to run Dragon from Eclipse environment.

First, you have to install a Tomcat runtime. See Apache Tomcat web site for installing instructions. Dragon need a Tomcat 5.5.x version.

Then you could add a Tomcat server setting to Eclipse IDE. Go to **Windows -> Show View... -> Other...** and then select **Server -> Servers**. The following view is added to your workspace:

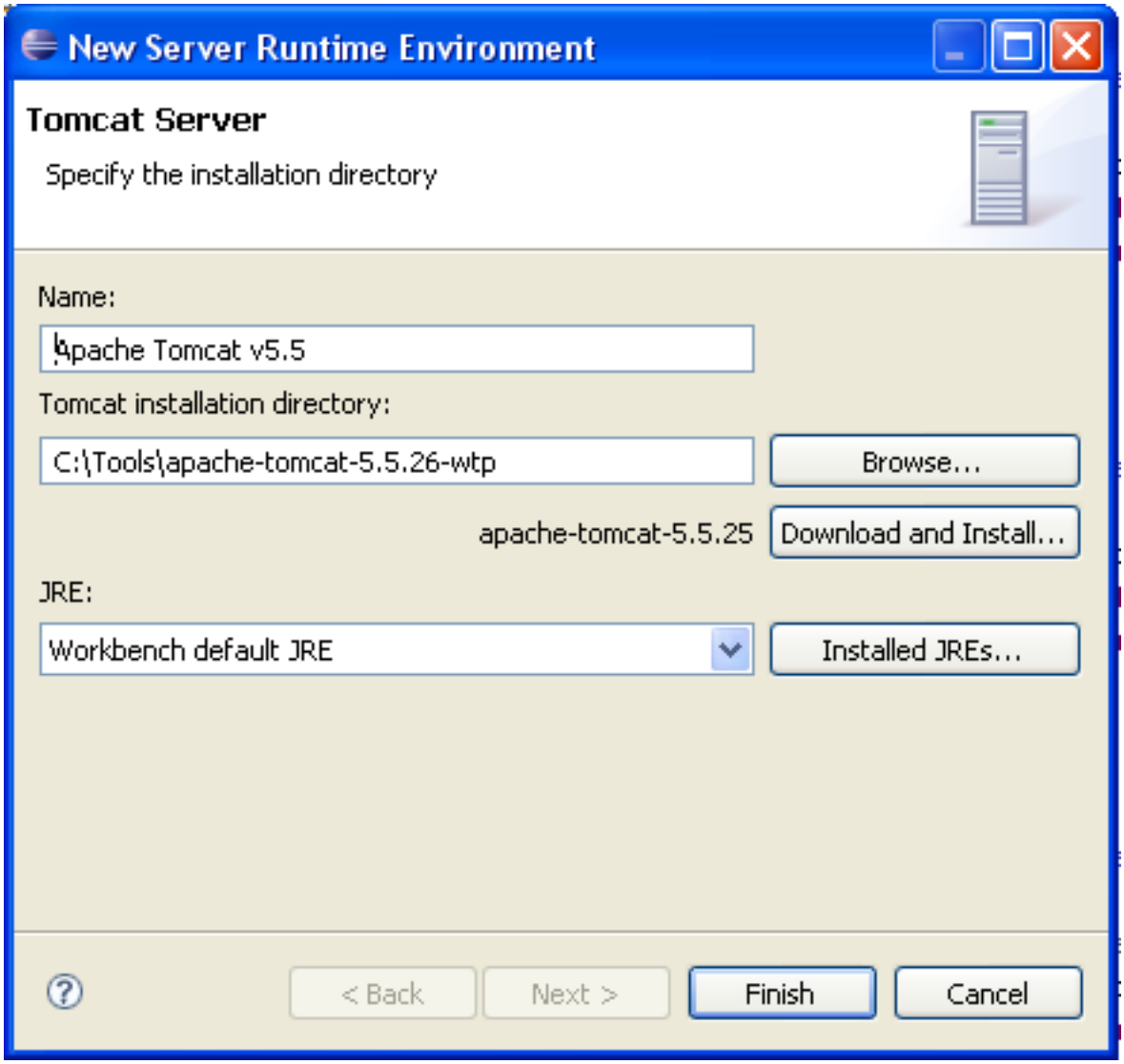
Figure 3.7. Servers view



To add a Tomcat for Dragon server config like the one in the previous snapshot, follow these steps:

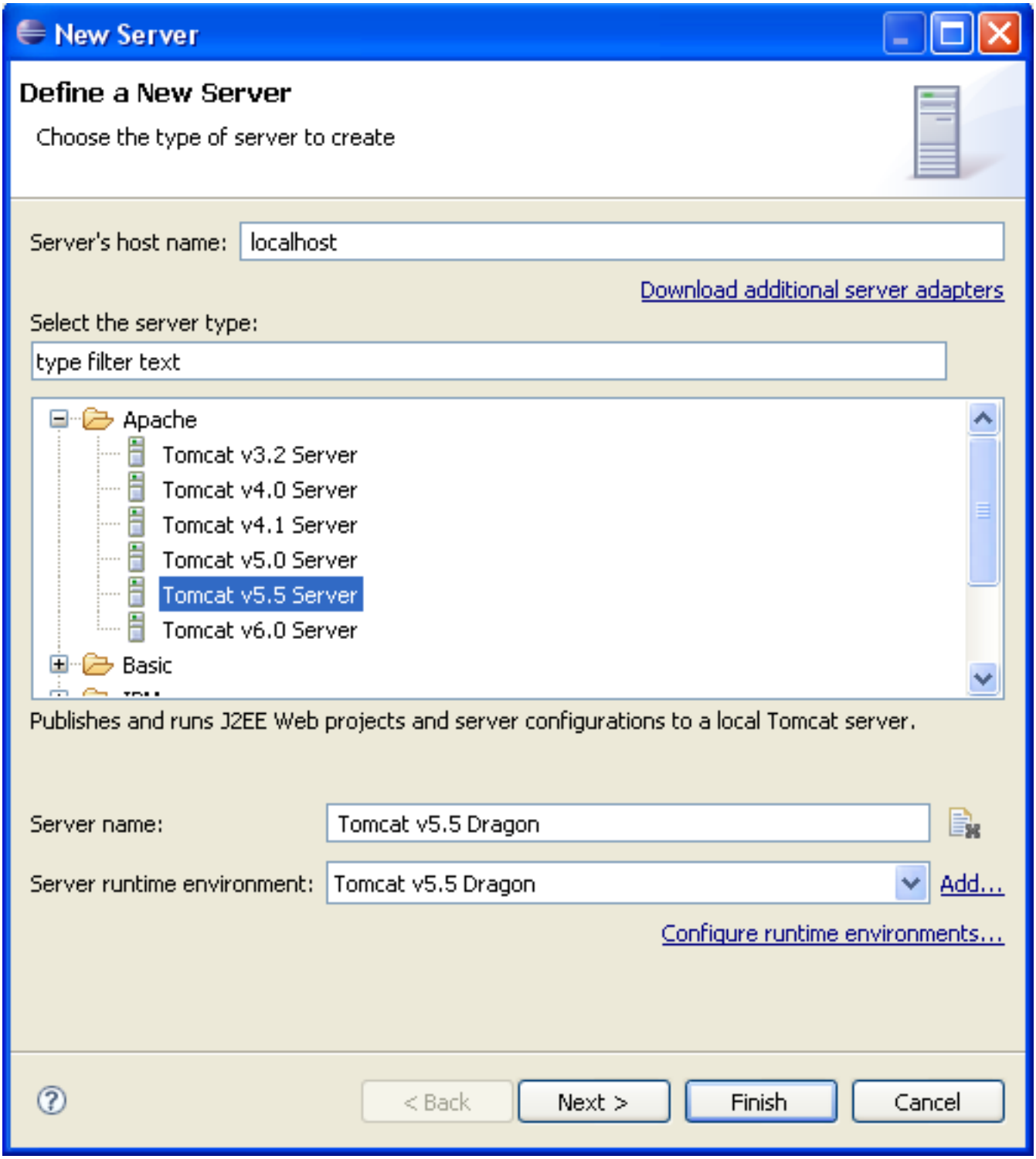
- Right click on the view and go to **New -> Server**
- Create a new runtime environement by clicking on the **Add..** link next to the "Server runtime environment" labelled field. Fill the displayed form like in the following snapshot. The "Tomcat installation directory" field need to point to your Tomcat installation root folder. Then click "**Finish**".

Figure 3.8. New Runtime environment



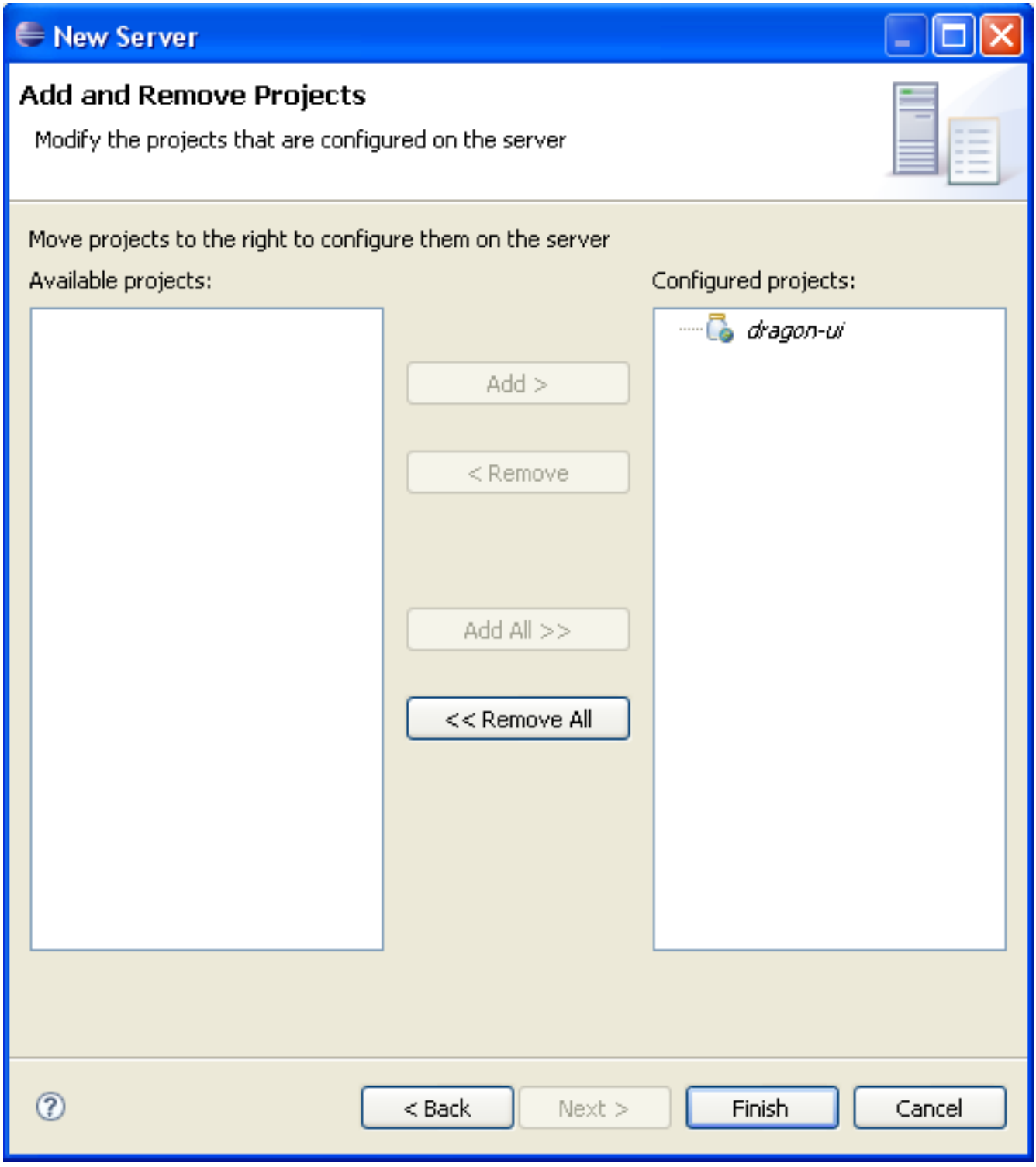
- Then fill the "New Server" form like in the following snapshot, and click "**Next**":

Figure 3.9. New server form



- Add dragon-ui has a configured project like this and click "**Finish**":

Figure 3.10. Dragon configured project



- Your Tomcat for Dragon server config is created. Due to a bug on the maven-eclipse-plugin wtp part for multimodule project, you need to run "mvn eclipse:eclipse" into the "<DRAGON_SRC_HOME>/dragon-ui" folder, to allow this config to work.

Caution

Don't forget to do the last step each time you run a "mvn eclipse:eclipse" into the Dragon top source directory (<DRAGON_SRC_HOME>), otherwise you could accounter some trouble with the WTP Eclipse plugin.

Chapter 4. Releasing Dragon

4.1. Releasing a module

We use the Maven release plugin to release all the Dragon modules.

4.1.1. Preparing the release

mvn release:prepare -Prelease -Dusername=<developer-login>

The previous command will execute:

1. Check that there are no uncommitted changes in the sources
2. Check that there are no SNAPSHOT dependencies
3. Change the version in the poms from x-SNAPSHOT to a new version (you will be prompted for the versions to use)
4. Transform the SCM information in the POM to include the final destination of the tag
5. Run the project tests against the modified POMs to confirm everything is in working order
6. Commit the modified POMs
7. Tag the code in the SCM with a version name (this will be prompted for)
8. Bump the version in the POMs to a new value y-SNAPSHOT (these values will also be prompted for)
9. Commit the modified POMs

You must provide :

- The -Prelease option to activate the release profile in order to generate optimized classes
- Your OW2 developer name that will be used for scm connection and in generated POM files

4.1.2. Performing the release

mvn release:perform -Prelease

This command will execute :

1. Checkout from an SCM URL with optional tag
2. Run the predefined Maven goals to release the project (by default, deploy site-deploy)

This command needs to have the following configuration in your M2_HOME/conf/settings.xml, with the right values :

```
<servers>
 - - -<server>
 - - - - - -<id>ow2</id>
 - - - - - -<username>xxxxxx</username>
 - - - - - -<password>yyyyyy</password>
 - - -</server>
</servers>
```

You can get additional information on the release plugin page: http://maven.apache.org/plugins/maven-release-plugin/.

4.2. Maintenance Release

When a module is released, a new tag is created under the tags SCM directory. For example, when the dragon-core v 1.0 module is released, the tags/dragon-core-1.0 directory is created under the SCM.

This tag **MUST** be used to create a branch for maintenance. The maintenance branch should have the same name as the tag one.

Once the branch is created, you have to modify the pom.xml file :

- Set the version as maintenance SNAPSHOT. For example, a 1.0 version becomes 1.0.1-SNAPSHOT
- Modify the SCM attributes. The maven release plugin has created SCM elements for the previously created tag. You have to change the URL to be the branch one.

You can now checkout the branch, and work on it as usual. The release process is the same as the trunk modules one.

4.3. Specific settings to release under Windows environment

To be able to release Dragon modules from a windows environment, you need to follows these steps before starting the first release process:

- Install TortoiseSVN
- Add an enviroment variable: "**SVN_SSH = <TORTOISE_SVN>\\bin\\TortoisePlink.exe**" where <TORTOISE_SVN> is the root folder of your TortoiseSvn install directory. For exemple if TortoiseSvn is intalled in the "**c:\Tools\TortoiseSvn**" folder your env variable is "**SVN_SSH = c:\\Tools\\TortoiseSvn\\bin\\TortoisePlink.exe**".

Then you could start releasing by following the previously described steps.

Chapter 5. Best practices

5.1. Managing bugs and feature requests

5.1.1. Bugs

- When a bug is created; the category, summary and description fields must contain relevant information to unterderstand the bug. **The GROUP field has to match the component version where the bug is found**.
- The Dragon leader set the priority and affect a developer to the task. **The correction starts here**.
- When the bug is fixed, the developer set the *RESOLUTION* field to *FIXED*. Moreover, he set the *STATE* field to *PENDING*.
- When a release occurs for this component, the Dragon leader set the *STATE* field to *CLOSED* and the version of the component is set as a comment.

5.1.2. Features

- When a feature is created; the category, summary and description fields must contain relevant information to unterderstand the feature. **The GROUP field remains empty**.
- The Dragon leader evaluates the request and, if accepted, sets a priority.
- The Dragon leader affects a developer to the task. **The implementation starts here**.
- When the feature is implemented, the developer sets the *STATE* field to *PENDING*.
- When a release occurs for this component, the Dragon leader sets the *STATE* field to *CLOSED* and **the GROUP field is set with the version where the feature can be found**.