JDO Tools Guide (v5.1)

Table of Contents

The DataNucleus project provides a few tools to assist in your use of DataNucleus and the JDO API.

Maven Plugin

Apache Maven is a project management and build tool that is quite common in organisations. Using DataNucleus and JDO with Maven is simple since the DataNucleus jars, JDO API jar and Maven plugin are present in the Maven central repository, so you don't need to define any repository to find the artifacts.

`pom.xml` Integration

The first thing to do is identify which artifacts are required for your project, and updating your `pom.xml` accordingly.

Firstly, you will need the following for *compile time* building against the JDO API.

```
<project>
    ...
    <dependencies>
        <dependency>
            <groupId>org.datanucleus</groupId>
            <artifactId>javax.jdo</artifactId>
            <version>3.2.0-m6</version>
        </dependency>
    </dependencies>
    ...
</project>
```

If using any DataNucleus API extensions in your code then you will also need `datanucleus-core` at *compile time*.

At runtime you will need the DataNucleus artifacts present also, so this becomes

```
<project>
    ...
    <dependencies>
        ...
        <dependency>
            <groupId>org.datanucleus</groupId>
            <artifactId>javax.jdo</artifactId>
            <version>3.2.0-m6</version>
        </dependency>
        <dependency>
            <groupId>org.datanucleus</groupId>
            <artifactId>datanucleus-core</artifactId>
            <version>[5.0.0-release, )</version>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.datanucleus</groupId>
            <artifactId>datanucleus-api-jdo</artifactId>
            <version>[5.0.0-release, )</version>
        </dependency>
        <dependency>
            <groupId>org.datanucleus</groupId>
            <artifactId>datanucleus-rdbms</artifactId>
            <version>[5.0.0-release, )</version>
            <scope>runtime</scope>
        </dependency>
    </dependencies>
    ...
</project>
```

Obviously replace the `datanucleus-rdbms` jar with the jar for whichever datastore you are using. If running your app using Maven "exec" plugin then the runtime specification may not be needed.

Please note that you can alternatively use the convenience artifact for JDO+RDBMS (or JDO+ whichever datastore you're using).

```
<project>
    ...
    <dependencies>
        ...
        <dependency>
            <groupId>org.datanucleus</groupId>
            <artifactId>datanucleus-accessplatform-jdo-rdbms</artifactId>
            <version>4.2.0-m1</version>
            <type>pom</type>
        </dependency>
    </dependencies>
    ...
</project>
```

Enhancement and SchemaTool

Now that you have the DataNucleus jars available to you, via the repositories, you want to perform DataNucleus operations. The primary operations are enhancement and SchemaTool. If you want to use the DataNucleus Maven plugin for enhancement or SchemaTool add the following to your `pom.xml`

```
<project>
    ...
    <build>
        <plugins>
            <plugin>
                <groupId>org.datanucleus</groupId>
                <artifactId>datanucleus-maven-plugin</artifactId>
                <version>5.0.0.release</version>
                <configuration>
                    <api>JDO</api>
                    <props>${basedir}/datanucleus.properties</props>
                    <log4jConfiguration>
${basedir}/log4j.properties</log4jConfiguration>
                    <verbose>true</verbose>
                </configuration>
                <executions>
                    <execution>
                        <phase>process-classes</phase>
                        <goals>
                            <goal>enhance</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```

This will use your Maven dependencies defined for your project, so will need to have `datanucleus-core.jar`, and either of `datanucleus-api-jdo.jar` or `datanucleus-api-jpa.jar` depending on what type of metadata is being used. SchemaTool will also need the `datanucleus-{datastore}.jar` for whichever datastore is being used (where you will create the schema).

The *executions* part of that will make enhancement be performed immediately after compile, so automatic. See also the Enhancer docs

To run the enhancer manually you do

```
mvn datanucleus:enhance
```

DataNucleus SchemaTool is achieved similarly, via

```
mvn datanucleus:schema-create
```

Eclipse Plugin

Eclipse provides a powerful development environment for Java systems. DataNucleus provides its own plugin for use within Eclipse, giving access to many features of DataNucleus from the convenience of your development environment.

Plugin Installation

The DataNucleus plugin requires Eclipse 3.1 or above. To obtain and install the DataNucleus Eclipse plugin select *Help → Software Updates → Find and Install*

On the panel that pops up select *Search for new features to install*

Select *New Remote Site*, and in that new window set the URL as **http://www.datanucleus.org/downloads/eclipse-update/** and the name as DataNucleus. Now select the site it has added "DataNucleus", and click "Finish". This will then find the releases of the DataNucleus plugin. **Select the latest version of the DataNucleus Eclipse plugin**. Eclipse then downloads and installs the plugin. Easy!

Plugin configuration

The DataNucleus Eclipse plugin allows saving of preferences so that you get nice defaults for all subsequent usage. You can set the preferences at two levels :-

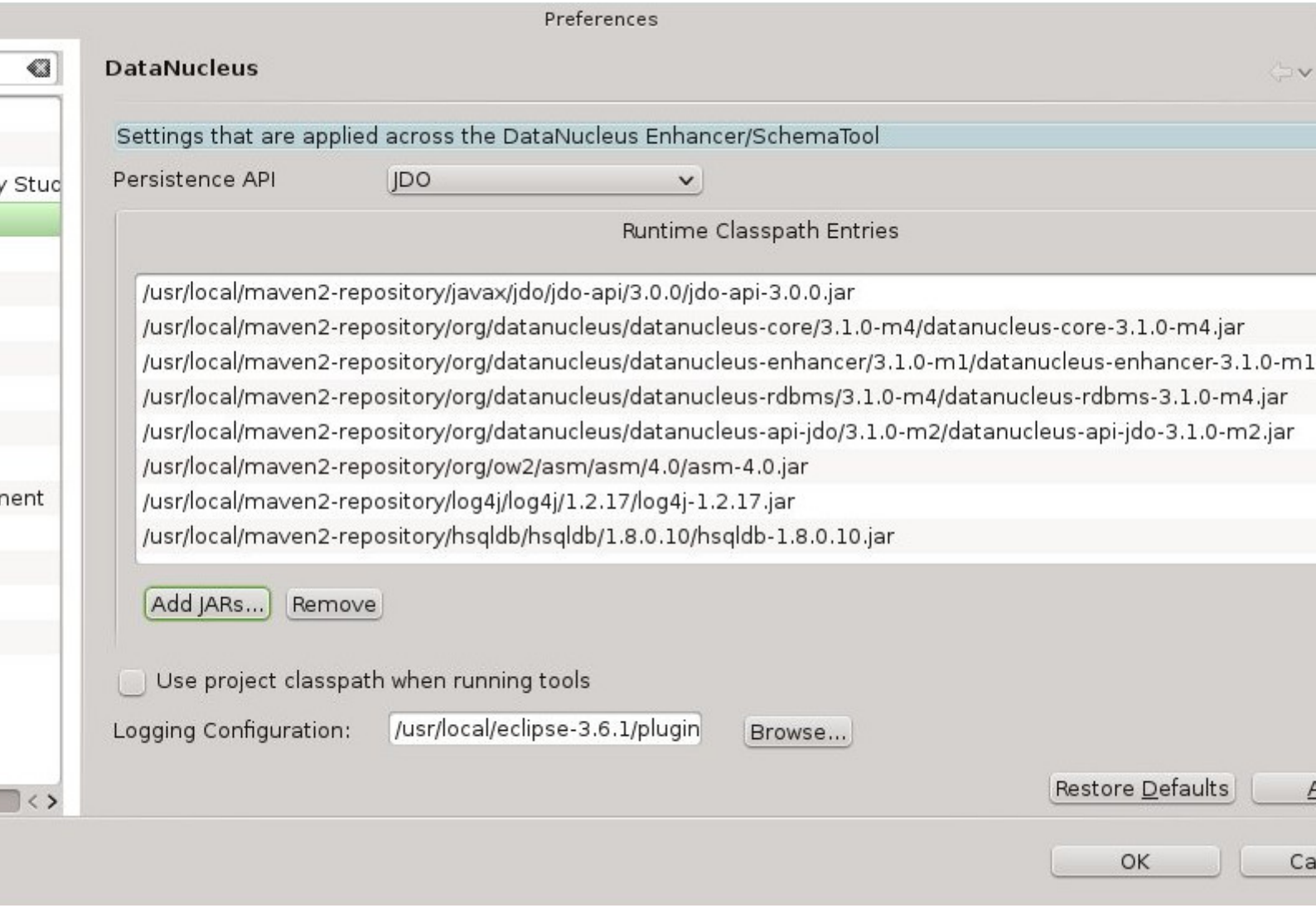
- **Globally for the Plugin** : Go to *Window → Preferences → DataNucleus Eclipse Plugin* and see the options below that
- **For a Project** : Go to *{your project} → Properties → DataNucleus Eclipse Plugin* and select "Enable project-specific properties"

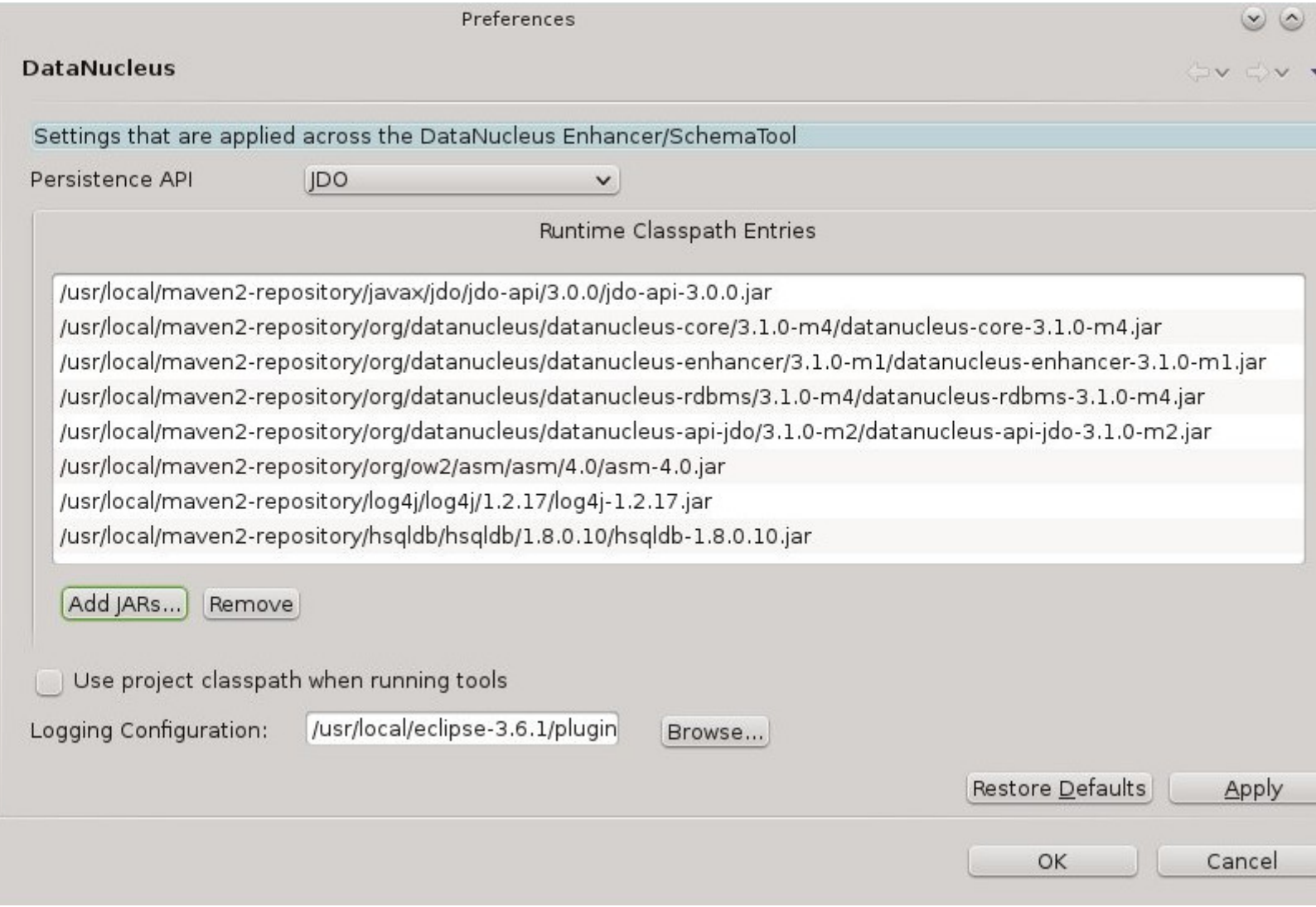
Plugin configuration - General

Firstly open the main plugin preferences page, set the API to be used, and configure the libraries needed by DataNucleus. These are in addition to whatever you already have in your projects CLASSPATH, but to run the DataNucleus Enhancer/SchemaTool you will require the following

- `javax.jdo.jar` - the JDO API jar
- `datanucleus-core`
- `datanucleus-api-jdo` - the DataNucleus JDO API implementation
- `datanucleus-rdbms` - for running SchemaTool for RDBMS (use the appropriate `datanucleus-{datastore}` for your chosen datastore.
- Datastore driver jar (e.g JDBC) : for running SchemaTool

Below this you can set the location of a configuration file for Log4j to use. This is useful when you want to debug the Enhancer/SchemaTool operations.

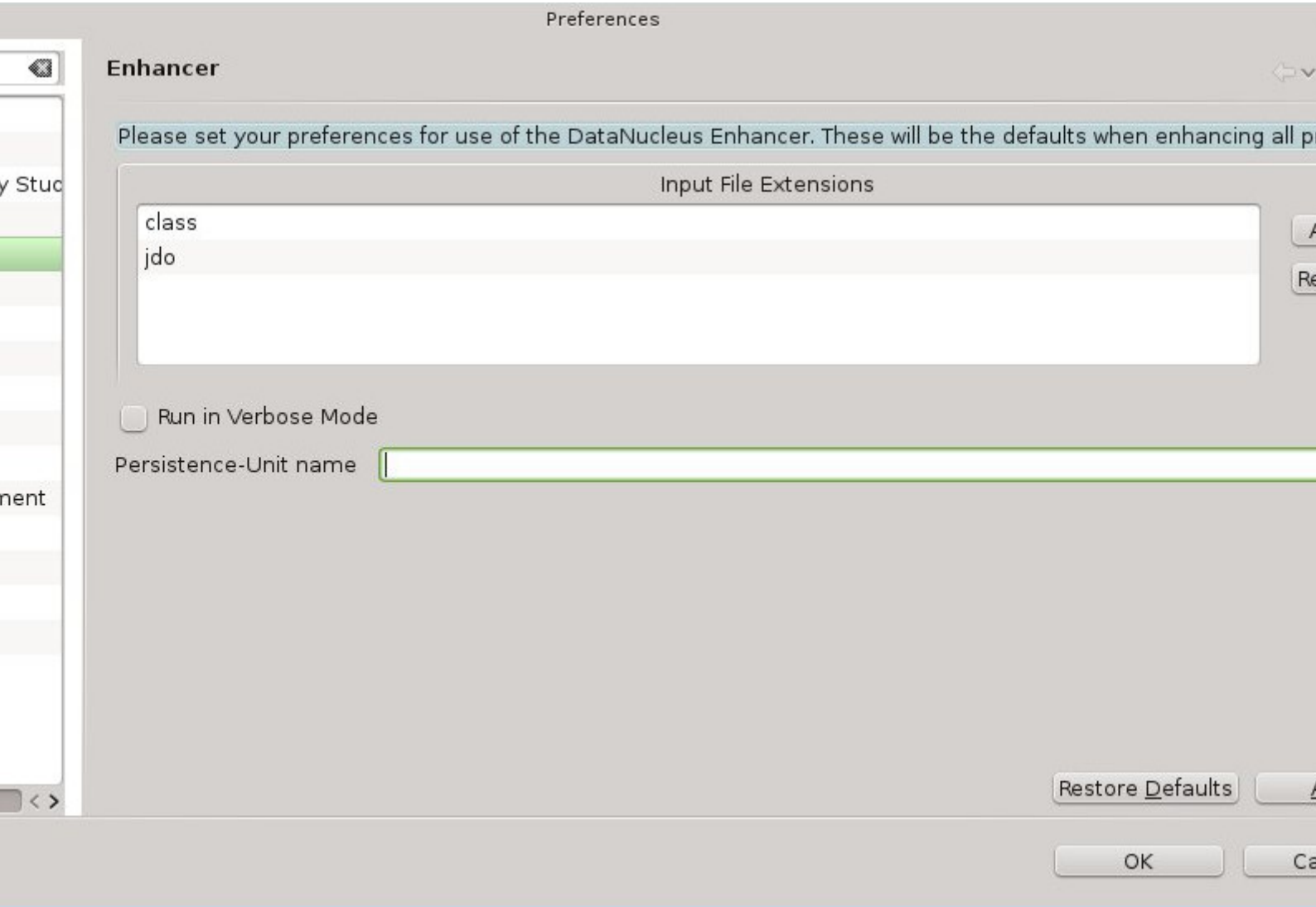


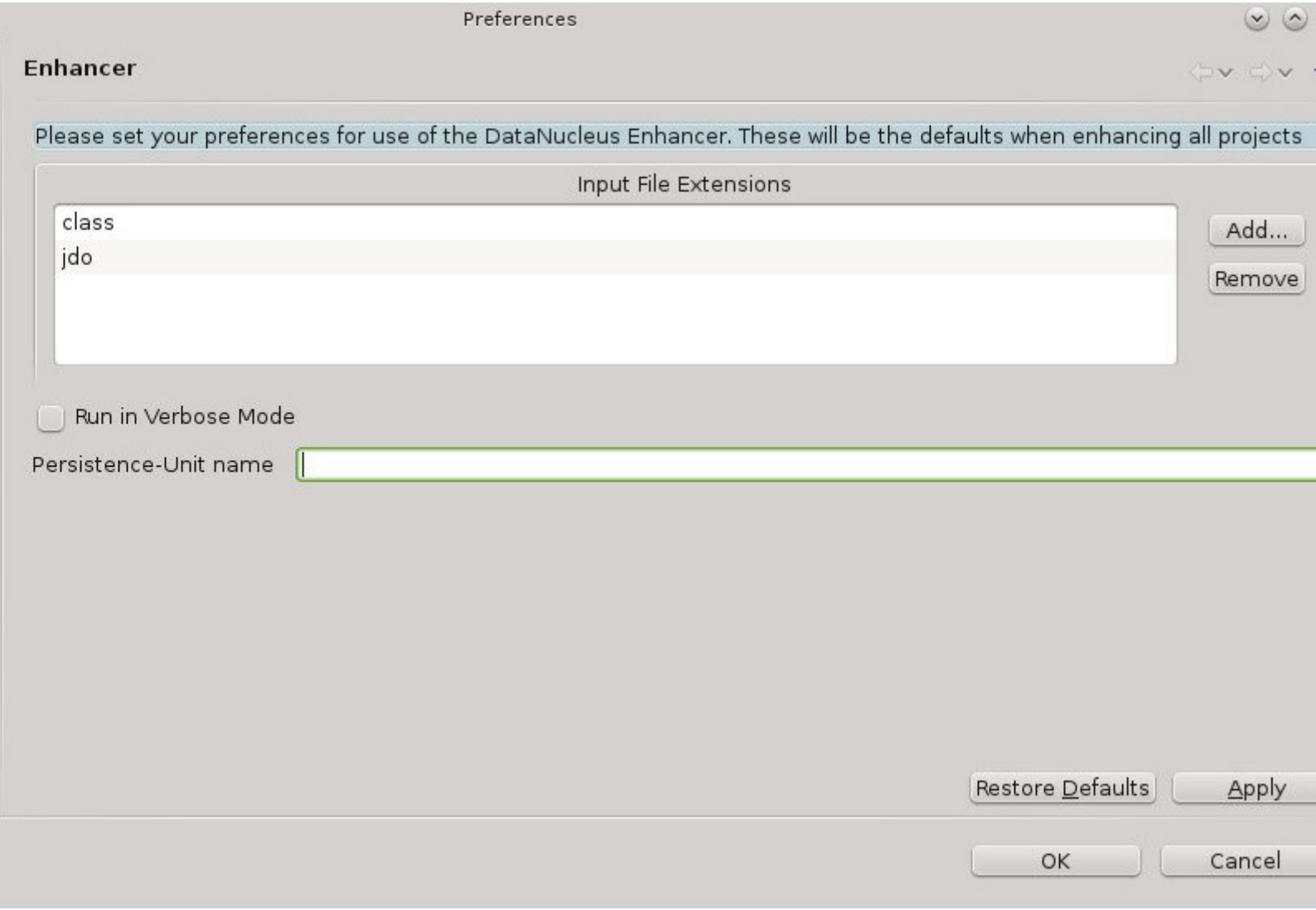


Plugin configuration - Enhancer

Open the "Enhancer" page. You have the following settings

- **Input file extensions** : the enhancer accepts input defining the classes to be enhanced. This is typically performed by passing in the JDO XML MetaData files. When you use annotations you need to pass in *class* files. So you select the suffices you need
- **Verbose** : selecting this means you get much more output from the enhancer
- **PersistenceUnit** : Name of the persistence unit if enhancing a persistence-unit

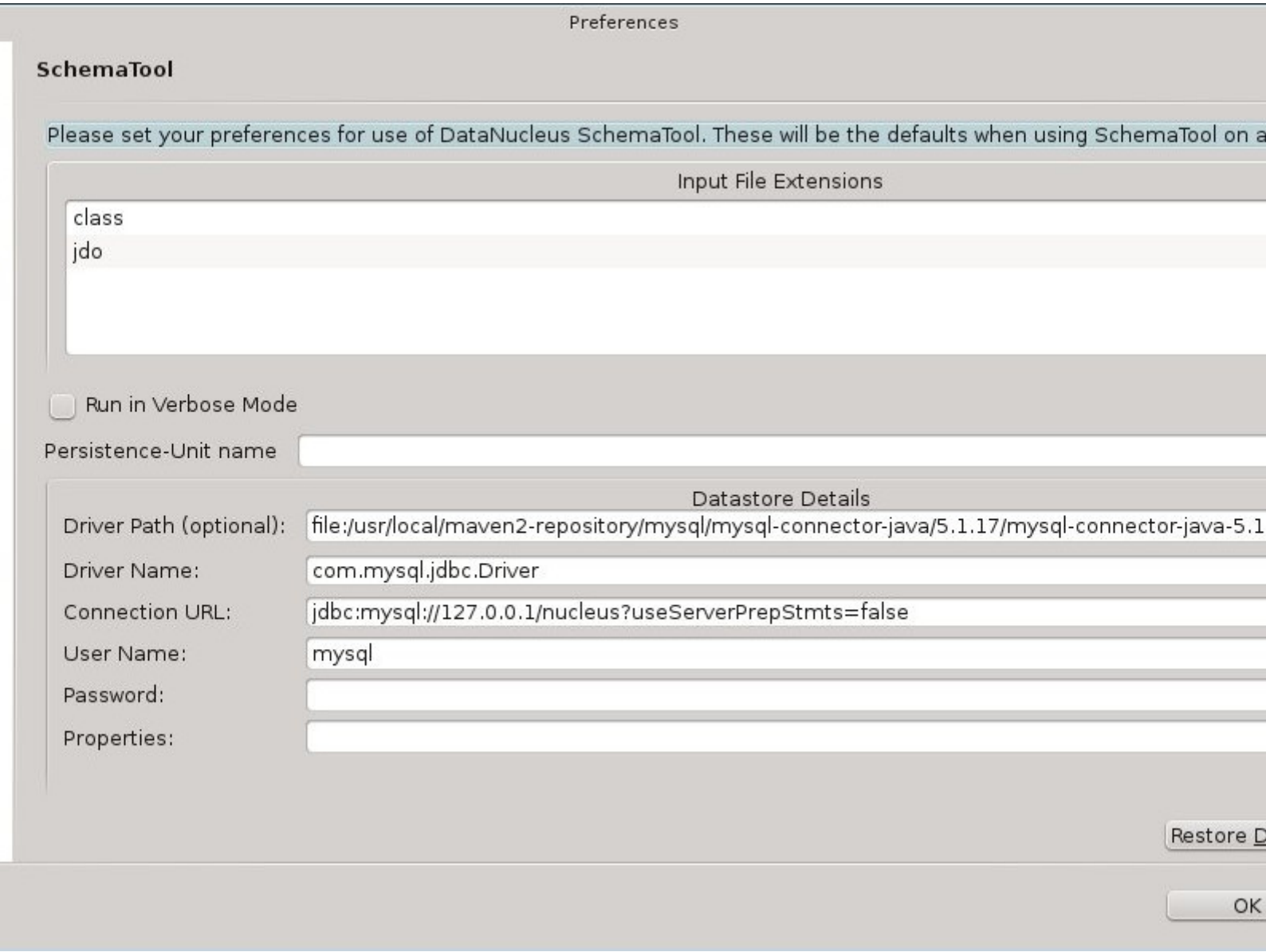

Preferences
Enhancer
Please set your preferences for use of the DataNucleus Enhancer. These will be the defaults when enhancing all p
Input File Extensions
class
jdo
Run in Verbose Mode
Persistence-Unit name
Restore Defaults
OK

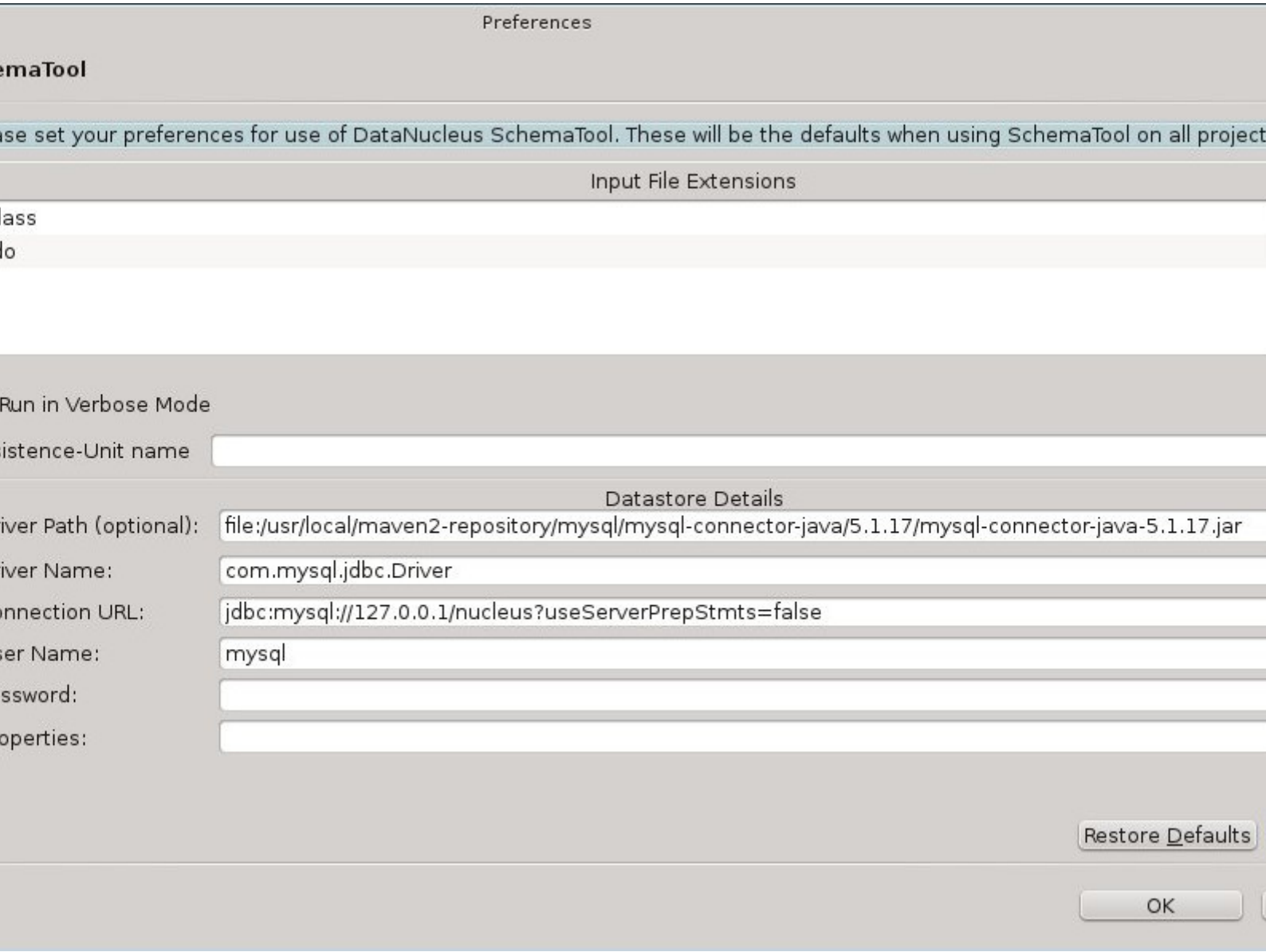


Plugin configuration - SchemaTool

Open the "SchemaTool" page. You have the following settings

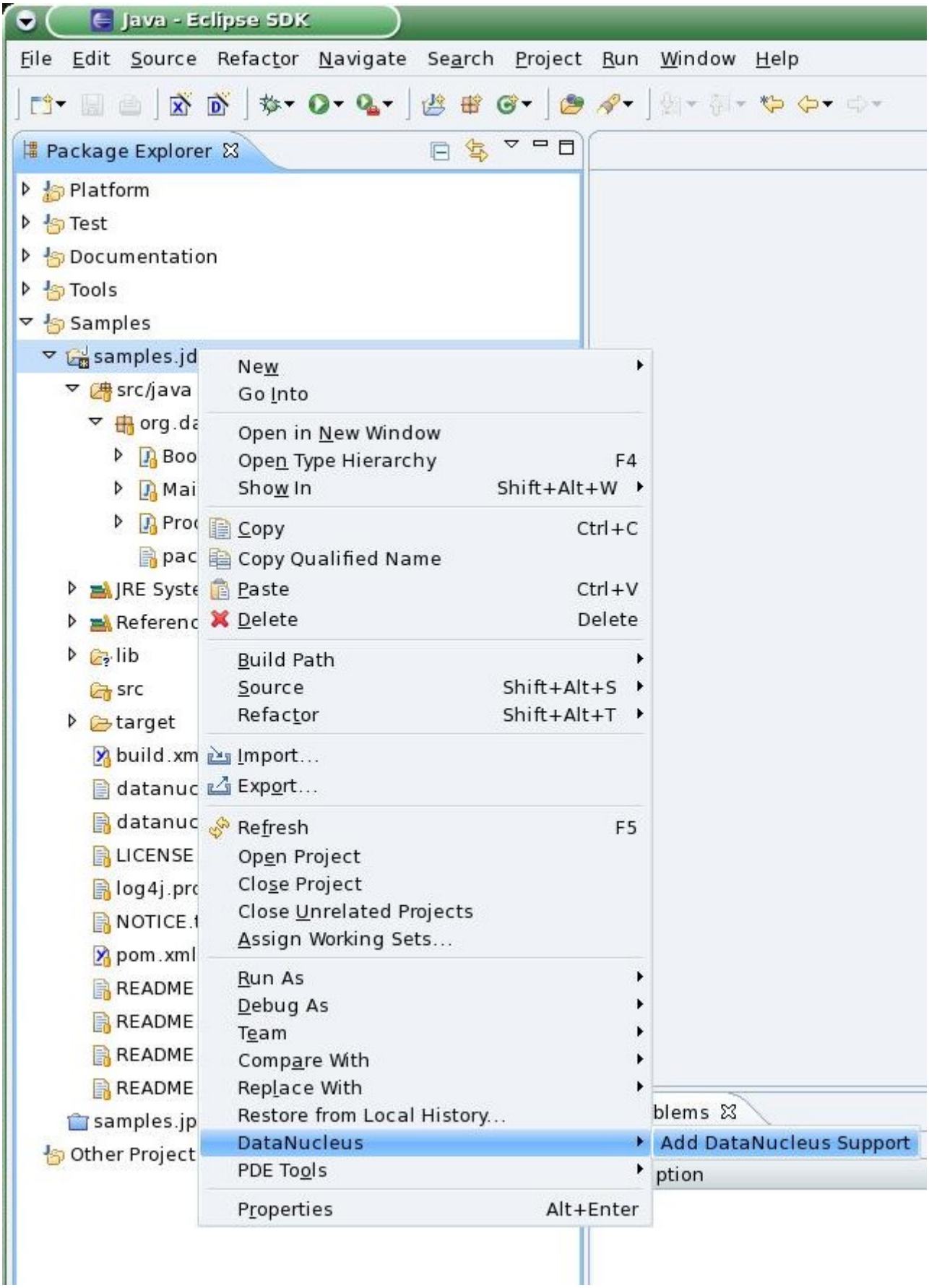
- **Input file extensions** : SchemaTool accepts input defining the classes to have their schema generated. This is typically performed by passing in the JDO XML MetaData files. When you use annotations you need to pass in *class* files. So you select the suffices you need
- **Verbose** : selecting this means you get much more output from SchemaTool
- **PersistenceUnit** : Name of the persistence unit if running SchemaTool on a persistence-unit
- **Datastore details** : You can either specify the location of a properties file defining the location of your datastore, or you supply the driver name, URL, username and password.


Preferences
SchemaTool
Please set your preferences for use of DataNucleus SchemaTool. These will be the defaults when using SchemaTool on a
Input File Extensions
class
jdo
Run in Verbose Mode
Persistence-Unit name
Datastore Details
Driver Path (optional):
file:/usr/local/maven2-repository/mysql/mysql-connector-java/5.1.17/mysql-connector-java-5.1
Driver Name:
com.mysql.jdbc.Driver
Connection URL:
jdbc:mysql://127.0.0.1/nucleus?useServerPrepStmts=false
User Name:
mysql
Password:
Properties:
Restore D
OK



Enabling DataNucleus support

First thing to note is that the DataNucleus plugin is for Eclipse "Java project"s only. After having configured the plugin you can now add DataNucleus support on your projects. Simply right-click on your project in **Package Explorer** and select *DataNucleus→"Add DataNucleus Support"* from the context menu.



Defining JDO XML Metadata

It is standard practice to define the MetaData for your persistable classes in the same package as these classes. You now define your MetaData, by right-click on a package in your project and select

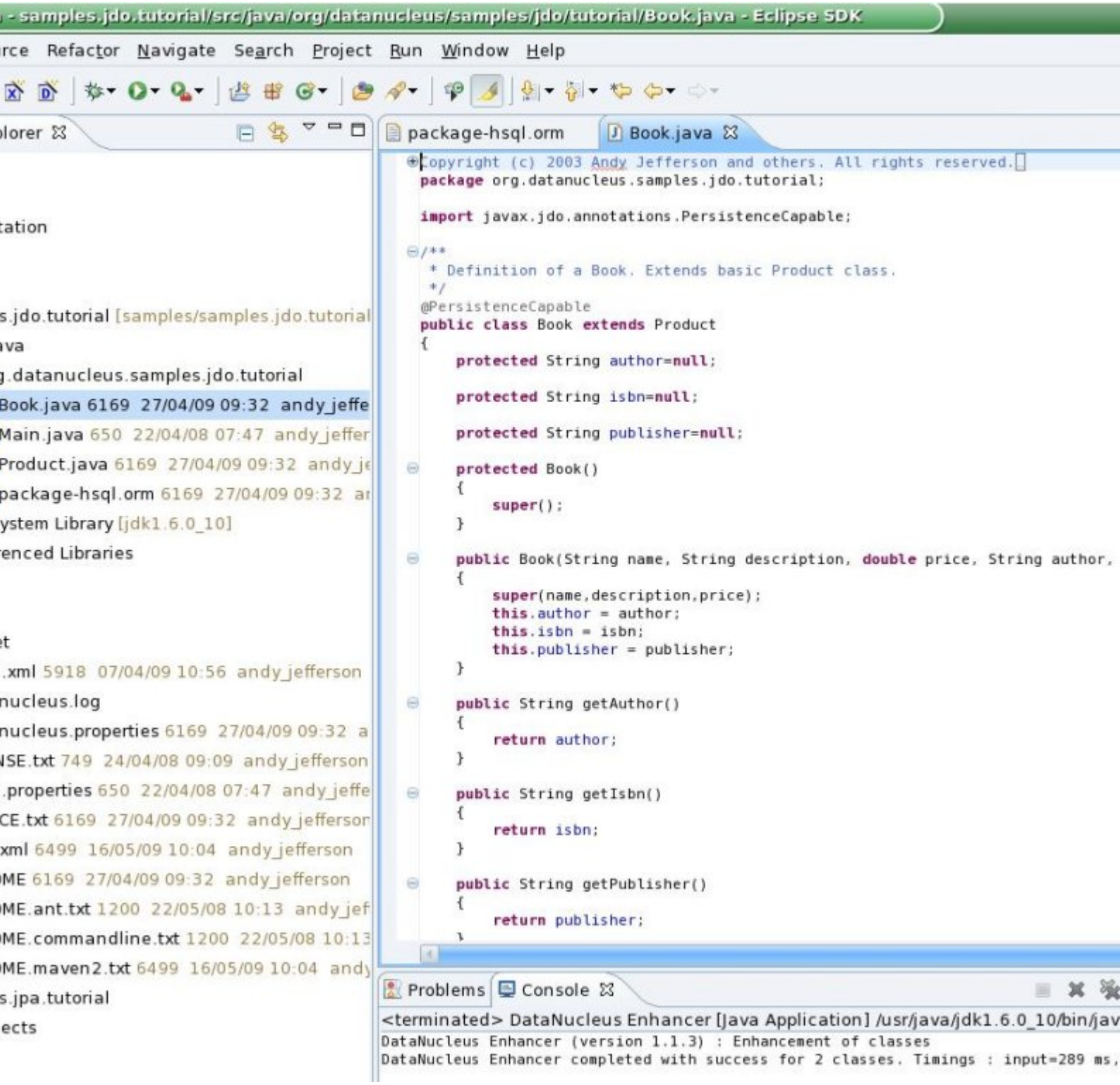
"Create JDO 2.0 Metadata File" from DataNucleus context menu. The dialog prompts for the file name to be used and creates a basic Metadata file for all classes in this package, which can now be adapted to your needs. You can also perform same steps as above on a *.java file, which will create the metadata for the selected file only. Please note that the wizard will overwrite existing files without further notice.

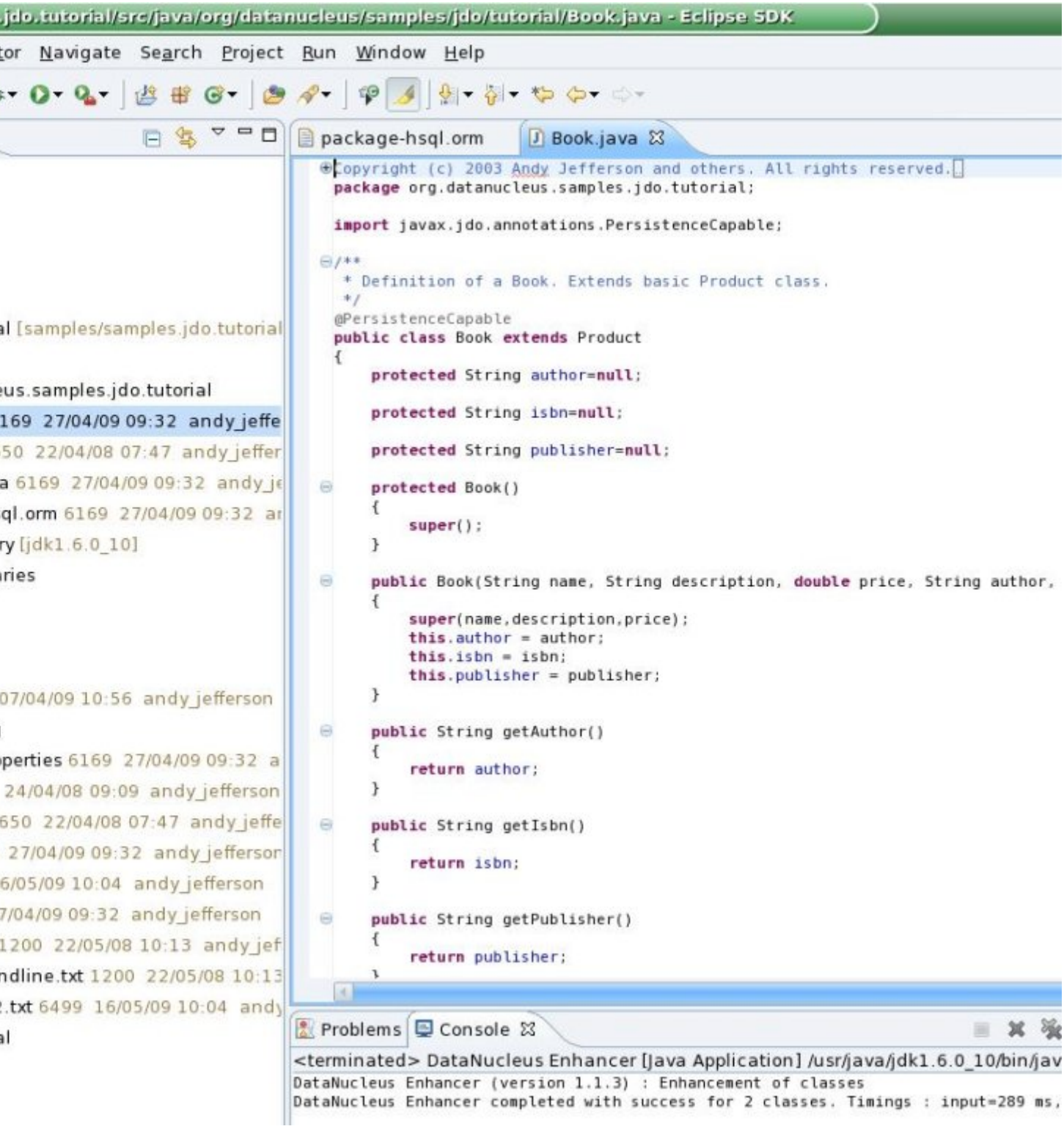
Defining 'persistence.xml'

You can also use the DataNucleus plugin to generate a `persistence.xml` file adding all classes into a single *persistence-unit* (only xml metadata is supported currently). You do this by right-clicking on a package in your project, and selecting the option. The `persistence.xml` is generated under META-INF for the source folder. Please note that the wizard will overwrite existing files without further notice.

Enhancing the classes

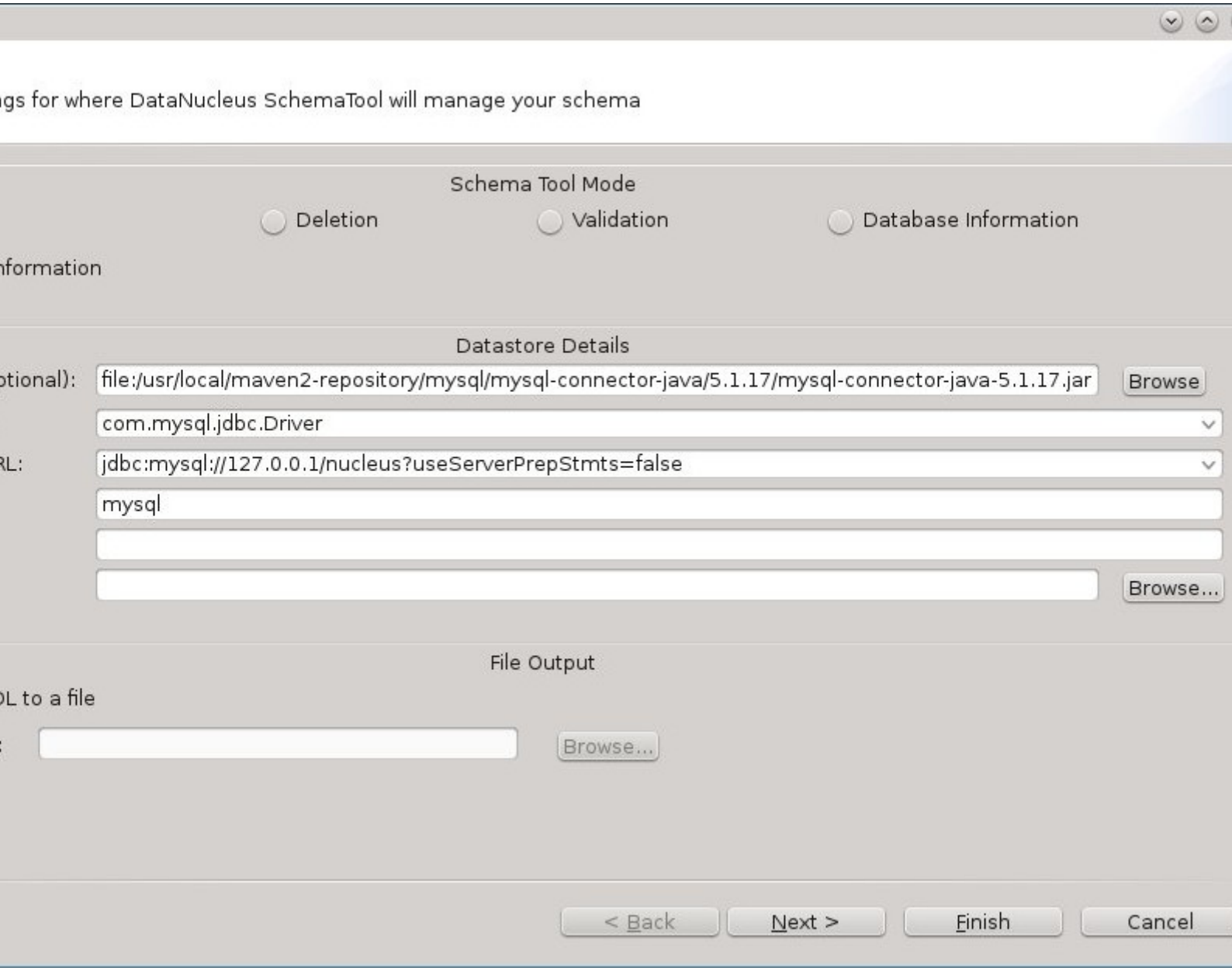
The DataNucleus Eclipse plugin allows you to easily byte-code enhance your classes using the DataNucleus enhancer. Right-click on your project and select "Enable Auto-Enhancement" from the DataNucleus context menu. Now that you have the enhancer set up you can enable enhancement of your classes. The DataNucleus Eclipse plugin currently works by enabling/disabling automatic enhancement as a follow on process for the Eclipse build step. This means that when you enable it, every time Eclipse builds your classes it will then enhance the classes defined by the available "jdo" MetaData files. Thereafter every time that you build your classes the JDO enabled ones will be enhanced. Easy! Messages from the enhancement process will be written to the Eclipse Console. **Make sure that you have your Java files in a source folder, and that the binary class files are written elsewhere** If everything is set-up right, you should see the output below.

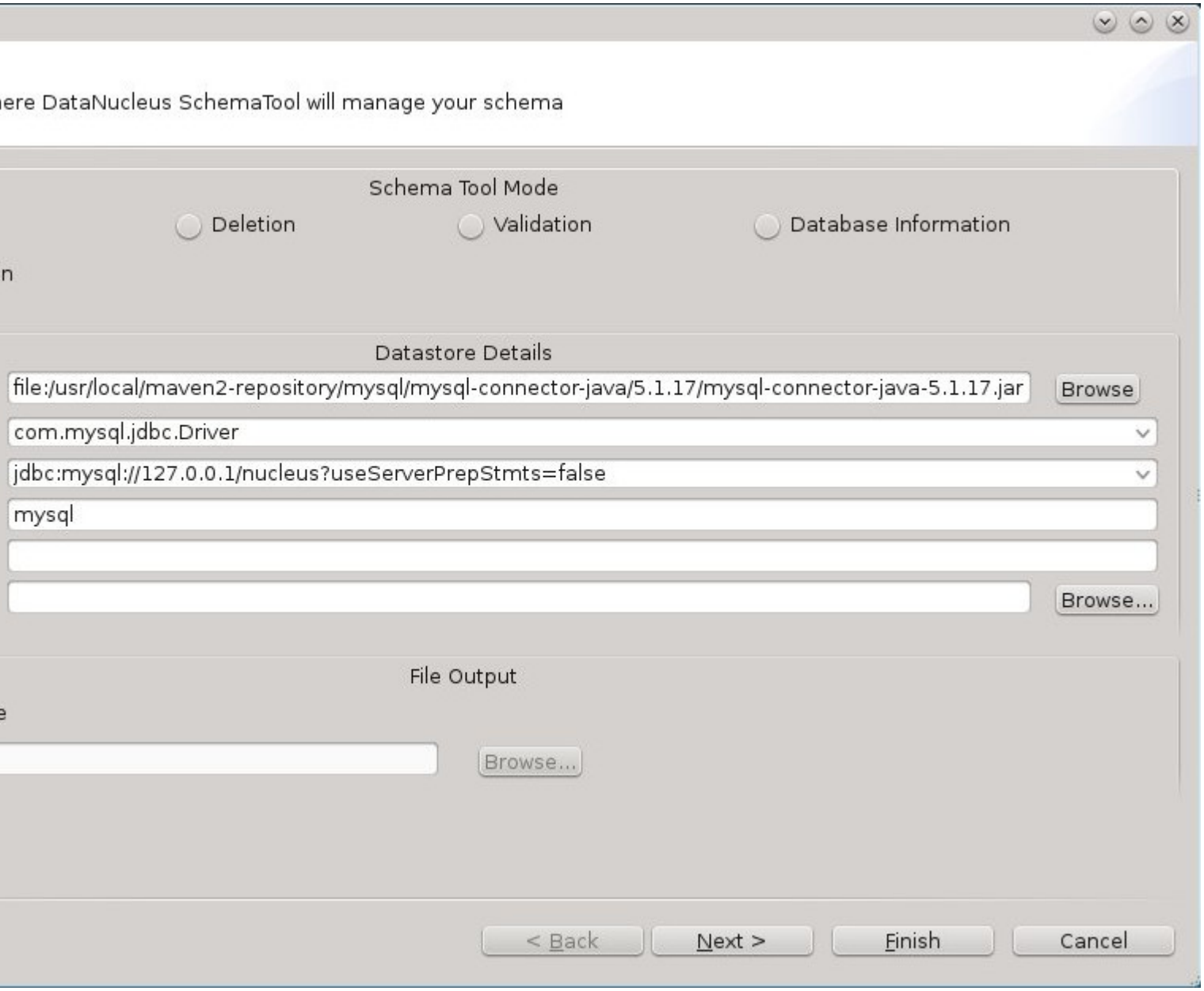

- samples.jdo.tutorial/src/java/org/datanucleus/samples/jdo/tutorial/Book.java - Eclipse SDK
rce Refactor Navigate Search Project Run Window Help
package-hsql.orm
Book.java
Copyright (c) 2003 Andy Jefferson and others. All rights reserved.
package org.datanucleus.samples.jdo.tutorial;
import javax.jdo.annotations.PersistenceCapable;
/**
* Definition of a Book. Extends basic Product class.
*/
@PersistenceCapable
public class Book extends Product
{
protected String author=null;
protected String isbn=null;
protected String publisher=null;
protected Book()
{
super();
}
public Book(String name, String description, double price, String author,
{
super(name,description,price);
this.author = author;
this.isbn = isbn;
this.publisher = publisher;
}
public String getAuthor()
{
return author;
}
public String getIsbn()
{
return isbn;
}
public String getPublisher()
{
return publisher;
Problems
Console
<terminated> DataNucleus Enhancer [Java Application] /usr/java/jdk1.6.0_10/bin/jav
DataNucleus Enhancer (version 1.1.3) : Enhancement of classes
DataNucleus Enhancer completed with success for 2 classes. Timings : input=289 ms,



Generating your database schema

Once your classes have been enhanced you are in a position to create the database schema (assuming you will be using a new schema - omit this step if you already have your schema). Click on the project under "Package Explorer" and under "DataNucleus" there is an option "Run SchemaTool". This brings up a panel to define your database location (URL, login, password etc). You enter these details and the schema will be generated.


gs for where DataNucleus SchemaTool will manage your schema
Schema Tool Mode
Deletion
Validation
Database Information
nformation
Datastore Details
otional):
file:/usr/local/maven2-repository/mysql/mysql-connector-java/5.1.17/mysql-connector-java-5.1.17.jar
Browse
com.mysql.jdbc.Driver
RL:
jdbc:mysql://127.0.0.1/nucleus?useServerPrepStmts=false
mysql
Browse...
File Output
DL to a file
Browse...
< Back
Next >
Finish
Cancel



Messages from the SchemaTool process will be written to the Eclipse Console.

Netbeans

This guide is based around Netbeans v7.0

Perhaps the most important step in developing applications with DataNucleus is the enhancement of compiled classes. NetBeans provides a convenient way of integrating this procedure into the build process without the need for any additional tools or plugins. This is possible because NetBeans has native integration with Maven and Ant. Any DataNucleus project based on Maven will open and run as is with Netbeans. No changes are needed in the project nor in Netbeans. When using Ant Class enhancement thus becomes a simple matter of adding a new task to the existing `build.xml` generated by NetBeans.

This tutorial shows how to integrate DataNucleus with NetBeans 7.0 to simplify the development of JDO applications. **Please contribute any updates to this guide that you have since the developers of DataNucleus don't use Netbeans**

Requirements

The following components are required to complete this tutorial successfully:

- DataNucleus AccessPlatform
- DataNucleus Samples
- NetBeans
- Hsqldb

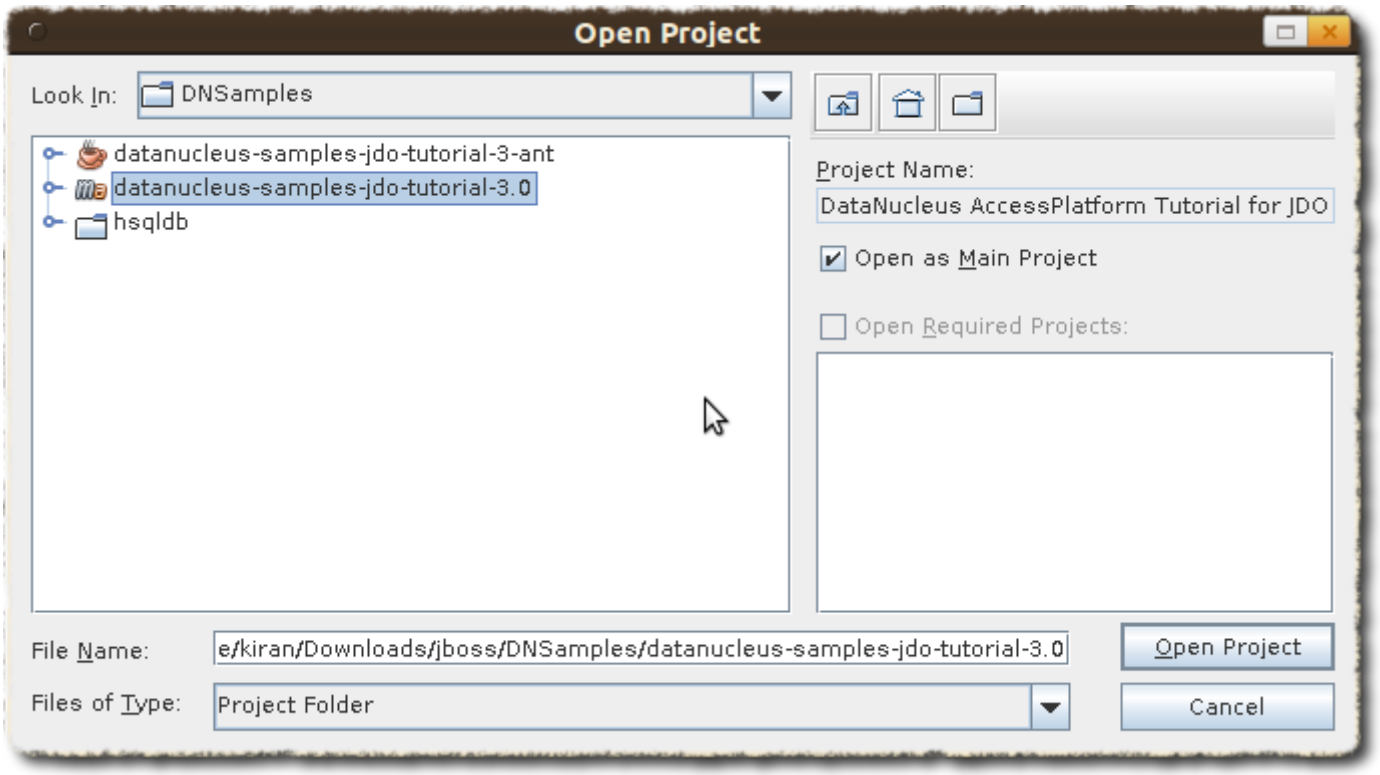
Maven : Working with a DataNucleus Maven Project

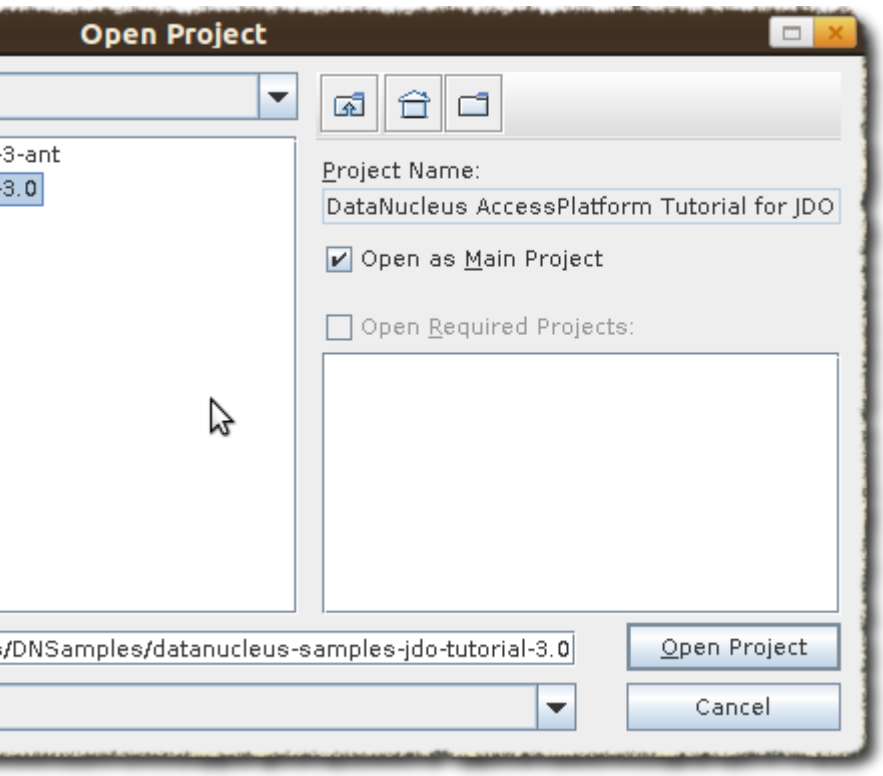
NetBeans has native integration with Maven. If your project builds with Maven then it will also build with NetBeans without any change. Unzip the sample project to *$home/DNSamples/datanucleus-samples-jdo-tutorial-3.0*

The sample project comes with a `pom.xml`. When you try to open this project NetBeans will automatically detect it as maven project.

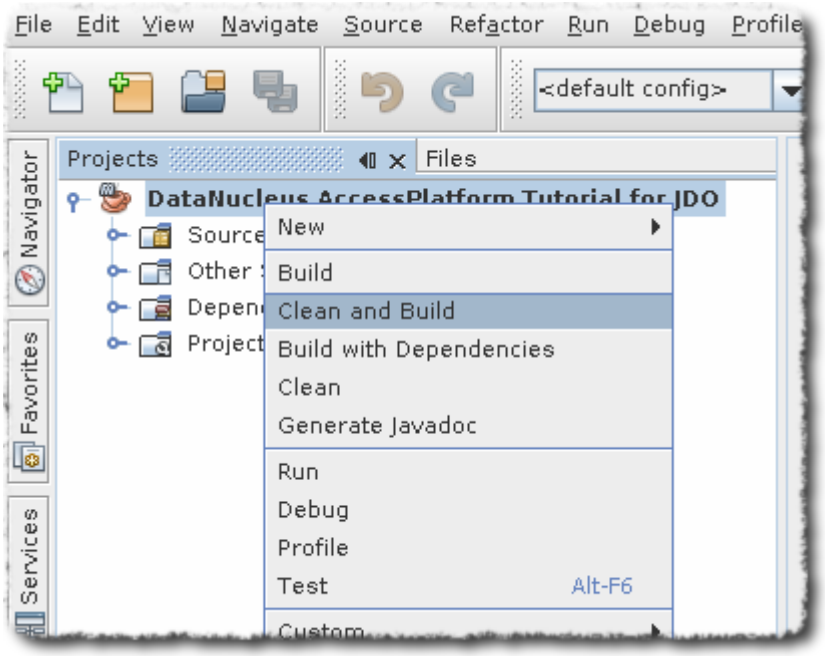
Start Netbeans

Open Maven Project





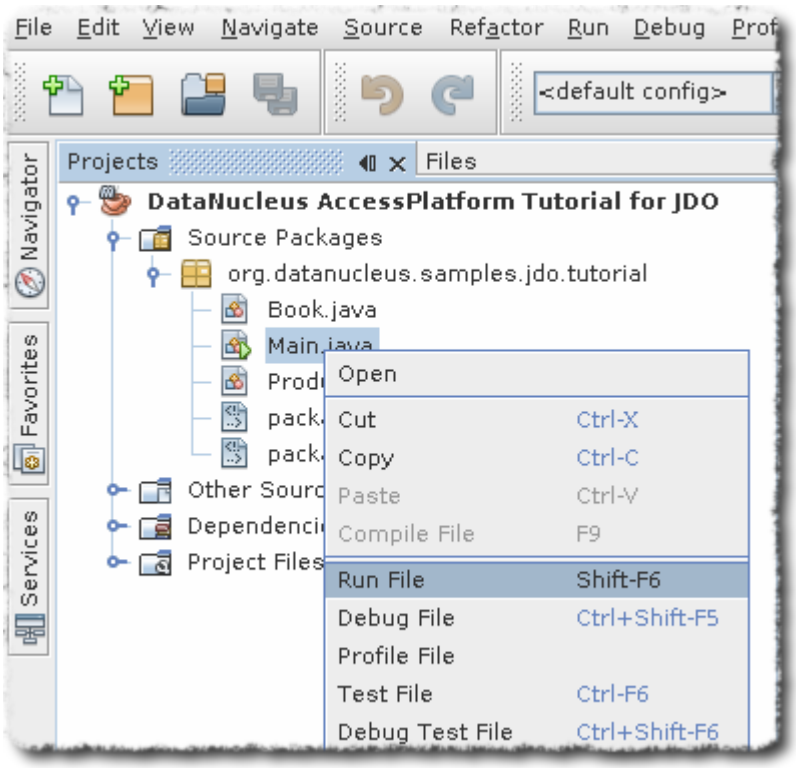
Clean Build the project



You will see a success message as below

```
BUILD SUCCESS

Total time: 5.406s
Finished at: Fri Aug 05 09:23:53 IST 2011
Final Memory: 14M/108M
```

Run Main Class



The results can be seen in the output window

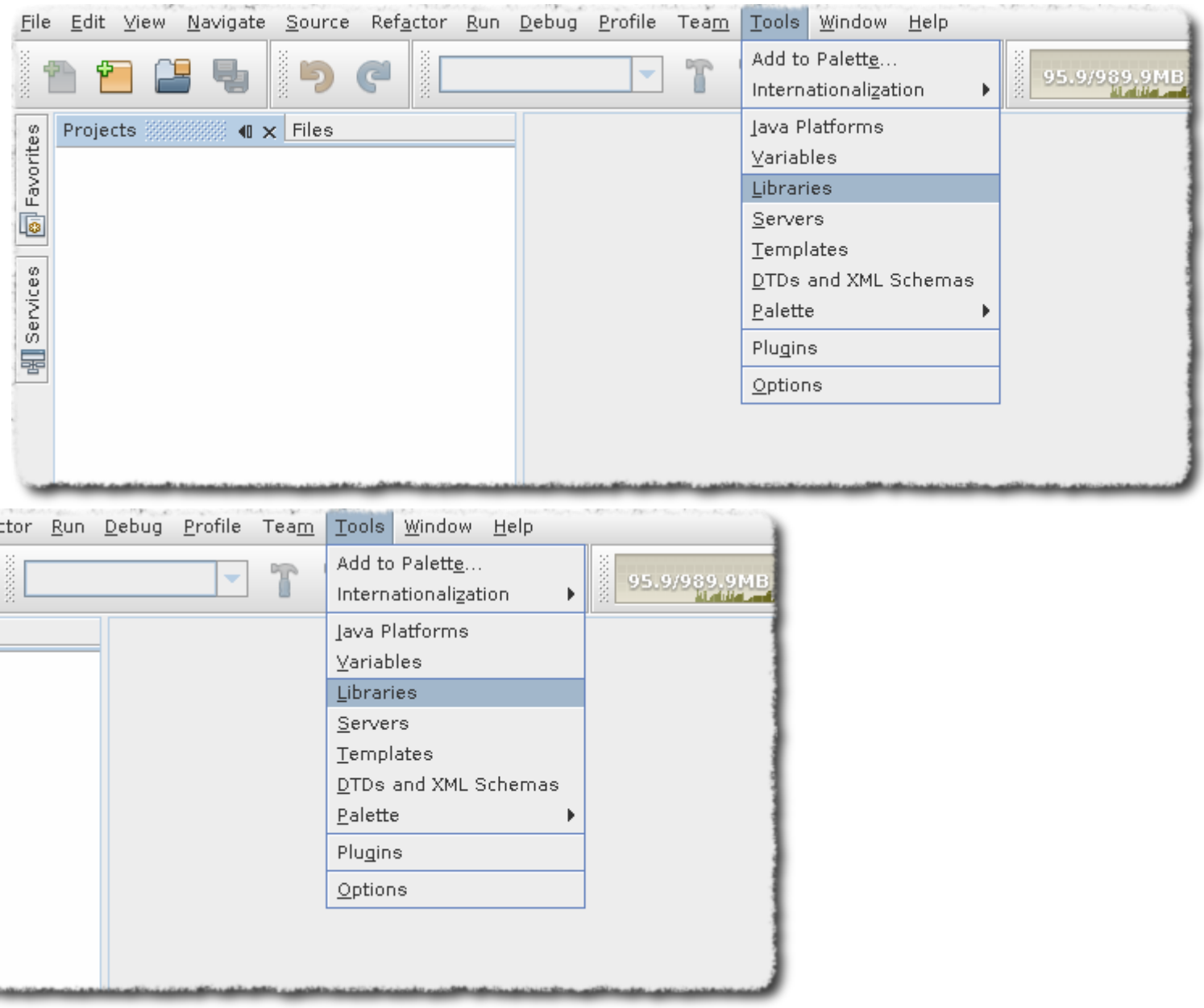
```
DataNucleus AccessPlatform with JPA

Persisting products
Product and Book have been persisted

Retrieving Extent for Products
>> Book : JRR Tolkien - Lord of the Rings by Tolkien
>> Product : Sony Discman [A standard discman from Sony]

Executing Query for Products with price below 150.00
>> Book : JRR Tolkien - Lord of the Rings by Tolkien

Deleting all products from persistence
Deleted 2 products

End of Tutorial

BUILD SUCCESS
Total time: 6.221s
Finished at: Fri Aug 05 09:27:43 IST 2011
Final Memory: 7M/106M
```

Setting up NetBeans for DataNucleus ANT use

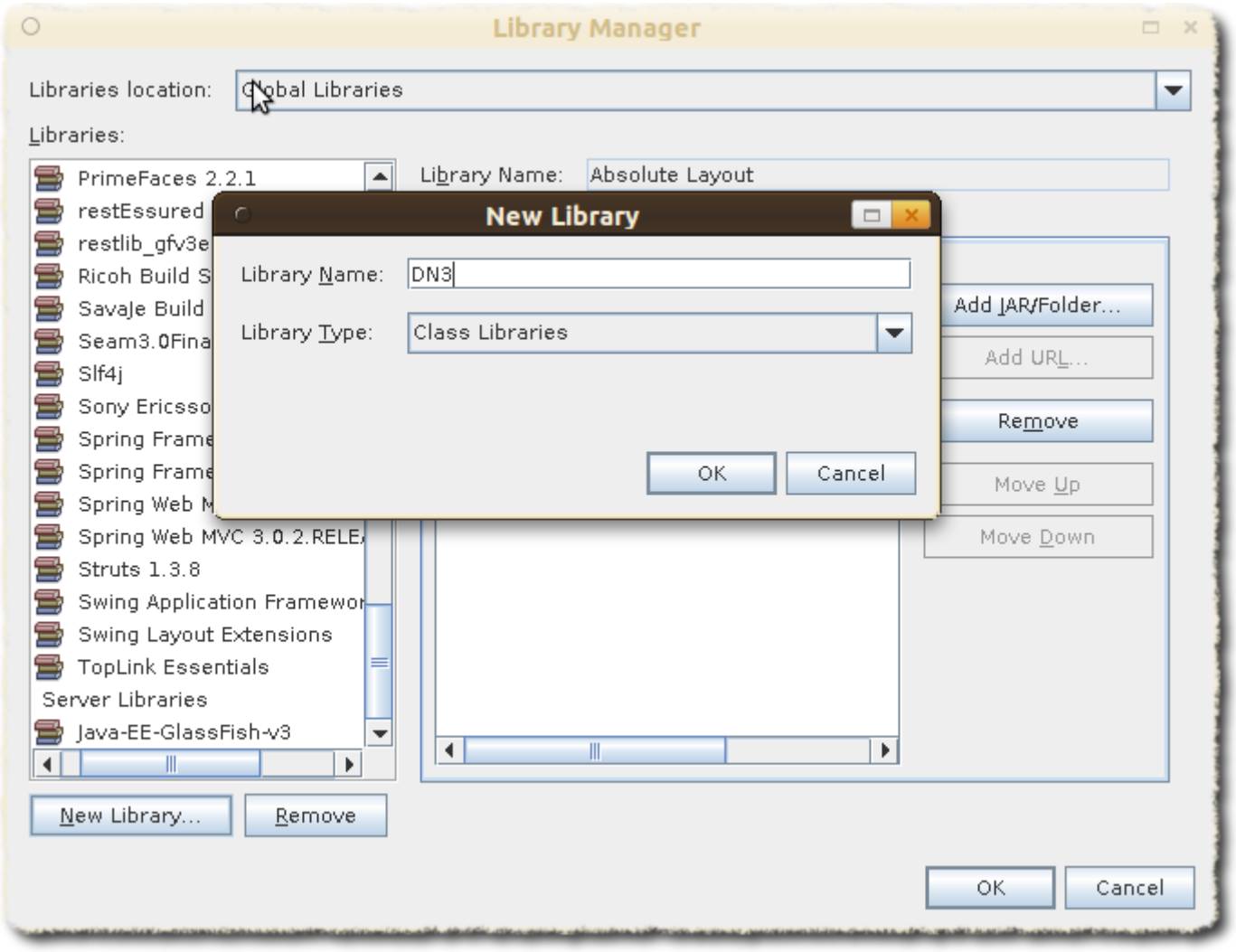
The first thing to do is to register the DataNucleus components in the **Library Manager** of NetBeans 7 so that these become available to any project created with the IDE. This involves creating a new

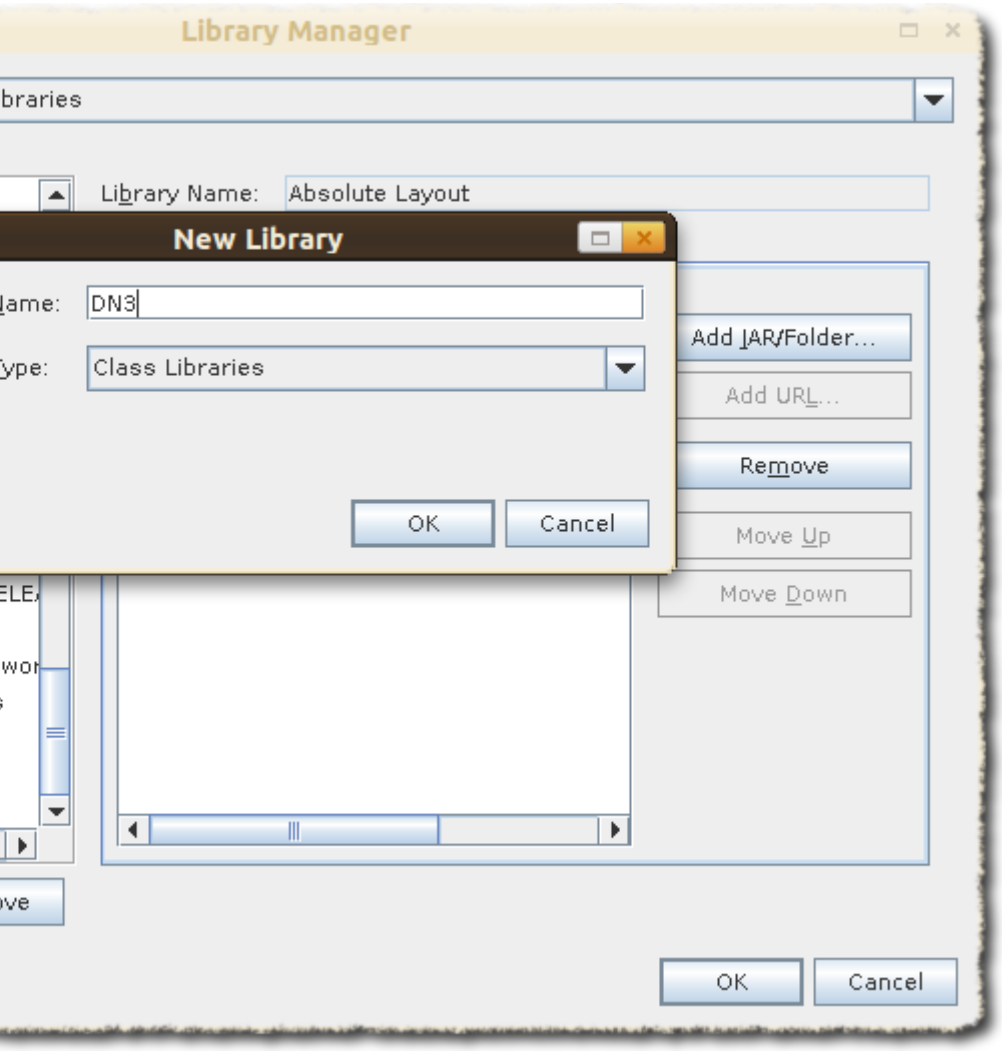
library and adding the JAR files to it, as shown in the following screenshots

Open Library Manager

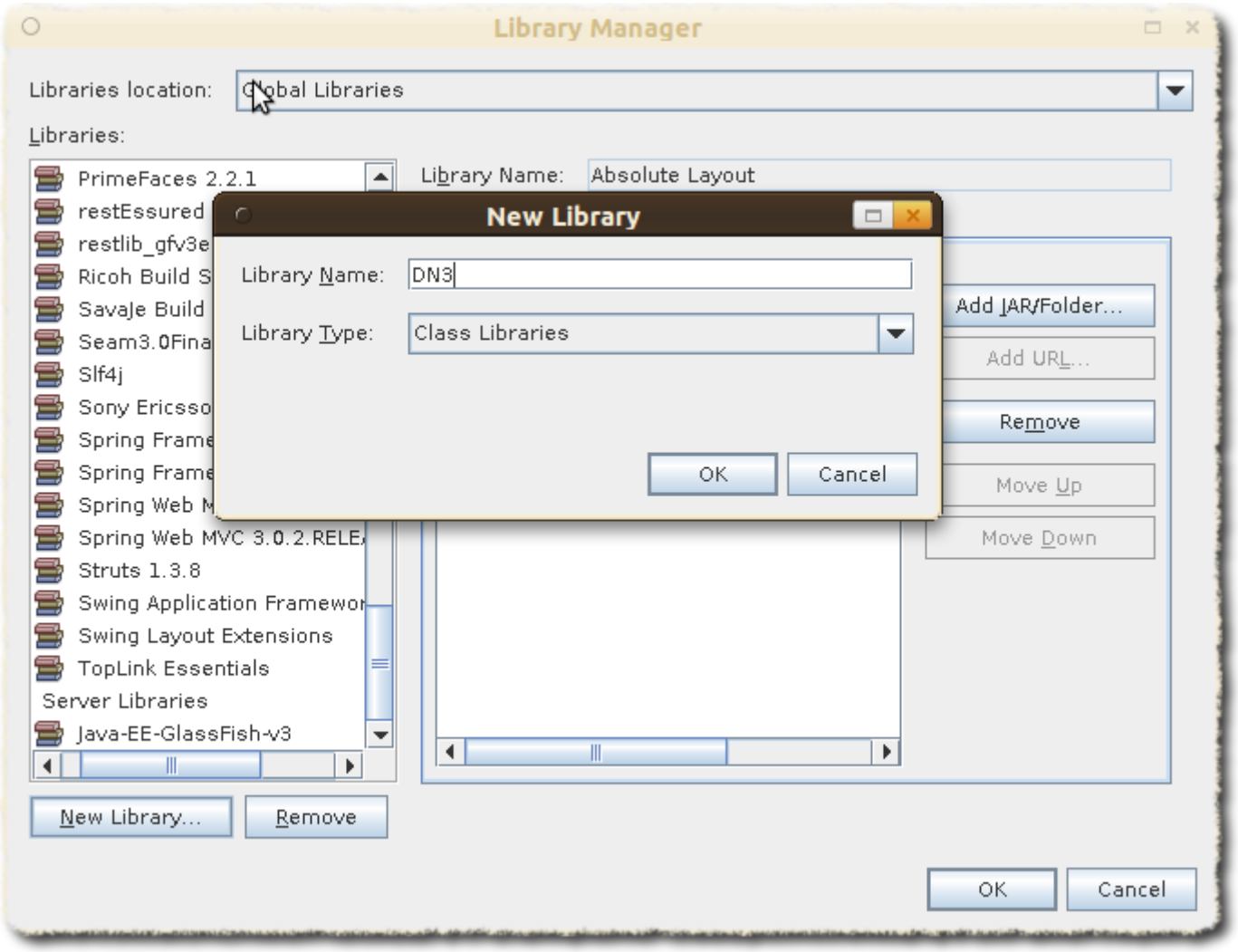


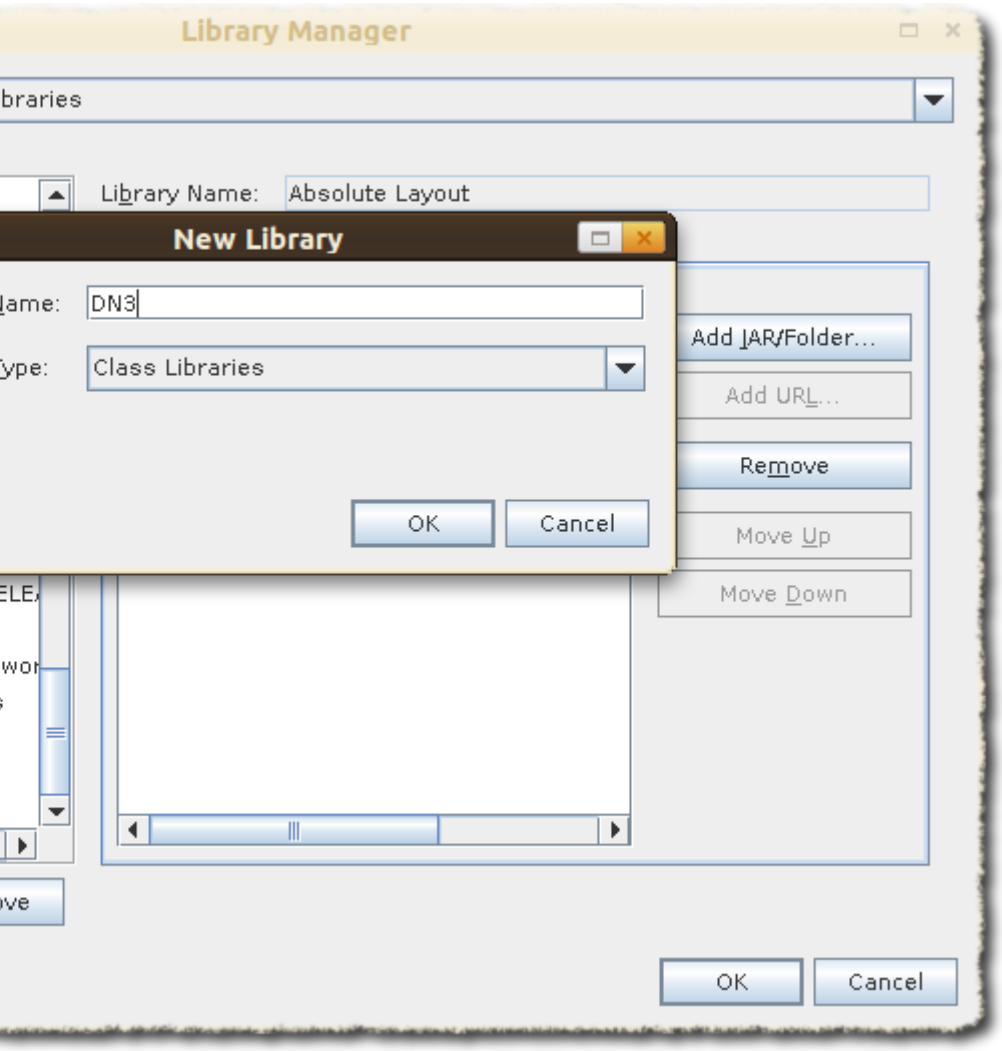
Add New Library with Name DN3 - Repeat the process for DN3Deps


Library Manager
Libraries location:
Global Libraries
Libraries:
PrimeFaces 2.2.1
restEssured
restlib_gfv3e
Ricoh Build S
SavaJe Build
Seam3.0Fina
Slf4j
Sony Ericsso
Spring Frame
Spring Frame
Spring Web M
Spring Web MVC 3.0.2.RELE
Struts 1.3.8
Swing Application Framewo
Swing Layout Extensions
TopLink Essentials
Server Libraries
Java-EE-GlassFish-v3
Library Name:
Absolute Layout
Add JAR/Folder...
Add URL...
Remove
Move Up
Move Down
New Library...
Remove
OK
Cancel
New Library
Library Name:
DN3
Library Type:
Class Libraries
OK
Cancel



Add all the libraries under *$home/datanucleus-accessplatform-full-deps-3.0.0-release/lib* to DN3. Add all the libraries under *$home/datanucleus-accessplatform-full-deps-3.0.0-release/deps* to DN3Deps


Library Manager
Libraries location:
Global Libraries
Libraries:
PrimeFaces 2.2.1
restEssured
restlib_gfv3e
Ricoh Build S
SavaJe Build
Seam3.0Fina
Slf4j
Sony Ericsso
Spring Frame
Spring Frame
Spring Web M
Spring Web MVC 3.0.2.RELE
Struts 1.3.8
Swing Application Framewo
Swing Layout Extensions
TopLink Essentials
Server Libraries
Java-EE-GlassFish-v3
Library Name:
Absolute Layout
New Library
Library Name:
DN3
Library Type:
Class Libraries
OK
Cancel
Add JAR/Folder...
Add URL...
Remove
Move Up
Move Down
New Library...
Remove
OK
Cancel

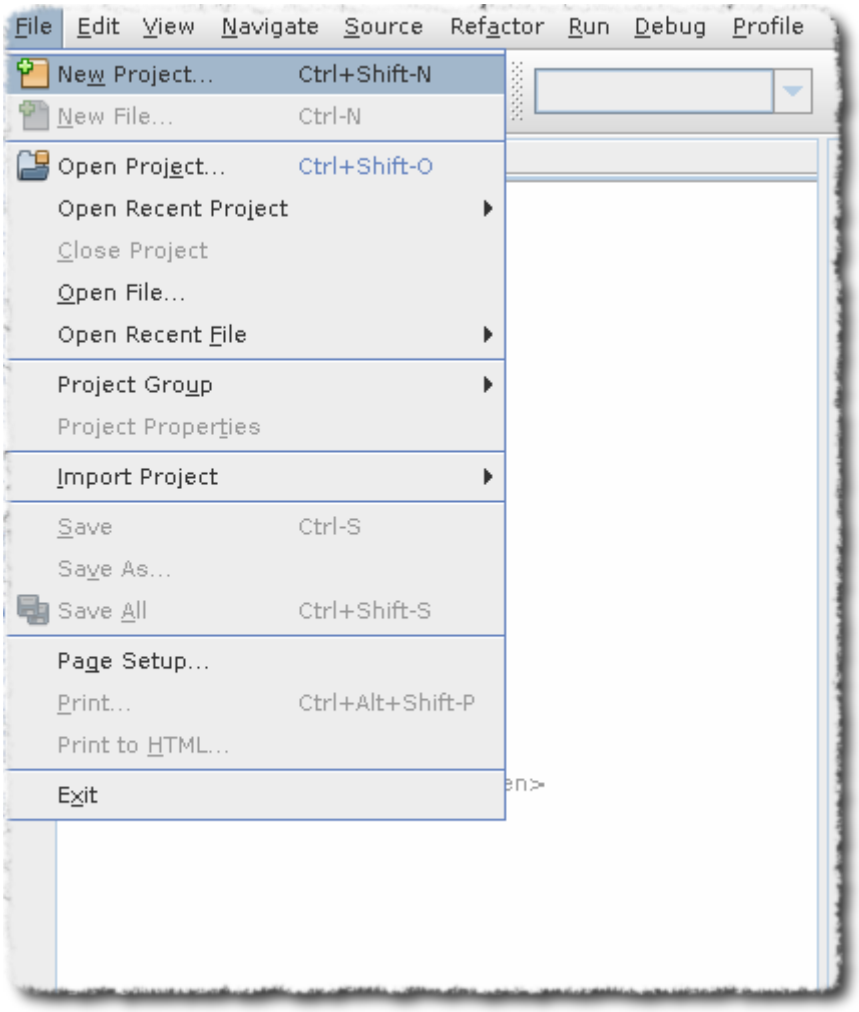


Once this is done, the NetBeans 7 will add the JAR files to the classpath whenever the newly-created DN3 and DN3Deps library is selected for a project.

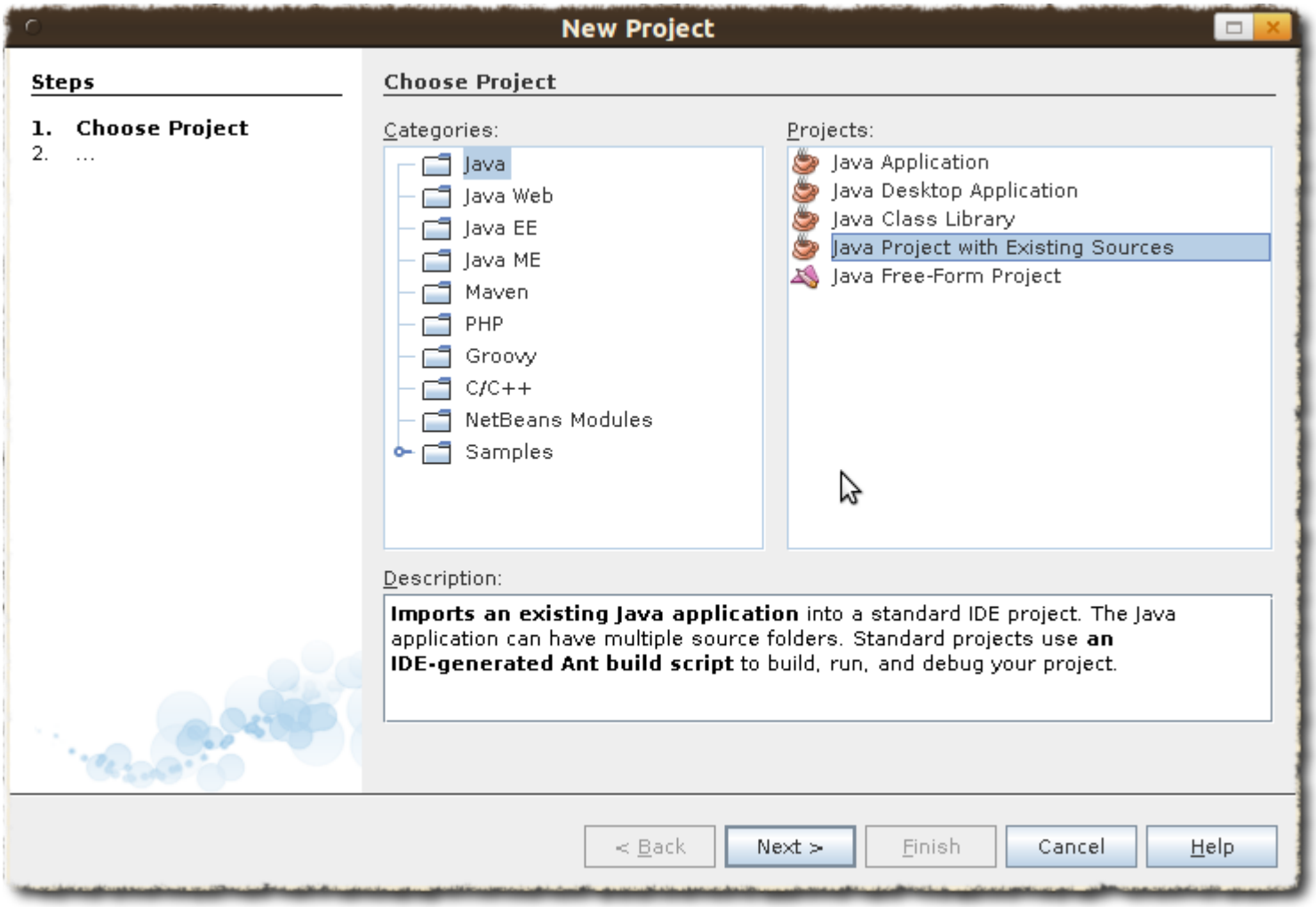
Ant : Setting up a new project

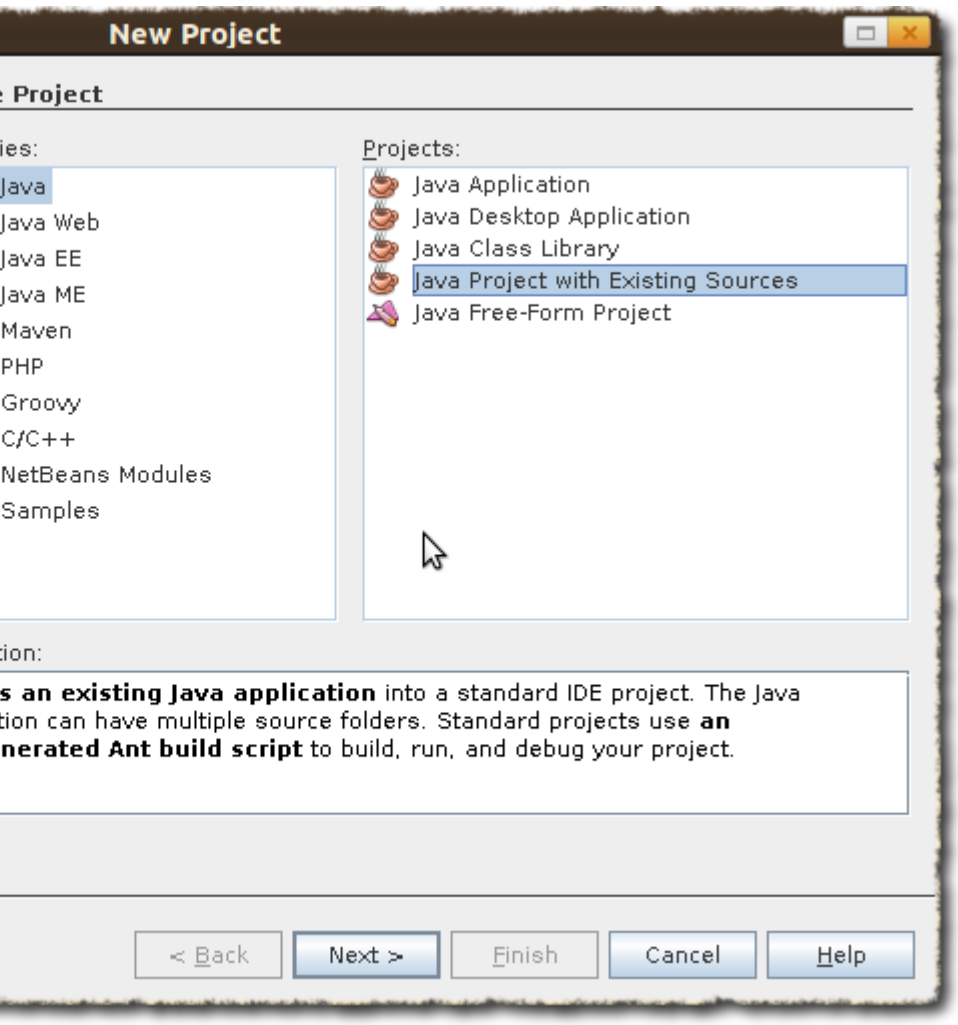
Delete `pom.xml` from the downloaded samples project. NetBeans treats any folder with `pom.xml` as a Maven project. Copy the `datanucleus.properties` to src/java in the downloaded samples project.

Now create a new project from existing sources.

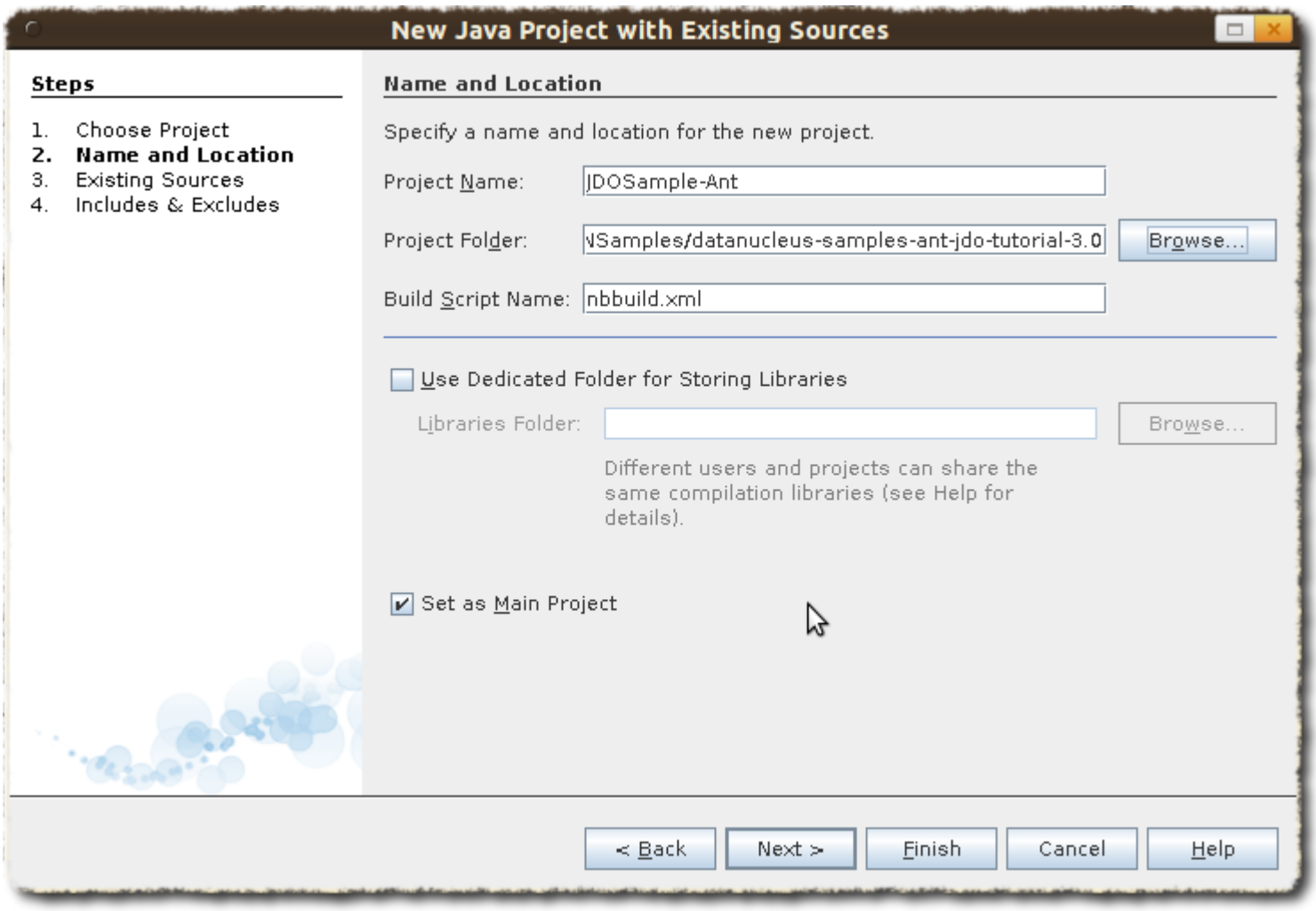


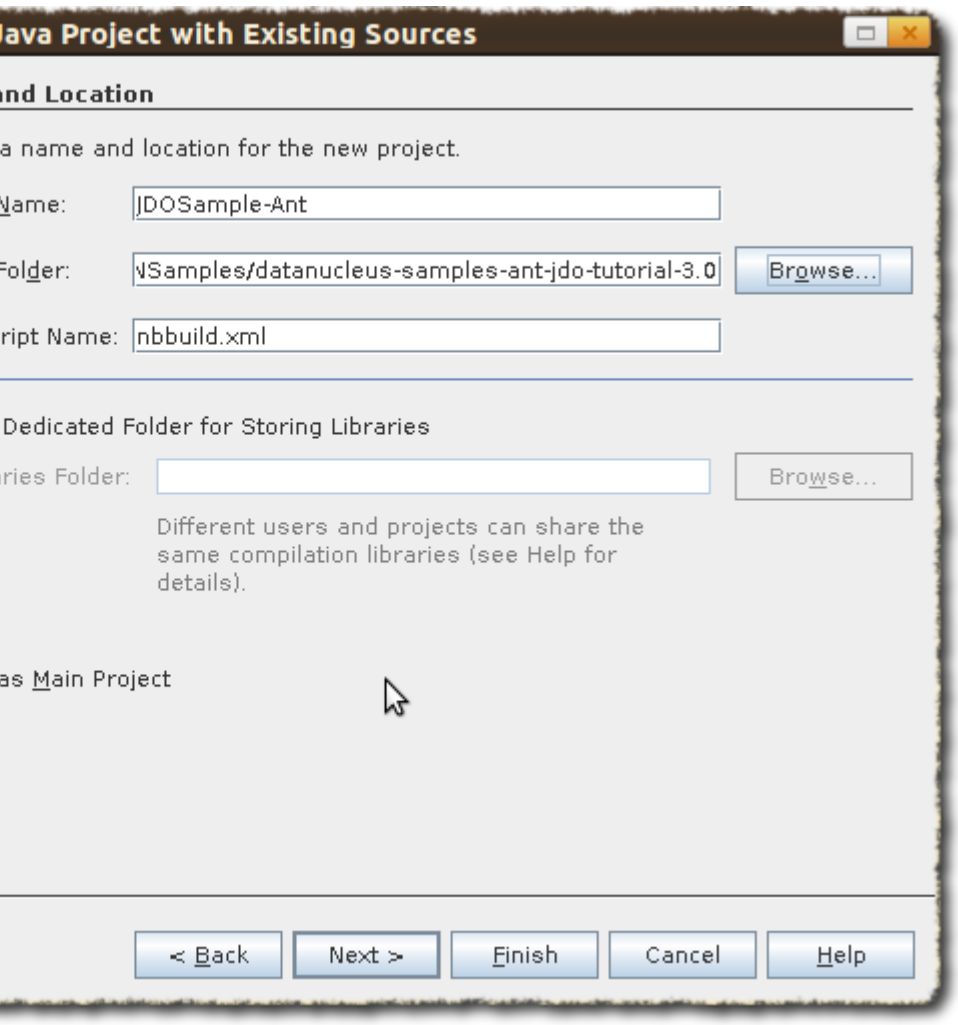
Remember to Select **Java Project with Existing Source**


New Project
Steps
1. Choose Project
2. ...
Choose Project
Categories:
Java
Java Web
Java EE
Java ME
Maven
PHP
Groovy
C/C++
NetBeans Modules
Samples
Projects:
Java Application
Java Desktop Application
Java Class Library
Java Project with Existing Sources
Java Free-Form Project
Description:
Imports an existing Java application into a standard IDE project. The Java application can have multiple source folders. Standard projects use an IDE-generated Ant build script to build, run, and debug your project.
< Back
Next >
Finish
Cancel
Help



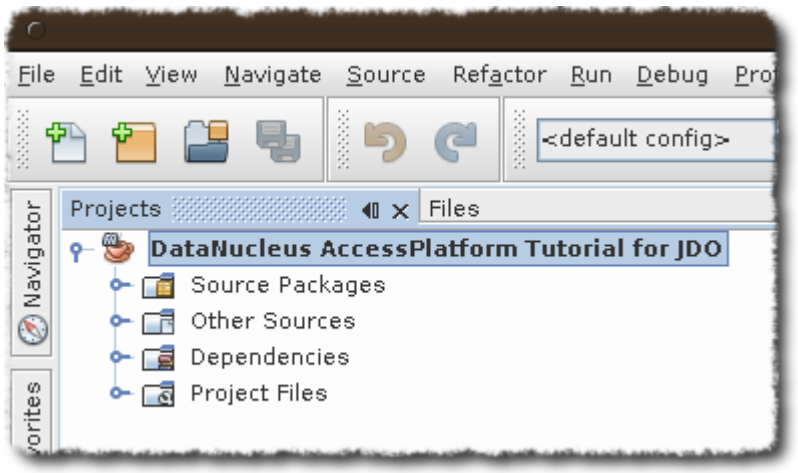
Click Next and Add Folder. The source folder should point to *datanucleus-samples-jdo-tutorial-3.0/src/java*


New Java Project with Existing Sources
Steps
1. Choose Project
2. Name and Location
3. Existing Sources
4. Includes & Excludes
Name and Location
Specify a name and location for the new project.
Project Name: JDOSample-Ant
Project Folder: NSamples/datanucleus-samples-ant-jdo-tutorial-3.0
Browse...
Build Script Name: nbbuild.xml
Use Dedicated Folder for Storing Libraries
Libraries Folder:
Browse...
Different users and projects can share the same compilation libraries (see Help for details).
Set as Main Project
< Back
Next >
Finish
Cancel
Help

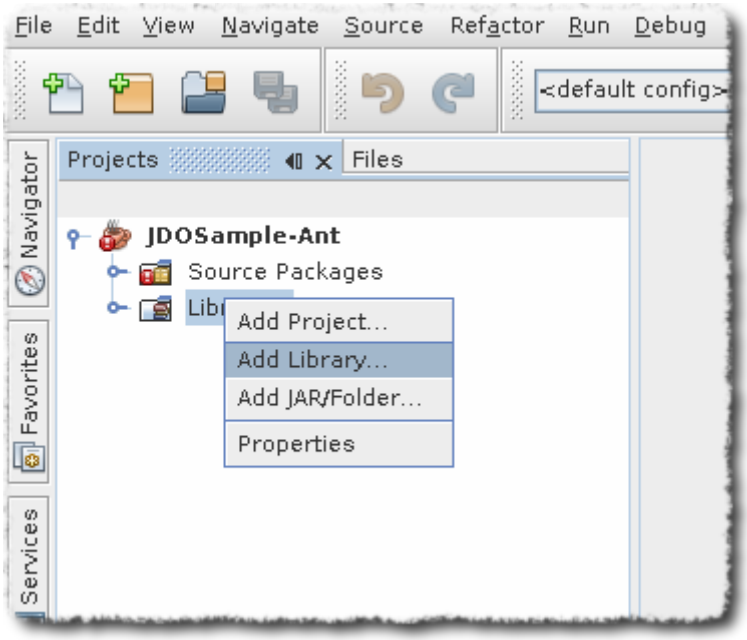


Then click Next and Finish

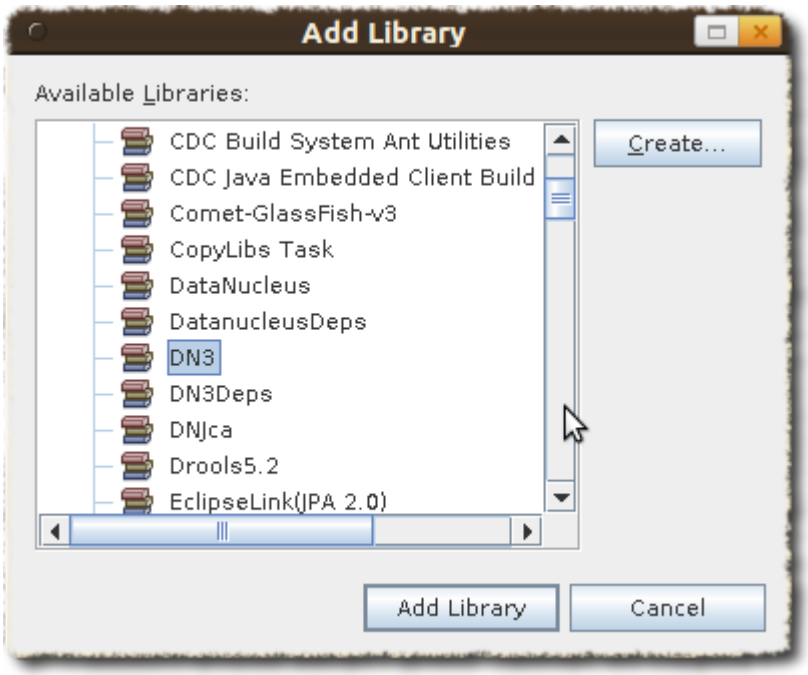
Now we have the sucessfuly created the sample project using Ant



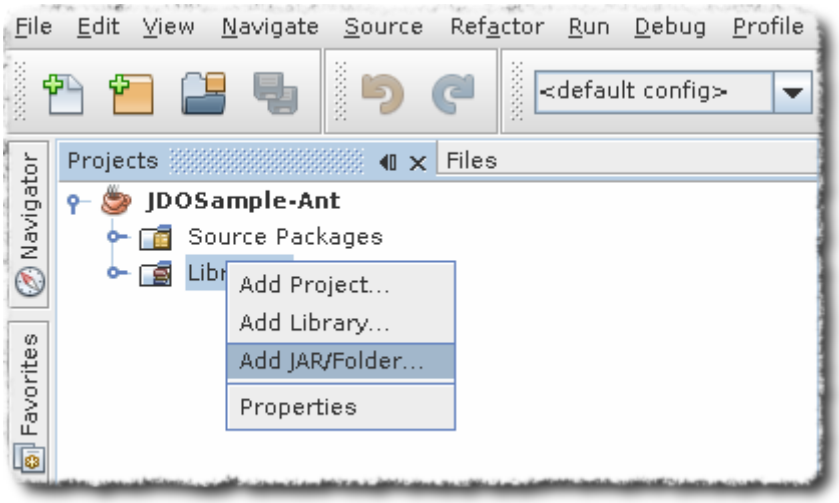
Add the Libraries created in first set to this project. Right-Click on Libraries in the Projects Tab



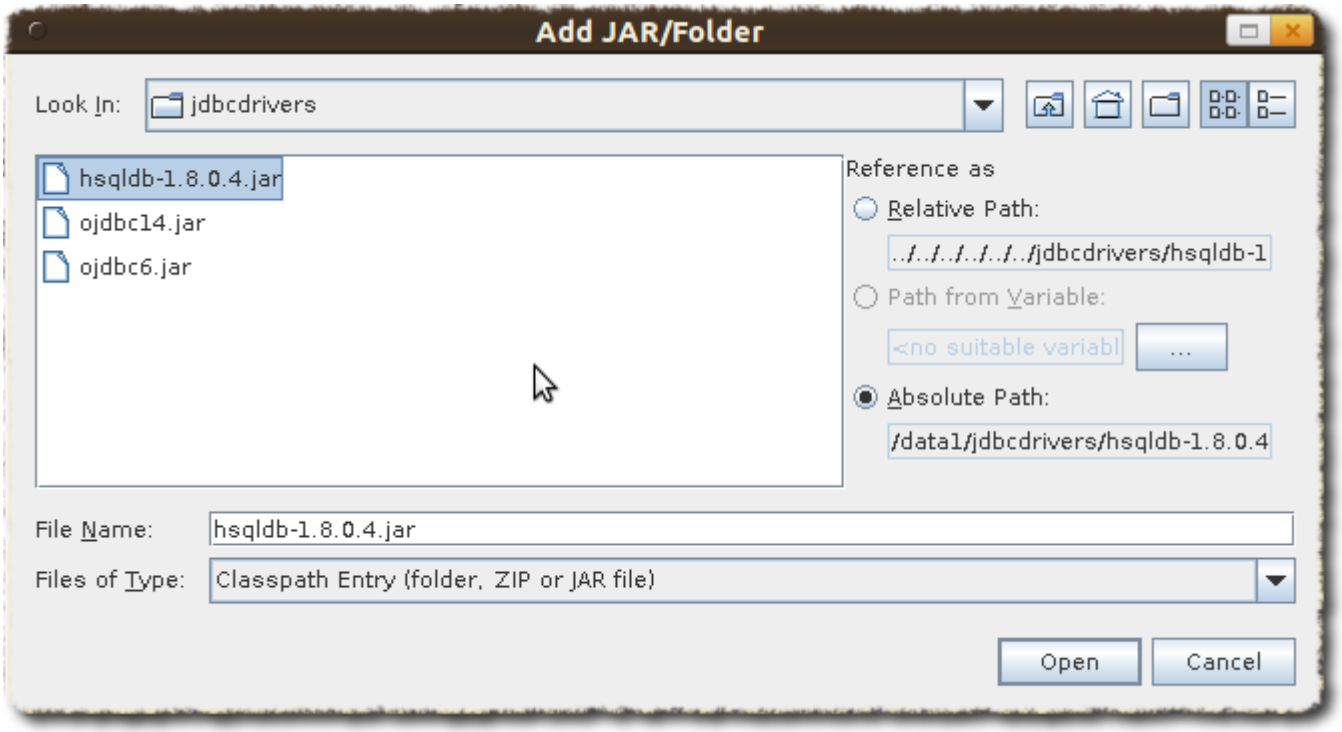
Add DN3 library to your project. Also add DN3Deps

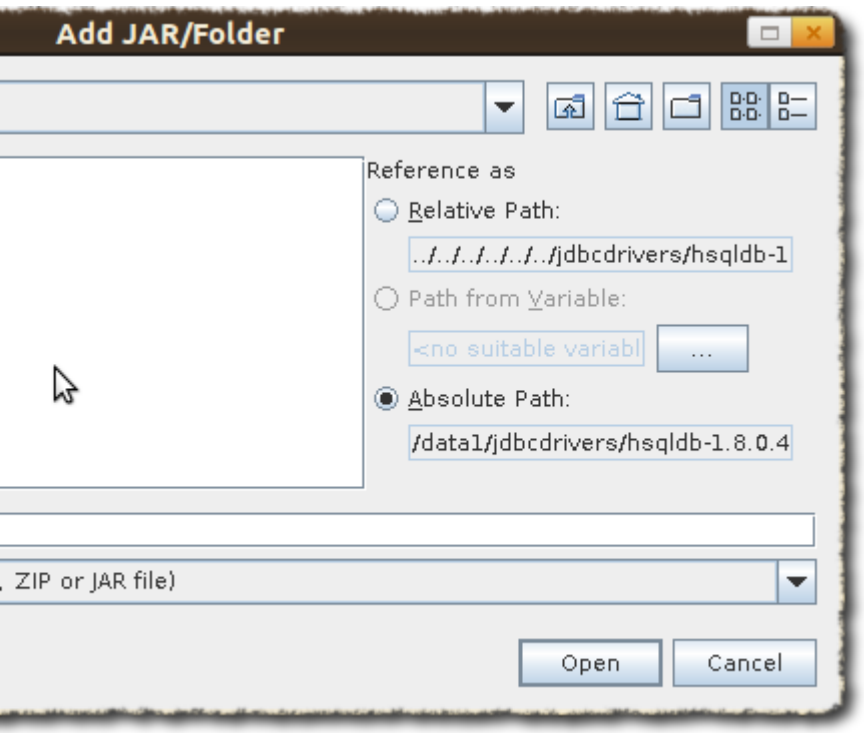


We also need to **add JDBC support to our project**. Add this using add "Add jars" option. Right-Click on Libraries in the Projects Tab and select "Add jars"


File Edit View Navigate Source Refactor Run Debug Profile
<default config>
Navigator
Favorites
Projects
Files
JDOSample-Ant
Source Packages
Add Project...
Add Library...
Add JAR/Folder...
Properties

Add hsqldb


Add JAR/Folder
Look In:
jdbcdrivers
hsqldb-1.8.0.4.jar
ojdbc14.jar
ojdbc6.jar
Reference as
Relative Path:
../../../../../../jdbcdrivers/hsqldb-1
Path from Variable:
<no suitable variabl
...
Absolute Path:
/data1/jdbcdrivers/hsqldb-1.8.0.4
File Name:
hsqldb-1.8.0.4.jar
Files of Type:
Classpath Entry (folder, ZIP or JAR file)
Open
Cancel

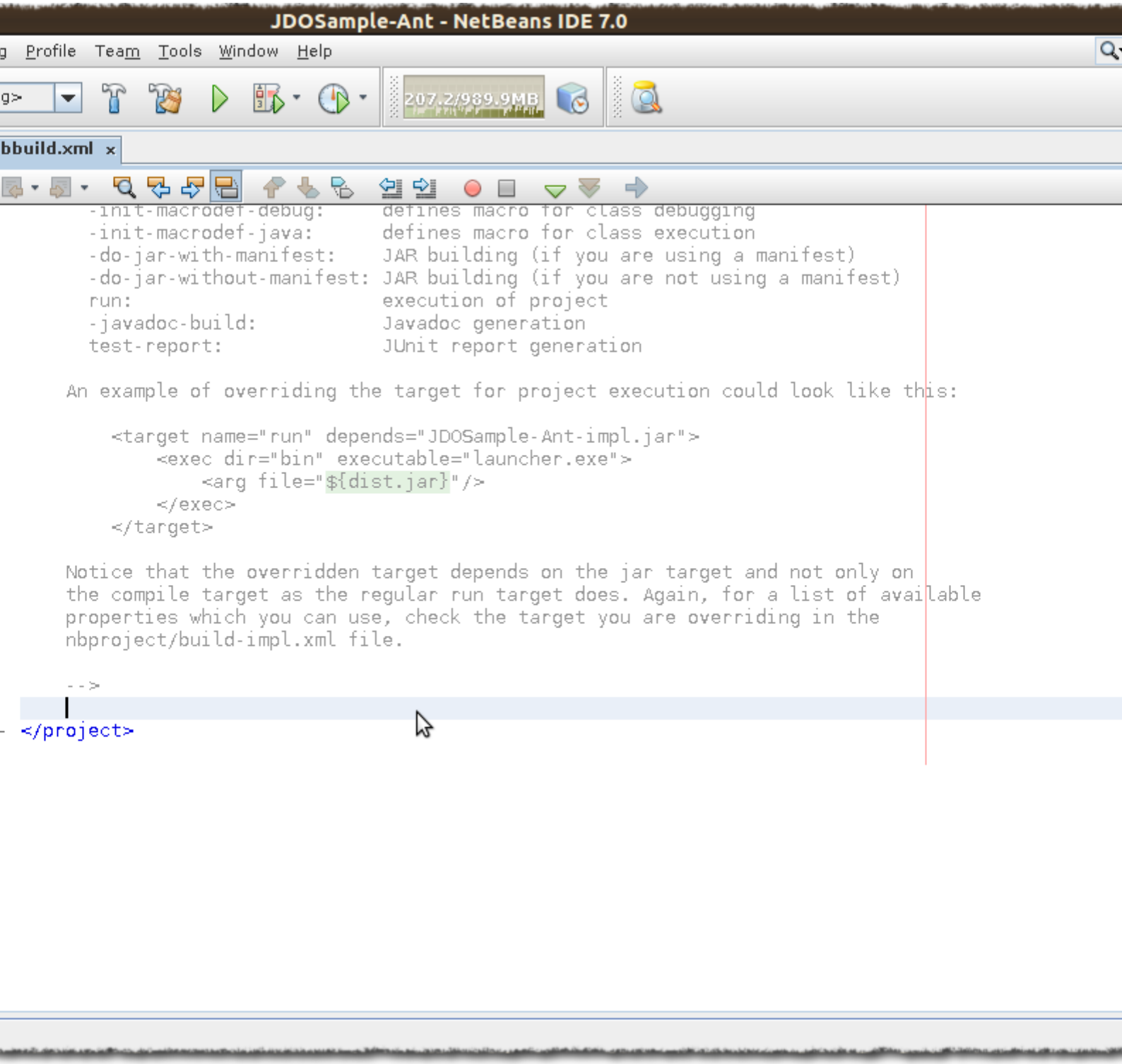


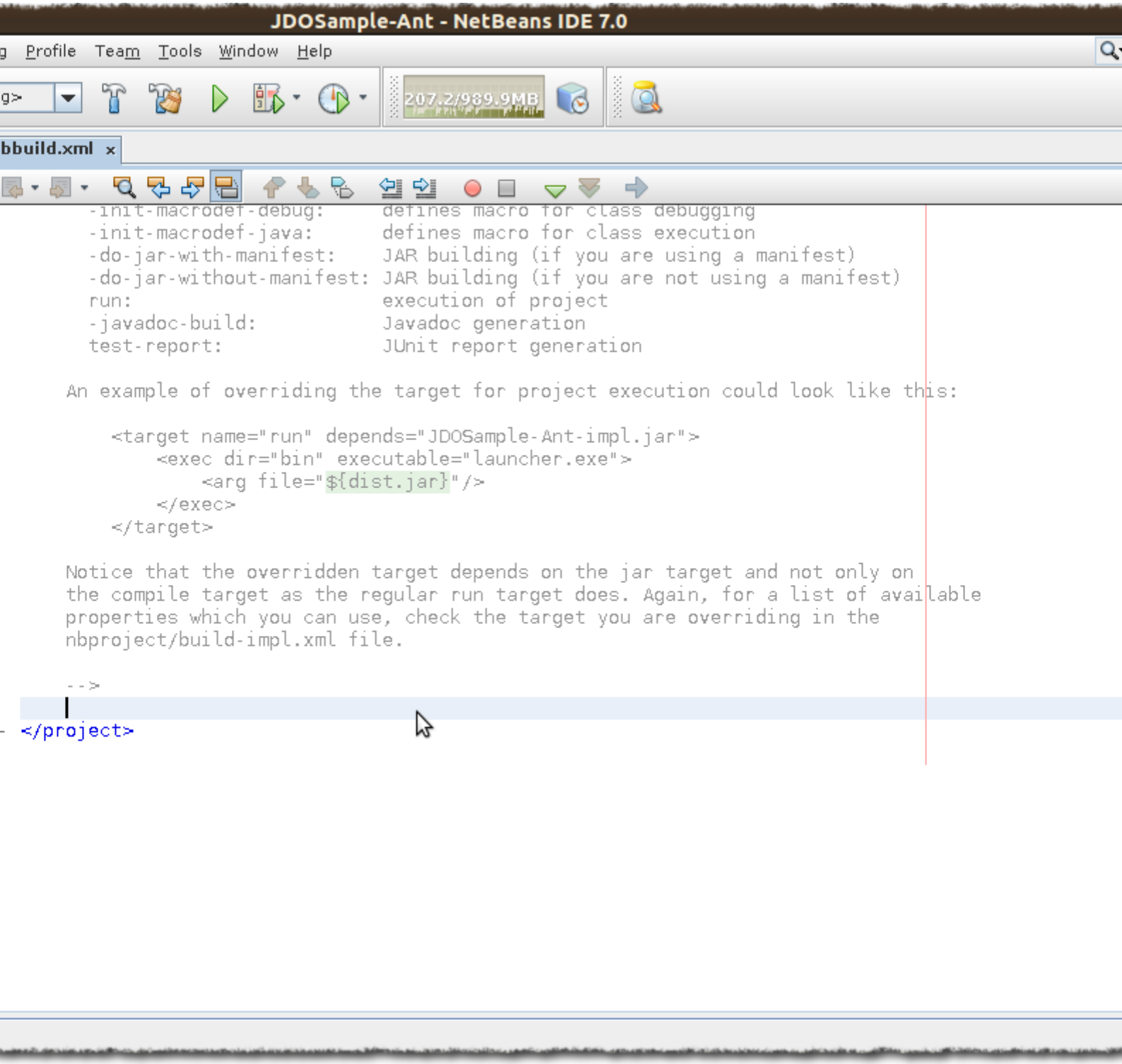
Ant : Enhancing the classes

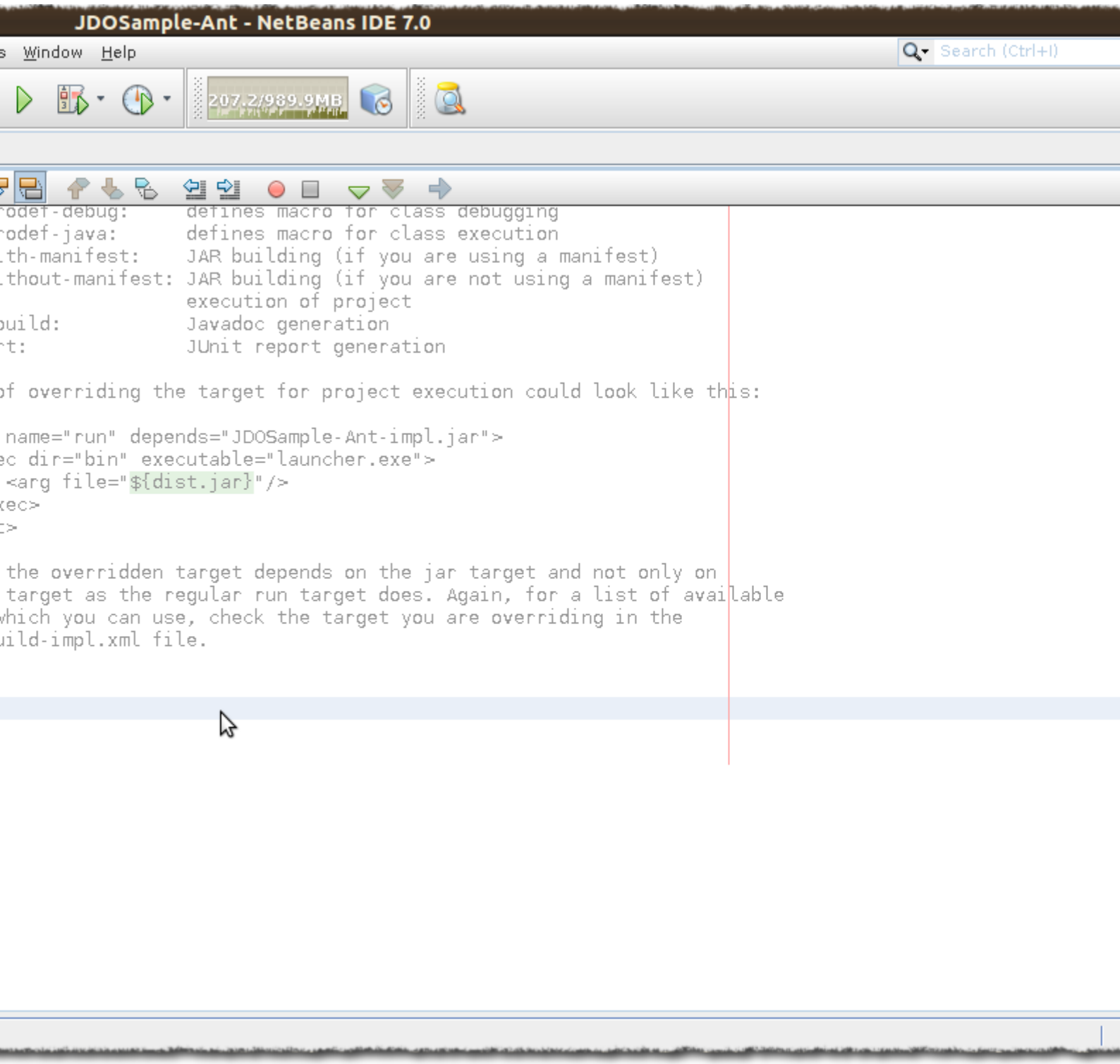
The enhancement process needs to be defined and integrated into the build process. As stated in the introduction, this requires a simple change to the `nbbuild.xml` file.

TypeSafe Queries :- Datanucleus generates additional code for supporting Criteria queries. Ensure that "Enable Annotation processing" check box is selected, which is under Project Properties | build | compiling

Click on the **Files** tab, expand the project tree, then open `nbbuild.xml`


JDOSample-Ant - NetBeans IDE 7.0
Profile Team Tools Window Help
207.2/989.9MB
bbuild.xml
-init-macrodef-debug: defines macro for class debugging
-init-macrodef-java: defines macro for class execution
-do-jar-with-manifest: JAR building (if you are using a manifest)
-do-jar-without-manifest: JAR building (if you are not using a manifest)
run: execution of project
-javadoc-build: Javadoc generation
test-report: JUnit report generation
An example of overriding the target for project execution could look like this:
<target name="run" depends="JDOSample-Ant-impl.jar">
<exec dir="bin" executable="launcher.exe">
<arg file="${dist.jar}"/>
</exec>
</target>
Notice that the overridden target depends on the jar target and not only on
the compile target as the regular run target does. Again, for a list of available
properties which you can use, check the target you are overriding in the
nbproject/build-impl.xml file.
-->
</project>


JDOSample-Ant - NetBeans IDE 7.0
Profile Team Tools Window Help
207.2/989.9MB
bbuild.xml
-init-macrodef-debug: defines macro for class debugging
-init-macrodef-java: defines macro for class execution
-do-jar-with-manifest: JAR building (if you are using a manifest)
-do-jar-without-manifest: JAR building (if you are not using a manifest)
run: execution of project
-javadoc-build: Javadoc generation
test-report: JUnit report generation
An example of overriding the target for project execution could look like this:
<target name="run" depends="JDOSample-Ant-impl.jar">
<exec dir="bin" executable="launcher.exe">
<arg file="${dist.jar}"/>
</exec>
</target>
Notice that the overridden target depends on the jar target and not only on
the compile target as the regular run target does. Again, for a list of available
properties which you can use, check the target you are overriding in the
nbproject/build-impl.xml file.
-->
</project>



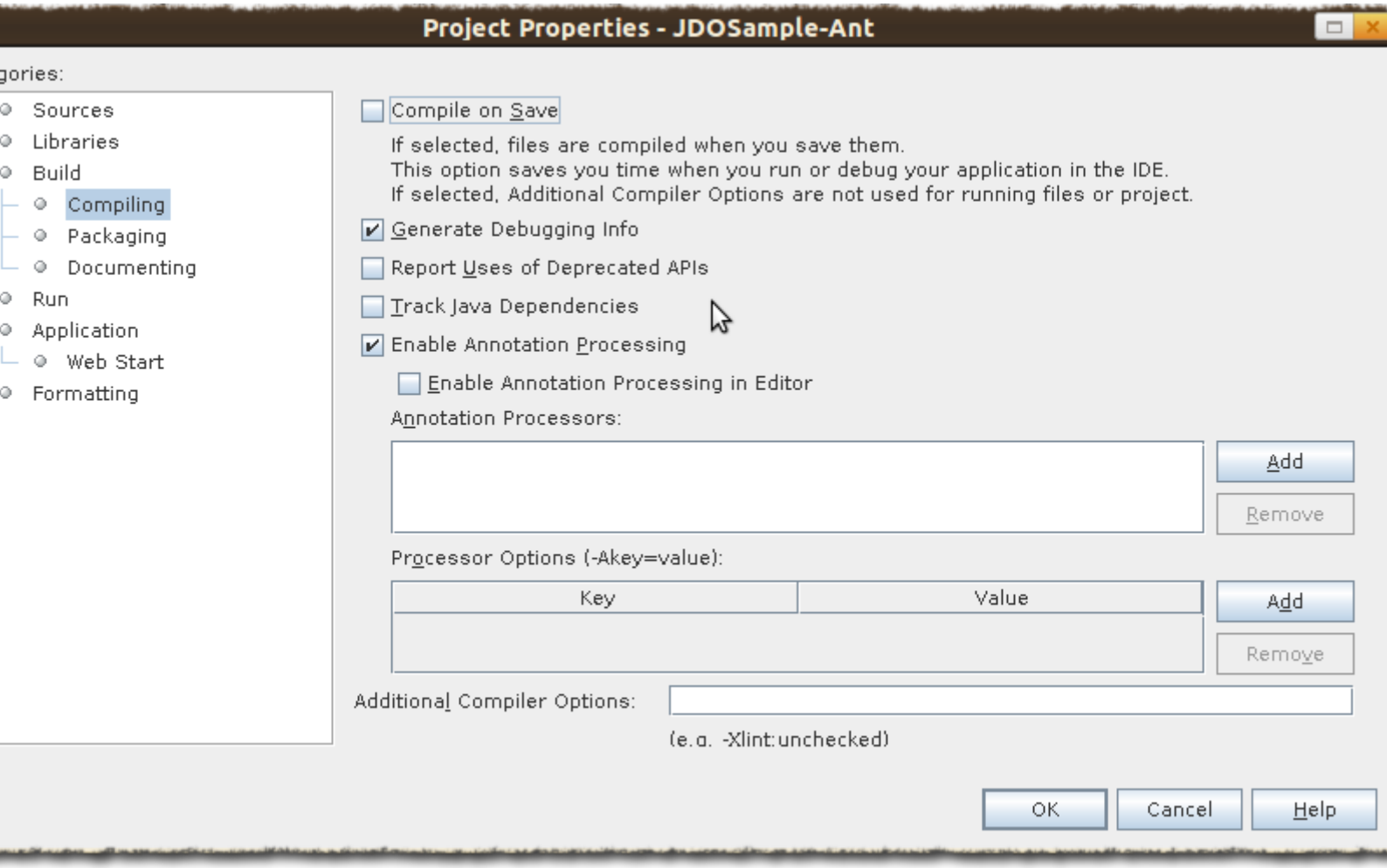
Override the * -post-compile * task/target with the following Ant instructions.

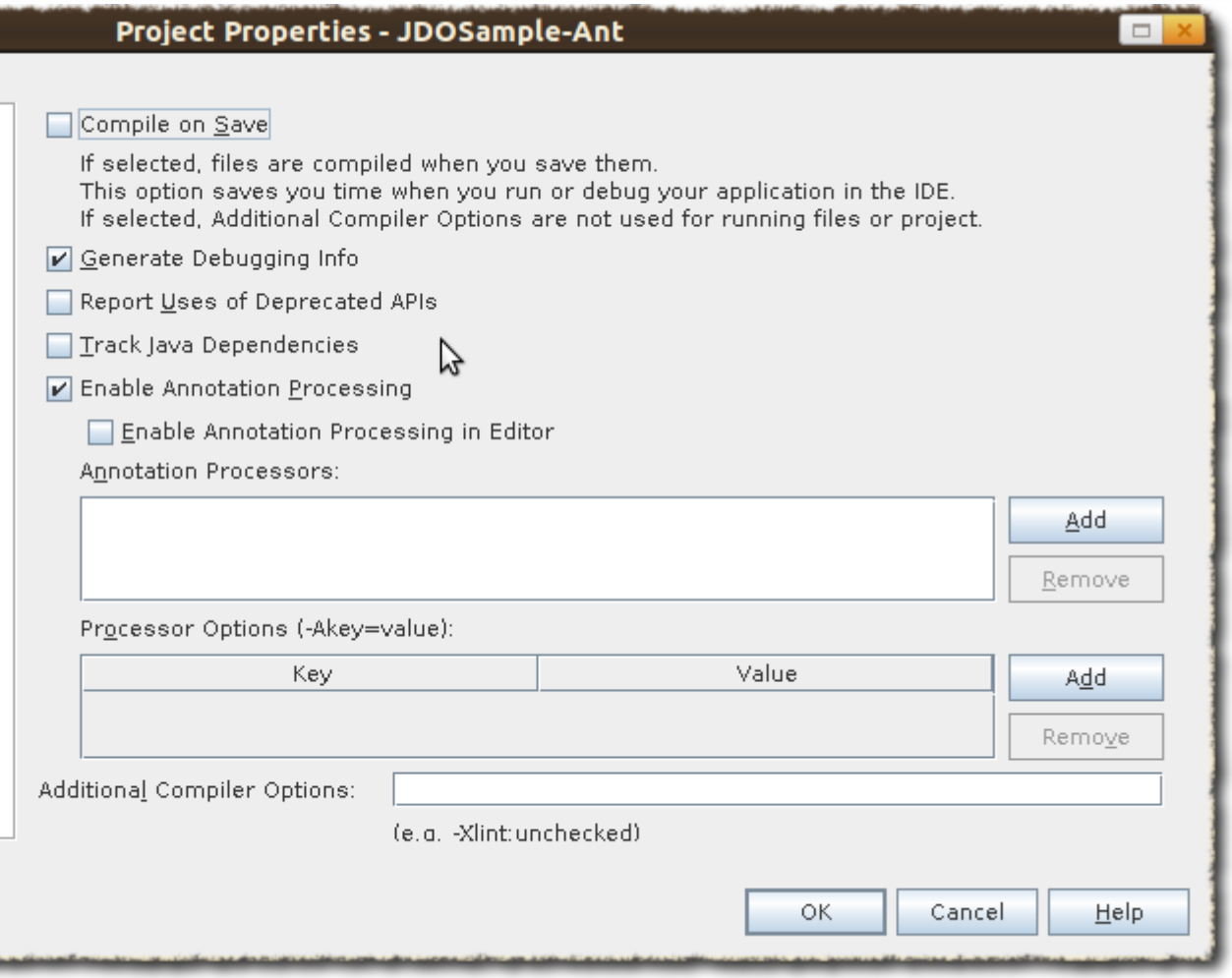
```
<target name="-post-compile" depends="init">
    <path id="module.enhancer.classpath">
        <pathelement path="${javac.classpath}"/>
        <pathelement location="${build.classes.dir}"/>
    </path>
    <taskdef name="datanucleusenhancer" classpathref="module.enhancer.classpath"
classname="org.datanucleus.enhancer.EnhancerTask"/>
    <echo message="start datanucleusenhancer"/>
    <datanucleusenhancer classpathref="module.enhancer.classpath"
dir="${build.classes.dir}" verbose="true">
        <fileset dir="${build.classes.dir}">
            <include name="**/*.class"/>
        </fileset>
    </datanucleusenhancer>
    <echo message="end datanucleusenhancer"/>
</target>
```

This target is the most convenient for enhancing classes because it occurs just after all classes have been compiled and is called in any case, whether the project is being built, tested or deployed. This ensures that classes are always enhanced.

Ensure that the Compile on Save option is turned off.

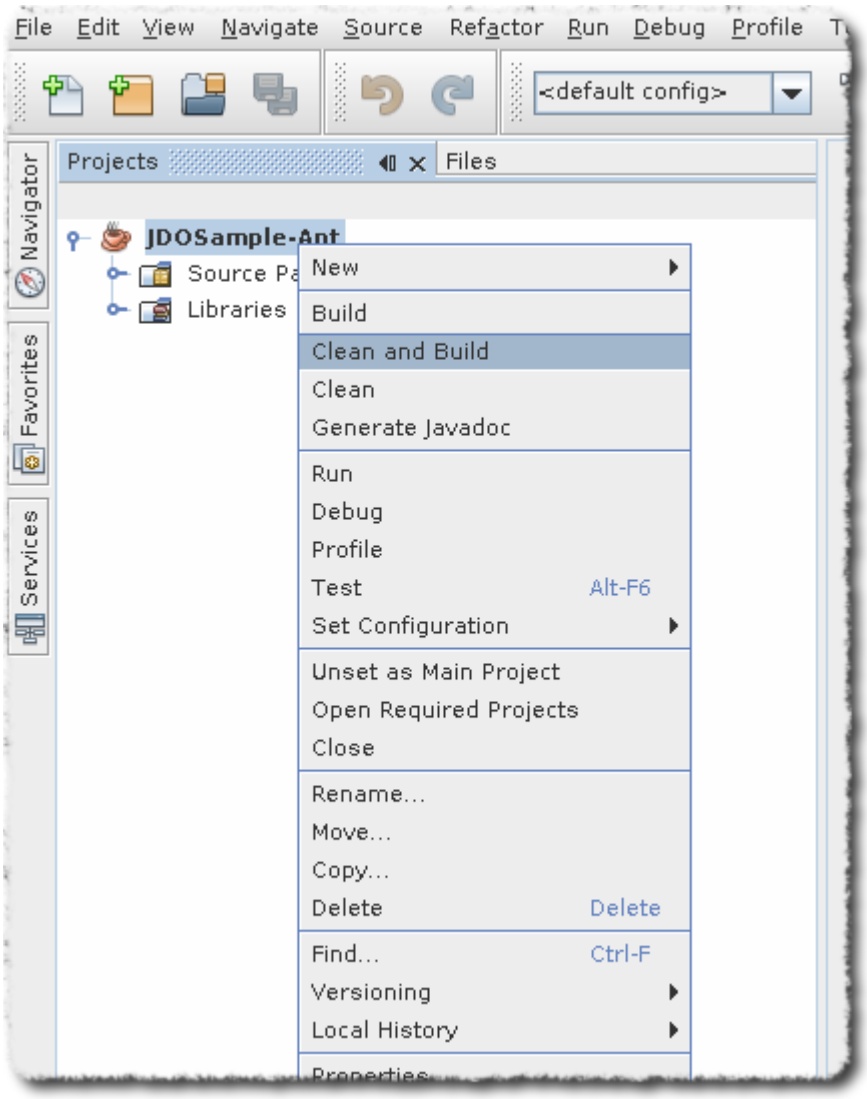
Enable Annotation Processing should be turned ON


Project Properties - JDOSample-Ant
gories:
Sources
Libraries
Build
Compiling
Packaging
Documenting
Run
Application
Web Start
Formatting
Compile on Save
If selected, files are compiled when you save them.
This option saves you time when you run or debug your application in the IDE.
If selected, Additional Compiler Options are not used for running files or project.
Generate Debugging Info
Report Uses of Deprecated APIs
Track Java Dependencies
Enable Annotation Processing
Enable Annotation Processing in Editor
Annotation Processors:
Add
Remove
Processor Options (-Akey=value):
Key
Value
Add
Remove
Additional Compiler Options:
(e.g. -Xlint:unchecked)
OK
Cancel
Help


Project Properties - JDOSample-Ant
Compile on Save
If selected, files are compiled when you save them.
This option saves you time when you run or debug your application in the IDE.
If selected, Additional Compiler Options are not used for running files or project.
Generate Debugging Info
Report Uses of Deprecated APIs
Track Java Dependencies
Enable Annotation Processing
Enable Annotation Processing in Editor
Annotation Processors:
Add
Remove
Processor Options (-Akey=value):
Key
Value
Add
Remove
Additional Compiler Options:
(e.g. -Xlint:unchecked)
OK
Cancel
Help

Ant : Building the project

The project can now be built, with the knowledge that the classes will be enhanced in the process.



Output window will show some thing similar to this

```
BUILD SUCCESSFUL (total time: 2 seconds)
```

Run the main class. The output window will show as below

```
Product and Book have been persisted

Retrieving Extent for Products
>> Book : JRR Tolkien - Lord of the Rings by Tolkien
>> Product : Sony Discman [A standard discman from Sony]

Executing Query for Products with price below 150.00
>> Book : JRR Tolkien - Lord of the Rings by Tolkien

Deleting all products from persistence
Deleted 2 products

End of Tutorial
BUILD SUCCESSFUL (total time: 2 seconds)
```

Conclusion

This concludes our tutorial on how to integrate DataNucleus with NetBeans 7. As can be seen, thanks to NetBeans project system based on Ant and Native Maven support, development of JDO applications is largely simplified. This tutorial was provided by a user of this software, Kiran Kumar.

Gradle Plugin

Gradle is a build tool that is used in some organisations. DataNucleus does not currently have an official plugin for using Gradle to build DataNucleus projects. If you develop such a tool, to perform enhancement, and SchemaTool operations then please contact us so that it can be distributed as part of the DataNucleus project.