JPA Enhancement Guide (v5.0)

Table of Contents

DataNucleus requires that all JPA entities implement Persistable and Detachable. Rather than requiring that a user add this themself, we provide an enhancer that will modify your compiled classes to implement all required methods. This is provided in *datanucleus-core.jar*.

- The use of this interface means that you get **transparent persistence**, and your classes always remain *your* classes; ORM tools that use a mix of reflection and/or proxies are not totally transparent.
- DataNucleus' use of *Persistable* provides transparent change tracking. When any change is made to an object the change creates a notification to DataNucleus allowing it to be optimally persisted. ORM tools that dont have access to such change tracking have to use reflection to detect changes. The performance of this process will break down as soon as you read a large number of objects, but modify just a handful, with these tools having to compare all object states for modification at transaction commit time.
- OpenJPA requires a similar bytecode enhancement process also, and EclipseLink and Hibernate both allow it as an option since they also now see the benefits of this approach over use of proxies and reflection.

In the DataNucleus bytecode enhancement contract there are 3 categories of classes. These are *Entity*, *PersistenceAware* and normal classes. The Meta-Data (XML or annotations) defines which classes fit into these categories. To give an example, we have 3 classes. Class *A* is to be persisted in the datastore, class *B* directly updates the fields of class *A* but doesn't need persisting, and class *C* is not involved in the persistence process. We would define these classes as follows

```
@Entity
public class A
{
    String myField;
    ...
}

@org.datanucleus.api.jpa.annotations.PersistenceAware
public class B
{
    ...
}

public class C {...}
```

So our MetaData is mainly for those classes that are *Entity* (or

MappedSuperclass/Embeddable) and are to be persisted to the datastore. For *PersistenceAware* classes we simply notate that the class knows about persistence. We don’t define MetaData for any class that has no knowledge of persistence.

You can read more about the precise details of the bytecode enhancement contract later in this section.

The enhancement process is very quick and easy.

> You **cannot** enhance classes that are in a JAR/WAR file. They must be unpacked, enhanced and then repacked.

> If the MetaData is changed in any way during development, the classes should always be recompiled and re-enhanced afterwards.

How to use the DataNucleus Enhancer depends on what environment you are using. Below are some typical examples.

- Post-compilation
  - Using Maven via the DataNucleus Maven plugin
  - Using Ant
  - Manual invocation at the command line
  - Using the Eclipse DataNucleus plugin
- At runtime
  - Runtime Enhancement
  - Programmatically via an API

Maven

Maven operates from a series of plugins. There is a DataNucleus plugin for Maven that allows enhancement of classes. Go to the Download section of the website and download this. Once you have the Maven plugin, you then need to set any properties for the plugin in your `pom.xml` file. Some properties that you may need to change are below

Property	Default	Description
persistenceUnitName		Name of the persistence-unit to enhance. **Mandatory**
metadataDirectory	${project.build.outputDirectory}	Directory to use for enhancement files (classes/mappings). For example, you could set this to ${project.build.testOutputDirectory} when enhancing Maven test classes
metadataIncludes	/**.jdo, /**.class	Fileset to include for enhancement (if not using persistence-unit)
metadataExcludes		Fileset to exclude for enhancement (if not using persistence-unit)
log4jConfiguration		Config file location for Log4J (if using it)
jdkLogConfiguration		Config file location for JDK1.4 logging (if using it)
api	JDO	API to enhance to (JDO, JPA). **Mandatory : Set this to JPA**
verbose	false	Verbose output?
quiet	false	No output?
targetDirectory		Where the enhanced classes are written (default is to overwrite them)
fork	true	Whether to fork the enhancer process (e.g if you get a command line too long with Windows).
generatePK	true	Generate a PK class (of name {MyClass}_PK) for cases where there are multiple PK fields yet no PK class is defined.
generateConstructor	true	Generate a default constructor if not defined for the class being enhanced.
detachListener	false	Whether to enhance classes to make use of a detach listener for attempts to access an undetached field.
ignoreMetaDataForMissingClasses	false	Whether to ignore when we have metadata specified for classes that aren't found

You will need to add `datanucleus-core.jar` and `datanucleus-api-jpa.jar` into the CLASSPATH (of the plugin, or your project) for the enhancer to operate. Similarly *javax.persistence* (but then you almost certainly will have that in your project CLASSPATH anyway).

You then run the Maven DataNucleus plugin, as follows

```
mvn datanucleus:enhance
```

This will enhance all classes for the specified persistence-unit. If you want to check the current status of enhancement you can also type

```
mvn datanucleus:enhance-check
```

Or alternatively, you could add the following to your POM

```
<build>
    ...
    <plugins>
        <plugin>
            <groupId>org.datanucleus</groupId>
            <artifactId>datanucleus-maven-plugin</artifactId>
            <version>5.0.2</version>
            <configuration>
                <api>JPA</api>
                <persistenceUnitName>MyUnit</persistenceUnitName>
                <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>
```

So you then get auto-enhancement after each compile. Please refer to the Maven JPA guide for more details.

Ant

Ant provides a powerful framework for performing tasks, and DataNucleus provides an Ant task to enhance classes. You need to make sure that the `datanucleus-core.jar`, `datanucleus-api-jpa.jar`, `log4j.jar` (optional), and `javax.persistence.jar` are in your CLASSPATH. If using JDO metadata then you will also need `javax.jdo.jar` and `datanucleus-api-jdo.jar` in the CLASSPATH. In the DataNucleus Enhancer Ant task, the following parameters are available

Parameter	Description	values
destination	Optional. Defining a directory where enhanced classes will be written. If omitted, the original classes are updated.	
api	Defines the API to be used when enhancing	Set this to **JPA**
persistenceUnit	Defines the "persistence-unit" to enhance.	
checkonly	Whether to just check the classes for enhancement status. Will respond for each class with "ENHANCED" or "NOT ENHANCED". **This will disable the enhancement process and just perform these checks.**	true, **false**
verbose	Whether to have verbose output.	true, **false**
quiet	Whether to have no output.	true, **false**
generatePK	Whether to generate PK classes as required.	**true**, false
generateConstructor	Whether to generate a default constructor as required.	**true**, false
if	Optional. The name of a property that must be set in order to the Enhancer Ant Task to execute.	
ignoreMetaDataForMissingClasses	Optional. Whether to ignore when we have metadata specified for classes that aren't found	

The enhancer task extends the Apache Ant Java task, thus all parameters available to the Java task are also available to the enhancer task.

So you could define something *like* the following, setting up the parameter **enhancer.classpath**, and **log4j.config.file** to suit your situation.

```xml
<target name="enhance" description="DataNucleus enhancement">
    <taskdef name="datanucleusenhancer" classpathref="enhancer.classpath"
classname="org.datanucleus.enhancer.EnhancerTask" />
    <datanucleusenhancer persistenceUnit="MyUnit" failonerror="true" verbose="true">
        <jvmarg line="-Dlog4j.configuration=${log4j.config.file}"/>
        <classpath>
            <path refid="enhancer.classpath"/>
        </classpath>
    </datanucleusenhancer>
</target>
```

Manually

If you are building your application manually and want to enhance your classes you follow the instructions in this section. You invoke the enhancer as follows

```
java -cp classpath  org.datanucleus.enhancer.DataNucleusEnhancer [options]
    where options can be
        -pu {persistence-unit-name} : Name of a "persistence-unit" to enhance the
classes for
        -d {target-dir-name} : Write the enhanced classes to the specified directory
        -api {api-name} : Name of the API we are enhancing for (JDO, JPA). Set this to
JPA
        -checkonly : Just check the classes for enhancement status
        -v : verbose output
        -q : quiet mode (no output, overrides verbose flag too)
        -generatePK {flag} : generate any PK classes where needed ({flag} should be
true or false - default=true)
        -generateConstructor {flag} : generate default constructor where needed
({flag} should be true or false - default=true)
        -ignoreMetaDataForMissingClasses : ignore classes that have defined metadata
but are missing

    where "mapping-files" and "class-files" are provided when not enhancing a
persistence-unit,
        and give the paths to the mapping files and class-files that define the
classes being enhanced.

    where classpath must contain the following
        `datanucleus-core.jar`
        `datanucleus-api-jpa.jar`
        `javax.persistence.jar`
        `log4j.jar` (optional)
        your classes
        your meta-data files
```

The input to the enhancer should be the name of the "persistence-unit" to enhance. To give an example of how you would invoke the enhancer

```
# Linux/Unix :
java -cp target/classes:lib/datanucleus-core.jar:lib/datanucleus-api-
jpa.jar:lib/javax.persistence.jar:lib/log4j.jar
     -Dlog4j.configuration=file:log4j.properties
     org.datanucleus.enhancer.DataNucleusEnhancer -api JPA -pu MyUnit

# Windows :
java -cp target\classes;lib\datanucleus-core.jar;lib\datanucleus-api-jpa.jar;lib
\javax.persistence.jar;lib\log4j.jar
     -Dlog4j.configuration=file:log4j.properties
     org.datanucleus.enhancer.DataNucleusEnhancer -api JPA -pu MyUnit

# [should all be on same line. Shown like this for clarity]
```

So you pass in the persistence-unit name as the final argument(s) in the list, and include the respective JAR's in the classpath (-cp). The enhancer responds as follows

```
DataNucleus Enhancer (version 5.0.2) for API "JPA"

DataNucleus Enhancer : Classpath
>>  /home/andy/work/myproject//target/classes
>>  /home/andy/work/myproject/lib/log4j.jar
>>  /home/andy/work/myproject/lib/javax.persistence.jar
>>  /home/andy/work/myproject/lib/datanucleus-core.jar
>>  /home/andy/work/myproject/lib/datanucleus-api-jpa.jar

ENHANCED (persistable): org.mydomain.mypackage1.Pack
ENHANCED (persistable): org.mydomain.mypackage1.Card
DataNucleus Enhancer completed with success for 2 classes. Timings : input=422 ms,
enhance=490 ms, total=912 ms.
     ... Consult the log for full details
```

If you have errors here relating to "Log4J" then you must fix these first. If you receive no output about which class was ENHANCED then you should look in the DataNucleus enhancer log for errors. The enhancer performs much error checking on the validity of the passed MetaData and the majority of errors are caught at this point. You can also use the DataNucleus Enhancer to check whether classes are enhanced. To invoke the enhancer in this mode you specify the **checkonly** flag. This will return a list of the classes, stating whether each class is enhanced for persistence under JPA or not. The classes need to be in the CLASSPATH

A CLASSPATH should contain a set of JAR's, and a set of directories. It should NOT explictly include class files, and should NOT include parts of the package names. If in doubt please consult a Java book)

Runtime Enhancement

When operating in a JavaEE environment (JBoss, WebSphere, etc) instead set the persistence property datanucleus.jpa.addClassTransformer to *true*. Note that this is only for a real JavaEE server that implements the JavaEE parts of the JPA spec.

To enable runtime enhancement in other environments, the *javaagent* option must be set in the java command line. For example,

```
java -javaagent:datanucleus-core.jar=-api=JPA Main
```

The statement above will mean that all classes, when being loaded, will be processed by the ClassFileTransformer (except class in packages "java.**", "javax.**", "org.datanucleus.*"). This means that it can be slow since the MetaData search algorithm will be utilised for each. To speed this up you can specify an argument to that command specifying the names of package(s) that should be processed (and all others will be ignored). Like this

```
java -javaagent:datanucleus-core.jar=-api=JPA,mydomain.mypackage1,mydomain.mypackage2
Main
```

so in this case only classes being loaded that are in *mydomain.mypackage1* and *mydomain.mypackage2* will be attempted to be enhanced.

Please take care over the following when using runtime enhancement

- When you have a class with a field of another entity type make sure that you mark the field with the relation annotation (@OneToOne, @OneToMany, @ManyToOne, @ManyToMany etc) since with runtime enhancement at that point the related class is likely not yet enhanced so will likely not be marked as persistent otherwise. **Be explicit**
- If the agent jar is not found make sure it is specified with an absolute path.

Programmatic API

You could alternatively programmatively enhance classes from within your application.

```
import org.datanucleus.enhancer.DataNucleusEnhancer;

DataNucleusEnhancer enhancer = new DataNucleusEnhancer("JPA", null);
enhancer.setVerbose(true);
enhancer.addPersistenceUnit("MyPersistenceUnit");
enhancer.enhance();
```

This will look in META-INF/persistence.xml and enhance all classes defined by that unit. **Please note that you will need to load the enhanced version of the class into a different ClassLoader after performing this operation to use them**.

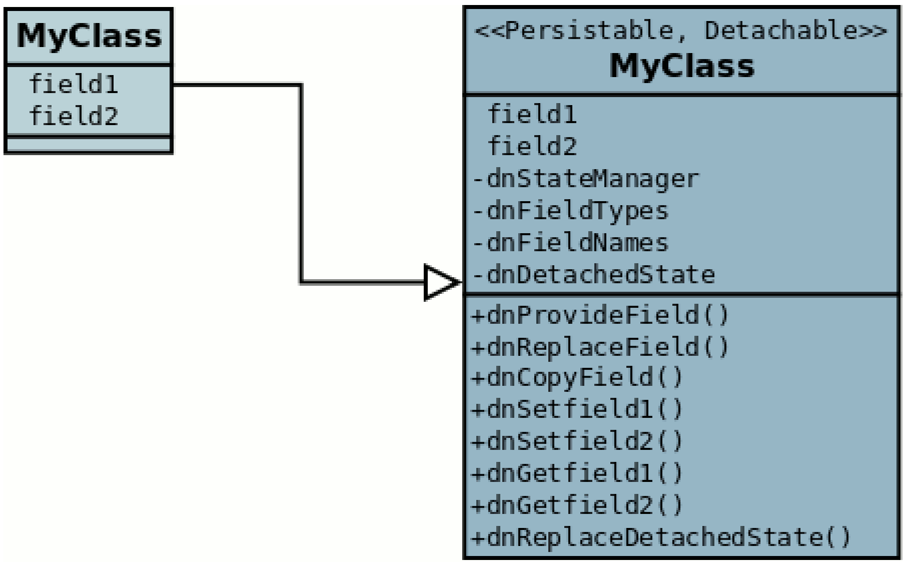
Enhancement Contract Details

Persistable

JPA allows implementations to bytecode-enhance persistable classes to implement some interface to provide them with change tracking etc. DataNucleus provides its own byte-code enhancer (in the *datanucleus-core.jar*) to enhance users entity classes to implement this *Persistable* interface. If we start off with the following class

```
@Entity
public class MyClass
{
    String field1;
    String field2;
    ...
}
```

This is bytecode enhanced for JPA to implement Persistable and Detachable.



The example above doesn't show all *Persistable* methods, but demonstrates that all added methods and fields are prefixed with "dn" to distinguish them from the users own methods and fields. Also each persistent field of the class will be given a dnGetXXX, dnSetXXX method so that accesses of these fields are intercepted so that DataNucleus can manage their "dirty" state. Regarding the *Detachable* interface, the main thing to know is that the detached state (object id of the datastore object, the version of the datastore object when it was detached, and which fields were detached is stored in "dnDetachedState") is stored in the object when it is detached, and available to be merged later on.

Byte-Code Enhancement Myths

Some groups (e.g Hibernate) in the past perpetuated arguments against "byte-code enhancement" saying that it was somehow 'evil'. The most common were :-

- *Slows down the code-test cycle*. This is erroneous since you only need to enhance just before test and the provided tools for Ant, Eclipse and Maven all do the enhancement job automatically and rapidly.
- *Is less "lazy" than the proxy approach since you have to load the object as soon as you get a pointer to it*. In a 1-1 relation you **have to load** the object then since you would cause issues with null pointers otherwise. With 1-N relations you load the elements of the collection/map only when you access them and not the collection/map. Hardly an issue then is it!
- *Fail to detect changes to public fields unless you enhance your client code*. Firstly very few people will be writing code with public fields since it is bad practice in an OO design, and secondly, this is why we have "PersistenceAware" classes.

So as you can see, there are no valid reasons against byte-code enhancement, and the pluses are that runtime detection of dirty events on objects is much quicker, hence your persistence layer operates faster without any need for iterative reflection-based checks. The fact is that Hibernate itself also now has a mode whereby you can do bytecode enhancement although not the default mode of Hibernate. So maybe it wasn’t so evil after all ?

Decompilation

Many people will wonder what actually happens to a class upon bytecode enhancement. In simple terms the necessary methods and fields are added so as to implement *Persistable* and *Detachable* as described above. If you want to check this, just use a Java decompiler such as JD. It has a nice GUI allowing you to just select your class to decompile and shows you the source.