MavenGroupIDChange

Changing Maven Group Id

DRAFT DRAFT DRAFT DRAFT DRAFT DRAFT DRAFT

Several Commons components are using Maven Group Ids other than *org.apache.commons*. This makes more work for Nexus and Maven Central maintenance.

This page is intended to collect together information on the issues to consider when changing a Maven Group Id.

Examples that follow assume that the Commons component Foo currently uses groupId=commons-foo and artifactId=commons-foo, i.e. commons-foo:commons-foo for short. The classes are currently in the org.apache.commons.foo Java package hierarchy.

Assume also that there is a Bar application which depends on two libraries, lib1 and lib2. lib1 depends on commons-foo:commons-foo:1 and lib2 depends on commons-foo:commons-foo:2.

Classpath considerations

The Java classpath can contain multiple versions of the same classname (in different jars or directories). However, at most one version of a given class can be loaded into a single class-loader.

So if a class is updated such that its API is no longer backwards-compatible, it's not possible to use both in the same classpath. The only solution is to change the class name, e.g. by changing its package name - or one could change just the classname. [In both of these cases the code could be made compatible again by keeping the original code alongside the new. However, at some point the old code will probably need to be deleted, at which time the versions will be incompatible anyway.]

If there are multiple versions of the same class on the classpath, there's no guarantee which version will be loaded, so in general the classpath should only ever contain a single version of each class.

What is jar-hell ?

Suppose project A depends on the jars B and C, and these in turn depend on jar D. So the classpath consists of (A,B,C,D). No problem so far.

D is updated to add some new features (call it D2), and jar B is updated to use them (B2). Provided that D2 is binary compatible with D, the a classpath which contains (A,B2,C,D2) will still allow jar C to work correctly.

Now suppose that D2 is not compatible with D, such that jar C does not operate with jar D2. The classpath would have to contain both D and D2. However B2 requires the class from D2, wheres C requires the version of the class from jar D. It's not possible to satisfy both of these. Even for compatible classes in D and D2, there is a problem, because there's no way to determine whether classes will be loaded from D or D2.

Jar-hell is when the classpath needs to contains multiple copies of the same classes in different jars.

Maven dependency resolution

Maven assumes that artifacts with the same GroupId and ArtifactId represent the same item, and ensures that only one instance of each such artifact is added to the classpath.

In the above example, only commons-foo:commons-foo:2 would be added to the classpath.

If the groupId is changed, then org.apache.commons:commons-foo will be treated as a different artifact, and the Maven classpath could potentially contain two copies of the same component.

This can cause a problem, for which there are several possible solutions - neither of which is ideal:

- use relocation POMs
- change the Foo package name

Relocation POMS

A relocation POM can be set up to redirect references from commons-foo:commons-foo to org.apache.commons:commons-foo. Both would be seen as being the same item, avoiding duplicates on the classpath.

This sounds ideal, but the relocation POMs may not always be processed??

TBA details of relocation poms

Change of package name

If the change from commons-foo:commons-foo to org.apache.commons:commons-foo is accompanied by a change to the Java package name, e.g. to org.apache.commons.foo3, then there will be no classpath issue, as both Maven and Java treat the artifacts as different.

There are two possible scenarios here

- The new version of the code is binary incompatible with the old version.
- The new version is binary compatible with the old version.

However, the change of Java package name is neither binary nor source-compatible, and can require a lot of work for users of Commons Foo. This may be acceptable if the new version has incompatible changes to the API, but not otherwise - why should users (who may not even use Maven) be forced to change their code just to upgrade to the latest version (James Carman: the user will thank us when they try to use a library that requires the older version, we shouldn't discount this too much. This approach solves the "jar hell" issue) (Sebb: there is no "jar hell" if the versions are binary compatible)?

For binary-compatible releases, the Java package name should NOT be changed, as that causes unnecessary work for all users. It follows that the Maven groupID should not be changed either, unless relocation POMs are guaranteed to work.

As a concrete example, Logging uses the groupId commons-logging. Changing the package name merely to allow the groupId to be changed would cause an awful lot of work, for almost no benefit.