BANNED dependencies

Your Rating: Results: 14 rates

Problem - using a parent pom prior to 20

You are trying to build Magnolia or a Magnolia project or module, and you get the following Maven error :

```
[INFO] Failed to resolve artifact.

Missing:
----------
1) commons-logging:commons-logging:jar:BANNED
  Path to dependency:
      1) info.magnolia:foobar:jar:1.0-SNAPSHOT
      2) <abc:abc>
      3) ...
      4) <xyz:xyz>
      5) commons-logging:commons-logging:jar:BANNED
```

What does this mean ?

Two things:

1. The library at (4) (`<xyz>`) has a dependency on commons-logging
2. Our parent poms manages the version of commons-logging in its `<dependencyManagement` section and forces it to an non-existing version called "`BANNED`".

Problem - using a parent pom version 20 or newer.

You get the following message:

```
We try to avoid having any of the following dependencies to end up in Magnolia projects:
* xml-apis : cause endless issues and conflicts with the xml apis already
  present in JDKs.
* commons-logging: Magnolia ships with SLF4J and its jcl-over-slf4j companion.
  For that reason, we need to avoid ending up with commons-logging in Magnolia
  projects. See http://slf4j.org/faq.html
* commons-beanutils:commons-beanutils-core: Magnolia ships with
  commons-beanutils:commons-beanutils, which contains all of
  commons-beanutils:commons-beanutils-core. Exclude commons-beanutils-core to
  avoid version conflicts.
Please add exclusions appropriately, or override the enforced-banned-dependencies rules.
```

Since parent poms version 20, we use the enforcer plugin in conjunction with the bannedDependencies rule for this.

What the hell ?

Why do we do this, you ask ? Well, those "banned" libraries should, in 99% of cases, not be shipped with any Magnolia project.

By version-managing those dependencies, we avoid them creeping into our builds. See http://day-to-day-stuff.blogspot.com/2007/10/announcement-version-99-does-not-exist.html for explanations of a similar trick.

We could also use Maven Enforcer banned dependencies rules, however, those would be much more complicated to override in projects where they're actually needed.

Why are those dependencies banned ?

In the case of `commons-logging`, it's fairly simple: we use SLF4j, which provides a **better** replacement for `commons-logging`, and we ship Magnolia with `jcl-over-slf4j` (see http://www.slf4j.org/legacy.html for details). Including `commons-logging` in a Magnolia project would make those 2 clashes and would prevent Magnolia from logging properly.

We also "ban" the following artifacts:

- `xml-apis`: these APIs are present in any JDK or redundant with Xerces (TBC - anyone with more details please correct this). They have caused enough pains and headaches in the past for us to decide to apply this hack to them too.
- `commons-beanutils-core`: we ship Magnolia with `commons-beanutils`, which includes `commons-beanutils-core`, so this is done as a preventive measure to avoid clashes.

So what can I do ?

Use the following two Maven commands to figure out where those unwanted dependencies are coming from:

- `mvn dependency:tree` (e.g. `mvn dependency:tree -Dincludes=commons-logging,commons-beanutils,xml-apis`)
- `mvn help:effective-pom`

If indeed you have a dependency that depends on one of the banned library, you'll need to exclude it. In the example above, you could already exclude it a (2) level (<abc>), or if you have an explicit dependency on <xyz>, exclude it there:

```
<dependencies>
  <...>
  <dependency>
    <groupId>xyz</groupId>
    <artifactId>xyz</artifactId>
    <version>1.0</version>
    <exclusions>
      <exclusion>
        <groupId>commons-logging</groupId>
        <artifactId>commons-logging</artifactId>
      </exclusion>
    </exclusions>
  </dependency>
  <...>
```

If you **really** need one of the banned dependencies (it should really not be the case, see above), you have two options, depending on the parent pom you use.

Up to version 19

You can override the managed-version in your own project's `<dependencyManagement>` section.

As from version 20

You could override the rules if really necessary (it probably isn't). Example, to allow commons-logging:

```
  <build>
    <pluginManagement>
      <plugins>
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-enforcer-plugin</artifactId>
          <executions>
            <execution>
              <id>enforce-banned-dependencies</id>
              <goals>
                <goal>enforce</goal>
              </goals>
              <configuration>
                <rules>
                  <bannedDependencies>
                    <includes>
                      <include>commons-logging:*</include>
                    </includes>
                  </bannedDependencies>
                </rules>
              </configuration>
            </execution>
          </executions>
        </plugin>
      </plugins>
    </pluginManagement>
  </build>
```

It is crucial that you keep the correct `enforce-banned-dependencies` id for the execution, in order to override the configuration of the parent pom.

See http://maven.apache.org/enforcer/enforcer-rules/bannedDependencies.html for more details.