

3 ways to add local jar to maven project


MAVEN  28 JUN 2016 13 COMMENTS 

Table of contents [\[hide\]](#)

- 1 Introduction
- 2 1- Install manually the JAR into your local Maven repository
- 3 2- Adding directly the dependency as system scope
- 4 3- Creating a different local Maven repository
- 5 4- Using Nexus repository manager
- 6 References

Introduction

You may need to add a custom JAR as a dependency to your Maven project. This tutorial shows 3 ways to do it:

1. Install manually the JAR into your local Maven repository
2. Adding the dependency as system scope
3. Creating a different local Maven repository
4. Using a Nexus repository manager

1- Install manually the JAR into your local Maven repository

The first solution is to add manually the JAR into your local Maven repository by using the Maven goal ***install:install-file***. The use of the plugin is very simple as below:

```
1 | mvn install:install-file -Dfile=<path-to-file>
```

Note that we didn't specify *groupId*, *artifactId*, *version* and *packaging* of the JAR to install. Indeed, since the version 2.5 of ***Maven-install-plugin***, these information can be taken from an optionally specified ***pomFile***.



These information can also be given in command line:

```
1 | mvn install:install-file -Dfile=<path-to-file> -DgroupId=<group-id> -DartifactId=<artifact-id> -Dversion=<version>
```

Where:

- **<path-to-file>**: Path to the JAR to install
- **<group-id>**: Group id of the JAR to install
- **<artifact-id>**: Artifact id of the JAR to install
- **<version>**: Version of the JAR

For example:

```
1 | mvn install:install-file -Dfile=C:\dev\app.jar -DgroupId=com.roufid.tutorials -DartifactId=example-app -Dversion=1.0
```

You can then add the dependency to your Maven project by adding those lines to your pom.xml file:

```
1 | <dependency>
2 |     <groupId>com.roufid.tutorials</groupId>
3 |     <artifactId>example-app</artifactId>
4 |     <version>1.0</version>
5 | </dependency>
```

This solution can be very expensive. Why? You have to consider that the day you change your local Maven repository you have to re-install the JAR. Or again, if there are many persons working on the project, each must install the JAR in his local repository. The portability of the project must be taken into account.

Another solution is to use the **maven-install-plugin** in your **pom.xml** which will install the jar during the Maven **"initialize"** phase. To do this, you must specify the location of the jar you want to install. The best way is to put the JAR in a folder created at the root of the project (*in the same directory as the pom.xml file*).

Let's consider that the jar is located under **<PROJECT_ROOT_FOLDER>/lib/app.jar**. Below the configuration of maven-



install-plugin:

```
1  <plugin>
2    <groupId>org.apache.maven.plugins</groupId>
3    <artifactId>maven-install-plugin</artifactId>
4    <version>2.5</version>
5    <executions>
6      <execution>
7        <phase>initialize</phase>
8        <goals>
9          <goal>install-file</goal>
10       </goals>
11       <configuration>
12         <groupId>com.roufid.tutorials</groupId>
13         <artifactId>example-app</artifactId>
14         <version>1.0</version>
15         <packaging>jar</packaging>
16         <file>${basedir}/lib/app.jar</file>
17       </configuration>
18     </execution>
19   </executions>
20 </plugin>
```



`${basedir}` represents the directory containing pom.xml.

You may encounter an error while adding the previous lines, add the following plugin to your project to allow the lifecycle mapping:

```
1  <pluginManagement>
2    <plugins>
3      <!--This plugin's configuration is used to store Eclipse m2e :
4        It has no influence on the Maven build itself. -->
5      <plugin>
6        <groupId>org.eclipse.m2e</groupId>
7        <artifactId>lifecycle-mapping</artifactId>
8        <version>1.0.0</version>
9        <configuration>
10          <lifecycleMappingMetadata>
11            <pluginExecutions>
12              <pluginExecution>
13                <pluginExecutionFilter>
```



```
14         <groupId>org.codehaus.mojo</groupId>
15         <artifactId>aspectj-maven-plugin</art:
16         <versionRange>[1.0,)</versionRange>
17         <goals>
18             <goal>test-compile</goal>
19             <goal>compile</goal>
20         </goals>
21     </pluginExecutionFilter>
22     <action>
23         <execute />
24     </action>
25 </pluginExecution>
26 <pluginExecution>
27     <pluginExecutionFilter>
28         <groupId>
29             org.apache.maven.plugins
30         </groupId>
31         <artifactId>
32             maven-install-plugin
33         </artifactId>
34         <versionRange>
35             [2.5,)
36         </versionRange>
37         <goals>
38             <goal>install-file</goal>
39         </goals>
40     </pluginExecutionFilter>
41     <action>
42         <execute>
43             <runOnIncremental>false</runOnInci
44         </execute>
45     </action>
46 </pluginExecution>
47 </pluginExecutions>
48 </lifecycleMappingMetadata>
49 </configuration>
50 </plugin>
51 </plugins>
52 </pluginManagement>
```



2- Adding directly the dependency as system scope

Another solution – dirty solution – is by adding the dependency as **system** scope and refer to it by its full path. Consider that the JAR is located in **<PROJECT_ROOT_FOLDER>/lib**. Then add the dependency in your pom.xml file as following:

```
1 <dependency>
2   <groupId>com.roufid.tutorials</groupId>
3   <artifactId>example-app</artifactId>
4   <version>1.0</version>
5   <scope>system</scope>
6   <systemPath>${basedir}/lib/yourJar.jar</systemPath>
7 </dependency>
```



***\${basedir}** represents the directory containing pom.xml.*

3- Creating a different local Maven repository

The third solution is quite similar to the first one, the difference lies in the fact that the JARs will be installed in a different local Maven repository.

Let's consider the new local Maven repository is named **"maven-repository"** and is located in **\${basedir}** (the directory containing pom.xml). First you have to do is **deploying** the local JARs in the new local maven repository as below:

```
1 mvn deploy:deploy-file -Dfile=<path-to-file> -DgroupId=<group-id> -Dart
```

Normally, the Maven **deploy:deploy-file** installs the artifact in a remote repository but in our case the repository is located in the local machine.



After installing the JARs you need to add the repository in your pom.xml file:

```
1 <repositories>
2   <repository>
3     <id>maven-repository</id>
4     <url>file:///${project.basedir}/maven-repository</url>
5   </repository>
6 </repositories>
```

Then you can add the dependency into your pom.xml

```
1 <dependency>
2   <groupId>com.roufid.tutorials</groupId>
3   <artifactId>example-app</artifactId>
4   <version>1.0</version>
5 </dependency>
```

4- Using Nexus repository manager

The best solution is to use a Nexus Repository Manager which will contain all your JARs and you will use it as repository to download the dependency.

This book from the official Nexus site will show you how to install and use Nexus repository manager.

References

- **Maven : installing 3rd party JARs**
- **Maven lifecycle**
- **Maven deploy:deploy-file goal**
- **Maven install:install-file goal**
- **Nexus**

Spread the word :



Tags: Maven

