OBR Maven Plugin

Contributed by Clement
Friday, 18 May 2007
Last Updated Monday, 13 August 2007

This Maven plug-in aims to automate OBR (OSGI Bundle Repository) management. It creates a local OBR repository and can create a remote OBR to distribute bundles. This plug-in allows you to install your bundles inside an OBR repository at the same time you install it in your Maven repository. It can also deploy your bundle in a remote Maven repository when you deploy your Maven artefact. The plug-in compute dependencies and requirements and edits the repository description file to add (or update) the bundle description.

Features:

The plug-in allows:
 - Installing Maven artefact inside an OBR repository, automatically or manually (for legacy bundles). This OBR could be remote or local.
 - Discovering bundle capabilities and requirements
 - Customizing bundle descriptions   Downloads & Changelog

The plug-in is available here .

Changelog :
 - 08132007 - Add deploy-file goal, solves a resolution bug in the install-file goal.

How to use the plug-in?

The plug-in offers four Maven goals:
 - A: The standard &lsquo;install&rsquo; goal: it will install your bundle in the repository at the same time it install it in your Maven repository.
 - B: An &lsquo;install-file&rsquo; goal which takes a jar file, and installs it in the local Maven repository and update the remote repository descriptor file.
 - C: A &lsquo;deploy&rsquo; goal which upload a bundle on a remote repository and update the remote repository descriptor file.
 - D: A &lsquo;deploy-file&rsquo; goal which upload jar file on a remote repository and update the remote repository descriptor file.   -   A: Maven Project Integration:

To use the plug-in, you need to update the pom file of your bundle project. By this way, you need to add the following XML snippet (after your packaging plug-in). By default no configuration is needed.

```
<plugin>
    <groupId>homega.tools</groupId>
    <artifactId>obrPlugin</artifactId>
        <executions>
          <execution>
            <goals>
              <goal>repository</goal>
            </goals>
          </execution>
        </executions>
</plugin>
```

In this metadata, you may specify the path to your OBR repository.xml file. By default the plug-in use a file named &lsquo;repository.xml&rsquo; at the root of your Maven repository. It will set the default repository name to &ldquo;OSGi Bundle Repository&rdquo;.

For each installed bundle, the plug-in use Maven metadata, Bindex and an external file to build a bundle description containing following attributes:

<resource   id   : calculated by the plug-in, and takes the first none-used number   presentationname : Ma
name   symbolicname   : Maven artefact Id   uri   : path to the artefact in the Maven local repository
artefact version

>   <size>   : size of the bundle jar file <description>   : Maven artefact description <documentation>
documentation URL

<category   id   : Maven artefact group id.

> <capability> & <require>: Calculated by the plug-in (using the bindex tools, and an external file).

When installing an existing bundle, the plug-in searches a bundle in the repository with the same presentation name, symbolic name and version number. If the bundle already exists, the description is replaced by the new one. Else the plug-in adds the new bundle description to the repository.

By default, the plug-in uses the pom file and Bindex to build the description. However, it is possible to add metadata &lsquo;manually&rsquo;. You could add categories, capabilities and/or requirements manually in your project. For this, you must include an obr.xml file in the resource folder.

This file contains metadata (using the OBR-syntax) such as: <resource>

```
    <capability name=&hellip;>
        &hellip;
    </capability>
    <require &hellip;>
         &hellip;
    </require>
    <category id=&hellip;/>
</resource>
```

Except the <resource>, all the other elements are optional. All information given in this file will be added to the final repository file. It does not override previously collected information. B: install-file goal

This second way, to use the plug-in, allows you adding an already existing bundle in the OBR. The bundle must be already in the Maven local repository (you can use the Maven install:install-file plug-in to install an external bundle in your Maven repository).

To use this goal, the user must provide information in command line: Variable name (preceded by &ndash;D) status description file require Path to the jar file (used to install bundle only) artefactId require artefactId of the bundle Use to determine path to the jar file in local maven Repository use as symbolic name if it isn&rsquo;t define in manifest groupId require groupId of the bundle Use to determine path to the jar file on local maven repository version require Version of the bundle Use to determine path to the jar file on local maven repository packaging require File type Use by maven to install the file (used to install bundle only) repository-path optional Path to the repository descriptor file (if not define: default path is : ${MavenRepo}\repository.xml) obr-file optional Path to the obr.xml, file which describe capabilities requirement and category given manually by user. (if not define: nothing is added to the resource description). Example of command line: mvn homega.tools:obrPlugin:install-file \

```
-Drepository-path=file:/c:\repository.xml \
-obr-file=file:/c:\project\homega\obr.xml \
-DartifactId=multicast.discovery \
-DgroupId=homega.utils \
-Dversion=1.0.0
```

By using this command, the plug-in will look at the artefact

${MavenRepo}\homega\utils\multicast.discovery\1.0.0\multicast.discovery-1.0.0.jar Then it will compute bundle description by using bindex and information form the command line (artifactId, groupId, version, and obr file (if set)). Note:

As said previously, the plug-in does not install the file in the Maven repository but target an already install artefact. However, it is possible to install the bundle in the repository and to add it in the OBR repository file in one command:

```
    mvn install:install-file homega.tools:obrPlugin:install-file \
 -Dfile=file:/c:\projet\homega\multicast.discovery-1.0.0.jar \
-Drepository-path=file:/c:\repository.xml \
-Dobr-file=file:/c:\project\homega\obr.xml \
-DartifactId=multicast.discovery \
-DgroupId=homega.utils \
-Dversion=1.0.0 \
-Dpackaging=jar
```

C: deploy goal

To use the deploy goal, you must change your pom.xml:
 - Add the &ldquo;deployment&rdquo; goal
 - Indicate the repository-name property. This is the name of the repository descriptor file (this property is optional; default value is &ldquo;repository.xml&rdquo;). This file is located on url provided by user. <plugin>

```
  <groupId>homega.tools</groupId>
  <artifactId>obrPlugin</artifactId>
```

```
        <configuration>
            <repository-name>repository.xml</repository-name>
        </configuration>
        <executions>
          <execution>
            <goals>
              <goal>repository</goal>
              <goal>deployment</goal>
            </goals>
          </execution>
        </executions>
</plugin>
```

You must also configure where the plugin will upload bundles and repository descriptor file. You could do it in maven as this: <distributionManagement>

```
    <repository>
     <id>ftp-repository</id>
     <name>RepoName</name>
     <url>ftp://ftp.youraddress.com/obr</url>
    </repository>
  </distributionManagement>
```

Now maven will upload your bundle on this ftp site.

In fact, all protocols are not supported by maven. For example ftp protocol as describe above require wagon plugin to run correctly. So you could add this part in your pom.xml to use wagon in ftp protocol:

&hellip; <build>

```
    <extensions>
      <extension>
        <groupId>org.apache.maven.wagon</groupId>
         <artifactId>wagon-ftp</artifactId>
         <version>1.0-alpha-6</version>
      </extension>
    </extensions>
```

By this way, the deploy goal supports all protocols supported by wagon. D: deploy-file goal

This goal is useful if you want add a non-maven project on your remote repository, for example a legacy bundles.The command line to use it is: Variable name (preceded by &ndash;D) status description file require Path to the jar file artefactId require artefactId of the bundle Use to determine path to the jar file in local maven Repository use as symbolic name if it isn&rsquo;t define in manifest groupId require groupId of the bundle Use to determine path to the jar file on local maven repository version require Version of the bundle Use to determine path to the jar file on local maven repository packaging require File type Use by maven to install the file (used to install bundle only) url require url to the remote server uniqueVersion optional Set it to false to avoid unique filename when you upload repositoryId optional Name of the repository use to upload file obr-file optional Path to the obr.xml, file which describe capabilities requirement and category given manually by user. (if not define: nothing is added to the resource description).

Example of complete command line:

```
mvn deploy:deploy-file homega.tools:obrPlugin:deploy-file \

-DartifactId=echo2

-DgroupId=tools

-Dversion=2.0.0

-Dpackaging=jar

-DrepositoryId=ftp-repository

-DuniqueVersion=false
```

-Durl=ftp://ftp.plop-plop.net/obr

-Dobr-file=c:\obr.xml

-Dfile=echo2.jar

To upload you file you must configure a repositoryManagement to indicates which remote server you use and component used for the protocol transfer. To do that you could create a new pom.xml for each jar file to upload (an example of this file is given below). This file must contain also groupId, artifactId version and packaging information (in fact those data are redundant with the command line, so you could omit them in command line).

Example of additional pom.xml:

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <packaging>jar</packaging>
  <groupId>tools</groupId>
  <artifactId>echo2</artifactId>
  <version>2.0.0</version>
  <distributionManagement>
   <repository>
    <id>ftp-repository</id>
    <name>RepoName</name>
    <url>ftp://ftp.plop-plop.net/obr</url>
   </repository>
  </distributionManagement>
  <build>
    <extensions>
      <extension>
        <groupId>org.apache.maven.wagon</groupId>
         <artifactId>wagon-ftp</artifactId>
         <version>1.0-alpha-6</version>
      </extension>
    </extensions>
  </build>
</project>
```

Plugin information:

In order to create the bundle description, the plug-in gets information from bindex, the pom.xml and the obr.xml file. Information can be overridden:

Bindex

| (overrides)

Pom.xml

| (overrides)

Obr.xml

A warning message is displayed each time already existing information is overridden. In case of install-file goal, the information given by user (i.e.: artefactId, groupId, version) is considered as pom.xml properties.

Known bugs & limitation:

1. Do not support relative paths when you specify the repository descriptor in command line use.
2. obr.xml (file given by the user to describe some properties not found by Bindex) must be correct, because the plug-in does not check the syntax.

Full example:

This section shows a full example with project integration:

First, the project must be compiled with maven 2. So we create an adequate pom.xml file, which contains plug-in directive.

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <packaging>bundle</packaging>
  <groupId>homega.remote.DPWS.bridge</groupId>
  <artifactId>remote.DPWS.bridge.binary.light</artifactId>
  <version>1.0.0</version>
  <name>Bridge for Binary Light</name>
  <dependencies>
    <dependency>
        <groupId>homega.utils</groupId>
        <artifactId>web.service.publisher</artifactId>
        <version>1.0.0</version>
    </dependency>
  </dependencies>           <build>
    <plugins>
      <plugin>
        &hellip;                                        </plugin>
      <plugin>
        <groupId>homega.tools</groupId>
        <artifactId>obrPlugin</artifactId>
        <configuration>
            <repository-path>file:/c:\repository.xml</repository-path >
        </configuration>
        <executions>
          <execution>
            <goals>
              <goal>repository</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</project>
```

This bundle comes from the H-Omega project; it affords a stub to control a simulated device (in this example a binary lamp) on another gateway. We use here iPOJO plugin but it is not necessary, our plug-in could be applied alone, or with another Maven plugin.

The compilation executes the plug-in which create the repository.xml file. By default, the plug-in use Bindex to get the requirement from a bundle from its manifest. To complete bundle requirements we use an obr.xml file, include in the resource folder of the project:

```
<resource>
        <capability name="service">
         <p n="service" v="fr.imag.adele.homega.devices.itf.BinaryLight"/>
         <p n="property" v="location"/>
         <p n="property" v="friendly.name"/>
         <p n="property" v="uuid"/>
         <p n="property" v="url.device"/>
        </capability>
        <require optional="false" multiple="true" name="service" extend="false"
            filter="(service=org.osgi.service.event.EventAdmin) "/>
        <require optional="false" multiple="true" name="service" extend="false"
            filter="(service=fr.imag.adele.remotesoap.RegisterRemote) "/>
                <category id="device"/>
</resource>
```

In this file we declare a provided service with 4 properties (BinaryLight), and use 2 others (EventAdmin and RegisterRemote).

Now we compile the project with maven (&lsquo;mvn clean install&rsquo;), the plug-in creates a new repository.xml file (if not already exists) with the bundle extracted information. It contains only our bundle, so we must compiled each bundle used in the project to add them in the obr.

This generated file is: <?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?> <repository lastmodified="20070417101307.234" name="MyRepository"> <resource id="0" presentationname="Bridge for Binary Light" symbolicname="remote.DPWS.bridge.binary.light" uri="file:/C:\projet\Stage\mvn_repo\homega\remote\DPWS\bridge\remote.DPWS.bridge.binary.light\1.0.0\remote.DPWS.bridge.binary.light-1.0.0.jar" version="1.0.0"> <description/> <size>8516</size> <documentation/> <category id="device"/> <category id="homega.remote.DPWS.bridge"/> <capability name="bundle"> <p n="manifestversion" v="2"/> <p n="presentationname" v="Bridge for Binary Light"/> <p n="symbolicname" v="remote.DPWS.bridge.binary.light"/> <p n="version" t="version" v="1.0.0"/> </capability> <capability name="service">   <p n="service" v="fr.imag.adele.homega.devices.itf.BinaryLight"/>   <p n="property" v="location"/
<p n="property" v="friendly.name"/>   <p n="property" v="uuid"/>   <p n="property" v="url.device"/> </capability> <require extend="false" filter="(service=org.osgi.service.event.EventAdmin))" multiple="true" name="service" optional="false"/> <require extend="false" filter="(service=fr.imag.adele.remotesoap.RegisterRemote)" multiple="true" name="service" optional="false"/> <require extend="false" filter="(&(package=fr.imag.adele.homega.devices.itf)(version>=0.0.0))" multiple="false" name="package" optional="false">

Import package fr.imag.adele.homega.devices.itf </require> <require extend="false" filter="(&(package=javax.xml.namespace)(version>=0.0.0))" multiple="false" name="package" optional="false">

Import package javax.xml.namespace </require> <require extend="false" filter="(&(package=javax.xml.rpc)(version>=0.0.0))" multiple="false" name="package" optional="false">

Import package javax.xml.rpc </require> <require extend="false" filter="(&(package=org.apache.axis.client)(version>=0.0.0))" multiple="false" name="package" optional="false">

Import package org.apache.axis.client </require> </resource>

<resource>

&hellip; </resource> </repository>
Now we can use this file in OSGi&trade;, with the obr bundle, we just load the correct file (located in c:\repository.xml):

-> obr add-url file:/c:\repository.xml

-> obr list

Alarm Manager (1.0.0)

Apache Felix Bundle Repository (0.8.0.incubator)

Apache Felix Shell Service (0.8.0.incubator)

Apache Felix Shell TUI (0.8.0.incubator)

Axis (1.0.0)

Bridge for Alarm Center (1.0.0)

Bridge for Binary Light (1.0.0)

Bridge for Brightness Sensor (1.0.0)

Bridge for Dimmer (1.0.0)

Bridge for Fridge (1.0.0)

Bridge for Heater (1.0.0)

Bridge for LCD Screen (1.0.0)

Bridge for Shutter (1.0.0)

Bridge for Thermometer (1.0.0)

Configuration For bridge (1.0.0)

Device Manager (1.0.0)

Devices Gateway Configuration For SCC (1.0.0)

DevicesCreator (1.0.0) Discovery Bridge (1.0.0)

Environment Control (1.0.0)

Event Admin (0.7.2)

Event Admin Handler (1.0.0) Gui Event Bridge (1.0.0)

HomeCinema (1.0.0)

Homega Interfaces (1.0.0)

```
Homega Manager GUI (1.0.0)

HTTP Server (1.1.0)

I Leave / I come back scene (1.0.0)

iPOJO (0.7.0.incubator-SNAPSHOT)

iPOJO Arch Command (0.0.0)

Mail Service (1.0.0)

Multicast Discovery Library (1.0.0)

osgi.compendium (4.0.0)

PresenceSimulation (1.0.0)

servlet (2.3.0)

Simulated Devices (1.0.0)

telnetd (1.0.0)

Wake up scene (1.0.0)

WeatherService (1.0.0)

Web Service Publisher (1.0.0)

->
```

And we start the adequate bundle:

```
-> obr start "Bridge for Binary Light"

Target resource(s):

-------------------

   Bridge for Binary Light (1.0.0)

Required resource(s):

---------------------

   Web Service Publisher (1.0.0)

   osgi.compendium (4.0.0)

   Homega Interfaces (1.0.0)

   servlet (2.3.0)

   Axis (1.0.0)

   Event Admin Handler (1.0.0)

   Event Admin (0.7.2)

   iPOJO (0.7.0.incubator-SNAPSHOT)

Deploying...DEBUG: WIRE: 5.0 -> javax.servlet.http -> 7.0
```

```
[&hellip;]

DEBUG: WIRE: 12.0 -> fr.imag.adele.util.eventadmin -> 12.0

done.

-> ps

START LEVEL 1

   ID   State         Level  Name

[   0] [Active     ] [    0] System Bundle (0.8.0.incubator)

[   1] [Active     ] [    1] Apache Felix Shell Service (0.8.0.incubator)

[   2] [Active     ] [    1] Apache Felix Shell TUI (0.8.0.incubator)

[   3] [Active     ] [    1] Script Commands Bundle (0.4.0)

[   4] [Active     ] [    1] Apache Felix Bundle Repository (0.9.0.incubator_SNAPSHOT)

[   5] [Active     ] [    1] Web Service Publisher (1.0.0)

[   6] [Active     ] [    1] osgi.compendium (4)

[   7] [Active     ] [    1] servlet (2.3)

[   8] [Active     ] [    1] Homega Interfaces (1.0.0)

[   9] [Active     ] [    1] Event Admin Handler (1.0.0)

[  10] [Active     ] [    1] Axis (1.0)

[  11] [Active     ] [    1] Bridge for Binary Light (1.0.0)

[  12] [Active     ] [    1] Event Admin (0.7.2)

[  13] [Active     ] [    1] iPOJO (0.7.0.incubator-SNAPSHOT)

->
```

Feedbacks

For any question or feedback, do not hesitate to send an email to maxime.vincent@hotmail.fr or clement.escoffier@imag.fr