Chapter 1. Install a JDBC driver

Table of Contents

This document describes the process to transform a JDBC driver jar file into an OSGi™ bundle suitable for usage in JOnAS.

It targets a developer audience. Some OSGi™ vocabulary is used in this document.

Note

This howto has been tested using `DataSource` provided using a Resource Adapter.

1.1. Why creating an OSGi bundle ?

JOnAS is performing an automatic bundle generation during startup for jars found in `$JONAS_ROOT/lib/ext/` and `$JONAS_BASE/lib/ext/`. That means that the jar file dropped into the `lib/ext/` directory will be analyzed and appropriate entries will be added in the MANIFEST of the jar.

Theses new entries are OSGi™ related attributes (`Bundle-SymbolicName`, `Bundle-ClassPath`, `Import-Package`, `Export-Package`, ...). They describe what the bundle is (symbolic name, ...), what it requires from the system (`Import-Package`) and what it gives to the system (`Export-Package`). Concretely the Import-Package entry contains a list of package that are necessary in order to have a working bundle and the Export-Package is also a list of packages "offered" to the system.

By default, JOnAS use very simple directives to generate a bundle that should *mostly* work:

- Every package required by the code in the jar file is placed in the Import-Package list, *with a resolution:=optional attribute*. That means that, during startup (when the bundle is installed then started), the OSGi™ framework will try to resolve as much required packages as possible, but if some are missing, there will be no failures. At the end, the bundle is ready to be used, but it's possible that it won't work as a really needed package could not be found during the resolution.
- Every package provided by the original jar file is placed in the `Export-Package` list. That means that all the classes that are in the jar file are potentially visible from other part of the server. This is bad for modularity.
- `Bundle-SymbolicName` is simply the jar file short name.

Creating a handmade bundle with controled content will also avoid seeing that kind of traces during JOnAS start-up sequence:

```
07/09/09 12:26:22 (I) ExtensionLoaderComponent.__start : Some jars have been found in
 [<JONAS_ROOT>\lib\ext, <JONAS_BASE>\lib\ext].
                                                         They have been transformed into
 bundles (see <JONAS_BASE>\work\ext-bundles directory).
                                                         This is NOT the prefered way to
 extends JOnAS libraries, prefer to use carefully
                                                         created and tested bundles, and
 place them in your deploy/ directory.
```

```
07/09/09 12:26:22 (I) ExtensionLoaderComponent.__start : file:/<JONAS_BASE>/work/ext-
bundles/ojdbc14.jar installed as a bundle.
07/09/09 12:26:22 (I) ExtensionLoaderComponent.__start : file:/<JONAS_BASE>/work/ext-
bundles/mysql-connector-java-3.1.10-bin.jar installed as a bundle.
```

1.2. Using Bnd to generate an OSGi bundle

Bnd is a bundle generation tool. It helps a lot the developer of OSGi™ bundles by automating MANIFEST generation and bundle content creation.

The references section contains links to some Bnd documentations and examples that a user should read to better understand the Bnd philisophy.

This howto focus on practical operations to be done to transform a jar file containing a JDBC driver into a suitable OSGi™ bundle.

1.2.1. Installing Bnd

Bnd can be downloaded here: http://www.aqute.biz/Code/Download.

This is an all-in-one executable jar file (sources can be seen in `/OSGI-OPT/src/`).

It can be executed using the following command line:

```
>$ java -jar bnd.jar
```

Note

Bnd requires a 1.5 JVM (at minimum, also works with 1.6).

Note

Later in this article, a command named **bnd** is used. It refers to a shell script that could looks like the following:

```
#!/bin/sh
java -jar /path/to/your/bnd.jar $*
```

1.2.2. Default jar wrapping

Bnd provides a utility wrapping mode. Here is a quote from the Bnd documentation:

“The wrap command takes an existing JAR file and guesses the manifest headers that will make this JAR useful for an OSGi Service Platform.”

Looks promising :) In fact, this mode the one used by JOnAS when a library is placed in `lib/ext/`. If the jar in lib/ext/ had already been transformed and tested as a bundle, it's probably safe to use it as is (transformed bundle are in `<JONAS_BASE>/work/bundle-ext/`).

The following command show how to use it using a MySQL driver jar:

```
>$ bnd wrap mysql-connector-java-5.1.8-bin.jar
mysql-connector-java-5.1.8-bin 239 0
One warning
1 : Superfluous export-package instructions: [org, com, com.mysql.jdbc.jdbc2, org.gjt,
 org.gjt.mm, com.mysql.jdbc.integration, com.mysql]
```

This command has created a `mysql-connector-java-5.1.8-bin.bar` file that is an OSGi™ bundle. It can be placed in `<JONAS_BASE>/deploy/` and renamed with a `.jar` extension and then JOnAS will automatically deploy it.

As a side note, here is what is imported and exported from the MySQL bundle after wrapping (this output can be useful for advanced users):

```
>$ bnd print -impexp mysql-connector-java-5.1.8-bin.bar
[IMPEXP]
Import-Package
  com.mchange.v2.c3p0                   {resolution:=optional}
  javax.naming                          {resolution:=optional}
  javax.naming.spi                      {resolution:=optional}
  javax.net                             {resolution:=optional}
  javax.net.ssl                         {resolution:=optional}
  javax.sql                             {resolution:=optional}
  javax.transaction.xa                  {resolution:=optional}
  javax.xml.parsers                     {resolution:=optional}
  javax.xml.stream                      {resolution:=optional}
  javax.xml.transform                   {resolution:=optional}
  javax.xml.transform.dom               {resolution:=optional}
  javax.xml.transform.sax               {resolution:=optional}
  javax.xml.transform.stax              {resolution:=optional}
  javax.xml.transform.stream            {resolution:=optional}
  org.apache.commons.logging            {resolution:=optional}
  org.apache.log4j                      {resolution:=optional}
  org.jboss.resource.adapter.jdbc       {resolution:=optional}
  org.jboss.resource.adapter.jdbc.vendor{resolution:=optional}
  org.w3c.dom                           {resolution:=optional}
  org.xml.sax                           {resolution:=optional}
  org.xml.sax.helpers                   {resolution:=optional}
Export-Package
  com.mysql.jdbc
  com.mysql.jdbc.configs
  com.mysql.jdbc.exceptions
  com.mysql.jdbc.exceptions.jdbc4
  com.mysql.jdbc.integration.c3p0
  com.mysql.jdbc.integration.jboss
  com.mysql.jdbc.interceptors
  com.mysql.jdbc.jdbc2.optional
  com.mysql.jdbc.log
  com.mysql.jdbc.profiler
  com.mysql.jdbc.util
  org.gjt.mm.mysql
```

1.2.3. Advanced wrapping

This section is for lib/ext/ bundles that cannot be installed with the default process or address developers wanting to control exactky their bundle's content.

The wrap command's default values can be changed using a property file:

```
Import-Package: !com.mchange.v2.c3p0,\                    1
                !org.jboss.*,\
                javax.transaction.*;version=1.1,\         2
                *;resolution:=optional                    3

Export-Package: !com.mysql.jdbc.integration.*,\           4
                *;version=5.1.8                           5

Bundle-Version 5.1.8                                      6

Bundle-SymbolicName MySQL Connector/J                     7
```

1. Inside JOnAS, the c3p0 and jboss related packages don't have to be imported (they are not even exported, so the dependencies can never be resolved). Using the '!' prefix exclude them.
2. Specify a version for this required package (not optional). The bundle will be installed only if a package with the correct version is exported.
3. The trainling '*' is required. If omitted, Bnd will not discover the imported packages of the jar.
4. Similar to Import-Package exclusions, Bnd can avoid exporting some packages that are part of the original jar by prefixing their package name with '!'
5. That line indicates to Bnd that all exported packages will be "tagged" with a particular version. It's a good practice to always have a version on exported packages.
6. By default, the value will be '0'. For consistency, it should be set accordingly to the version of the driver.

7 By default, the value will be the jar file name.

This file can then be used on the command line:

```
>$ bnd wrap -properties mysql-connector-java-5.1.8-bin.bnd mysql-connector-java-5.1.8-
bin.jar
mysql-connector-java-5.1.8-bin 239 728542
```

Note

That section was only an example of an advanced bundle generation. the import/export directives shown above may not be suitable for your own bundle. It is advised to read the links provided in the references section to better understand and see other capabilities of Bnd.

1.3. References

1. Bnd: Bundle Tool [http://www.aqute.biz/Code/Bnd]
2. Example: Using Bnd To Do a Quick Wrap of Hibernate [http://www.aqute.biz/Code/BndHibernate]
3. SpringSource: Creating OSGi bundles [http://blog.springsource.com/2008/02/18/creating-osgi-bundles/]
4. Documentation: Maven bundle Plugin [http://felix.apache.org/site/apache-felix-maven-bundle-plugin-bnd.html] (FAQ [http://felix.apache.org/site/apache-felix-bundle-plugin-faq.html])
5. Download Bnd [http://www.aqute.biz/Code/Download]