OSGi Bundle Utility 1.0

Product Documentation

Jeremias Märki <dev@jeremias-maerki.ch>

OSGi Bundle Utility 1.0: Product Documentation

by Jeremias Märki

This document was written in DocBook 5 [http://www.docbook.org/tdg5/]. It was converted to PDF using Apache FOP [http://xmlgraphics.apache.org/fop/].

The HTML version of this document uses the CSS stylesheet from the Subversion book [http://svnbook.googlecode.com/svn/tags/en-1.5-final/src/en/book/styles.css] by Ben Collins-Sussman, Brian W. Fitzpatrick & C. Michael Pilato. It is licensed under the Creative Commons Attribution License [http://creativecommons.org/licenses/by/2.0/].

Table of Contents

List of Tables

List of Examples

Chapter 1. Introduction

1.1. What's the OSGi Bundle Utility?

The OSGi Bundle Utility is a package that helps you produce OSGi bundles using Apache Ant [http://ant.apache.org]. It is designed to serve as an alternative to Peter Kriens' *Bnd* [http://www.aqute.biz/Code/Bnd]. But it does not aim to handle all possible cases. The idea is to follow the 80:20 rule to ensure ease of use for most use cases.

1.2. Why did I write this utility?

I found Bnd difficult to manage and to get to work with Apache Ant. I wanted something simpler because in most cases, you don't need all the flexibility Bnd provides. Finally, I wanted the *bundle descriptor* to be written in XML. It turned out to work quite well, at least for me, and was a good way for me to learn more about OSGi. Hopefully, I've followed all the best practice.

1.3. A quick peek

Here's a simple example demonstrating what the utility does. The most important part is the bundle descriptors:

Example 1.1. A simple example (bundle.xml)

```
<bundle xmlns="http://www.jeremias-maerki.ch/ns/osgi-bundle">
  <name>JM :: Apache FOP OSGi Integration</name>
  <symbolic-name>ch.jm.fop.osgi</symbolic-name>
  <headers>
    <header name="Bundle-Activator">ch.jm.fop.osgi.Activator</header>
  </headers>
  <imports>
    <import match="org.xml.sax" add="true"/>
  </imports>
</bundle>
```

If you know a little about OSGi you will surely recognize the meaning of the tags. For detailed documention, please refer to the following documention.

Finally, the utility provides Ant tasks for creating bundles, converting JARs into bundles and to produce manifest files for OSGi bundles.

Chapter 2. The Bundle Descriptor

The bundle descriptor is an XML file that contains meta-information about the bundle to be created.

2.1. The XML Schema

There is an XML Schema for the bundle descriptor presented in this chapter. You can find it under `src/java/ch/jm/osgi/util/bundle/bundle.xsd` in the source distribution or under `ch/jm/osgi/util/bundle/bundle.xsd` in the JAR file. The bundle descriptors are automatically validated against the schema when they are loaded by the Ant tasks.

Note

Please note that not the full structure of the bundle descriptor is described in prose here. Please refer to the XML Schema when in doubt about the structuring.

2.2. The Namespace

The bundle descriptor uses the namespace `http://www.jeremias-maerki.ch/ns/osgi-bundle` for all elements.

2.3. The "bundle" Element (Root Element)

The `bundle` element serves as the root element of the bundle descriptor. It contains the actual meta-information for the bundle. This includes:

Table 2.1. Immediate children of the "bundle" element

Entry	Bundle descriptor	Manifest
the bundle name	name	Bundle-Name
the bundle description	description	Bundle-Description
the bundle's symbolic name	symbolic-name	Bundle-SymbolicName
the bundle category	category	Bundle-Category

Note

You'll note the absence of an element for the version number. The `Bundle-Version` is usually expected to be set within the Apache Ant build script as part of a manifest attribute.

2.3.1. The "headers" Element

The `headers` element allows to directly specify a number of manifest headers.

Note

It is theoretically possible to generate, for example, the `Bundle-SymbolicName` header here, but there is no guarantee which value will prevail if it collides with the `symbolic-name` element from above.

The `headers` element takes one or more `header` elements. Each element takes a mandatory `name` attribute. The text value is provided as the element's content.

Example 2.1. Header example

```
<bundle xmlns="http://www.jeremias-maerki.ch/ns/osgi-bundle">
  [..]
  <headers>
    <header name="Bundle-Activator">ch.jm.fop.osgi.Activator</header>
    <header name="Bundle-Vendor">Jeremias Märki</header>
    <header name="Service-Component">OSGI-INF/configurator.xml</header>
  </headers>
  [..]
</bundle>
```

2.3.2. The "exports" Element

The `exports` element allows to directly specify a number of packages to be exported.

The `exports` element takes one or more `export`elements. The package name is provided as the element's content. To export a package including all subpackages, you may use a "*" as a wildcard.

Each `export` element takes an optional `version` attribute to override the version specified by the `Bundle-Version` header. If the `version` attribute is absent, each package export will have the bundle's version added as the exported version. All exported bundles will also automatically be imported with the same version number. This is in line with OSGi best practice.

Example 2.2. Export example

```
<bundle xmlns="http://www.jeremias-maerki.ch/ns/osgi-bundle">
  [..]
  <exports>
    <export>org.apache.fop*</export>
    <!-- ...or ... -->
    <export>ch.jm.lpr</export>
    <export>ch.jm.lpr.client</export>
  </exports>
  [..]
</bundle>
```

2.3.3. The "imports" Element

The `imports` element allows to directly specify a number of packages to be imported. Normally, the bundle utility will automatically determine all required imports through byte code analysis but sometimes it's necessary to add directives or attributes to the import, for example, if you need optional resolution for some packages.

The `imports` element takes one or more `import` elements. The package name is provided in the required `match` attribute. To match multiple packages at one you may use "*" as a wildcard. The directives an attributes will then be applied to all these matching packages.

Normally, only imports that have been identified by byte code analysis are matched with the above mechanism. If you want to explicitely import a package, you can use the `add` attribute. The datatype here is `xsd:boolean` and the default is `false` for this attribute. The wildcard ("*") is not allowed if `add="true"`.

Example 2.3. Import example

```
<bundle xmlns="http://www.jeremias-maerki.ch/ns/osgi-bundle">
  [..]
  <imports>
    <import match="org.apache.tools.ant*">
      <directive name="resolution">optional</directive>
    </import>
    <import match="com.acme.something" add="true">
      <attribute name="looney">true</attribute>
    </import
  </imports>
  [..]
</bundle>
```

Chapter 3. The Ant Tasks

3.1. The "bundle-manifest" Task

The `bundle-manifest` task allows you to create a JAR manifest with the necessary OSGi headers for the bundle to be created.

Example 3.1. Example using `bundle-manifest`

```
<project name="example.freespace" default="all">

  [..]
  <!-- Build manifest from bundle descriptor and add additional entries -->
  <bundle-manifest file="${build.classes.dir}/META-INF/MANIFEST.MF"
        classes="${build.classes.dir}"
        descriptor="${basedir}/bundle.xml">
    <attribute name="Bundle-Version" value="${version}"/>
    <attribute name="Bundle-Vendor" value="${implementation.vendor}"/>
    <attribute name="Bundle-DocURL" value="${implementation.url}"/>
    <attribute name="Implementation-Title"
        value="${Name} (${subproject-name})"/>
    <attribute name="Implementation-Version" value="${version}"/>
    <attribute name="Implementation-Vendor"
        value="${implementation.vendor}"/>
    <attribute name="Implementation-URL"
        value="${implementation.url}"/>
  </bundle-manifest>

  <!-- Now build the actual bundle. -->
  <jar jarfile="${build.dir}/${subproject-name}.jar"
        filesetmanifest="merge" manifestencoding="UTF-8">
    <fileset dir="${build.classes.dir}"/>
    <metainf dir="${project-root.dir}" includes="LICENSE.txt,NOTICE.txt"/>
  </jar>
  [..]
</project>
```

The example above shows a typical use of the `bundle-manifest` task. As you can see, the first step is to generate the manifest which is written to `/META-INF/MANIFEST.MF` (in the directory where you place the compiled classes). The `jar` task can then merge that generated manifest into the final manifest that will be found in the JAR file (see `filesetmanifest="merge"`).

The `file` attribute specifies the manifest file to be written.

The `descriptor` attribute specifies the bundle descriptor for the bundle.

The `classes` attribute specifies the directory containing the compiled Java classes to be analyzed to determine all necessary imports.

Additionally, you can add a nested `path` element (See Ant's path-like structures) as a child of `bundle-manifest` to specify additional JAR files that will be assumed to be embedded in the bundle (under `/META-INF/lib/`). These embedded JARs will also be scanned for external dependencies. You are, however, responsible for copying these JARs to the right place in the classes directory so they end up in the bundle/JAR and are found at runtime.

Finally, you can provide any number of header attributes just like is possible in Ant's `manifest` or `jar` tasks. It is recommended to provide at least the `Bundle-Version` header so the bundle's version number is applied to all exports.

Note

Headers specified in the bundle descriptor will override equally-named headers from the Ant task!

3.2. The "load-bundle-descriptor" Task

The `load-bundle-descriptor` task is used to set a number of properties based on values in a bundle descriptor.

Table 3.1. The properties set by the `load-bundle-descriptor` task

Ant property	Bundle descriptor value
`osgi-bundle.name`	`name` or `Bundle-Name` header
`osgi-bundle.symbolic-name`	`symbolic-name` or `Bundle-SymbolicName` header
`osgi-bundle.version`	`Bundle-Version` header

See below for an example of the task's usage.

3.3. The "convert-jar-to-bundle" Task

The `convert-jar-to-bundle` task is used to convert normal JARs into OSGi bundles. This is useful when OSGi-ifying third-party libraries.

Example 3.2. Example using `convert-jar-to-bundle`

```
<project name="thirdparty" default="all">

  [..]
  <load-bundle-descriptor descriptor="${basedir}/commons-id/bundle.xml"/>
  <property name="commons-id-org-version" value="0.1-SNAPSHOT"/>
  <property name="commons-id-version" value="0.1.0.SNAPSHOT"/>

  <convert-jar-to-bundle
        file="${basedir}/commons-id-${commons-id-org-version}.jar"
        targetfile=
          "${build.dir}/${osgi-bundle.symbolic-name}-${commons-id-version}.jar"
        descriptor="${basedir}/bundle.xml">
    <attribute name="Bundle-Version" value="${commons-id-version}"/>
  </convert-jar-to-bundle>
  [..]
</project>
```

In this example, Apache Commons Id [http://commons.apache.org/sandbox/id/] is converted to an OSGi bundle. You will note how its version number is being replaced by a valid OSGi version number. The `load-bundle-descriptor` task is used to retrieve the symbolic name of the bundle from the bundle descriptor to use it as part of the target filename of the bundle (`${osgi-bundle.symbolic-name}`).

The `file` and `descriptor` attributes as well as the nested `attribute` element are the same as for the `bundle-manifest` task.

The `targetfile` attribute specifies the target filename of the converted JAR. If it is omitted, the original JAR file will be replaced.

Chapter 4. How it works

This chapter shall quickly show how the OSGi bundle utility works.

4.1. Loading the Bundle Descriptor

As the first step, the bundle descriptor is loaded and interpreted.

Example 4.1. The bundle descriptor from the "create1" example in the distribution

```
<bundle xmlns="http://www.jeremias-maerki.ch/ns/osgi-bundle">
  <name>Example :: Free Space Logging Example</name>
  <symbolic-name>example.freespace</symbolic-name>
  <headers>
    <header name="Bundle-Activator">example.freespace.impl.Activator</header>
  </headers>
  <exports>
    <export>example.freespace</export>
  </exports>
</bundle>
```

The bundle descriptor will usually be loaded by one of the Ant tasks.

Example 4.2. Excerpt from the Ant build

```
<project name="example.freespace" default="all">

  [..]
  <!-- Build manifest from bundle descriptor and add additional entries -->
  <bundle-manifest file="${build.classes.dir}/META-INF/MANIFEST.MF"
        classes="${build.classes.dir}"
        descriptor="${basedir}/bundle.xml">
    <attribute name="Bundle-Version" value="${version}"/>
    <attribute name="Bundle-Vendor" value="${implementation.vendor}"/>
    <attribute name="Bundle-DocURL" value="${implementation.url}"/>
    <attribute name="Implementation-Title"
        value="${Name} (${subproject-name})"/>
    <attribute name="Implementation-Version" value="${version}"/>
    <attribute name="Implementation-Vendor"
        value="${implementation.vendor}"/>
    <attribute name="Implementation-URL"
        value="${implementation.url}"/>
  </bundle-manifest>

  <!-- Now build the actual bundle. -->
  <jar jarfile="${build.dir}/${subproject-name}.jar"
        filesetmanifest="merge" manifestencoding="UTF-8">
    <fileset dir="${build.classes.dir}"/>
    <metainf dir="${project-root.dir}" includes="LICENSE.txt,NOTICE.txt"/>
  </jar>
  [..]
</project>
```

4.2. Analyzing dependencies

The second step involves parsing all compiled Java classes (".class" files) and determining all dependent classes not provided inside the bundle. The external classes are reduced to a set of package names which are then noted as packages to be imported. The class parsing is done through Apache BCEL [http://jakarta.apache.org/bcel/], a byte code engineering library.

4.3. Generating the manifest

Finally, the manifest (`/META-INF/MANIFEST.MF`) for the OSGi bundle is generated from a mixture of information gathered from the bundle descriptor, the attributes specified through the Ant task and the decompiled Java class files.

Example 4.3. The resulting manifest file

```
Manifest-Version: 1.0
Ant-Version: Apache Ant 1.7.1
Created-By: 1.5.0_22-b03 (Sun Microsystems Inc.)
Bundle-ManifestVersion: 2
Build-Id: 20101117114513 (Jeremias [Windows XP 5.1 x86, Java 1.5.0_22-
 b03])
Bundle-Activator: example.freespace.impl.Activator
Bundle-DocURL: http://www.jeremias-maerki.ch
Bundle-Name: Example :: Free Space Logging Example
Bundle-SymbolicName: example.freespace
Bundle-Vendor: Jeremias Märki, Switzerland
Bundle-Version: 1.0.0.dev
Implementation-Title: Free Space Example (example.freespace)
Implementation-URL: http://www.jeremias-maerki.ch
Implementation-Vendor: Jeremias Märki, Switzerland
Implementation-Version: 1.0.0.dev
Export-Package: example.freespace;version=1.0.0.dev
Import-Package: example.freespace;version=1.0.0.dev,org.apache.commons
 .io,org.osgi.framework
```

Looking at the manifest, you can see how the "Free Space" example has a dependency on Apache Commons IO and the OSGi core framework. Some header values have been set from Ant properties, others come from the bundle descriptor (like the `Bundle-Activator`).

Appendix A. The XML Schema for the Bundle Descriptor

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
 Copyright 2009-2010 Jeremias Maerki.

 Licensed under the Apache License, Version 2.0 (the "License");
 you may not use this file except in compliance with the License.
 You may obtain a copy of the License at

      http://www.apache.org/licenses/LICENSE-2.0

 Unless required by applicable law or agreed to in writing, software
 distributed under the License is distributed on an "AS IS" BASIS,
 WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 See the License for the specific language governing permissions and
 limitations under the License.
-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    targetNamespace="http://www.jeremias-maerki.ch/ns/osgi-bundle"
    xmlns:bndl="http://www.jeremias-maerki.ch/ns/osgi-bundle">
  <xs:element name="bundle">
    <xs:complexType>
      <xs:all>
        <xs:element name="name" type="xs:string" minOccurs="0"/>
        <xs:element name="description" type="xs:string" minOccurs="0"/>
        <xs:element name="symbolic-name" type="xs:string" minOccurs="0"/>
        <xs:element name="category" type="xs:string" minOccurs="0"/>
        <xs:element ref="bndl:headers" minOccurs="0"/>
        <xs:element ref="bndl:exports" minOccurs="0"/>
        <xs:element ref="bndl:imports" minOccurs="0"/>
      </xs:all>
    </xs:complexType>
  </xs:element>
  <xs:element name="headers">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="bndl:header" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="header" type="bndl:name-value-pair-type"/>
  <xs:element name="exports">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="bndl:export"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="export">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="bndl:package-name-match-type">
          <xs:attribute name="version" type="bndl:simple-version" use="optional"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
  <xs:simpleType name="fragment-attachment-type">
    <xs:restriction base="xs:NCName">
      <xs:enumeration value="always"/>
      <xs:enumeration value="never"/>
```

```
      <xs:enumeration value="resolve-time"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="package-name-match-type">
    <xs:restriction base="xs:string">
      <xs:pattern value="([a-zA-Z0-9_\-\.\*])*"></xs:pattern>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="simple-version">
    <xs:restriction base="xs:string">
      <xs:pattern value="[0-9]([0-9\.])+"></xs:pattern>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="imports">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="bndl:import"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="import">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="bndl:directive" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="bndl:attribute" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="match" type="bndl:package-name-match-type" use="required"/>
      <xs:attribute name="add" type="xs:boolean" default="false"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="directive" type="bndl:name-value-pair-type"/>
  <xs:element name="attribute" type="bndl:name-value-pair-type"/>
  <xs:complexType name="name-value-pair-type">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="name" type="xs:NCName" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:schema>
```

Index