

Bundlor User Guide

**Ben Hale
Glyn Normington
Juliet Shackell**



1.1.1.RELEASE

Table of Contents

Copyright	iv
License	v
1. Introduction to Bundlor	1
1.1. About Bundlor	1
2. Getting Bundlor	2
2.1. Getting the Bundlor ZIP	2
2.2. Getting Bundlor with Ivy	2
2.3. Getting Bundlor with Maven	2
3. Quickstart	4
3.1. Command Line Quickstart	4
3.2. Apache ANT Quickstart	4
3.3. Apache Maven Quickstart	5
4. Usage	6
4.1. Command-Line Usage	6
Command Syntax	6
Command Line Reference	6
Command Line Parameters	6
Command Line Property Values	7
4.2. Apache ANT Usage	7
ANT Setup	7
ANT Task Reference	8
Task Attributes	8
Inline Manifest Template	9
Inline OSGi Profile	10
Inline Property Values	10
ANT Task Examples	10
Creating a manifest	10
Creating a manifest with placeholder replacement	10
4.3. Apache Maven Usage	11
Maven Setup	11
Maven Plugin Reference	12
Plugin Configuration	12
Inline Manifest Template	13
Inline OSGi Profile	13
Inline Property Values	14
Maven Plugin Examples	14
Creating a manifest	14
Creating a manifest with placeholder replacement	15
5. Manifest Templates	16
5.1. Introduction	16

5.2. Manifest Template Format	16
5.3. Specifying property placeholders	17
5.4. Specifying automatic version expansion of imported packages based on a pattern	18
Re-using version patterns	19
5.5. Example Bundlor Manifest Template	20
6. OSGi Profiles and Bundlor	22
6.1. Overview of OSGi profiles	22
6.2. Using OSGi profiles with Bundlor	22
7. Detecting Manifest Requirements	24
7.1. Java Detection Criteria	24
Export Package	24
Import Package	24
7.2. Spring Context Configuration Detection Criteria	25
Spring Context Values	25
7.3. Blueprint Service Configuration Detection Criteria	26
Blueprint Configuration Values	26
7.4. Web Application File Detection Criteria	27
web.xml Values	27
7.5. Bundle-Classpath File Detection Criteria	28
7.6. JPA Detection Criteria	28
persistence.xml Values	28
orm.xml Values	28
7.7. EclipseLink Detection Criteria	29
eclipselink-orm.xml Values	29
7.8. Hibernate Mapping File Detection Criteria	31
Hibernate Attributes	31
Hibernate Keywords	32
7.9. JSP File Detection Criteria	34
JSP Values	34
7.10. Log4J Configuration Detection Criteria	34
Log4J Configuration Values	34
7.11. Static Resource Detection Criteria	34
8. Detecting Manifest Issues	35
8.1. Import Version Range Warning Criteria	35
8.2. Import of Exported Packages Warning Criteria	35
8.3. Signed JAR Warning Criteria	35
8.4. Versioned Imports Warning Criteria	35
8.5. Versioned Exports Warning Criteria	35
8.6. Bundle-SymbolicName Warning Criteria	35
8.7. Manifest-Version Warning Criteria	35

Copyright

Copyright 2008-2012, VMware Inc.

Licensed Under the terms and conditions of the Eclipse Public License Version 1.0 ("EPL"). A copy of the EPL is available at <http://www.eclipse.org/legal/epl-v10.html>.

License

Eclipse Public License - v 1.0

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS ECLIPSE PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

1. DEFINITIONS

"Contribution" means:

a) in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and
b) in the case of each subsequent Contributor:
i) changes to the Program, and
ii) additions to the Program;
where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

"Contributor" means any person or entity that distributes the Program.

"Licensed Patents" mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Program" means the Contributions distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

2. GRANT OF RIGHTS

a) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form.

b) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.

c) Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances

are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient's responsibility to acquire that license before distributing the Program.

d) Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

3. REQUIREMENTS

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

a) it complies with the terms and conditions of this Agreement; and

b) its license agreement:

i) effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;

ii) effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;

iii) states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and

iv) states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

a) it must be made available under this Agreement; and

b) a copy of this Agreement must be included with each copy of the Program.

Contributors may not remove or alter any copyright notices contained within the Program.

Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

4. COMMERCIAL DISTRIBUTION

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and

indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

5. NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

6. DISCLAIMER OF LIABILITY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. GENERAL

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it

fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. The Eclipse Foundation is the initial Agreement Steward. The Eclipse Foundation may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.

1. Introduction to Bundlor

1.1 About Bundlor

With the increasing focus on OSGi in Enterprise Java, there has been increasing focus on creating OSGi bundles for deployment. When a development team is creating their own bundles, bundlor simplifies the creation and maintenance of the OSGi metadata of each bundle.

Bundlor also helps in the use of third-party enterprise libraries, many of which are not packaged as OSGi bundles. In this case, developers must add OSGi metadata to the library before use.

Bundlor helps in both these scenarios. It can be very hard for developers to keep track of the dependencies needed by a JAR file. Bundlor is a tool that automates the detection of dependencies and the creation of OSGi manifest directives for JARs after their creation. Bundlor takes as input a JAR and a template consisting of a superset of the standard OSGi manifest headers. Bundlor analyses the source code and support files contained in the JAR, applies the template to the results, and generates a manifest.

The use of Bundlor can take different forms, from an Apache ANT task and an Apache Maven plugin, to simple command line execution.

2. Getting Bundlor

2.1 Getting the Bundlor ZIP

Eclipse Virgo Bundlor is distributed as a ZIP file.

1. Download the ZIP file from the Virgo download page.

The Virgo download page is located at <http://www.eclipse.org/virgo/download/>.

2.2 Getting Bundlor with Ivy

Eclipse Virgo Bundlor can be obtained from an Ivy repository.

1. Add the Virgo resolver to the `ivysettings.xml` file

```
<url name="eclipse.virgo.build.read.resolver">
  <ivy pattern="http://build.eclipse.org/rt/virgo/ivy/bundles/release/[organisation]/[module]/[revision]/[artifact]">
  <artifact pattern="http://build.eclipse.org/rt/virgo/ivy/bundles/release/[organisation]/[module]/[revision]/[artifact]">
</url>
```

2. Download the Eclipse Virgo Bundlor dependency in the `build.xml` file

```
<ivy:cachedpath resolveId="bundlor.classpath" pathid="bundlor.classpath" organisation="org.eclipse.virgo.bundlor"
  module="org.eclipse.virgo.bundlor.ant" revision="1.1.1.RELEASE" conf="ant" inline="true"
  type="jar" log="download-only"/>
```

2.3 Getting Bundlor with Maven

Eclipse Virgo Bundlor can be obtained from a Maven repository.

1. Add the Eclipse Virgo build and SpringSource Enterprise Bundle Repository resolvers to the `pom.xml` file

```
<repository>
  <id>eclipse.virgo.build.bundles.release</id>
  <name>Eclipse Virgo Build</name>
  <url>http://build.eclipse.org/rt/virgo/maven/bundles/release</url>
</repository>
<repository>
  <id>com.springsource.repository.bundles.external</id>
  <name>SpringSource Enterprise Bundle Repository - External Bundle Releases</name>
  <url>http://repository.springsource.com/maven/bundles/external</url>
</repository>
```

2. Add a dependency to the `pom.xml` file

```
<dependencies>
  <dependency>
    <groupId>org.eclipse.virgo.bundlor</groupId>
    <artifactId>org.eclipse.virgo.bundlor.maven</artifactId>
    <version>1.1.1.RELEASE</version>
    <scope>compile</scope>
  </dependency>
</dependencies>
```

3. Quickstart

3.1 Command Line Quickstart

The command line client allows Bundlor to be run from the command line.

1. Change directory to the `$BUNDLOR_HOME/bin` directory where `$BUNDLOR_HOME` is a directory into which the bundlor ZIP file distribution has been unzipped.
2. Run `bundlor.sh` or `bundlor.bat` scripts. See Section 4.1, “Command-Line Usage” for details.

```
% ./bundlor.sh \  
-i ./org.springframework.integration.jar \  
-m ./template.mf \  
-o ./target/org.springframework.integration.jar  
  
Transformed bundle written to ./target/org.springframework.integration.jar  
%
```

3.2 Apache ANT Quickstart

The ANT task allows Bundlor to be run from inside any ANT based build system.

1. Define a bundlor namespace

```
<project name="bundlor-sample-ant"  
  xmlns:bundlor="antlib:org.eclipse.virgo.bundlor.ant">
```

2. Import the bundlor task into your build

```
<target name="bundlor.init">  
  <ivy:cachedpath resolveId="bundlor.classpath" pathid="bundlor.classpath" organisation="org.eclipse.virgo.bundlor"  
    module="org.eclipse.virgo.bundlor.ant" revision="1.1.1.RELEASE" conf="ant" inline="true"  
    type="jar" log="download-only"/>  
  <taskdef resource="org/eclipse/virgo/bundlor/ant/antlib.xml" uri="antlib:org.eclipse.virgo.bundlor.ant"  
    classpathref="bundlor.classpath"/>  
</target>
```

3. Use the bundlor task. See Section 4.2, “Apache ANT Usage” for details about the parameters of the task.

```
<bundlor:bundlor  
  inputPath="${basedir}/target/classes"  
  outputPath="${basedir}/target/classes"  
  bundleVersion="1.0.2.BUILD-${timestamp}"  
  manifestTemplatePath="${basedir}/template.mf"/>
```

3.3 Apache Maven Quickstart

The Maven plugin allows Bundlor to be run from inside any Maven project.

1. Add the Eclipse Virgo build and SpringSource Enterprise Bundle Repository to the `pom.xml` file

```
<pluginRepositories>
  <pluginRepository>
    <id>eclipse.virgo.build.bundles.release</id>
    <name>Eclipse Virgo Build</name>
    <url>http://build.eclipse.org/rt/virgo/maven/bundles/release</url>
  </pluginRepository>
  <pluginRepository>
    <id>com.springsource.repository.bundles.external</id>
    <name>SpringSource Enterprise Bundle Repository - External Bundle Releases</name>
    <url>http://repository.springsource.com/maven/bundles/external</url>
  </pluginRepository>
  ...
</pluginRepositories>
```

2. Use the `bundlor` plugin in the `pom.xml` file. See Section 4.3, “Apache Maven Usage” for details about the parameters of the plugin.

```
<build>
  <plugins>
    <plugin>
      <groupId>org.eclipse.virgo.bundlor</groupId>
      <artifactId>org.eclipse.virgo.bundlor.maven</artifactId>
      <version>1.1.1.RELEASE</version>
      <executions>
        <execution>
          <id>bundlor</id>
          <goals>
            <goal>bundlor</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jar-plugin</artifactId>
      <version>2.4</version>
      <configuration>
        <archive>
          <manifestFile>
            target/classes/META-INF/MANIFEST.MF
          </manifestFile>
        </archive>
      </configuration>
    </plugin>
    ...
  </plugins>
  ...
</build>
```

4. Usage

4.1 Command-Line Usage

The command line client allows Bundlor to be run from the command line of any platform

Command Syntax

To use Bundlor run the following for UNIX and Windows respectively.

```
$BUNDLOR_HOME/bin/bundlor.sh [options]
```

```
%BUNDLOR_HOME%\bin\bundlor.bat [options]
```

Command Line Reference

Command Line Parameters

The following table lists all the parameters that you can specify for the `bundlor` command line client.

Table 4.1. Attributes

Attribute	Description	Required
-f	Whether Bundlor should cause a build failure when there are warnings about the resulting manifest	No - defaults to <code>false</code>
-i <path>	The path to the input to create a manifest for. This can either be a directory or a JAR file.	Yes
-m <path>	The path to the manifest template. See Chapter 5, <i>Manifest Templates</i> for details.	No
-p <path>	The path to the OSGi profile. See ??? for details.	No
-o <path>	The path to write the manifest to. This can either be a directory, a JAR file, or not specified.	No - defaults to <code>System.out</code>

Attribute	Description	Required
	<p>If a directory is specified, the manifest will be written to <code>\${directory}/META-INF/MANIFEST.MF</code>.</p> <p>If a JAR file is specified, the manifest will be written as the manifest for that JAR file.</p> <p>If nothing is specified, the manifest will be written to <code>System.out</code>.</p>	
<code>-r <path></code>	The path to a properties file used for substitution. See Section 5.3, “Specifying property placeholders” for details.	No

Command Line Property Values

Property substitution values can be optionally specified on the command line instead of as an external file using the `-Dproperty=value` parameter.

```
% ./bundlor.sh \
-i ./org.springframework.integration.jar \
-m ./template.mf \
-o ./target/org.springframework.integration.jar \
-Dname="Spring Integration"

Transformed bundle written to ./target/org.springframework.integration.jar
%
```

See Section 5.3, “Specifying property placeholders” for details.

4.2 Apache ANT Usage

The ANT task allows you to run Bundlor from inside any ANT based build system

ANT Setup

The following procedure shows how to set up Bundlor inside of an existing ANT build file

1. Define a bundlor namespace

```
<project name="bundlor-sample-ant"
  xmlns:bundlor="antlib:org.eclipse.virgo.bundlor.ant">
```

2. Import the bundlor task into your build

```
<target name="bundlor.init">
  <ivy:classpath resolveId="bundlor.classpath" pathid="bundlor.classpath" organisation="org.eclipse.virgo.bundlor"
    module="org.eclipse.virgo.bundlor.ant" revision="1.1.1.RELEASE" conf="ant" inline="true"
    type="jar" log="download-only"/>
  <taskdef resource="org/eclipse/virgo/bundlor/ant/antlib.xml" uri="antlib:org.eclipse.virgo.bundlor.ant"
    classpathref="bundlor.classpath"/>
</target>
```

This example uses a very simplistic method for building the bundlor task classpath. It is possible to use a dependency manager such as Ivy to better manage the classpath of Bundlor.

3. Use the bundlor task, as shown in the following example. See the section called “ANT Task Reference” for details about the parameters of the task.

```
<bundlor:bundlor
  inputPath="${basedir}/target/classes"
  outputPath="${basedir}/target/classes"
  bundleVersion="1.0.2.BUILD-${timestamp}"
  manifestTemplatePath="${basedir}/template.mf" >
  <property name="name" value="${ant.project.name}" />
</bundlor:bundlor>
```

ANT Task Reference

Task Attributes

The following table lists all the attributes that you can specify for the bundlor ANT task.

Table 4.2. Attributes

Attribute	Description	Required
bundleSymbolicName	The OSGi Bundle-SymbolicName for the resulting manifest	No
bundleVersion	The OSGi Bundle-Version for the resulting manifest	No
enabled	Whether Bundlor should create a manifest	No - defaults to true
failOnWarnings	Whether Bundlor should cause a build failure when there are warnings about the resulting manifest	No - defaults to false
inputPath	The path to the input to create a manifest for. This can either be a	Yes

Attribute	Description	Required
	directory or a JAR file.	
manifestTemplatePath	The path to the manifest template. See Chapter 5, <i>Manifest Templates</i> for details.	No
OSGiProfilePath	The path to the OSGi profile. See Chapter 6, <i>OSGi Profiles and Bundlor</i> for details.	No
outputPath	<p>The path to write the manifest to. This can either be a directory, a JAR file, or not specified.</p> <p>If a directory is specified, the manifest will be written to <code>\${directory}/META-INF/MANIFEST.MF</code>.</p> <p>If a JAR file is specified, the manifest will be written as the manifest for that JAR file.</p> <p>If nothing is specified, the manifest will be written to <code>System.out</code>.</p>	No - defaults to <code>System.out</code>
propertiesPath	The path to a properties file used for substitution. See Section 5.3, “Specifying property placeholders” for details.	No

Inline Manifest Template

Manifest templates can be optionally specified inline instead of as an external file using the `<manifestTemplate/>` element.

```
<bundlor:bundlor>
  <manifestTemplate>
    Bundle-ManifestVersion: 2
    Bundle-Name: Bundlor Core
    Bundle-SymbolicName: org.eclipse.virgo.bundlor
    Bundle-Version: 0
  </manifestTemplate>
</bundlor:bundlor>
```

See Chapter 5, *Manifest Templates* for details.

Inline OSGi Profile

OSGi profiles can be optionally specified inline instead of as an external file using the `<OSGiProfile/>` element.

```
<bundlor:bundlor>
  <OSGiProfile>
    org.OSGi.framework.system.packages = \
    org.eclipse.virgo.osgi.extensions.equinox.hooks,\
    javax.accessibility,\
    javax.activation,\
    javax.activation.version="1.1.1",\
    javax.activity,\
    javax.annotation,\
    ...

    org.OSGi.framework.bootdelegation = \
    org.eclipse.virgo.kernel.authentication,\
    com.sun.*,\
    javax.xml.*,\
    ...
  </OSGiProfile>
</bundlor:bundlor>
```

See Chapter 6, *OSGi Profiles and Bundlor* for details.

Inline Property Values

Property substitution values can be optionally specified inline instead of as an external file using the `<property/>` and `<propertySet/>` elements.

```
<bundlor:bundlor>
  <property name="bundle.name" value="Kernel test bundle"/>
  <property name="bundle.version" value="1.0.2.BUILD-${timestamp}"/>
  <propertyset>
    <propertyref builtin="all"/>
  </propertyset>
</bundlor:bundlor>
```

See Section 5.3, “Specifying property placeholders” for details.

ANT Task Examples

Creating a manifest

```
<bundlor:bundlor
  inputPath="${basedir}/target/classes"
  outputPath="${basedir}/target/classes"
  bundleVersion="1.0.2.BUILD-${timestamp}"
  manifestTemplatePath="${basedir}/template.mf"/>
```

Creating a manifest with placeholder replacement

```
<bundlor:bundlor
```

```

    inputPath="${basedir}/target/classes"
    outputPath="${basedir}/target/target/classes"
    bundleVersion="1.0.2.BUILD-${timestamp}"
    manifestTemplatePath="${basedir}/template.mf">
    <property name="bundle.name" value="Kernel test bundle"/>
    <property name="bundle.version" value="1.0.2.BUILD-${timestamp}"/>
  </bundlor:bundlor>

```

4.3 Apache Maven Usage

The Maven plugin allows Bundlor to be run from inside any Maven project.

Maven Setup

The following procedure shows how to set up Bundlor inside of an existing Maven POM file.

1. Add the Eclipse Virgo build and SpringSource Enterprise Bundle Repository to the `pom.xml` file.

```

<pluginRepositories>
  <pluginRepository>
    <id>eclipse.virgo.build.bundles.release</id>
    <name>Eclipse Virgo Build</name>
    <url>http://build.eclipse.org/rt/virgo/maven/bundles/release</url>
  </pluginRepository>
  <pluginRepository>
    <id>com.springsource.repository.bundles.external</id>
    <name>SpringSource Enterprise Bundle Repository - External Bundle Releases</name>
    <url>http://repository.springsource.com/maven/bundles/external</url>
  </pluginRepository>
  ...
</pluginRepositories>

```

2. Use the bundlor plugin, as shown in the following example. See the section called “Maven Plugin Reference” for details about the parameters of the plugin.

```

<build>
  <plugins>
    <plugin>
      <groupId>org.eclipse.virgo.bundlor</groupId>
      <artifactId>org.eclipse.virgo.bundlor.maven</artifactId>
      <version>1.1.1.RELEASE</version>
      <executions>
        <execution>
          <id>bundlor</id>
          <goals>
            <goal>bundlor</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jar-plugin</artifactId>
      <version>2.4</version>
      <configuration>
        <archive>
          <manifestFile>
            target/classes/META-INF/MANIFEST.MF
          </manifestFile>
        </archive>
      </configuration>
    </plugin>
  </plugins>
</build>

```

```

        </manifestFile>
      </archive>
    </configuration>
  </plugin>
  ...
</plugins>
...
</build>

```

Maven Plugin Reference

Plugin Configuration

The following table lists all the elements that you can specify for the bundlor Maven plugin.

Table 4.3. Elements

Attribute	Description	Required
bundleSymbolicName	The OSGi Bundle-SymbolicName for the resulting manifest	No - defaults to <code>\${project.artifactId}</code>
bundleVersion	The OSGi Bundle-Version for the resulting manifest	No - defaults to <code>\${project.version}</code>
enabled	Whether Bundlor should create a manifest	No - defaults to true
failOnWarnings	Whether Bundlor should cause a build failure when there are warnings about the resulting manifest	No - defaults to false
inputPath	The path to the input to create a manifest for. This can either be a directory or a JAR file.	No - defaults to <code>\${project.build.outputDirectory}</code>
manifestTemplatePath	The path to the manifest template. See Chapter 5, <i>Manifest Templates</i> for details.	No - defaults to <code>\${basedir}/template.mf</code>
OSGiProfilePath	The path to the OSGi profile. See Chapter 6, <i>OSGi Profiles and Bundlor</i> for details.	No
outputPath	The path to write the manifest to. This can either be a directory, a JAR file, or not specified.	No - defaults to <code>\${project.build.outputDirectory}</code>

Attribute	Description	Required
	<p>If a directory is specified, the manifest will be written to <code>\${directory}/META-INF/MANIFEST.MF</code>.</p> <p>If a JAR file is specified, the manifest will be written as the manifest for that JAR file.</p>	
propertiesPath	The path to a properties file used for substitution. See Section 5.3, “Specifying property placeholders” for details.	No

Inline Manifest Template

Manifest templates can be optionally specified inline instead of as an external file using the `<manifestTemplate/>` element.

```
<execution>
  <id>bundlor</id>
  <goals>
    <goal>bundlor</goal>
  </goals>
  <configuration>
    <manifestTemplate>
Bundle-ManifestVersion: 2
Bundle-Name: Bundlor Core
Bundle-SymbolicName: org.eclipse.virgo.bundlor
Bundle-Version: 0
    </manifestTemplate>
  </configuration>
</execution>
```

See Chapter 5, *Manifest Templates* for details.

Inline OSGi Profile

OSGi profiles can be optionally specified inline instead of as an external file using the `<OSGiProfile/>` element.

```
<execution>
  <id>bundlor</id>
  <goals>
    <goal>bundlor</goal>
  </goals>
  <configuration>
    <OSGiProfile>
org.OSGi.framework.system.packages = \
org.eclipse.virgo.osgi.extensions.equinox.hooks,\
javax.accessibility,\
javax.activation,\
```

```
javax.activation;version="1.1.1",\  
javax.activity,\  
javax.annotation,\  
...  
  
org.osgi.framework.bootdelegation = \  
org.eclipse.virgo.kernel.authentication,\  
com.sun.*,\  
javax.xml.*,\  
...  
    </OSGiProfile>  
    </configuration>  
</execution>
```

See Chapter 6, *OSGi Profiles and Bundlor* for details.

Inline Property Values

Property substitution values can be optionally specified inline instead of as an external file using the `<properties/>` element.

```
<project>  
...  
  <properties>  
    <bundle.name>${project.name}</bundle.name>  
    <bundle.version>2.0.0.RELEASE</bundle.version>  
  </properties>  
  ...  
</project>
```

See Section 5.3, “Specifying property placeholders” for details.

Maven Plugin Examples

Creating a manifest

```
<project>  
...  
  <build>  
    <plugins>  
      <plugin>  
        <groupId>org.eclipse.virgo.bundlor</groupId>  
        <artifactId>org.eclipse.virgo.bundlor.maven</artifactId>  
        <executions>  
          <execution>  
            <id>bundlor</id>  
            <goals>  
              <goal>bundlor</goal>  
            </goals>  
          </execution>  
        </executions>  
      </plugin>  
    </plugins>  
  </build>  
  ...  
</project>
```

Creating a manifest with placeholder replacement

```
<project>
...
  <properties>
    <bundle.name>${project.name}</bundle.name>
    <bundle.version>2.0.0.RELEASE</bundle.version>
  </properties>
...
  <build>
    <plugins>
      <plugin>
        <groupId>org.eclipse.virgo.bundlor</groupId>
        <artifactId>org.eclipse.virgo.bundlor.maven</artifactId>
        <executions>
          <execution>
            <id>bundlor</id>
            <goals>
              <goal>bundlor</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
...
</project>
```

5. Manifest Templates

5.1 Introduction

A manifest template is a file that Bundlor uses during the generation of OSGi-compliant manifest entries in a JAR's manifest. The format of the manifest template is the same as that of a standard Java manifest file, i.e. a series of 'key: value' pairs.

From this template, Bundlor recognizes a specific set of directives and uses them to generate the OSGi-compliant manifest entries. Bundlor will also add any other headers that are specified in the template to the generated manifest. This is typically used to specify things like the bundle's symbolic name and version.

You can also specify property placeholders, or variables, in your manifest template that Bundlor substitutes with actual values at runtime. With this feature, your manifest templates become more dynamic and useful across a variety of your projects. A particularly handy use for this feature is to tell Bundlor to automatically expand versions of imports based on a pattern of your choosing. See Section 5.3, “Specifying property placeholders” for details.

5.2 Manifest Template Format

The following table lists the headers you can add to the manifest template, in addition to the standard manifest headers.

Table 5.1. Headers for Manifest Template

Header	Description
Excluded-Exports	A comma-separated list of packages that must not be added to the manifest's <code>Export-Package</code> header. This is useful for preventing implementation packages from being exported.
Excluded-Imports	By default, Bundlor adds imports for every package that Bundlor determines is referenced by the code or for special files in the jar. Use this header to specify a comma-separated list of packages for which imports Bundlor will <i>not</i> generate.
Export-Template	By default, Bundlor versions all exported packages at the specified <code>Bundle-Version</code> . Use this header to specify that individual exported packages be exported at different versions. For

Header	Description
	example, <code>Export-Template: com.foo.*;version="1.5"</code> results in Bundlor versioning any <code>Export-Package</code> entries for <code>com.foo</code> or its subpackages at 1.5.
Ignored-Existing-Headers	If the JAR for which you are generating a manifest already contains an OSGi-compliant manifest, use this template header to list headers in the original manifest which Bundlor should ignore.
Import-Template	Use this header to augment package imports that Bundlor generates via bytecode and special file analysis. Typically you use the header to version the import and, in some cases, to mark them as optional. When you use this header to version the import, you can optionally specify a version expansion pattern so that Bundlor sets the version to a range rather than a single version. To use the header, set its value to a comma-separated list of package names and attributes.
Version-Patterns	Use this header to declare one or more version expansion patterns and give each one a name. You can then use these named patterns in the <code>Import-Template</code> header if you want to specify an expansion pattern for the version of an imported package. This feature is described in detail later in this section.

A wildcard '*' at the end of the package name is supported to match multiple packages. For example, the header `Import-Template: com.foo;version=[1.0,2.0);resolution:=optional,com.bar.*;version="[1.5,1.6)"` will cause any import generated for the `com.foo` package to be versioned at 1.0 (inclusive) to 2.0 (exclusive) and to be considered optional, and for any import of `com.bar` or its sub-packages to be versioned at 1.5 (inclusive) to 1.6 (exclusive).

5.3 Specifying property placeholders

To specify a property placeholder in your manifest template, use the form `${property.name}`, where `property.name` refers to the name of the property placeholder. The method in which the manifest template actually gets the value of the property placeholder at runtime depends on the Bundlor front end you use (command line, ANT, or Maven); the details are described later.

The following example shows how to use a property placeholder for the `Bundle-Name` manifest header rather than a literal.

```
Bundle-Name: ${bundle.name}
```

5.4 Specifying automatic version expansion of imported packages based on a pattern

When you use the `Import-Template` template header to augment package imports that Bundlor generates in the manifest file, you use the `version` attribute to specify a version range of the imported package.

```
Import-Template:
org.eclipse.virgo.kernel.*;version="[1.2.0, 2.0.0)"
org.apache.commons.logging;version="[1.1.1, 2.0.0)"
```

The preceding example specifies that Bundlor should import the `org.eclipse.virgo.kernel.*` packages in the range `[1.2.0, 2.0.0)` and the `org.apache.commons.logging` package in the range `[1.1.1, 2.0.0)` in the generated manifest file. This works just fine for many use cases, but sometimes the use of literal versions in this manner can be restrictive.

In order to make the manifest template more dynamic and useful, you can specify that Bundlor automatically expand the package version into a version range using an expansion pattern of your choosing. The pattern uses as a base a property placeholder that you define (as described in Section 5.3, “Specifying property placeholders”) and set to a valid OSGi version number. Then, based on the expansion pattern you specify, Bundlor generates a version range using the 4 parts of an OSGi version: major, minor, micro, and qualifier.

The way to tell Bundlor to automatically expand a package import version is to specify the property placeholder to the right of the `version` directive of the package in the `Import-Template` header, and then within the property placeholder, specify the pattern for both sides of the version range. The following manifest template snippet shows how to use this feature; the example is described in detail after the table.

```
Import-Template:
org.eclipse.virgo.kernel.*;version="${org.eclipse.virgo.kernel:[=.=.=, +1.0.0)}",
org.apache.commons.logging.*;version="${org.apache.commons.logging:[=.=.=, =.=.+1)}"
```

The following table lists the symbols you can use in the expansion pattern.

Table 5.2. Expansion Pattern Symbols

Symbol	Description	Location Allowed
=	Use the same value from the variable.	Valid only in the first three segments (major, minor, micro)

Symbol	Description	Location Allowed
		of the version pattern.
[+/-]n	Adjust the value from the variable by this amount. For example, +1 means to add 1 to the value from the variable.	Valid only in the first three segments (major, minor, micro) of the version pattern.
n	Substitute this value for the one in the variable. Typically you only use this for putting in a 0.	Valid only in the first three segments (major, minor, micro) of the version pattern.
Any legal qualifier value	Substitute this value for the one in the variable.	Valid only in the fourth (qualifier) segment of the version pattern.

Based on the descriptions of the symbols, we can now understand how the examples above work. First assume that you have set the property `${org.eclipse.virgo.kernel}` to the value `1.2.0`. Based on the expansion pattern, Bundlor sets the version range of the imported `org.eclipse.virgo.kernel.*` packages to `[1.2.0, 2.0.0)`. The pattern in this case first specifies that the beginning of the version range stay exactly the same as the value of the property. The pattern then specifies that at the end of the version range, the major part of the version should be one integer larger than what the property is originally set to (1); the pattern then specifies that the minor and micro segments of the version both be set to 0.

Similarly, assume that you set the `${org.apache.commons.logging}` property to `1.4.0`. Bundlor generates a version range of `[1.4.0, 1.4.1)`. Again, the beginning of the range is exactly the same as the property value. The pattern specifies that, in the end of the range, only the micro segment of the version increase by one; the major and minor segments stay the same.

Re-using version patterns

If you use the same version expansion pattern for several imports, you can name the pattern using the `Version-Patterns` header in the manifest template, and then use this name in the particular import of `Import-Template`.

Use the form `pattern.name;pattern="pattern"` to specify a named pattern, where `pattern.name` is the name of the pattern and `pattern` is the pattern, such as `[.=.=.=, +1.0.0)`.

```
Version-Patterns:
apache;pattern="[.=.=.=, +1.0.0)",
hibernate;pattern="[.=.=.=, =.=.+1)"
```

The preceding example shows two named patterns: `apache` and `hibernate`. The `apache` pattern specifies a version range from the one provided in the property up to but not including the next major

version. The `hibernate` pattern specifies a version range of the one provided up to but not including the next micro version.

To use a named pattern, simply substitute it in the `Import-Template` header in the place where you would put the in-line pattern.

```
Import-Template:
org.apache.commons.codec.*;version="${org.apache.commons.codec:apache}",
org.apache.commons.logging.*;version="${org.apache.commons.logging:apache}",
org.hibernate.*;version="${org.hibernate:hibernate}"
org.myorg.*;version="${org.myorg:[]=.=.=, =.+1.0.=)}"
```

In the example, the `apache` named pattern is used twice, for the two `org.apache` imports, and the `hibernate` pattern is used once. Also note that you can also include an import whose version is specified with an in-line pattern.

5.5 Example Bundlor Manifest Template

The following shows a simple example of a Bundlor manifest template file, with a description after the sample.

```
Bundle-ManifestVersion: 2
Bundle-SymbolicName: org.springframework.binding
Bundle-Name: ${bundle.name}
Import-Package:
  ognl;version="[2.6.9, 3.0.0)";resolution:=optional,
  org.jboss.el;version="[2.0.0, 3.0.0)";resolution:=optional
Import-Template:
  org.springframework.*;version="[2.5.4.A, 3.0.0)",
  org.apache.commons.logging;version="[1.1.1, 2.0.0)",
  javax.el;version="[2.1.0, 3.0.0)";resolution:=optional,
  ognl;version="[2.6.9, 3.0.0)";resolution:=optional,
  org.jboss.el;version="[2.0.0, 3.0.0)";resolution:=optional
```

The headers marked in bold are required in all manifest templates unless the jar already contains a manifest with those headers.

- **Bundle-ManifestVersion:** This should always be 2
- **Bundle-SymbolicName:** specifies a unique name for the bundle of `org.springframework.binding`
- **Bundle-Name:** specifies a human-readable name for the bundle. The example shows how to use a property placeholder `${bundle.name}`, which at runtime Bundlor will substitute with an actual value, such as `Spring Binding`.
- **Import-Package:** hard-codes two packages that will be imported (`ognl` and `org.jboss.el` in the generated manifest. Bundlor isn't infallible; this lets you add imports that it misses.
- **Import-Template:** specifies the versions for the package imports that Bundlor generates, marking

`javax.el`, `ognl`, and `org.jboss.el` optional.

6. OSGi Profiles and Bundlor

When managing and transforming bundles it can become difficult to remember which packages are boot delegated, which are exported from the system bundle, and which are from other bundles in your system. This information is important because you typically do not want to import packages into your own application that are boot delegated, you want to import system bundle packages at version 0, and you want to define custom imports for all the rest of the bundles. Trying to keep track of which packages are in each of these categories can be error prone; similarly, defining template entries for them in your manifest template can be time-consuming and tedious.

To solve this problem, you can specify that Bundlor take an [OSGi profile](#) as input and automatically add template entries for boot delegated packages and system bundles. These import entries would ignore boot-delegated packages and set the version of system bundles to `version="0"`. This feature is available for all Bundlor front ends: command-line, ANT and Maven.

6.1 Overview of OSGi profiles

An OSGi profile defines the packages that a particular OSGi runtime (such as Virgo) exports from the system bundle and the packages that it delegates to the boot class loader. An OSGi profile isn't an actual file; rather, it is two properties that are well known to an OSGi runtime. However, when you pass these properties to Bundlor, you pass them as a file, as described in the next section. The properties that make up an OSGi profile are as follows.

- The `org.OSGi.framework.system.packages` property defines the packages exported from the system bundle.
- The `org.OSGi.framework.bootdelegation` property defines the packages that are boot delegated.

If you are using Virgo as your OSGi runtime, see the file `$VIRGO_HOME/configuration/java6-server.profile` for its OSGi profile, where `$VIRGO_HOME` refers to the main installation directory of Virgo. If you are using another OSGi runtime, such as Equinox, then see their documentation for their OSGi profile.

For additional information about the syntax of the values of these properties, see the [OSGi Core specification](#).

6.2 Using OSGi profiles with Bundlor

The first step in using OSGi profiles with Bundlor is to create a file that contains a textual representation of the two properties that make up an OSGi profile: `org.OSGi.framework.system.packages` and `org.OSGi.framework.bootdelegation`. What you include in this file is up to you, but

typically you start with the OSGi profile of the OSGi runtime you are using, and then customize it to fit your environment.

If you are using Virgo as your OSGi runtime, you can start by copying the section of the file `$VIRGO_HOME/configuration/java6-server.profile` that refers to the two properties and pasting it into your text file. If you are using another runtime, consult their documentation.

The following snippet shows a partial OSGi profile for Virgo; for clarity only a few packages are shown. The example shows the format in which you should create your own OSGi profile file.

```
org.OSGi.framework.system.packages = \  
  org.eclipse.virgo.osgi.extensions.equinox.hooks,\  
  javax.accessibility,\  
  javax.activation,\  
  javax.activation;version="1.1.1",\  
  javax.activity,\  
  javax.annotation,\  
  ...  
  
org.OSGi.framework.bootdelegation = \  
  org.eclipse.virgo.kernel.authentication,\  
  com.sun.*,\  
  javax.xml.*,\  
  ...
```

Once you've created your OSGi profile file, the method of passing it to Bundlor depends on the front end you are using to generate a manifest. For detailed information about using the various front ends, see Chapter 4, *Usage*.

7. Detecting Manifest Requirements

Bundlor's main function is to scan an existing JAR file and determine its runtime dependencies. With this information it can then generate the OSGi-compliant manifest headers needed for proper runtime operation. This analysis is comprised of looking for class references and class names in Java classes and certain well-known file types.

7.1 Java Detection Criteria

Bundlor scans any Java class it can find in the artifact created by the underlying build system. This means that if a build process has custom behavior (i.e. weaving with AspectJ or `jarjaring`), Bundlor will be able to see and analyze the changes made by that process as long as the changes are in the artifact created by the build system.

There are a number of places in a Java class that another Java type can be referenced from. Bundlor detects these references and adds manifest requirements for them.

Export Package

Bundlor exports any package that contains a class.

Import Package

The following is a list of the places that Bundlor will search for type names

- Declared Type Superclass Types
- Declared Type Implemented Interfaces Types
- Declared Type Annotation Types
- Declared Field Types
- Declared Field Values Types
- Declared Method Argument Types
- Declared Method Return Types
- Declared Method Exception Types
- Declared Method Annotation Types
- Reference To Field Owner Type

- Reference To Field Type
- Declared Local Variable Type
- Reference to Method Declaring Type
- Reference to Method Return Type
- Reference to Method Argument Types
- Allocation of Array Type
- Declared Parameter Annotation Types
- Caught Exception Type
- Instantiated Type
- Cast Target Type
- Instanceof Type
- Declared Constant Type

7.2 Spring Context Configuration Detection Criteria

Bundlor scans for Spring context configuration files. If it detects this file type, it scans the file for a number of values that contain class names.

Spring Context Values

Using XPath syntax, the following is a list of values searched for type names

- `//beans:bean/@class`
- `//aop:declare-parents/@implement-interface`
- `//aop:declare-parents/@default-impl`
- `//context:load-time-weaver/@weaver-class`
- `//context:component-scan/@name-generator`
- `//context:component-scan/@scope-resolver`
- `//jee:jndi-lookup/@expected-type`

- `//jee:jndi-lookup/@proxy-interface`
- `//jee:remote-slsb/@home-interface`
- `//jee:remote-slsb/@business-interface`
- `//jee:local-slsb/@business-interface`
- `//jms:listener-container/@container-class`
- `//lang:jruby/@script-interfaces`
- `//lang:bsh/@script-interfaces`
- `//oxm:class-to-be-bound/@name`
- `//oxm:jibx-marshaller/@target-class`
- `//osgi:reference/@interface`
- `//osgi:service/@interface`
- `//util:list/@list-class`
- `//util:map/@map-class`
- `//util:set/@set-class`
- `//webflow:flow-builder/@class`
- `//webflow:attribute/@type`
- `//osgi:service/osgi:interfaces/beans:value`
- `//osgi:reference/osgi:interfaces/beans:value`
- `//context:component-scan/@base-package`

7.3 Blueprint Service Configuration Detection Criteria

Bundlor scans for Blueprint Service configuration files. If it detects this file type, it scans the file for a number of values that contain class names.

Blueprint Configuration Values

Using XPath syntax, the following is a list of values searched for type names

- `//bp:bean/bp:argument/@type`
- `//bp:bean/@class`
- `//bp:service/@interface`
- `//bp:reference/@interface`
- `//bp:reference-list/@interface`
- `//bp:map/@key-type`
- `//bp:map/@value-type`
- `//bp:list/@value-type`
- `//bp:set/@value-type`
- `//bp:array/@value-type`
- `//bp:interfaces/bp:value`

7.4 Web Application File Detection Criteria

Bundlor scans for the Servlet `web.xml` file located in the `WEB-INF` directory. If it detects this file, it scans the file for a number of values that contain class names.

web.xml Values

Using XPath syntax, the following is a list of values searched for type names

- `//context-param/param-values`
- `//filter/filter-classs`
- `//filter/init-param/param-values`
- `//listener/listener-classs`
- `//servlet/servlet-classs`
- `//servlet/init-param/param-values`
- `//error-page/exception-types`
- `//env-entry/env-entry-types`

- `//ejb-ref/homes`
- `//ejb-ref/remotes`
- `//ejb-local-ref/local-homes`
- `//ejb-local-ref/locals`
- `//service-ref/service-interfaces`
- `//resource-ref/res-types`
- `//resource-env-ref/resource-env-ref-types`
- `//message-destination-ref/message-destination-type`

7.5 Bundle-Classpath File Detection Criteria

Bundlor scans for JAR files located anywhere in the bundle. If it detects this file, it runs the entire set of analyzers against it. The imports and exports of the JAR file are added to the bundle's manifest and the JAR file is placed on the bundle's Bundle-Classpath.

7.6 JPA Detection Criteria

Bundlor scans for the JPA `persistence.xml` and `orm.xml` files located in the `META-INF` directory. If it detects this file it scans the file for a number of values that contain class names and package names. If the class name is unqualified (i.e. has no `'.'` in it), the classname is prepended with the content of the entity-mapping tag's package element.

persistence.xml Values

Using XPath syntax, the following is a list of values searched for type names

- `//persistence-unit/provider`
- `//persistence-unit/class`

orm.xml Values

Using XPath syntax, the following is a list of values searched for type names

- `//element-collection/@target-class`

- `//embeddable/@class`
- `//entity/@class`
- `//entity-listener/@class`
- `//entity-result/@entity-class`
- `//id-class/@class`
- `//many-to-many/@target-entity`
- `//many-to-one/@target-entity`
- `//map-key-class/@class`
- `//mapped-superclass/@class`
- `//named-native-query/@result-class`
- `//one-to-many/@target-entity`
- `//one-to-one/@target-entity`

7.7 EclipseLink Detection Criteria

Bundlor scans for the EclipseLink `eclipselink-orm.xml` files located in the `META-INF` directory. If it detects this file it scans the file for a number of values that contain class names and package names. If the class name is unqualified (i.e. has no `'.'` in it), the classname is prepended with the content of the `entity-mapping` tag's `package` element.

`eclipselink-orm.xml` Values

Using XPath syntax, the following is a list of values searched for type names

- `//cache-interceptor/@class`
- `//converter/@class`
- `//copy-policy/@class`
- `//customizer/@class`
- `//discriminator-class/@value`
- `//id-class/@class`

- `//element-collection/@target-class`
- `//entity/@class`
- `//entity-listener/@class`
- `//entity-result/@entity-class`
- `//embeddable/@class`
- `//many-to-many/@target-entity`
- `//many-to-one/@target-entity`
- `//map-key-class/@class`
- `//mapped-superclass/@class`
- `//named-native-query/@result-class`
- `//named-stored-procedure-query/@result-class`
- `//object-type-converter/@data-type`
- `//object-type-converter/@object-type`
- `//one-to-many/@target-entity`
- `//one-to-one/@target-entity`
- `//property/@value-type`
- `//query-redirectors/@all-queries`
- `//query-redirectors/@read-all`
- `//query-redirectors/@read-object`
- `//query-redirectors/@report`
- `//query-redirectors/@update`
- `//query-redirectors/@insert`
- `//query-redirectors/@delete`
- `//read-transformer/@transformer-class`
- `//stored-procedure-parameter/@type`

- `//struct-converter/@converter`
- `//type-converter/@data-type`
- `//type-converter/@object-type`
- `//variable-one-to-one/@target-interface`
- `//write-transformer/@transformer-class`

7.8 Hibernate Mapping File Detection Criteria

Bundlor scans for any file that ends with a `.hbm` extension. If it detects one of these files it scans the file for a number of attributes that can contain class names. If the class name is unqualified (i.e. has no `'.'` in it), the classname is prepended with the content of the `hibernate-mapping` tag's `package` attribute. Many of the attributes that can contain class names can also contain Hibernate keywords corresponding to Hibernate-known types. When these are detected, no manifest requirements are added.

Hibernate Attributes

Using XPath syntax, the following is a list of attributes searched for type names

- `//class/@name`
- `//id/@type`
- `//generator/@class`
- `//composite-id/@class`
- `//discriminator/@type`
- `//property/@type`
- `//many-to-one/@class`
- `//one-to-one/@class`
- `//one-to-many/@class`
- `//many-to-many/@class`
- `//version/@type`
- `//component/@class`

- `//dynamic-component/@class`
- `//subclass/@name`
- `//joined-subclass/@name`
- `//union-subclass/@name`
- `//import/@class`

Hibernate Keywords

The following is a list of reserved Hibernate keywords that will not trigger the addition of manifest requirements

- `assigned`
- `big_decimal`
- `big_integer`
- `binary`
- `blob`
- `boolean`
- `byte`
- `calendar`
- `calendar_date`
- `character`
- `class`
- `clob`
- `currency`
- `date`
- `double`
- `float`
- `foreign`

- guid
- hilo
- identity
- imm_binary
- imm_calendar
- imm_calendar_date
- imm_date
- imm_serializable
- imm_time
- imm_timestamp
- increment
- integer
- locale
- long
- native
- select
- seqhilo
- sequence
- sequence-identity
- serializable
- short
- string
- text
- time
- timestamp

- `timezone`
- `true_false`
- `uuid`
- `yes_no`

7.9 JSP File Detection Criteria

Bundlor scans for the JSP files. If it detects this file, it scans the file for a number of values that contain class names.

JSP Values

Using Regular expression syntax, the following is a list of values searched for type names

- `<%@ page.*import=\"(.*?)\".*%>`

7.10 Log4J Configuration Detection Criteria

Bundlor scans for Log4J configuration files. If it detects this file type, it scans the file for a number of values that contain class names.

Log4J Configuration Values

Using XPath syntax, the following is a list of values searched for type names

- `//appender/@class`
- `//layout/@class`

7.11 Static Resource Detection Criteria

Bundlor scans for any static resource and exports that package.

8. Detecting Manifest Issues

Bundlor's second function is to scan an existing manifest and identify any potential issues with it.

8.1 Import Version Range Warning Criteria

Bundlor checks that all entries in the `Import-Package` header have a sensible version range declared. This ensures that there are no version ranges that are reversed (`[2, 1)`), and no version ranges that are empty (`[1, 1)`).

8.2 Import of Exported Packages Warning Criteria

Bundlor checks that the manifest does not import any package that it exports. This behavior is usually indicative of a package split between two bundles.

8.3 Signed JAR Warning Criteria

Bundlor checks that the manifest does not contain headers indicating that it is from a signed JAR. Running Bundlor against a signed JAR will render that JAR invalid as the manifest will have changed from when it was signed.

8.4 Versioned Imports Warning Criteria

Bundlor checks that all entries in the `Import-Package` header have a version range declared.

8.5 Versioned Exports Warning Criteria

Bundlor checks that all entries in the `Export-Package` header have a version declared.

8.6 Bundle-SymbolicName Warning Criteria

Bundlor checks that the manifest contains a `Bundle-SymbolicName` header.

8.7 Manifest-Version Warning Criteria

Bundlor checks that the manifest contains a `Bundle-ManifestVersion` header with a value of 2.