

# Bundlor User Guide

Ben Hale  
Juliet Shackell



1.0.0.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 .....	3
3.1. Command Line Quickstart .....	3
3.2. Apache ANT Quickstart .....	3
3.3. Apache Maven Quickstart .....	4
4. Usage .....	5
4.1. Command-Line Usage .....	5
Command Syntax .....	5
Command Line Reference .....	5
Command Line Parameters .....	5
Command Line Property Values .....	6
ANT Task Examples .....	6
Creating a manifest .....	6
Creating a manifest with placeholder replacement .....	7
4.2. Apache ANT Usage .....	7
ANT Setup .....	7
ANT Task Reference .....	8
Task Attributes .....	8
Inline Manifest Template .....	9
Inline OSGi Profile .....	9
Inline Property Values .....	10
ANT Task Examples .....	10
Creating a manifest .....	10
Creating a manifest with placeholder replacement .....	10
4.3. Apache Maven Usage .....	10
Maven Setup .....	10
Maven Plugin Reference .....	11
Plugin Configuration .....	11
Inline Manifest Template .....	12
Inline OSGi Profile .....	13
Inline Property Values .....	13
Maven Plugin Examples .....	13
Creating a manifest .....	13

Creating a manifest with placeholder replacement .....	14
5. Manifest Templates .....	15
5.1. Introduction .....	15
5.2. Manifest Template Format .....	15
5.3. Specifying property placeholders .....	16
5.4. Specifying automatic version expansion of imported packages based on a pattern .....	17
Re-using version patterns .....	18
5.5. Example Bundlor Manifest Template .....	19
6. OSGI Profiles and Bundlor .....	21
6.1. Overview of OSGI profiles .....	21
6.2. Using OSGI profiles with Bundlor .....	21
7. Detecting Manifest Requirements .....	23
7.1. Java Detection Criteria .....	23
Export Package .....	23
Import Package .....	23
7.2. Spring Context Configuration Detection Criteria .....	24
Spring Context Values .....	24
7.3. Blueprint Service Configuration Detection Criteria .....	25
Blueprint Configuration Values .....	25
7.4. Web Application File Detection Criteria .....	26
web.xml Values .....	26
7.5. Bundle-Classpath File Detection Criteria .....	27
7.6. JPA Detection Criteria .....	27
persistence.xml Values .....	27
7.7. Hibernate Mapping File Detection Criteria .....	27
Hibernate Attributes .....	28
Hibernate Keywords .....	28
7.8. JSP File Detection Criteria .....	30
JSP Values .....	30
7.9. Log4J Configuration Detection Criteria .....	31
Log4J Configuration Values .....	31
7.10. Static Resource Detection Criteria .....	31
8. Detecting Manifest Issues .....	32
8.1. Import Version Range Warning Criteria .....	32
8.2. Import of Exported Packages Warning Criteria .....	32
8.3. Signed JAR Warning Criteria .....	32
8.4. Versioned Imports Warning Criteria .....	32
8.5. Versioned Exports Warning Criteria .....	32
8.6. Bundle-SymbolicName Warning Criteria .....	32
8.7. Manifest-Version Warning Criteria .....	32

# Copyright



Copyright 2008-2009, SpringSource

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>

# License

Apache License  
Version 2.0, January 2004  
<http://www.apache.org/licenses/>

## TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

### 1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.
4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
  - (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and
  - (b) You must cause any modified files to carry prominent notices stating that You changed the files; and
  - (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
  - (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.
6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) 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. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.
8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

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.



# 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, it is easy for them to focus on identifying Imports and Exports and ensuring that their manifests are correct. When it comes time to use third-party enterprise libraries however, things start to break down.

Many of these libraries are not packaged as OSGi bundles. In fact, many of them have internal architectures and designs that do not lend themselves to controls that OSGi provides. In this case, a user must retrofit OSGi directives to the library before use.

This is where SpringSource Bundlor comes in. It can be very hard for a developer to know exactly what dependencies are needed by a library using simple inspection. 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 tasks for Apache ANT and plugins for Apache Maven, to simple command line execution for integration into any existing build system.

Before you start using Bundlor, please take a moment to look at the SpringSource Enterprise Bundle Repository; you may find that what you are looking for has already been updated. If you convert a library for your own use and think that it might be useful to the community at large, please submit your bundle information to the Enterprise Bundle Repository for inclusion.

## 2. Getting Bundlor

### 2.1 Getting the Bundlor ZIP

SpringSource Bundlor is distributed as a ZIP file.

1. Download ZIP file from Bundlor download page.

The Bundlor download page is located at <http://www.springsource.org/bundlor>.

### 2.2 Getting Bundlor with Ivy

SpringSource Bundlor can be obtained from an Ivy repository

1. Add the SpringSource Enterprise Bundle Repository resolvers to the `ivysettings.xml` file

```
<url name="com.springsource.repository.bundles.release">
  <ivy pattern="http://repository.springsource.com/ivy/bundles/release/[organisation]/[module]/[revision]/[artifact]">
  <artifact pattern="http://repository.springsource.com/ivy/bundles/release/[organisation]/[module]/[revision]/[artifact]">
</url>
```

2. Download the SpringSource Bundlor dependency in the `build.xml` file

```
<ivy:cachedpath resolveId="bundlor.classpath" pathid="bundlor.classpath" organisation="com.springsource.bundlor"
  module="com.springsource.bundlor.ant" revision="1.0.0.RELEASE" conf="ant" inline="true"
  type="jar" log="download-only"/>
```

### 2.3 Getting Bundlor with Maven

SpringSource Bundlor can be obtained from a Maven repository

1. Add the SpringSource Enterprise Bundle Repository resolvers to the `pom.xml` file

```
<repository>
  <id>com.springsource.repository.bundles.release</id>
  <url>http://repository.springsource.com/maven/bundles/release</url>
</repository>
```

## 3. Quickstart

### 3.1 Command Line Quickstart

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

1. Change directory to the \$BUNDLOR\_HOME/bin directory
2. Run `bundlor.sh` or `bundlor.bat` scripts. See Section 4.1, “Command-Line Usage” for details about the parameters the command line client.

```
% ./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:com.springsource.bundlor.ant">
```

2. Import the bundlor task into your build

```
<target name="bundlor.init">  
  <taskdef resource="com/springsource/bundlor/ant/antlib.xml"  
    uri="antlib:com.springsource.bundlor.ant">  
    <classpath id="bundlor.classpath">  
      <fileset dir="${bundlor.home}/dist"/>  
      <fileset dir="${bundlor.home}/lib"/>  
    </classpath>  
  </taskdef>  
</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. 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 SpringSource Enterprise Bundle Repository to the `pom.xml` file

```
<pluginRepositories>
  <pluginRepository>
    <id>com.springsource.repository.bundles.release</id>
    <name>SpringSource Enterprise Bundle Repository</name>
    <url>http://repository.springsource.com/maven/bundles/release</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>com.springsource.bundlor</groupId>
      <artifactId>com.springsource.bundlor.maven</artifactId>
      <version>1.0.0.RELEASE</version>
      <executions>
        <execution>
          <id>bundlor</id>
          <goals>
            <goal>bundlor</goal>
          </goals>
        </execution>
      </executions>
    </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 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 Chapter 6, <i>OSGi Profiles and Bundlor</i> for details.	No
-o <path>	The path to write the manifest to. This can either be a directory, a	No - defaults to <code>System.out</code>

Attribute	Description	Required
	<p>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>	
<code>-r &lt;path&gt;</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.

## ANT Task Examples

### Creating a manifest

```
% ./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
%
```

## Creating a manifest with placeholder replacement

```
% ./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  
%
```

## 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:com.springsource.bundlor.ant">
```

2. Import the bundlor task into your build

```
<target name="bundlor.init">  
  <taskdef resource="com/springsource/bundlor/ant/antlib.xml"  
    uri="antlib:com.springsource.bundlor.ant">  
    <classpath id="bundlor.classpath">  
      <fileset dir="${bundlor.home}/dist"/>  
      <fileset dir="${bundlor.home}/lib"/>  
    </classpath>  
  </taskdef>  
</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
<code>bundleSymbolicName</code>	The OSGi <code>Bundle-SymbolicName</code> for the resulting manifest	No
<code>bundleVersion</code>	The OSGi <code>Bundle-Version</code> for the resulting manifest	No
<code>enabled</code>	Whether Bundlor should create a manifest	No - defaults to <code>true</code>
<code>failOnWarnings</code>	Whether Bundlor should cause a build failure when there are warnings about the resulting manifest	No - defaults to <code>false</code>
<code>inputPath</code>	The path to the input to create a manifest for. This can either be a directory or a JAR file.	Yes
<code>manifestTemplatePath</code>	The path to the manifest template. See Chapter 5, <i>Manifest Templates</i> for details.	No
<code>osgiProfilePath</code>	The path to the OSGi profile. See Chapter 6, <i>OSGI Profiles and Bundlor</i> for details.	No
<code>outputPath</code>	<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>	No - defaults to <code>System.out</code>



Attribute	Description	Required
	If nothing is specified, the manifest will be written to <code>System.out</code> .	
<code>propertiesPath</code>	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-Vendor: SpringSource Inc.
Bundle-SymbolicName: com.springsource.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 = \
  javax.accessibility,\
  javax.activation,\
  javax.activation.version="1.1.0",\
  javax.activity,\
  javax.annotation,\
  ...
org.osgi.framework.bootdelegation = \
  com_cenqua_clover,\
  com.cenqua.*,\
  com.yourkit.*,\
  ...
  </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 SpringSource Enterprise Bundle Repository to the `pom.xml` file.

```
<pluginRepositories>
  <pluginRepository>
    <id>com.springsource.repository.bundles.release</id>
    <name>SpringSource Enterprise Bundle Repository</name>
```

```

    <url>http://repository.springsource.com/maven/bundles/release</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>com.springsource.bundlor</groupId>
      <artifactId>com.springsource.bundlor.maven</artifactId>
      <version>1.0.0.RELEASE</version>
      <executions>
        <execution>
          <id>bundlor</id>
          <goals>
            <goal>bundlor</goal>
          </goals>
        </execution>
      </executions>
    </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 <code>true</code>
failOnWarnings	Whether Bundlor should cause a build failure when there are warnings about the resulting manifest	No - defaults to <code>false</code>

Attribute	Description	Required
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	<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>	No - defaults to <code>\${project.build.outputDirectory}</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.

```

<execution>
  <id>bundlor</id>
  <goals>
    <goal>bundlor</goal>
  </goals>
  <configuration>
    <manifestTemplate>
Bundle-ManifestVersion: 2
Bundle-Name: Bundlor Core
Bundle-Vendor: SpringSource Inc.
Bundle-SymbolicName: com.springsource.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 = \
  javax.accessibility,\
  javax.activation,\
  javax.activation;version="1.1.0",\
  javax.activity,\
  javax.annotation,\
  ...

org.osgi.framework.bootdelegation = \
  com_cenqua_clover,\
  com.cenqua.*,\
  com.yourkit.*,\
  ...
    </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>com.springsource.bundlor</groupId>
      <artifactId>com.springsource.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>com.springsource.bundlor</groupId>
        <artifactId>com.springsource.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 real values at runtime. With this feature, your manifest templates become more dynamic and useful across a variety of your projects. A particularly useful use case 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.
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 example, <code>Export-Template</code>

Header	Description
	<code>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` directive to specify a version or version range of the imported package.

```
Import-Template:
com.springsource.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 version the `com.springsource.*` packages at `[2.5.4.A, 3.0.0)` and the `org.apache.commons.logging` package at `[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:
com.springsource.kernel.*;version="${com.springsource.kernel:[=.=.=, +1.0.0)}",
org.apache.commons.logging.*;version="${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) of the version pattern.
[+/-]n	Adjust the value from the	Valid only in the first three

Symbol	Description	Location Allowed
	variable by this amount. For example, +1 means to add 1 to the value from the variable.	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 `${com.springsource.kernel}` to the value `1.2.0`. Based on the expansion pattern, Bundlor sets the version range of the imported `com.springsource.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. The in-line pattern shows a more unusual usage. It will create a version range from, but not including, the provided version up to and including the next minor version with the same qualifier.

## 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}
Bundle-Vendor: SpringSource
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`.
- **Bundle-Vendor:** specifies the bundle's vendor
- **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 the many bundles included in the SpringSource Enterprise Bundle Repository, 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 Section 6.1, “Overview of OSGI profiles” 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 dm Server) 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 dm Server as your OSGi runtime, see the file `DM_SERVER_HOME/lib/java6-server.profile` for its OSGi profile, where `DM_SERVER_HOME` refers to the main installation directory of dm Server. 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 sections 3.8.3 and 3.8.5 of the [OSGI 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 dm Server as your OSGI runtime, you can start by copying the section of the file `$DM_SERVER_HOME/lib/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 dm Server; 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 = \  
  javax.accessibility,\  
  javax.activation,\  
  javax.activation/version="1.1.0",\  
  javax.activity,\  
  javax.annotation,\  
  ...  
  
org.osgi.framework.bootdelegation = \  
  com_cenqua_clover,\  
  com.cenqua.*,\  
  com.yourkit.*,\  
  ...
```

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 outer bundle's manifest and the JAR file is placed on the outer bundle's Bundle-Classpath.

## 7.6 JPA Detection Criteria

Bundlor scans for the JPA `persistence.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.

### **persistence.xml Values**

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

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

## 7.7 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 `hibernate-mapping tags' 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

## 7.8 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=\"(.*?)\".*%>`

## 7.9 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`

## 7.10 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 exclusive (`[ 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 the when it was signed.

### 8.4 Versioned Imports Warning Criteria

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