Multi-release JAR Support

Draft

10 Pages

Abstract

Java 9 adds the concept of Multi-release JAR files which contain versions of classes for different major versions of Java. OSGi needs to support Multi-release JAR files as they are being adopted by the Java community.

0 Document Information

0.1 License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

The OSGi Alliance hereby grants you a limited copyright license to copy and display this document (the “Distribution”) in any medium without fee or royalty. This Distribution license is exclusively for the purpose of reviewing and providing feedback to the OSGi Alliance. You agree not to modify the Distribution in any way and further agree to not participate in any way in the making of derivative works thereof, other than as a necessary result of reviewing and providing feedback to the Distribution. You also agree to cause this notice, along with the accompanying consent, to be included on all copies (or portions thereof) of the Distribution. The OSGi Alliance also grants you a perpetual, non-exclusive, worldwide, fully paid-up, royalty free, limited license (without the right to sublicense) under any applicable copyrights, to create and/or distribute an implementation of the Distribution that: (i) fully implements the Distribution including all its required interfaces and functionality; (ii) does not modify, subset, superset or otherwise extend the OSGi Name Space, or include any public or protected packages, classes, Java interfaces, fields or methods within the OSGi Name Space other than those required and authorized by the Distribution. An implementation that does not satisfy limitations (i)-(ii) is not considered an implementation of the Distribution, does not receive the benefits of this license, and must not be described as an implementation of the Distribution. "OSGi Name Space" shall mean the public class or interface declarations whose names begin with "org.osgi" or any recognized successors or replacements thereof. The OSGi Alliance expressly reserves all rights not granted pursuant to these limited copyright licenses including termination of the license at will at any time.

EXCEPT FOR THE LIMITED COPYRIGHT LICENSES GRANTED ABOVE, THE OSGi ALLIANCE DOES NOT GRANT, EITHER EXPRESSLY OR IMPLIEDLY, A LICENSE TO ANY INTELLECTUAL PROPERTY IT, OR ANY THIRD PARTIES, OWN OR CONTROL. Title to the copyright in the Distribution will at all times remain with the OSGi Alliance. The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted therein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

THE DISTRIBUTION IS PROVIDED "AS IS," AND THE OSGi ALLIANCE (INCLUDING ANY THIRD PARTIES THAT HAVE CONTRIBUTED TO THE DISTRIBUTION) MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DISTRIBUTION ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.
NEITHER THE OSGi ALLIANCE NOR ANY THIRD PARTY WILL BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THE DISTRIBUTION.

Implementation of certain elements of this Distribution may be subject to third party intellectual property rights, including without limitation, patent rights (such a third party may or may not be a member of the OSGi Alliance). The OSGi Alliance is not responsible and shall not be held responsible in any manner for identifying or failing to identify any or all such third party intellectual property rights.

The Distribution is a draft. As a result, the final product may change substantially by the time of final publication, and you are cautioned against relying on the content of this Distribution. You are encouraged to update any implementation of the Distribution if and when such Distribution becomes a final specification.

The OSGi Alliance is willing to receive input, suggestions and other feedback (“Feedback”) on the Distribution. By providing such Feedback to the OSGi Alliance, you grant to the OSGi Alliance and all its Members a non-exclusive, non-transferable,

worldwide, perpetual, irrevocable, royalty-free copyright license to copy, publish, license, modify, sublicense or otherwise distribute and exploit your Feedback for any purpose. Likewise, if incorporation of your Feedback would cause an implementation of the Distribution, including as it may be modified, amended, or published at any point in the future ("Future Specification"), to necessarily infringe a patent or patent application that you own or control, you hereby commit to grant to all implementers of such Distribution or Future Specification an irrevocable, worldwide, sublicenseable, royalty free license under such patent or patent application to make, have made, use, sell, offer for sale, import and export products or services that implement such Distribution or Future Specification. You warrant that (a) to the best of your knowledge you have the right to provide this Feedback, and if you are providing Feedback on behalf of a company, you have the rights to provide Feedback on behalf of your company; (b) the Feedback is not confidential to you and does not violate the copyright or trade secret interests of another; and (c) to the best of your knowledge, use of the Feedback would not cause an implementation of the Distribution or a Future Specification to necessarily infringe any third-party patent or patent application known to you. You also acknowledge that the OSGi Alliance is not required to incorporate your Feedback into any version of the Distribution or a Future Specification.

I HEREBY ACKNOWLEDGE AND AGREE TO THE TERMS AND CONDITIONS DELINEATED ABOVE.

0.2 Trademarks

OSGi™ is a trademark, registered trademark, or service mark of the OSGi Alliance in the US and other countries. Java is a trademark, registered trademark, or service mark of Oracle Corporation in the US and other countries. All other trademarks, registered trademarks, or service marks used in this document are the property of their respective owners and are hereby recognized.

0.3 Feedback

This document can be downloaded from the OSGi Alliance design repository at https://github.com/osgi/design The public can provide feedback about this document by opening a bug at https://www.osgi.org/bugzilla/.

0.4 Table of Contents

0.5 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in 10.1.

`Source code is shown in this typeface.`

0.6 Revision History

The last named individual in this history is currently responsible for this document.

Revision	Date	Comments
Initial	2017-12-12	Initial draft. BJ Hargrave, IBM
2nd draft	2017-12-13	Second draft after first EG review. BJ Hargrave, IBM
3rd draft	2018-01-02	Third draft after CPEG review. Added section on Bundle.getLastModified. BJ Hargrave, IBM

1 Introduction

Java SE 9 added support for the Java Platform Module System to Java SE. One of the main goals of adding module support to Java SE was to modularize the Java platform API itself. This is in support of encapsulating implementation details of the platform to avoid application code from being dependent on implementation details as was common in many applications. A common example of this is the sun.misc.Unsafe class which is used by many applications.

Once the platform is modularized and can hide implementation detail, new API can be added to replace the platform implementation API which applications has become dependent upon. But this means that different implementations of the application may be needed for pre-Java 9 usage and post-Java 8 usage. The pre-Java 9 implementation would use the older platform implementation details and the post-Java 8 implementation would use the new-to-Java 9 API which replaces the now inaccessible platform implementation details.

So the application provider would be in the situation of having to deliver 2 artifacts for their application: One which runs on pre-Java 9 platforms and one which runs on post-Java 8 platforms. This is awkward for both the application provider as well as consumers of the application.

To ameliorate this issue, the Java SE 9 platform also introduced the concept of Multi-release JAR files in JEP 238 [3]. A multi-release JAR file is a normal JAR where the base version of the application code is in the normal place but the JAR contains the manifest header `Multi-Release: true` as well as classes in the META-INF/versions folder which is for specific major versions of the Java platform.

The OSGi framework, and tooling generating bundles, must support multi-release JAR file as they begin to be utilized by the Java community.

2 Application Domain

A bundle is a JAR file with additional OSGi metadata which further describes the bundle. Many developers, specifically in open source projects, generate artifacts which can be used as both vanilla JAR files as well as OSGi bundles. They often use available tools such as maven plugins based upon Bnd which decorate the JAR file with OSGi metadata based upon analysis of the JAR contents. This way many JARs are available to the OSGi community directly from the JAR provider.

As developers move to support Java 9, independent of the Java Platform Module System, they find they may need different implementation of select classes to replace use of platform implementation details which are no longer accessible on Java 9 with the use of newly added API. These developers are then choosing to use a multi-release JAR so that can deliver a single artifact which contains the implementations for pre-Java 9 and post-Java 8 platforms.

Currently neither the OSGi framework nor tooling which generates bundle understand or support the multi-release JAR format [4]. So as developers begin to utilize multi-release JAR files, they will no longer be able to also support OSGi without the hassle of generating a separate OSGi-specific artifact.

3 Problem Description

The OSGi framework specification does not address the new Multi-release JAR specification [4]. The OSGi framework specification must be updated to support Bundle-ClassPath elements which conform to the Multi-release JAR specification.

4 Requirements

Support for multi-release JARs in OSGi must be done in the spirit of multi-release JARs where the purpose of multi-release JARs is to support alternate implementation of select classes to deal with changes in the visible APIs of the Java platform. That is, it is not meant as a means to supply new function or new API on different Java platform version.

MR0010 – Bundle class loaders must support the Multi-release JAR format for each element of the Bundle-ClassPath. This include the main bundle itself when '.' is on the Bundle-ClassPath. '.' is the default Bundle-ClassPath.

MR0020 – The framework must support replacement values for the Import-Package and Require-Capability manifest headers when the bundle is marked as a multi-release JAR.

5 Technical Solution

5.1 Bundle Metadata

Since different implementations of a Java class for different versions of the Java platform can affect the requirements of the bundle, the OSGi framework must support alternate values for the `Import-Package` and `Require-Capability` manifest headers for different versions of the Java platform.

When processing the metadata for a bundle, if, after processing the manifest in `META-INF/MANIFEST.MF`, the bundle is declared to be a multi-release JAR via the manifest header:

```
Multi-Release: true
```

the framework must then look for a supplemental manifest file `OSGI-INF/MANIFEST.MF` in a versioned folder. For example:

```
META-INF/versions/9/OSGI-INF/MANIFEST.MF
```

First the versioned folder for the major version of the Java platform is examined and then prior versioned folders in descending order. The first supplemental manifest file found is used and the framework must replace the values of the `Import-Package` and `Require-Capability` manifest headers in the manifest with the values of these headers, if present, in the supplemental manifest file. The supplemental manifest file can contain one or both of these headers. Any other headers in the supplemental manifest file must be ignored.

The framework APIs which provide access to the bundle metadata, such as `Bundle.getHeaders` and `BundleRevision` and `BundleWiring`, must present the supplemented manifest information. That is, the main manifest with the replacement values from a supplemental manifest, if any, for the Java platform version.

The above also applies to fragment bundles since they can also have different implementations of Java classes.

5.2 Bundle Class Loader

The bundle class loader provides access to classes and resources in the bundle and its attached fragments using the values of the `Bundle-ClassPath` manifest headers of the bundle and each attached fragments. The `Bundle-ClassPath` manifest header defines an ordered list of *container paths* to entries in the bundle such as folders and JAR files. A `Bundle-ClassPath` container path can be '`.`' or '`/`' which represent the root container of the bundle. If `Bundle-ClassPath` is not specified, the default value of '`.`' is used. The bundle class loader must search each container of the `Bundle-ClassPath`, in order, when searching for a class or resource.

Each container referenced by the `Bundle-ClassPath` can be a multi-release container independent of the other containers. For example, a bundle can embed a multi-release JAR and list the path of the entry to that JAR in the `Bundle-ClassPath` while other containers referenced by the `Bundle-ClassPath` are not multi-release. So each container referenced by the `Bundle-ClassPath` must declare if it is a multi-release container to be treated as such. This is done via the

```
Multi-Release: true
```

manifest header in the container's `META-INF/MANIFEST.MF` manifest.

For each `Bundle-ClassPath` container which declares itself to be a multi-release container, the bundle class loader must search the container's `META-INF/versions` folders as specified by the Multi-release JAR specification [4] and then the container's base folder when attempting to locate a class or resource in the container.

The bundle class loader and the framework API which provide access to bundle classes and resources, such as `Bundle.loadClass`, `Bundle.getResource`, `Bundle.getResources`, `BundleWiring.getClassLoader`, and `BundleWiring.listResources`, must all support multi-release `Bundle-ClassPath` containers. In addition to returning versioned resource names in the `META-INF/versions` folder,

In addition, when using the `BundleWiring.listResources` method, if the parameters would include a resource name in the results if the resource was in the base folder and the resource is in a versioned folder visible on the current Java version, the results must include the resource name in the base folder. For example, if the container contains:

```
META-INF/versions/9/com/foo/resources.txt
```

and the call `BundleWiring.listResources("/com/foo", "*.txt", 0)` is made when running on version 9, or later, of the Java platform, the result must include "`com/foo/resources.txt`".

5.3 Bundle Entries

The framework APIs which provide access to bundle content independent of the bundle class loader, such as `Bundle.getEntry`, `Bundle.getEntryPaths`, `Bundle.findEntries`, and `BundleWiring.findEntries`, are not affected by multi-release JAR support. Multi-release JAR support only affect access to bundle content through the bundle class loader. That is, class loading and resource loading.

5.4 Bundle Last Modified

The information in a bundle supporting multi-release can change when the Java version used to run the OSGi framework changes. The bundle metadata can change due to use of supplemental manifests and the resources and classes available from the bundle's class loader can change due to versioned folders. The framework must report a more recent last modified time from `Bundle.getLastModified` if it is possible the information inside a bundle has changed, so that other bundles which might need to know if the information inside a bundle has changed can be advised of potential changes.

5.5 Java Versions

Multi-release support only supports alternate versions for Java 9 and higher. This is consistent with [4].

5.6 Indexing

While not an issue at runtime for a framework, when tooling indexes a bundle, the tool does not need to index supplemental manifests. Since the purpose of multi-release jars is to handle changes in the Java platform, any changes to the requirements of a bundle for multi-release are against capabilities of the Java platform. Tooling that builds bundles must complain if supplemental manifest contains requirements to capabilities not supplied by the Java platform for the Java version associated with the supplemental manifest.

6 Data Transfer Objects

This design does not alter any existing DTOs nor define any new DTOs.

7 Javadoc

This design does not add or alter any Java API for the OSGi specifications.

8 Considered Alternatives

None.

9 Security Considerations

Support for multi-release JAR files does not add any additional security considerations. A multi-release bundle can be signed the same as any bundle.

10 Document Support

10.1 References

[1] Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997.

[2] Software Requirements & Specifications. Michael Jackson. ISBN 0-201-87712-0

[3] JEP 238: Multi-Release JAR Files. http://openjdk.java.net/jeps/238

[4] JAR File Specification. https://docs.oracle.com/javase/9/docs/specs/jar/jar.html

10.2 Author's Address

Name	BJ Hargrave
Company	IBM

10.3 Acronyms and Abbreviations

JAR – Java ARchive file

10.4 End of Document