Require java package capabilities

Draft

10 Pages

Abstract

Java 9 adds the concept of modules. With Java 9, the Java platform itself is composed of a set of modules which can be defined at launch time or built into a~~n~~ runtime image using the jlink tool. The set of java.* packages which are provided by the runtime instance are not a constant set for each version of Java. OSGi needs to support bundles that have dependencies on specific java.* packages which may not be enabled by default with the Java runtime instance.

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	2018-01-04	Initial draft. Thomas Watson, 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 allowing application modules to require specific modules provided by the Java platform itself or other third-party libraries.

Once the platform is modularized, the runtime can be configured to load only the modules which are required by the application. Java SE 9 has launching options for enabling specific sets of modules for the launched runtime instance. The jlink tool is also available for creating a runtime image with a specific set of modules. This allows for a smaller custom runtime that only includes what is needed by the application.

The set of `java.*` packages provided by the Java platform is no longer constant for a specific version of the Java Platform. It depends on the set of modules that are enabled for the runtime instance. OSGi bundles depend on

most packages by using requirements (i.e. by specifying Import-Package or Require-Bundle) but as of the OSGi Core R4 specification bundles have been prohibited from using Import-Package to specify requirements on the `java.*` packages. Instead OSGi bundles have used the osgi.ee namespace to specify a dependency on specific versions of the Java platform and it has been assumed that the set of `java.*` packages available for a specific version of the platform is constant.

The OSGi framework, and tooling generating bundles, must provide a way for bundles to specify dependencies on packages from the `java.*` namespace (e.g. java.sql).

2 Application Domain

A bundle is a JAR file with additional OSGi metadata which further describes the bundle. One aspect of this description is to specify what is required by the OSGi bundle in order to resolve and function properly. There are three namespaces in OSGi that are used to specify capabilities and requirements for Java APIs or code.

1. osgi.wiring.package – A specific java package name
2. osgi.wiring.bundle – A specific OSGi bundle host identity
3. osgi.ee – The running execution environment

The resolution wires for the `osgi.wiring.package` and `osgi.wiring.bundle` namespaces both influence the OSGi bundle class loader delegation which allows an OSGi bundle to have access to packages provided by other bundles resolved in the OSGi framework. The `osgi.ee` namespace does not influence class loader delegation but it does allow a framework to depend on two aspects of the running platform or execution environment

1. The Java byte code level which is supported by the running platform.
2. The set of `java.*` ~~sub-~~packages ~~versions~~ provided by the running platform.

An OSGi R6 Framework must boot delegate all class load requests for classes in `java.*` ~~sub-~~packages. Bundles are not allowed to use Import-Package or Export-Package for `java.*` ~~sub-~~packages ~~of `java.*`~~.

With Java SE 9, it is no longer safe to assume a specific set of `java.*` ~~sub-~~packages will be available at runtime for a specific version of the Java Platform or execution environment. Only the `java.*` packages included in `java.base` module are guaranteed to be available. The `osgi.ee` namespace cannot be used to depend on a specific set of `java.*` packages. It should only be used to specify a minimal level of Java that must be supported by the running execution environment. Dependencies on specific `java.*` ~~sub-~~packages ~~names~~ need to be specified with a different namespace from the `osgi.ee` namespace.

3 Problem Description

The OSGi framework specification does not provide a way to specify requirements on packages from the `java.*` namespace. The OSGi framework specification must be updated to support requirements on `java.*` packages (e.g. java.sql)

4 Requirements

RJ0010 – Bundles must have way to specify a requirement on a specific `java.*` ~~sub-~~package of ~~`java.*`~~ (e.g. java.sql).

RJ0020 – Bundles must continue to have access to all available ~~java~~`java.*` ~~sub-~~packages even if the bundle does not specify a requirement on the ~~java~~`java.*` ~~sub-~~package (e.g. java.lang).

RJ0030 – Only the system bundle must provide capabilities which satisfy requirements on ~~java~~`java.*` ~~sub-~~packages. Other bundles are prohibited from providing these capabilities.

RJ0040 – Bundles that require `java.*` ~~sub-~~packages ~~of `java.*`~~ must work on all Java levels supported by the framework as long as the `java.*` package is available in the running Java platform.

RJ0050 – It must be possible for a single bundle to work on both OSGi R6 and R7 but on R7 be able to specify requirements on specific `java.*` package names.

5 Technical Solution

5.1 Capabilities

Since the `osgi.ee` namespace can no longer accurately describe the set of `java.*` ~~sub-~~packages available in the running Java platform, a new set of capabilities must be provided by the system bundle which describes the available `java.*` ~~sub-~~packages. The existing `osgi.wiring.package` namespace is used to specify the available `java.*` ~~sub-~~packages. The system bundle must discover all the available `java.*` ~~sub-~~packages in the

running Java platform and automatically provide the each `java.*` ~~sub-~~package as a separate system package (See `Constants.FRAMEWORK_SYSTEMPACKAGES`). Besides the `osgi.wiring.package` attribute, no additional attributes are required to be specified for the `java.*` ~~sub-~~packages. The Java modules providing `java.*` ~~sub-~~packages are versioned along with the Java platform itself. The Java platform version attribute is still expressed using the `osgi.ee` namespace.

The system bundle is the only bundle able to provide `osgi.wiring.package` capabilities. Normal bundles that attempt to use Export-Package with a `java.*` ~~sub-~~package will result in a `BundleException` indicating a manifest syntax error. Framework implementations must provide `java.*` ~~sub-~~package capabilities ~~for~~on all Java platform versions the framework implementation supports~~, not only when running on Java 9 or greater~~.

5.2 Requirements

Bundles may require `java.*` ~~sub-~~packages using the `osgi.wiring.package` namespace. The `osgi.wiring.package` namespace requirements must only be specified using the `Import-Package` header. Besides the `java.*` ~~sub-~~package name, no other matching attribute should be specified for the imported packages. ~~The `osgi.ee` namespace must be required t~~To specify the version of the Java platform required by the bundle, the `osgi.ee` namespace must be required.

The bundle class loaders will continue to have ~~free~~full ~~access~~visibility to all available `java.*` ~~sub-~~packages available in the running Java platform even when the bundle does not specify any Import-Package header. This is because class loads for classes in the `java.*` ~~sub-~~packages will continue to be boot delegated ~~in an OSGi R7~~by the framework implementation. The bundle class loaders must boot delegate `java.*` packages even in the absence of any `java.*` package requirements because the internals of the JVM implementation make assumptions about all class loaders having visibility to all provided `java.*` packages. Particularly (but may not be exclusively) from the `java.base` module.

BJH Does the wiring (including DTOs) show the java.* package wires? If so, I would assume only for the specific java.* packages mentioned in Import-Package.

5.3 Supporting OSGi R6 Frameworks

The OSGi R6 core specification prohibits bundles from importing `java.*` ~~sub-~~packages and will throw a `BundleException` if such a bundle is attempted to be installed. Bundles that must ~~support~~work on OSGi R6 and earlier Frameworks but want to utilize this new OSGi R7 functionality to import `java.*` ~~sub-~~packages may be packaged as a multi-release JAR. Multi-Release JAR support is new to R7 and allows bundles to provide alternative values for the `Import-Package` header depending on the version of Java being used at runtime. An OSGi R6 or earlier Framework will not understand the multi-release JAR versioned content and therefore will only ~~load~~use the `Import-Package` specified in the ~~base~~main bundle manifest (`META-INF/MANIFEST.MF`).

A bundle that must still be able to be installed on an OSGi R6 or earlier Framework must ~~only~~not specify ~~`non-java.*`~~ ~~sub-~~packages in their ~~base~~main bundle manifest `Import-Package` header.

A ~~versioned~~supplemental manifest must also be included~~, for at least Java 9~~ For example:

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

This ~~bundle~~supplemental manifest must contain the original `Import-Package` header from the ~~base~~main bundle manifest as well as any ~~of the bundles required~~ `java.*` ~~sub-~~package ~~names~~required by the bundle. When running on an OSGi R6 Framework, this ~~versioned~~supplemental manifest will be ignored and therefore the bundle will install successfully as ~~if~~ the bundle did not specify an import for the `java.*` ~~sub-~~packages in the main manifest.

If a bundle will always require OSGi R7 or greater Framework, there is no need to use multi-release JARs for this purpose even when the bundle must support Java 8 or earlier. This is because OSGi R7 frameworks must provide the `java.*` sub-package capabilities for on all Java platform versions the framework implementation supports, not only when running on Java 9 or greater. As long as the `java.*` sub-package is available on the running Java platform, the bundle with an import for the `java.*` sub-package must resolve.

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

An alternative solution considered using a separate namespace (`osgi.ee.package`) for the `java.*` sub-package names. This approach has the following advantages:

- Historically the `osgi.wiring.package` namespace wires have been used to influence class loader delegation. For the `java.*` packages, this willwould no longernot be true because these packages will continue to be boot delegated which effectively will cause any `osgi.wiring.package` wires to the `java.*` packages to be ignored.

- A separate namespace would allow the possibility of resolving a bundle that requires specific `java.*` packages on an R6 framework implementation by configuring the framework launch property `org.osgi.framework.system.capabilities.extra` with the necessary `osgi.ee.package` capabilities for the running Java platform.

- Provide a consistent resolution error even when running on an OSGi R6 framework. The new namespace requirements will be recognized by an R6 framework and cause the same resolution error which would be seen on an R7 framework when a required package is not available. The difference being that on an R6 framework the resolution error can only be solved by extra framework configuration. Using the `osgi.wiring.package` namespace for `java.*` packages will always result in a manifest syntax error at install time on and OSGi R6 framework.

This alternative solution was not agreed upon because of the following reasons

- A future OSGi specification may decide to enforce `java.*` package wires for class loader delegation. A separate namespace will make that more complicated and confusing in the future.

- A new namespace to describe a capability/requirement for a package name is confusing when there is already an existing `osgi.wiring.package` namespace. Besides the fact that the `java.*` packages are boot delegated they should be treated like other packages because `java.*` packages are real packages.

9 Security Considerations

TODO Should bundles be granted import `PackagePermission` for `java.*` implicitly since bundles always have access to `java.*` packages without having a package requirement? Is there value in preventing resolution of `java.*` packages by using permission checks?

BJH So this would be about 2.4.1 in Core Spec (Implied Permission). I think we do need to grant PackagePermission("java.*",IMPORT) to each bundle. In general, access to the Java platform APIs is necessary for normal operation. Furthermore, as noted above, (1) the JVM may expect visibility to some types by the bundles and (2) we will bootdelegate loading anyway. So the act of "importing" a java.* package is just a resolution operation and not a class loading operation per se.

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

10.2 Author's Address

Name	Tom Watson
Company	IBM

10.3 Acronyms and Abbreviations

`java.*` Packages whose name starts with "java.*".

10.4 End of Document