



RFC 197 – OSGi Package and Type Annotations

Draft

15 Pages

Abstract

OSGi may soon move to full Java 5 language support include annotations. Some standard OSGi annotations, such as package version, type role, etc., need to be defined. This RFC proposes an initial set of standard OSGi package and type annotations.

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 Document Information.....	2
0.1 License.....	2
0.2 Trademarks.....	3
0.3 Feedback.....	3
0.4 Table of Contents.....	3
0.5 Terminology and Document Conventions.....	4
0.6 Revision History.....	4
1 Introduction.....	4
2 Application Domain.....	5
2.1 Package Versioning.....	5
2.2 Type Role.....	5
2.2.1 Eclipse Build.....	6
2.2.2 Bnd.....	6
3 Problem Description.....	6
3.1 Package Versioning.....	6
3.2 Type Role.....	7
4 Requirements.....	7
5 Technical Solution.....	7
5.1 Package Versioning.....	8
5.1.1 Version.....	8
5.2 Type Role.....	8

5.2.1 ProviderType.....	8
5.2.2 ConsumerType.....	8
6 Javadoc.....	9
7 Considered Alternatives.....	14
7.1 @Export.....	14
7.2 Concurrency Roles.....	14
8 Security Considerations.....	15
9 Document Support.....	15
9.1 References.....	15
9.2 Author's Address.....	15
9.3 Acronyms and Abbreviations.....	15
9.4 End of Document.....	15

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 9.1.

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	March, 2013	David Bosschaert, Red Hat, initial version.
2 nd draft	5 March 2013	Repurpose the RFC as the initial set of standard OSGi annotations.
<u>3rd draft</u>	<u>18 March 2013</u>	<u>Change RFC name to be more explicit about its purpose.</u>

1 Introduction

There are several OSGi concepts that would be useful to express in Java source code such as package versions, the role of a type (producer or consumer) in a package. To date, there has been no OSGi standard way of expressing these things in the Java source code. OSGi will soon move to full Java 5 language feature support including annotations. When Java 6 reaches end-of-life and OSGi moves to Java 7, javac will no longer support

the `-target jsr14` option. So when we transition to `-target 1.5`, we will be able to use standard Java annotations like `@Override` and `@Deprecated`. We will also be able to define and use standard OSGi [package and type](#) annotations.

2 Application Domain

2.1 Package Versioning

Package versioning is a key part of OSGi development. In the OSGi Alliance we carefully version our packages. Others also need to version their packages. The current version number of a package must be recorded. JDK has the `package-info.java` source file which contains package level information such as documentation and annotations but there is no standard package version annotation.

2.2 Type Role

One of the key concepts in OSGi is the practice of Semantic Versioning. This is described in more details in the Semantic Versioning White-paper [3]. Part of the Semantic Versioning mechanics concern whether an API is implemented by providers of the technology or by consumers of the technology. This distinction governs how the version of this API is updated given a change to the Interface.

OSGi bundles describe their bundle and package versions in the Bundle Manifest. The following is an example:

```
Bundle-ManifestVersion: 2
Bundle-SymbolicName: test.bundle
Bundle-Version: 2.7.1.Final
Export-Package: org.acme.foo; version="1.7"
```

The bundle version itself is `2.7.1.Final`, but the version of the `org.acme.foo` package inside the bundle is `1.7`. Assume that the bundle contains a Java type `org.acme.foo.Bar`:

```
package org.acme.foo;

public interface Bar {
    void operation();
}
```

Modifying this interface will always result in a version change to a certain degree, but especially when adding a new method to the interface the role of the type becomes significant in relation to how the package version changes. Assume the following modification to the `Bar` interface:

```
package org.acme.foo;

public interface Bar {
    void operation();
    void operation2();
}
```

This modification is not *source* compatible with any existing implementors of the `Bar` interface (compilation will fail because the `operation2()` method was added), but it is *binary* compatible with users of the `Bar` interface. Existing users will not be aware of the new operation and will continue to function.

How this change affects the version number of the exported `org.acme.foo` package depends on the *role* of the Bar type.

If the Bar interface is only implemented by providers of the technology with which Bar is associated then consumers will continue to work. For example, if the Bar interface is used for a service registered in the OSGi Service registry and this service is provided as part of the technology of which Bar is part, then consumers of this service will continue to function given the change above. In this case the version of the `org.acme.foo` package will have its minor number increased to become 1.8.

However, if the interface is to be implemented by consumers of the technology the impact of the change above is much larger as a change as it will cause compilation errors for all the consumers. Examples where an interface is to be implemented by a consumer is where the consumer provides a callback object via an API or where the consumer registers a service implementing this API in the Service Registry which is then picked up by the technology to be handled. The latter is known as the Whiteboard Pattern.

Tools such as BND, the Eclipse Build system and the Aries Versioning Plugin have ways to manage version number changes automatically and some of these provide means to mark an interface to be either implemented by consumers or providers.

2.2.1 Eclipse Build

The Eclipse Build and the OSGi internal build system use a Javadoc taglet to mark an interface as not being implemented by consumers. Interfaces without this taglet are assumed to be implemented by consumers.

```
/**
 * @noimplement
 */
public interface Foo {
    ...
}
```

The Eclipse build system provides a mechanism to transfer this information to the binary bundle by preserving this metadata in a `.api_description` file.

2.2.2 Bnd

Bnd also provides a mechanism to specify the role of an interface. This is done using annotations. The `@aQute.bnd.annotation.ProviderType` and `@aQute.bnd.annotation.ConsumerType` annotations can be added to interface types to declare their role. These annotations have retention policy `CLASS`, which means that they introduce a compile-time dependency, but no runtime dependency. The information can however be read by tools that process the `.class` files of these interfaces.

3 Problem Description

3.1 Package Versioning

There is no OSGi standard mechanism to declare a package version in source code. We currently use a `packageinfo` file which is a format defined by bnd. All package maintainers need a standard way to record the current version of a package.

3.2 Type Role

Today there is no OSGi standard way to express the role of a given interface, which makes it impossible to write portable tools that handle versioning of packages over their long-term life cycle.

At this point in time there seem to be two prevalent ways of declaring the role of an exported interface. Given a particular bundle, finding out the role of a certain interface is challenging since the presence of this information depends on the build system used. Moreover it may not be present at all as other build systems being used do not record this information at present.

A common mechanism to specify and inspect the role of an interface needs to be specified so that tools and humans have a well-defined way of learning about the role of a given exported interface in an OSGi bundle.

4 Requirements

A01 – The solution **MUST** provide a mechanism to describe the Role of a type in a bundle.

A02 – The solution **MUST** allow the specification of the following roles: implemented by consumers, implemented by providers.

A03 – The solution **MUST** make it possible to find out the role of a type without the need to have access to the type's source code.

A04 – The solution **MUST** make the Role information accessible in such a way that it can be read by tools.

A05 – The solution **MUST** provide a mechanism to describe the version of a package.

A06 – The solution **MUST** allow the information to be place in the Java source files next to the package or type declaration.

A07 – The solution **MUST NOT** require the use of special “side” files or source processing tools to make the information available to other tools after compilation.

A08 – The solution **MUST NOT** require the presence of any annotations at runtime or the presence of any annotation at compile-time when compiling against marked types.

5 Technical Solution

Annotations will be used as the solution. This will allow the use of existing tooling for compiling and reading annotations. The annotation information will be stored directly in class files which avoid the need for “side” files and keeping them in sync with the source files.

CLASS retention will be used to keep the information in the class files while avoiding the need for the annotation types at runtime or when compiling classes referencing the annotated types.

5.1 Package Versioning

The `org.osgi.annotation.versioning.Version` package annotation is defined. It can be applied to the package declaration in `package-info.java` to specify the version of the package.

5.1.1 Version

Specify the version of a package.

This annotation is not retained at runtime. It is for use by tools to generate bundle manifests or otherwise process the version of a package.

For example:

```
@Version("1.0")  
package org.osgi.annotation.versioning;
```

5.2 Type Role

Two type annotations are defined. `org.osgi.annotation.versioning.ProducerType` is used to mark a type as being implemented by the producer role and `org.osgi.annotation.versioning.ConsumerType` is used to mark a type as being implemented by the consumer role.

5.2.1 ProviderType

A type implemented by the Provider Role.

A non-binary compatible change to a provider type normally requires incrementing the minor version of the type's package. This change will require all providers to be updated to handle the change, but consumers of that package will not require changes since they only use, and do not implement, the provider type.

A type can be marked `ConsumerType` or `ProviderType` but not both. A type is assumed to be `ConsumerType` if it is not marked either `ConsumerType` or `ProviderType`.

This annotation is not retained at runtime. It is for use by tools to understand the semantic version of a package. When a bundle implements a provider type from an imported package, then the bundle's import range for that package must require the package's exact major and minor version.

For example:

```
@ProviderType  
public interface SomeInterface
```

5.2.2 ConsumerType

A type implemented by the Consumer Role.

A non-binary compatible change to a consumer type normally requires incrementing the major version of the type's package. This change will require all providers and all consumers to be updated to handle the change since consumers implement the consumer type and all providers must understand the change in the consumer type.

A type can be marked `ConsumerType` or `ProviderType` but not both. A type is assumed to be `ConsumerType` if it is not marked either `ConsumerType` or `ProviderType`.

This annotation is not retained at runtime. It is for use by tools to understand the semantic version of a package. When a bundle implements a consumer type from an imported package, then the bundle's import range for that package must require the exact major version and a minor version greater than or equal to the package's version.

For example:

@ConsumerType

public interface SomeInterface

6 Javadoc

OSGi Javadoc
3/7/13 11:13 AM

Package Summary		Page
org.osgi.annotation.versioning	OSGi Versioning Annotations Package Version 1.0.	Error: Reference source not found

Package org.osgi.annotation.versioning

[@Version](#) (value="1.0")

OSGi Versioning Annotations Package Version 1.0.

See:

[Description](#)

Annotation Types Summary		Page
ConsumerType	A type implemented by the Consumer Role.	Error: Reference source not found
ProviderType	A type implemented by the Provider Role.	Error: Reference source not found
Version	Specify the version of a package.	Error: Reference source not found

Package org.osgi.annotation.versioning Description

OSGi Versioning Annotations Package Version 1.0.

This package is not used at runtime.

See Also:

[Semantic Versioning](#)

Annotation Type ConsumerType

[org.osgi.annotation.versioning](#)

@Documented

@Retention(value=RetentionPolicy.CLASS)

@Target(value=ElementType.TYPE)

public @interface ConsumerType

A type implemented by the Consumer Role.

A non-binary compatible change to a consumer type normally requires incrementing the major version of the type's package. This change will require all providers and all consumers to be updated to handle the change since consumers implement the consumer type and all providers must understand the change in the consumer type.

A type can be marked [ConsumerType](#) or [ProviderType](#) but not both. A type is assumed to be [ConsumerType](#) if it is not marked either [ConsumerType](#) or [ProviderType](#).

This annotation is not retained at runtime. It is for use by tools to understand the semantic version of a package. When a bundle implements a consumer type from an imported package, then the bundle's import range for that package must require the exact major version and a minor version greater than or equal to the package's version.

See Also:

[Semantic Versioning](#)

Annotation Type *ProviderType*

[org.osgi.annotation.versioning](#)

@Documented

@Retention(value=RetentionPolicy.CLASS)

@Target(value=ElementType.TYPE)

public @interface *ProviderType*

A type implemented by the Provider Role.

A non-binary compatible change to a provider type normally requires incrementing the minor version of the type's package. This change will require all providers to be updated to handle the change, but consumers of that package will not require changes since they only use, and do not implement, the provider type.

A type can be marked [ConsumerType](#) or [ProviderType](#) but not both. A type is assumed to be [ConsumerType](#) if it is not marked either [ConsumerType](#) or [ProviderType](#).

This annotation is not retained at runtime. It is for use by tools to understand the semantic version of a package. When a bundle implements a provider type from an imported package, then the bundle's import range for that package must require the package's exact major and minor version.

See Also:

[Semantic Versioning](#)

```
Annotation Type Version
org.osgi.annotation.versioning
@Documented
@Retention(value=RetentionPolicy.CLASS)
@Target(value=ElementType.PACKAGE)
public @interface Version
```

Specify the version of a package.
This annotation is not retained at runtime. It is for use by tools to generate bundle manifests or otherwise process the version of a package.
See Also:

[Semantic Versioning](#)

Required Element Summary		Page
String	value The version of the annotated package.	rror: Refer ence sourc e not found

Element Detail

value

```
public abstract String value
    The version of the annotated package.
    The version must be a valid OSGi version string.
```

Java API documentation generated with [DocFlex/Doclet](#) v1.5.6
DocFlex/Doclet is both a multi-format Javadoc doclet and a free edition of [DocFlex/Javadoc](#). If you need to customize your Javadoc without writing a full-blown doclet from scratch, DocFlex/Javadoc may be the only tool able to help you! Find out more at www.docflex.com

7 Considered Alternatives

7.1 @Export

@Export from bnd was suggested as a possible OSGi annotation. The @Export annotation seems to handle many things. It allows mandatory and non-mandatory attributes to be specified. But this sort of information does not seem to be an inherent pat of a package but rather a decision of the bundle exporting the package.
The @Export annotation also allows specifying the include and exclude directives. These directives are for use by existing packages which are poorly design in that implementation specific types are part of a package which is shared as public by the bundle. While a bundle may need to export such a package and control visible to types within that package, if the developer can modify the source code of the package to add OSGi annotations, it would be better to add annotations to the types rather than package-info. That is, a type annotation like @NoExport would be better. The developer could then apply this annotation to all types in the package that should not be exported.

7.2 Concurrency Roles

I originally considered adding concurrency annotations like [those](#) in Java Concurrency In Practice. However, I left these out for several reasons. First, these annotations already exist and people are free to use the existing annotations. Even OSGi could use them (except they are RUNTIME retention which seems to create compile-time issue if the annotations are not on the compile classpath). Second, there is some possibility that JSR 305 may bring them into the space of standard Java annotations (although that JSR seems dead). Finally, concurrency

annotations are really outside the scope of OSGi “expertise”. That is, they are orthogonal to modularity and services.

8 Security Considerations

The proposed annotations present no security issues.

9 Document Support

9.1 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]. Semantic Versioning Whitepaper. OSGi Alliance.
<http://www.osgi.org/wiki/uploads/Links/SemanticVersioning.pdf>

9.2 Author’s Address

Name	David Bosschaert
Company	Red Hat
e-mail	david@redhat.com

Name	BJ Hargrave
Company	IBM Corporation
e-mail	hargrave@us.ibm.com

9.3 Acronyms and Abbreviations

9.4 End of Document