RFC 180 Portable Java SE/EE Contracts

Proposed Final Draft

13 Pages

Abstract

One of the promises of OSGi is the ability of a bundle to know that the runtime provides the capabilities it needs. At one level this is based on the code package required, however this is insufficient. Packages evolve over time with new methods, interfaces and classes being added. The provider of a package can express the version of the package so client can select the minimum version it needs. Making use of semantic versioning also allows the provider to make breaking API changes without affecting client bundles. This is the promise, however the Java platform itself does not version its packages. This RFC defines how bundles using Java packages can express dependencies on particular versions of Java packages.

0 Document Information

0.1 License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

0.2 Trademarks

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 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
0	22/06/12	Initial Draft Alasdair Nottingham, IBM, not@uk.ibm.com
0.1	28/09/12	Updates from Basel face to face
0.2	4th April 2013	Updates for draft publication (largely contract name style) Graham Charters, IBM, charters@uk.ibm.com
0.3	12th September 2013	Updates from Hursley face to face. Removed javax.servlet.resources and javax.servlet.jsp.resources which only exist in the tomcat version of the API jar. Updated the Servlet 3 example. Added JSR numbers Alasdair Nottingham, IBM, not@uk.ibm.com
0.4	September 16, 2013	Prepare for vote

1 Introduction

As Enterprise OSGi gains adoption in the industry it is important to ensure that there is a clear statement of how a client bundle obtains access to the Java EE packages in a safe and compatible way. For example, it is important that a Web Application Bundle that requires Servlet 3.0 packages can express the requirement such that it will not resolve if the WAB extender only supports Servlet 2.5. It is also important that having written an application against one vendors extender that the Web Application Bundle can then be run using an alternative vendors implementation. Although the Web Application Bundle specification is Servlet 2.5 only many implementations of the specification support Servlet 3, however they take different approaches to versioning the package. Some version semantically from the WAB specification (i.e. 2.6), some version according to the JSR version (3.0). This inhibits portability and produces confusion in the development community.

This RFC seeks to bring clarity to the current confusion around Java platform package versioning such that client bundles can be written to be declare their dependency on Java platform packages in a version compatable and portable way.

2 Application Domain

Enterprise application development has traditionally made extensive use of a set of Java standards collectively known as Java Enterprise Edition. Typically applications written to use these standards are deployed into an application server. There are various different popular application servers and a key value of the Java EE standards is the ability to easily move an application between different servers. This means compatibility is a key requirement for application server vendors.

Currently the OSGi Alliance is standardizing how to integrate various Java EE standards into an OSGi environment, however many vendors are implementing ahead of the standards and are therefore making decisions and choices which impact portability when future standards are defined. An example of this exists within the Web Application Bundle chapter of the OSGi Enterprise Specification. This defines the version of the Servlet 2.5 packages, but does not speak about the versions for Servlet 3.0, or the as yet unreleased 3.1. Application server vendors have still provided implementations of the Web Application Bundle specification based on Servlet 3, but have chosen different versioning mechanisms for the packages. This mismatch is counter to the goals and principals of Java EE and OSGi which seek to promote portability between runtimes. This incompatibility is likely to be a significant inhibitor to the uptake of Enterprise OSGi.

3 Problem Description

How can an OSGi application express a dependency on a Java EE standard without being tied to a particular provider of its packages. In the absense of a definitive statement from either the JCP or the OSGi Alliance on what version a specific package should be exported at in OSGi we have ended up with different vendors taking different views on the appropriate version. Typically one of the following approaches has been taken by providers:

- Version semantically from a baseline. In some cases a baseline is defined, javax.servlet 2.5 is defined in the Web Application Bundle specification, in some cases a baseline was arbitarily chosen.
- Version packages based on the Java EE specification marketing version.
- Do both of the previous two.

4 Requirements

10 – The specification MUST provide a portable mechanism by which bundles can express dependencies on Java EE packages.

20 – The specification MUST provide a portable package dependency mechanism in a way that accommodates different vendors using different versioning schemes for individual Java EE packages.

5 Technical Solution

Rather than define package versions for each individual package in all Java platform packages, a task that will be neither fun, nor will produce consensus. This RFC will propose a set of OSGi contracts for the specifications. These will then be versioned according to the JSR version. A client bundle will then import the packages required with no version and express a dependency on the OSGi contract that defines the packages and the exact specification version required. A provider of the contract then expresses every version of the contract they support.

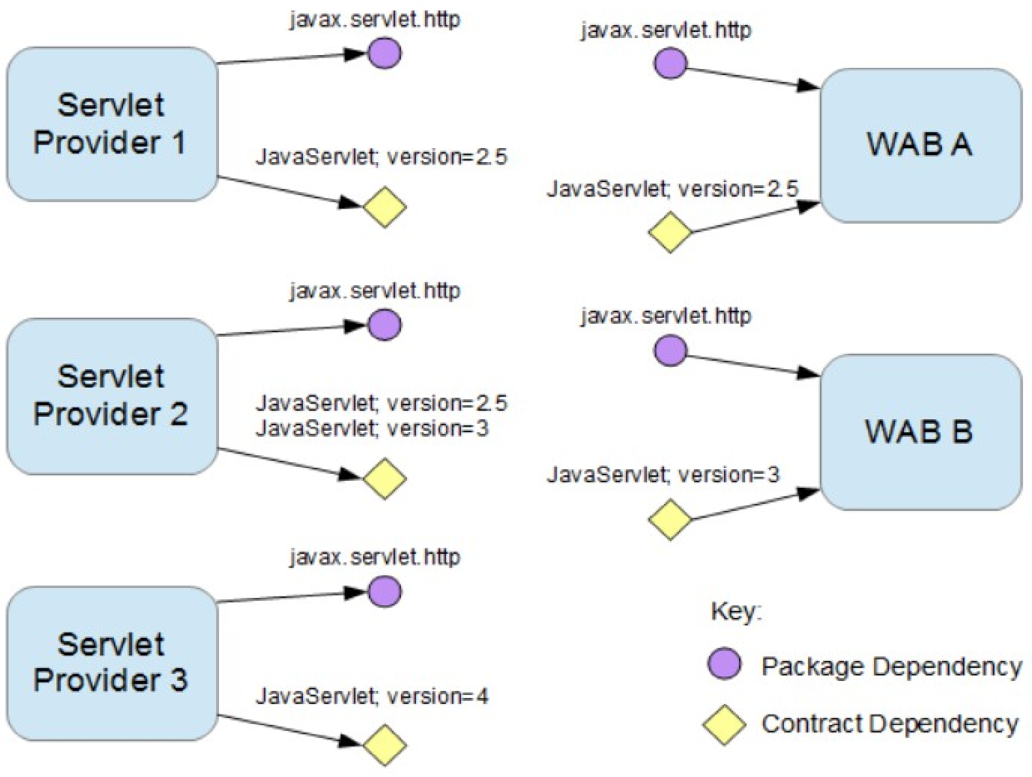

Figure 1: Example of using contracts for using Servlet APIs

As show in figure 1 we have three providers of the JavaServlet contract. The first provides JavaServlet 2.5, the second provides JavaServlet 3.0. The Servlet 3.0 specification is specifically written to be backwardly compatible, so a Servlet 3.0 runtime can support Servlet 2.5 applications. As a result it provides two contracts, one for JavaServlet at 2.5 and one for JavaServlet at 3.0. The third one provides a mythical JavaServlet at version 4 that is NOT compatible with JavaServlet 3.0 and JavaServlet 2.5. In this example WAB A can run against Servlet Provider 1 and 2, WAB B can run against Servlet provider 2, neither can run against Servlet Provider 3. By inverting the versioning scheme for these contracts, when compared with semantic versions for packages, we remove from the client the risk associated with the undefined mechanism the Java Community Process and the JSR committees use when choosing new versions for their specifications. Vendors are then free to version individual packages in accordance with their existing schemes, thus allowing compatibility between their releases, while being able to support portability across different vendors too. The JCP typically versions specifications based more on marketing considerations rather than based on making a statement about the new specifications compatibility with previous releases of the specifications.

OSGi contract names are advised to be upper camel case (aka Pascal case) with the first segment or segments corresponding to a namespace to ensure uniqueness. The OSGi alliance reserves the first segment “OSGi”. Similarly this design uses the first segment “Java” to define contracts corresponding to the Java based standards defined by the JCP. The “Javax” prefix is also be reserved, but is not used. What follows is the set of contracts:

Name	Version	JSR	Packages	Comments
JavaEJB	2.1	153	javax.ejb javax.ejb.spi	

Name	Version	JSR	Packages	Comments
JavaEJB	3	220	javax.ejb javax.ejb.spi	Compatible with 2.1
JavaEJB	3.1	318	javax.ejb javax.ejb.embeddable javax.ejb.spi javax.interceptor	Compatible with 3, 2.1
JavaEJBLite	3.1	318	javax.ejb javax.interceptor	EJBlite is a subset of the EJB specification. The package content is logically subset. A provider of java.ejb is required to also provide this contract. A provider of java.ejb.lite is not required to provide java.ejb.
JavaExpressionLanguage	2.1	245	javax.el	
JavaExpressionLanguage	2.2	245	javax.el	Compatible with 2.1
JavaCDI	1	299	javax.decorator javax.enterprise.context javax.enterprise.context.spi javax.enterprise.event javax.enterprise.inject javax.enterprise.inject.spi javax.enterprise.util	
JavaJMS	1.1	914	javax.jms	
JavaJPA	1	220	javax.persistence javax.persistence.spi	
JavaJPA	2	317	javax.persistence javax.persistence.criteria javax.persistence.metamodel javax.persistence.spi	Compatible with 1
JavaJCA	1.5	112	javax.resource javax.resource.cci javax.resource.spi javax.resource.spi.endpoint javax.resource.spi.security javax.resource.spi.work	
JavaJCA	1.6	322	javax.resource javax.resource.cci javax.resource.spi javax.resource.spi.endpoint javax.resource.spi.security javax.resource.spi.work	Compatible with 1.5
JavaJASPIC	1	196	javax.security.auth.message javax.security.auth.message.callback javax.security.auth.message.config javax.security.auth.message.module	
JavaJACC	1.1	115	javax.security.jacc	
JavaJACC	1.4	115	javax.security.jacc	Compatible with 1.1
JavaServlet	2.5	154	javax.servlet javax.servlet.http	

Name	Version	JSR	Packages	Comments
JavaServlet	3	315	javax.servlet javax.servlet.annotation javax.servlet.descriptor javax.servlet.http	Compatible with 2.5
JavaJSP	2	152	javax.servlet.jsp javax.servlet.jsp.el javax.servlet.jsp.tagext	
JavaJSP	2.1	245	javax.servlet.jsp javax.servlet.jsp.el javax.servlet.jsp.tagext	Compatible with 2
JavaJSP	2.2	245	javax.servlet.jsp javax.servlet.jsp.el javax.servlet.jsp.resources javax.servlet.jsp.tagext	Compatible with 2.1, 2
JavaJSTL	1	52	javax.servlet.jsp.jstl.core javax.servlet.jsp.jstl.fmt javax.servlet.jsp.jstl.sql javax.servlet.jsp.jstl.tlv	
JavaJSTL	1.1	52	javax.servlet.jsp.jstl.core javax.servlet.jsp.jstl.fmt javax.servlet.jsp.jstl.sql javax.servlet.jsp.jstl.tlv	Compatible with 1
JavaJSTL	1.2	52	javax.servlet.jsp.jstl.core javax.servlet.jsp.jstl.fmt javax.servlet.jsp.jstl.sql javax.servlet.jsp.jstl.tlv	Compatible with 1.1
JavaJTA	1.1	907	javax.transaction javax.transaction.xa	
JavaJTAJRE	1.1	907	javax.transaction javax.transaction.xa	This contains a subset of the package javax.transaction. It only contains 3 exceptions. A provider of java.jta MUST also provide this contract. The OSGi system bundle MUST provide this contract.
JavaBeanValidatio n	1	303	javax.validation javax.validation.bootstrap javax.validation.constraints javax.validation.groups javax.validation.metadata javax.validation.spi	
JavaJAXRS	1.1	311	javax.ws.rs javax.ws.rs.core javax.ws.rs.ext	
JavaJAXWS	2.1	224	javax.xml.ws javax.xml.ws.handler javax.xml.ws.handler.soap javax.xml.ws.http javax.xml.ws.soap javax.xml.ws.spi javax.xml.ws.wsaddressing	
JavaJAXWS	2.2	224	javax.xml.ws	Compatible with 2.1

Name	Version	JSR	Packages	Comments
			javax.xml.ws.handler javax.xml.ws.handler.soap javax.xml.ws.http javax.xml.ws.soap javax.xml.ws.spi javax.xml.ws.spi.http javax.xml.ws.wsaddressing	
JavaJAXBinding	2.1	222	javax.xml.bind javax.xml.bind.annotation javax.xml.bind.annotation.adapters javax.xml.bind.attachment javax.xml.bind.helpers javax.xml.bind.util	
JavaJAXBinding	2.2	222	javax.xml.bind javax.xml.bind.annotation javax.xml.bind.annotation.adapters javax.xml.bind.attachment javax.xml.bind.helpers javax.xml.bind.util	Compatible with 2.1
JavaAnnotation	1	250	javax.annotation javax.annotation.security	
JavaAnnotation	1.1	250	javax.annotation javax.annotation.security javax.annotation.sql	Compatible with 1
JavaInject	1	330	javax.inject	

Examples

In this example a bundle is written to use the Servlet 3 API. It expresses the dependencies in the Bundle headings showin in listing 1.

```
Bundle-SymbolicName: my.company.wab
Bundle-Version: 2
Import-Package: javax.servlet, javax.servlet.http
Require-Capability: osgi.contract; filter:="(&(osgi.contract=JavaServlet)
  (version=3.0))"
```

Listing 1: A bundle that uses JavaServlet 3.

The bundle can then make use of the Servlet API provided by the following providers of the API

```
Bundle-SymbolicName: jee.vendor1
Bundle-Version: 3
Export-Package: javax.servlet, javax.servlet.http
Provide-Capability: osgi.contract; osgi.contract=JavaServlet; version:Version=3;
 uses:="javax.servlet, javax.servlet.http", osgi.contract;
 osgi.contract=JavaServlet; version:Version=2.5; uses:="javax.servlet,
 javax.servlet.http"
```

Listing 2: A bundle that provides servlet packages unversioned

```
Bundle-SymbolicName: jee.vendor2
Bundle-Version: 3
Export-Package: javax.servlet; version=2.6, javax.servlet.http; version=2.6
Provide-Capability: osgi.contract; osgi.contract=JavaServlet; version:Version=3;
 uses:="javax.servlet, javax.servlet.http", osgi.contract;
 osgi.contract=JavaServlet; version:Version=2.5; uses:="javax.servlet,
 javax.servlet.http"
```

Listing 3: A bundle that provides servlet packages using semantic versions

```
Bundle-SymbolicName: jee.vendor3
Bundle-Version: 3
Export-Package: javax.servlet; version=3, javax.servlet.http; version=3
Provide-Capability: osgi.contract; osgi.contract=JavaServlet; version:Version=3;
 uses:="javax.servlet, javax.servlet.http", osgi.contract;
 osgi.contract=JavaServlet; version:Version=2.5; uses:="javax.servlet,
 javax.servlet.http"
```

Listing 4: A bundle that provides servlet packages using marketing versions

The bundle would not wire to the provider shown in Listing 5 because it does not support JavaServlet at version 3.

```
Bundle-SymbolicName: jee.vendor4
Bundle-Version: 2.5
Export-Package: javax.servlet; version=2.5, javax.servlet.http; version:Version=2.5
Provide-Capability: osgi.contract; osgi.contract=JavaServlet; version:Version=2.5;
 uses:="javax.servlet, javax.servlet.http"
```

Listing 5: A bundle that provides JavaServlet 2.5, but not 3

The bundle shown in listing 6 would be able to use the providers from listing 2-5.

```
Bundle-SymbolicName: my.company.wab2
Bundle-Version: 2
Import-Package: javax.servlet, javax.servlet.http
Require-Capability: osgi.contract; filter:="(&(osgi.contract=JavaServlet)
 (version=2.5))"
```

Listing 6: A bundle that uses JavaServlet 2.5.

The bundle shown in listing 6 would not match the bundle shown in listing 7 because it provides JavaServlet 3 compatibility, but not JavaSerlvet 2.5. This is in reality invalid, but it does illustrate how versioning works with this scheme.

```
Bundle-SymbolicName: jee.vendor5
Bundle-Version: 3
Export-Package: javax.servlet; version=3, javax.servlet.http; version=3,
javax.servlet.annotation; version=3, javax.servlet.descriptor; version=3
Provide-Capability: osgi.contract; osgi.contract=JavaServlet; version:Version=3;
 uses:="javax.servlet, javax.servlet.http"
```

Listing 7: A bundle that provides JavaServlet 3, but not JavaServlet 2.5

6 Considered Alternatives

Many alternatives were discussed and discounted. The following proposals were made:

1. Version packages according to the JSR marketing version. This was not liked because it violates the semantic versioning best practice.

2. Version packages semantically. This follows the precedent set by the JPA specification which versioned the javax.persistence package from JPA 2.0 at 1.1. Applying this to Servlet 3.0 would result in javax.servlet being at 2.6.

3. Version packages using the JSR version, but adding 100 to the version. This would result in things like Servlet 3 being 102.6, Servlet 2.5 being 102.5. This was not liked because it isn't obvious and looks ridiculous.

Fundamentally these options were all discounted because agreement on a good solution couldn't be reached. There were essentially two groups in the argument. The first group believes that package versions should be semantically done. The second group believes that Java platform packages are an exception and semantic versioning shouldn't apply and the JSR marketing version should instead. A major argument on this side was that Java EE developers are too stupid to get semantic versions. Both groups could agree on option 3, but it was felt the solution was ridiculous so it was abandoned.

7 Security Considerations

TBD

8 Document Support

8.1 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

8.2 Author’s Address

Name	Alasdair Nottingham
Company	IBM
e-mail	not@uk.ibm.com

8.3 Acronyms and Abbreviations

8.4 End of Document