DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

DATE OF DISTRIBUTION: 16 May 2011

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.
Preface

This document is Early Draft 2011.05 of the OSGi Service Platform Release 4 specifications. As an early draft, it contains non-final specification work and it is not organized in the format normally associated with final release OSGi specifications. This document contains copies of OSGi design documents which either propose to modify existing published OSGi specifications from the OSGi Service Platform Release 4 specification documents or propose new specifications to potentially be incorporated in future OSGi Service Platform Release 4 Specification documents.

Since this early draft is not a complete specification document, the reader is expected to be familiar with OSGi Technology and the currently published OSGi Service Platform Release 4 specification documents. The reader should refer to http://www.osgi.org/About/Technology for more information on the OSGi Technology. There the reader can find a description of the OSGi Technology, as well as links to whitepapers and the OSGi Service Platform Release 4 specification documents, which are all available for download.

Pursuant to the Distribution and Feedback License above, the OSGi expert groups welcome your feedback on this early draft. Feedback can be provided by opening a bug at https://www.osgi.org/bugzilla/enter_bug.cgi?product=OSGi%20Specification.

BJ Hargrave
Chief Technical Officer
OSGi Alliance
Abstract

This document describes a bundle repository for the OSGi Alliance. This repository consists of a web site (bundles.osgi.org) that hosts an XML resource that describes a federated repository managed by the OSGi Alliance. This repository can be browsed on the web site. Additionally, the repository can be used directly from any OSGi Framework to deploy bundles from the repository (if supported by the bundle’s licensing). This document defines the format of the XML and the OSGi service to access and use the repository.

This document is an experimental draft and not an official OSGi specification. With this experimental draft the OSGi Alliance wants to gather feedback and support. There is currently no commitment to turn this experimental draft into an official specification.
DISTRIBUTION AND FEEDBACK LICENSE

Copyright © OSGi Alliance 2006. All Rights Reserved.

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 expressly reserves all rights not granted pursuant to this limited copyright license including termination of the license at will at any time.

EXCEPT FOR THE LIMITED COPYRIGHT LICENSE 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.

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 right 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.

Comments about this specification can be mailed to: speccomments@mail.osgi.org
0 Document Information

0.1 Table of Contents

0 Document Information .................................................................................................................. 3
  0.1 Table of Contents .................................................................................................................. 3
  0.2 Terminology and Document Conventions .............................................................................. 6
  0.3 Revision History ...................................................................................................................... 6

1 Introduction ................................................................................................................................ 6
  1.1 Acknowledgements .................................................................................................................. 6
  1.2 Introduction .............................................................................................................................. 6

2 Application Domain ..................................................................................................................... 7

3 Problem Description ..................................................................................................................... 8

4 Requirements ................................................................................................................................ 8
  4.1 Functional .................................................................................................................................. 8
  4.2 Discovery ..................................................................................................................................... 8
  4.3 Dependency Resolution ............................................................................................................... 9
  4.4 Security ....................................................................................................................................... 9
  4.5 Non Functional ............................................................................................................................ 9

5 Technical Solution ....................................................................................................................... 9
  5.1 Entities ....................................................................................................................................... 9
  5.2 Overview ................................................................................................................................... 11
    5.2.1 Resource ............................................................................................................................. 12
    5.2.2 Capabilities .......................................................................................................................... 13
    5.2.3 Requirements ........................................................................................................................ 13
    5.2.4 Extends ................................................................................................................................ 14
  5.3 Repository Admin ...................................................................................................................... 14
    5.3.1 Discovery ............................................................................................................................. 14
    5.3.2 Resolving .............................................................................................................................. 14
    5.3.3 Admin ................................................................................................................................... 15
  5.4 Resolving .................................................................................................................................... 15
  5.5 XML Schema ............................................................................................................................. 15
    5.5.1 Namespace ............................................................................................................................ 15
    5.5.2 The XML Structure .............................................................................................................. 15
    5.5.3 Repository ............................................................................................................................ 15
    5.5.4 Referral ................................................................................................................................ 16
    5.5.5 Resource .............................................................................................................................. 16
7.6.2 getFilter.......................................................... 30
7.6.3 isMultiple...................................................... 30
7.6.4 isOptional...................................................... 30
7.6.5 isExtend.......................................................... 30
7.6.6 getComment.................................................... 30
7.6.7 isSatisfied....................................................... 31

7.7 Interface Resolver............................................... 31
  7.7.1 add............................................................. 31
  7.7.2 getUnsatisfiedRequirements.......................... 32
  7.7.3 getOptionalResources.................................... 32
  7.7.4 getReason.................................................... 32
  7.7.5 getResources.................................................. 32
  7.7.6 getRequiredResources.................................... 32
  7.7.7 getAddedResources........................................ 32
  7.7.8 resolve....................................................... 32
  7.7.9 deploy....................................................... 32

7.8 Interface Resource............................................... 32
  7.8.1 LICENSE_URL............................................... 34
  7.8.2 DESCRIPTION............................................... 34
  7.8.3 DOCUMENTATION_URL................................. 34
  7.8.4 COPYRIGHT.................................................. 34
  7.8.5 SOURCE_URL............................................... 34
  7.8.6 SYMBOLIC_NAME.......................................... 34
  7.8.7 PRESENTATION_NAME..................................... 34
  7.8.8 ID............................................................. 35
  7.8.9 VERSION..................................................... 35
  7.8.10 URL.......................................................... 35
  7.8.11 SIZE........................................................ 35
  7.8.12 KEYS........................................................ 35
  7.8.13 getProperties.............................................. 35
  7.8.14 getSymbolicName......................................... 35
  7.8.15 getPresentationName.................................... 35
  7.8.16 getVersion.................................................. 35
  7.8.17 getld....................................................... 35
  7.8.18 getURL...................................................... 36
  7.8.19 getRequirements........................................ 36
  7.8.20 getCapabilities.......................................... 36
  7.8.21 getCategories............................................ 36
  7.8.22 getRepository.............................................. 36

8 Security Considerations................................................................................................. 36
  8.1 Repository Permission.................................................................................. 36

9 Document Support........................................................................................................... 37
  9.1 References......................................................................................... 37
  9.2 Author's Address................................................................................. 37
  9.3 Acronyms and Abbreviations..................................................................... 37
  9.4 End of Document................................................................................. 37
0.2 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 11.1.

Source code is shown in this typeface.

0.3 Revision History

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

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>DEC 22 2005</td>
<td>Peter Kriens, Initial draft</td>
</tr>
<tr>
<td>0.1</td>
<td>MAR 16 2006</td>
<td>Peter Kriens, Prepared for release.</td>
</tr>
<tr>
<td>0.2</td>
<td>FEB 20 2009</td>
<td>Hal Hildebrand, resurrected RFC from zombie status, added schema for OBR</td>
</tr>
<tr>
<td>0.3</td>
<td>FEB 25 2009</td>
<td>Hal Hildebrand, updated the API to reflect current state of Felix OBR implementation</td>
</tr>
<tr>
<td>0.4</td>
<td>JUNE 10 2010</td>
<td>David Savage, fixed formatting and numbering issues</td>
</tr>
<tr>
<td>0.5</td>
<td>JUNE 10 2010</td>
<td>David Savage, separated RepositoryAdmin interface and ResolverFactory interface to allow flexibility for providers. Added discussion on open issues</td>
</tr>
<tr>
<td>0.6</td>
<td>JUNE 25 2010</td>
<td>David Savage, added discussion of Resolver to open issues, updated discussion of uses, mention wiring API/RFC 154 relationship</td>
</tr>
<tr>
<td>0.7</td>
<td>JULY 13 2010</td>
<td>David Savage, moved various open issues to considered alternatives or technical solution as discussed in conference call. Updated API to latest proposal</td>
</tr>
<tr>
<td>0.8</td>
<td>JULY 23 2010</td>
<td>David Savage, Fleshed out build and management usecases, diagram updates and API</td>
</tr>
<tr>
<td>0.9</td>
<td>OCTOBER 19 2010</td>
<td>David Savage, Expand on capabilities model + examples, tidy up loose ends, some work on management usecases</td>
</tr>
<tr>
<td>0.10</td>
<td>OCTOBER 26 2010</td>
<td>David Savage, Change resolver api to be a stateless api and allow previous resolution state to be parsed explicitly in resolve method</td>
</tr>
<tr>
<td>0.11</td>
<td>OCTOBER 16 2010</td>
<td>David Savage, Clarified API issues wrt framework context resolution. Started work on xml schema</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Acknowledgements
This document is based on the excellent work done by a number of contributors to this area including Richard S. Hall with the Oscar Bundle Repository, Robert Dunne and David Savage on the Nimble Resolver and the Sigil development framework, and Pascal Rapicault on the P2 provisioning platform.

1.2 Introduction
The uptake of the OSGi Specifications by the open source communities like Eclipse, Apache, and Knopflerfish has multiplied the number of available bundles. This is causing a confusing situation for end users because it is hard to find suitable bundles; there is currently no central repository.

This document addresses this lack of a repository. Not only describes it a concrete implementation of the OSGi Alliance’s repository (which will link member’s repositories), it also provides an XML format and service interface.

Revision | Date           | Comments                                                                                                                                                                      |
----------|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
0.12      | JANUARY 19 2011| David Savage, updated API to remove ResolverFactory based on F2F feedback also opened discussion on Extends model. Further work on xml schema                                             |
0.13      | FEBRUARY 2 2011| David Savage, Add Javadoc. Change extends to requirement with extend attribute set. Updates to match namespaces used in RFC 154. Change CapabilityProvider to return Iterable vs Iterator. Add uri as attribute on Resource and add discussion on when it is mandatory. Fix xmlns in xmlschema header. State RepositoryListener api is not testable in CT |
0.14      | MAY 2 2011     | Work in progress update to document to reflect Environment/Map-Wiring updates to API                                                                                         |
0.15      | MAY 13 2011    | Synchronize OBR API design with sub-systems design, prepare document for public draft.                                                                                       |
2 Application Domain

2.1 Introduction

OSGi specifications are being adopted at an increasing rate. The number of bundles available worldwide is likely in the thousands, if not low ten thousands. Although many of these bundles are proprietary and not suitable for distribution, there are a large number of distributable bundles available. The current situation is that vendors have proprietary bundle repositories. However, there are a number of available solutions to downloading and installing bundles. These include:

- The Felix OSGi Bundle Repository which allows end users to discover bundles using a command line tool that runs on any OSGi Framework.
- The P2 provisioning platform used in Eclipse
- Maven repositories which are supported by a number of provisioning tools such as PAXRunner and Karaf
- Nimble Repositories which extend the OSGi Bundle Repository concepts to deal with active state dependencies.

Since bundles explicitly declare requirements in their manifest file, it is possible to define a bundle repository service that provides access to this metadata to enable remote reasoning about bundle dependencies.

In general, bundle requirements are satisfied by capabilities provided by other bundles, the environment, or other resources. Resolving bundle requirements to provided capabilities is a constraint solving process. Some constraints are of a simple provide/require nature, while other constraints can include notions of versions and version ranges. One of the more complex constraints is the uses directive, which is used by package exporters to constrain package importers.

The OSGi specification defines numerous types of bundle requirements, such as Import Package, Require Bundle, Fragment Host, and Execution Environment. However, it is expected that new types of requirements and capabilities for resolving them will be defined in the future. Additionally, not all capabilities will be provided by bundles; for example, screen size or available memory could be capabilities.

Conceptually, capabilities can simply be viewed as the properties or characteristics of a bundle or the environment and requirements can be viewed as a selection constraint over these capabilities. On the whole, requirements are more complex than capabilities. The selection constraint of a requirement has two orthogonal aspects: multiple and optional. For example, an imported package is not optional and not multiple, while an imported service could have multiple cardinality. Likewise, imported packages or services can be mandatory or optional.

Further, extends relationships allow a provider to extend another bundle. For example, a bundle fragment defines an extends relationship between a bundle and a host. Specifically, a given bundle requirement is a relationship that the bundle knows about in advance, as opposed to an extension, which may not have been known in advance by the bundle.

The process of resolving bundle requirements is complicated because it is non-trivial to find optimal solutions. The OSGi framework defines a run-time resolution process, which is concerned with many of the aspects
described above. However, a provisioning resolution process for bundle discovery and deployment is also necessary, which is similar to the framework resolution process, but more generic.  

2.2 Use cases

2.2.1 Provisioning

A repository can be used to simplify bundle provisioning by making it possible to create mechanisms to automate processing of deployment-related bundle requirements. The OSGi Framework already handles bundle requirement processing, such as resolving imported packages, required bundles, host bundles, and execution environments. However, the framework can only reason about and manage these requirements after bundles have been installed locally.

When a bundle is installed, all its requirements must be fulfilled. If its requirements cannot be resolved, the bundle will fail to install or resolve. The missing requirements can potentially be resolved by installing other bundles; however, these bundles not only provide new capabilities, but they can also add new requirements that need to be resolved. This is a recursive process.

Downloaded bundles are usually licensed. Licensing issues are complex and dependent on the vendor of the bundle. The way a bundle is licensed may seriously affect the way the bundle can be downloaded. Many organizations require their employees to read the license before they download the actual artifact because many licenses contain an implicit agreement.

2.2.2 Build

It is also possible to use OSGi bundle meta data to resolve compile time dependencies as shown in the open source project Sigil. This use case follows a similar pattern to the provisioning use case in that a known package is required to satisfy a compile time dependency so the bundle used to satisfy that dependency can be downloaded on demand.

As with provisioning during the build process it is often necessary for a developer to read and/or acknowledge a license before using a library.

Compared to the "static" repositories defined in the provisioning use case Sigil also treats the workspace as a repository from which to satisfy dependencies for other projects in the workspace.

2.2.3 Management

System administrators are tasked with managing the set of bundles used in a corporate environment. As they are often not the original developer of the bundles they greatly benefit from helpful diagnostic information that can tell them why a certain resolution failure occurs (in order to find out who to contact/blame to get the problem fixed).

System administrators require the ability to:

- Check that a given set of bundles can deploy within a framework – prior to deploying the application.
- Browse and search bundles from a repository to find specific characteristics such as bundles built on a certain date, or by a certain author.
- Navigate through bundle dependencies to find out what the impacts of deleting or upgrading a bundle may be
• Create repositories from various sources including existing enterprise archives, filesystems, public repositories such as maven central, etc.

Enterprise repositories are often federated to enable simplified management of sub sets of bundles, there may be many duplicate bundle entries in these federations. The rate of deployment to these federated repositories can vary depending on the work rate of the team managing the repository. For example in a finance environment front office traders are often releasing several artifacts a day to update algorithms used during trading where as back office systems are more likely to update on a weekly or monthly basis.

3 Problem Description

The problem this document addresses is that end users can not discover and deploy available bundles from a single, trusted, point of access.

3.1 Resolver Strategies

The resolver capability of OBR may do one of a number of things:

1. Resolve against an “empty” framework i.e. calculate the set of resources that are required to provide some top level function outside the current installed bundles. This usecase is applicable if you want to use OBR as a packaging tool - i.e. to create static lists of bundles to be deployed in "clean" frameworks.

2. Resolve against the current framework such that the resolution will resolve in the context of the existing bundles in the framework - i.e. deploy bundle x and y in a framework and drag in all bundles not yet deployed in the framework in the resolved state.

3.1.1 Existing resolver strategies

● The OBR implementation on Felix also supports resolving against an empty container (This being case (1) from above.

● In Sigil we support resolving against an empty initial set of dependencies (case 1 again) but in this case the usecase is resolving OSGi bundles that will satisfy classpath requirements vs runtime requirements and the deployment process is different in that the target is a file system/eclipse classpath container vs an OSGi framework.

● In Nimble the resolver pulls bundles (or other resources – using pluggable deployment schemes) into a running framework (2 - though 1 is a trivial extension) based on requirements and policies and deploys bundles in different states (active/resolved/etc).

This specification will address both usecases.
4 Requirements

4.1 Functional
- Must not preclude browsing access to a bundle repository via a web server
- Provide access to a bundle repository so that bundles can be directly installed after discovery
- Handle dependency resolution so that bundles can be deployed without generating errors
- Allow repositories to be linked, creating a federated repository
- Allow multiple repositories to be used during resolution
- Allow ordering on repositories such that certain repositories are preferred over others during resolution
- Provide programmatic (service) access to the repository

4.2 Discovery
- Search bundles by keywords
- Search by namespace
- Provide filtering capabilities on execution environment
- Licensing conditions must be available before downloading the artifacts

4.3 Dependency Resolution
- Must be able to find bundles that can solve any unresolved requirements
- Must not preclude the ability to attach fragments or other extensions as part of a resolution
- Must handle all the requirements/capabilities and their directives as defined in the OSGi R4 specifications
- Must reuse the semantics of requirements/capabilities defined in RFC 154 where ever possible

4.4 Security
- A repository provider must be able to control the members of a federated repository.

4.5 Non Functional
- The repository must be able to scale to ten thousand bundles
- Compliant with other OSGi services
5 Technical Solution

5.1 Entities

- Capability – A named set of properties
- Requirement – An assertion on a resource's capabilities
- Wire – A connection between a requirement and a capability associated with a resource
- Resource – A description of a bundle or other artifact that can be installed on a device. A resource provides capabilities and requires capabilities of other resources or the environment
- Environment – An environment provides options and constraints to the potential solution of a Resolver resolve operation
- Resolver – an object that can be used to find dependent and extension resources based on a set of requirements and a supplied environment
- Resolution – a Map of resources to wires
- Repository – Provides access to a set of resources
- RepositoryListener – a service that can listen to changes in a Repository's contents
- Repository File – An XML file that can be referenced by a URL. The content contains meta data of resources and referrals to other repository files. It can be a static file or generated by a server.
5.1.1 Domain Object Model

- **Environment**
  - 1 contains 0..n
  - 1 requires 0..n
  - 1 provides 0..n

- **Resource**
  - 1 requires 0..n
  - 1 provides 0..n

- **Requirement**
- **Capability**

5.1.2 Service Model

- **Resolution**
  - created by

- **ResolverClientImpl**
  - used by
  - 1 discovers 1..n

- **Resolver**
  - created by
  - discovers 0..n

- **Environment**
  - 1 discovers 0..n

- **Repository**
  - 1 creates
5.1.3 Repository Management Model

5.2 Overview

The key architecture of the OSGi Repository is a generic description of a resource and its dependencies. A resource can represent a physical or virtual resource. Physical resources include elements such as:

- A bundle,
- A certificate
- A configuration file

Virtual resources include elements such as:

- A service
- A particular bundle revision in a framework
- A process such as a web container

The purpose of the repository is to discover potential resources that can be instantiated by an OSGi framework and deploy these resources without causing install errors due to missing dependencies.

For this purpose, each resource has a list of requirements on other resources or the environment, a list of capabilities that are used to satisfy the requirements. This is depicted in the following picture.
5.2.1 Resource

A resource is identified by a map of attributes accessed by the getAttributes() method. There are three mandatory attributes which all resources must define:

- **namespace** – The type of this resource; for example, a bundle has the namespace "osgi.bundle". This maps to the RFC 154 concept of a namespace
- **symbolicName** – A name for the resource that is globally unique for the function of the resource. There can exist multiple resources with the same name but a different version or namespace. Two resources with the same namespace, name and version are considered to be identical. For a bundle, this is normally mapped to the Bundle-SymbolicName manifest header
- **version** – A version for the resource. This must be a version usable by the OSGi Framework version class. For a bundle this is mapped to the Bundle-Version manifest header

The resource can contain any user-defined attributes. The attributes can be obtained with the getAttributes method. Attributes are case sensitive. The following attributes are predefined for all resource types:

1. **copyright** – A copyright statement. This element is derived from the Bundle-Copyright manifest header
2. **description** – A textual description of the resource. This must be unformatted text. This element is derived from the Bundle-Description manifest header
3. **documentation** – A URI to the documentation. This element is derived from the Bundle-DocURL manifest header
4. **license** – A URI to the license file. This element is derived from the Bundle-License manifest header

For physical resources the following attributes are also defined:
1. checksum – The base64 encoded value of the checksum for this resource
2. checksumAlgo – The algorithm used to calculate the checksum for this resource
3. location – A URI where this resource can be downloaded
4. scm – A URI to a source distribution of the resource. This element is derived from the Bundle-Source manifest header.
5. size – The size of the resource in bytes.

The type of the attribute can be one of the following:
- boolean – A Java Boolean object
- string – Java String object
- uri – Java URI object
- long – A Java Long object
- double – A Java Double object
- list – A comma separated list of values. White space around the commas must be discarded. The values cannot contain commas.

Attributes that are of a specific type are compared and filtered according to their type.

5.2.2 Capabilities
A capability is anything that can be described with a set of attributes and directives. Examples of capabilities are:
- A package export
- A service export
- A fragment host
- A bundle
- A certificate
- A configuration record
- An Execution Environment
- A Display type
- Memory size
- Accessories
Capabilities are named. The reason they are named is so that they can only be provided to requirements with the same name. This is necessary because an attribute from two capabilities could have different meanings but still use the same name. To prevent these name clashes, the capabilities (and the requirements that they can resolve) are named. This specification defines names necessary to handle the capability/requirements of the OSGi Bundle Manifest.

Capabilities publish a set of attributes that identify the capability and allow requirements to be matched. In order to simplify the task of expressing capabilities in the rest of this document the following nomenclature will be defined:

capability <namespace> {
  <attributeName> = <attributeValue>
  <directiveName> := <directiveValue>
}

All capabilities must publish one identity attribute that matches the name of their namespace, e.g.

capability osgi.package {
  osgi.package = com.example
}

Along side attributes capabilities also support directives, directives provide information to a resolver to allow for extensions to resolution strategies for specific use cases. One example of this is the OSGi export-package “uses” directive that specifies the name of other packages that this capability is wired to.

Capabilities can originate from other resources, but they can also be innate in the environment. This specification allows any bundle to dynamically provide capabilities to the environment.

5.2.3 Requirements

A requirement expressed as a filter on a resource. Just like a capability, a requirement is named and support attributes and directives. All requirements should define a filter directive that matches the OSGi filter syntax. The filter must only be matched to capabilities with the same namespace. A requirement matches a capability when its filter matches any of the attributes defined in that capability.

The syntax of the filter is the OSGi filter syntax. A filter was chosen because it allows the specification of arbitrary complex assertions. The disadvantage is that a filter is more or less opaque for the software, making it harder to provide assistance to the end user why certain resources are included and other not. A requirement can therefore optionally contain a description attribute. This is a short description that is applicable when a requirement is the cause for the selection of a resource.

For example, a package import is translated to a requirement for a package capability. If a bundle exports this package and it is selected, then the reason is the requirement for the import package.

Requirements also support directives that can be used to define behaviors such as optional imports, multiple cardinality, mandatory capability attributes, etc. The following requirement directives are defined:

resolution: dynamic | optional | mandatory (default mandatory)
cardinality: single | multiple (default single)
mandatory: attributeName (, attributeName)*

Multiple requirements are satisfied when there are one or more solutions, all solutions are usable. Package imports are for example are neither optional nor multiple. When there are multiple capabilities provided, the resolver must choose one of the applicable solutions. This is further discussed in the resolver section.
5.2.4 Extends

Requirements select a set of useful or required resources, the Extend reverses this model; an Extend selects resources for which it might be useful. For example, a fragment can extend its host or a bundle can act as a plugin for another bundle by providing a certain service. In both cases, the bundle that provides the extension is aware of the host but the host not of the providers.

TODO describe extends usage via requirements

5.3 Repository

Repositories are services published to the OSGi registry they may be backed by a range of different technologies, including static XML files, databases, carrier pigeons, etc.

This specification does not provide any implementations of a Repository these are left open, however some suggested repository implementations are:

- SimpleRepository – takes an array of Resource objects to form an inmemory repository
- FederatedRepository – takes an array of sub repositories
- XMLRepository – takes an XML file URL as a construction parameter

The API of the Repository is:

- Iterator<Resource> discoverResources(String filterExpr) - Discover any resources that specify attributes that match the given filter.
- Resource getResource(String namespace, String symbolicName, String version) - Lookup a resource based on a supplied namespace, name and version.
- Collection<Capability> findProviders(Requirement... requirements) - Find any capabilities from resources contained in this repository that can potentially satisfy the supplied requirements.
- long getIncrement() - A counter to indicate the state of the repository, clients can use this to check if there have been any changes to the resources contained in this repository. A repository implementation that supports external modifications should return a different increment.
- RepositoryDelta getDelta(long sinceIncrement) - Provides a mechanism to query the changes that have occurred to this repository since the specified increment.

5.3.1 RepositoryDelta

A repository delta lists the changes to a repository between to increments. The API of a RepositoryDelta is:

- List<Resource> getAddedResources();
- List<Resource> getChangedResources();
- List<Resource> getRemovedResources();
5.3.2 RepositoryListener and RepositoryChangeEvent

Depending on the underlying technology used to implement a Repository it may be possible to update the list of Resources that a Repository provides. If this happens it is necessary to inform the Resolver and any associated management tools of these changes. The RepositoryListener is a service interface that is looked up using the white board pattern by Repository implementations to notify of such changes:

The API of the RepositoryListener is:

- void repositoryChanged(RepositoryChangeEvent event);

The API of the RepositoryChangeEvent is:

- long getIncrement()
- Repository getRepository()

5.4 Environment

In order for a resolver to solve a set of requirements it needs an environment to work against. The environment supplies the existing wires between resources that the resolver needs to take into account and provides additional capabilities that match requirements. The environment is a Java interface that can proxy many different underlying implementations.

The Environment interface provides the following methods:

- Collection<Capability> findProviders(Requirement... requirements) - Find any capabilities that can potentially provide a match to the supplied requirements. A resolver should use the iteration order or the returned capability collection to infer preference in the case where multiple capabilities match a requirement. Capabilities at the start of the iteration are implied to be preferred over capabilities at the end.

- Map<Resource, List<Wire>> getWiring() - An immutable map of wires between resources.

5.5 Resolving

The resolver is a complicated process requiring difficult choices that likely require user intervention and/or policies. This includes but is not limited to:

- The addition of optional resources.
- Attachment of fragments
- Resolve time policy for version ranges
- Decisions related to licensing, bundle size, performance etc

The implementation of the Resolver object can provide these capabilities as it sees fit. The mechanism by which a client configures these capabilities is not defined here and is instead left up to implementations to provide an API or a management interface as they see fit. Though one potential mechanism is to use requirement directives to influence the resolver strategy.
The API of the Resolver is:

```java
Map<Resource, List<Wire>> resolve(Environment environment, Requirement...requirements) - Attempt to resolve the requirements based on the specified environment and return any new resources or wires to the caller. For a given resolve call an environment should return a consistent set of capabilities and wires. The simplest mechanism of achieving this is by creating an immutable snapshot of the environment state and passing this to the resolve method.
```

The Repository resolver is in many ways similar to the Framework resolver. Implementations may therefore strive to use the same code. However, the problem that the Framework resolver solves is subtly different from what the Repository resolver solves. First, the Repository resolver is more generic; it handles more than packages and bundles. This is the reason for the generic requirement/capability model instead of using the manifest directly. Second, the Framework creates a wiring between a set of installed bundles. In contrast, the Repository resolver installs a set of bundles. Despite these subtle differences, the logic behind these resolvers is very similar and can clearly share implementation code.

### 5.6 Code Examples

#### 5.6.1 Install into framework

```java
public void installIntoFramework(PotentialResource resource)
    throws IllegalStateException, InterruptedException, BundleException {
    EnvironmentBuilder envBuilder = new EnvironmentBuilder();
    envBuilder.addEnvironment(_ctx);
    Environment framework = envBuilder.buildEnvironment();

    // require the resource
    RequirementBuilder reqBuilder = new RequirementBuilder();
    reqBuilder.addRequirement(resource);
    try {
        // attempt to resolve
        Map<Resource, List<Wire>> delta = _resolver
            .resolve(framework, reqBuilder.getRequirements());

        // for all resolved bundle resources install them
        for (Resource r : delta.keySet()) {
            String namespace = (String)
                r.getAttributes().get(Resource.NAMESPACE_ATTRIBUTE);
            if (Resource.BUNDLE_NAMESPACE.equals(namespace)) {
                _ctx.installBundle(r.getLocation());
            } else {
                System.err.println("No action defined for " + namespace);
            }
        }
    } catch (ResolutionException e) {
        System.out.println("Failure :(");
    }
}
```
public void resolveIntoEmpty(Resource resource) 
    throws InterruptedException {
    // build empty environment
    EnvironmentBuilder envBuilder = new EnvironmentBuilder();
    envBuilder.addRepositories(_ctx);
    Environment framework = envBuilder.buildEnvironment();

    // require the resource
    RequirementBuilder reqBuilder = new RequirementBuilder();
    reqBuilder.addRequirement(resource);

    // attempt to resolve
    _resolver.resolve(framework, reqBuilder.getRequirements());
}

5.6.3 Resolve optional dependencies for a previous resolution

public void listCompleteResolution(final Resource resource) 
    throws InterruptedException {
    EnvironmentBuilder envBuilder = new EnvironmentBuilder();
    envBuilder.addEnvironment(_ctx);
    Environment framework = envBuilder.buildEnvironment();

    // require the resource
    RequirementBuilder reqBuilder = new RequirementBuilder();
    reqBuilder.addRequirement(resource);

    try {
        // attempt to resolve
        Map<Resource, List<Wire>> resolution = _resolver
            .resolve(framework, reqBuilder.getRequirements());

        System.out.println("Success :)");

        for (Resource r : resolution.keySet()) {
            System.out.println("Resolved " + r + " for " + resource);
            for (Requirement req : r.getDeclaredRequirements(null)) {
                if (isOptional(req) && !isSatisfied(req, resolution)) {
                    // force mandatory
                    req = Requirements.mandatory(req);
                    boolean preserved = preserveStateListOptionalRequirementsFor(
                        resolution, req);
                    if (!preserved) {
                        ignoreStateListOptionalRequirementsFor(resolution, req);
                    }
                }
            }
        }
    } catch (ResolutionException e) {
        System.out.println("Failure :(");
    }
private boolean preserveStateListOptionalRequirementsFor(
    final Map<Resource, List<Wire>> resolution,
    final PotentialRequirement req) throws InterruptedException {
    EnvironmentBuilder envBuilder = new EnvironmentBuilder();
    envBuilder.addEnvironment(_ctx);
    envBuilder.addWires(resolution);

    try {
        Map<Resource, List<Wire>> subResolution = _resolver
            .resolve(envBuilder.buildEnvironment(), req);

        for (Wire r : subResolution.get(req.getResource())) {
            System.out.println("Optional requirement " + req + "+ from "+
                + req.getResource() + " is satisfied by" + r);
        }

        return true;
    } catch (ResolutionException e) {
        return false;
    }
}

private void ignoreStateListOptionalRequirementsFor(
    final Map<Resource, List<Wire>> resolution,
    final Requirement req) throws InterruptedException {
    EnvironmentBuilder envBuilder = new EnvironmentBuilder();
    envBuilder.addRepositories(_ctx);

    RequirementBuilder reqBuilder = new RequirementBuilder();
    for (Resource r : resolution.keySet()) {
        reqBuilder.addRequirement(r);
    }
    reqBuilder.addRequirement(req);

    try {
        Map<Resource, List<Wire>> subResolution = _resolver
            .resolve(envBuilder.buildEnvironment(), reqBuilder.getRequirements());

        for (Wire r : subResolution.get(req.getResource())) {
            System.out.println("Optional requirement " + req + "+ from "+
                + req.getResource() + " is satisfied by" + r);
        }
    } catch (ResolutionException e) {
        System.out.println("Optional requirement " + req + "+ from "+
            + req.getResource() + " cannot be satisfied");
    }
}
5.7 XML Schema

5.7.1 Namespace

The XML namespace is:

http://www.osgi.org/xmlns/obr/v1.0.0

```xml
<obr:repository name='Untitled' time='20051210072623.031'
    xmlns:obr="http://www.osgi.org/xmlns/obr/v1.0.0">
    ...
</obr:repository>
```

5.7.2 The XML Structure

The following BNF describes the element structure of the XML file:

```plaintext
repository ::= (referral | resource) *
resource ::= attribute * require * capability *
require ::= attribute * directive *
capability ::= attribute * directive *
```

5.7.3 Repository

The `<repository>` tag is the outer tag of the XML document. It must contain the following attributes:

1. `name` – The name of the repository. The name may contain spaces and punctuation.
2. `increment` – A long value counter to indicate the state of the repository, clients can use this to check if there have been any changes to the resources contained in this repository

The repository element can only contain referral and resource elements.

```xml
<obr:repository name='Untitled' increment='1'
    xmlns:obr="http://www.osgi.org/xmlns/scr/v1.0.0">
</obr:repository>
```

5.7.4 Referral

A referral points to another repository XML file. The purpose of this element is to create a federation of repositories that can be accessed as a single repository. The referral element can have the following attributes:

1. `depth` – The depth of referrals this repository acknowledges. If the depth is 1, the referred repository must included but it must not follow any referrals from the referred repository. If the depth is more than one, referrals must be included up to the given depth. Depths of referred repositories must also be obeyed. For example, if the top repository specifies a depth of 5, and the 3 level has a depth of 1, then a repository included on level 5 must be discarded, even though the top repository would have allowed it.
2. `url` – The URL to the referred repository. The URL can be absolute or relative from the given repository's URL.

For example:

```
<referral depth="1" url="http://www.agute.biz/bundles/repository.xml"/>
```
5.7.5 Resource

The `<resource>` element describes a general resource with attributes, requirements, and capabilities. The resource element has the following attributes.

1. `name` – The name of the resource. In case of a bundle, this is the Bundle Symbolic Name.
2. `namespace` – The namespace of the resource
3. `version` – The version of the resource. Version must follow the major, minor, micro, qualifier format as used the Framework’s version class.
4. `uri` – The location where this resource can be downloaded. This attribute is optional as not all resources have an associated physical resource

5.7.6 Require

The `<require>` element describes one of the requirements that the enclosing resource has on its environment. A requirement is of a specific named type and contains a filter that is applied to all capabilities of the given type. Therefore, the requirement element has the following attributes:

- `namespace` – The namespace of the requirement. The filter must only be applied to capabilities that have the same namespace
- `filter` – The filter expression. The syntax must follow the OSGi filter syntax. The filter must correctly compare versions.
- `extend` – If this requirement indicates it extends the resource it is wired to. The value is true or false. Default value is false.

The content of the require element is a description of the requirement. It may be used to explain to the user why a particular resource was selected.

For example:

```xml
<require namespace='package'
    filter='(&(name=org.osgi.test.cases.util)(version>=1.1.0))'>
    Import package org.osgi.test.cases.util;version=1.1.0
</require>
```

This example requires that there is at least one exporter of the org.osgi.test.cases.util package with a version higher than 1.1.0

5.7.7 Capability

The capability element is a named set of type properties. A capability can be used to resolve a requirement if the resource is included. A capability has the following attribute:

- `namespace` – The namespace of the capability

Only requirements with the same namespace must be able to match this capability.

The capability can contain two elements, attribute and directive. These elements have the same set of attributes.
- **name** – The name of the property
- **value** – The value of the property
- **type** – The type of the property. This must be one of:
  - string – A string value, which is the default.
  - version – An OSGi version as implemented in the OSGi Version class.
  - uri – A URI
  - long –
  - double –
  - list – A comma separated list of values. White space must be discarded, the values can not contain commas.

The following example shows a package export:

```xml
<capability namespace='package'>
  <attribute value='org.eclipse.core.internal.resources' name='name'/>
  <attribute value='0.0.0' type='version' name='version'/>
  <directive value='true' name='x-internal'/>
</capability>
```

### 5.8

#### 5.9 Sample XML File

```xml
<repository name='Untitled' time='20051210072623.031'>
  <resource version='3.0.0' name='org.osgi.test.cases.tracker'
    uri='org.osgi.test.cases.tracker-3.0.0.jar'>
    <attribute name='size' value='44405' />
    <attribute name='documentation' value='http://www.osgi.org' />
    <attribute name='copyright' value='Copyright (c) OSGi Alliance (2000, 2005). All Rights Reserved.' />
    <attribute name='category' value='osgi,test' />
    <capability namespace='osgi.wiring.bundle'>
      <attribute value='org.osgi.test.cases.tracker' name='symbolic.name'/>
      <attribute value='1' name='manifest.version'/>
      <attribute value='3.0.0' type='version' name='version'/>
    </capability>
    <capability namespace='osgi.wiring.package'>
      <attribute value='org.osgi.test.cases.tracker' name='name'/>
      <attribute value='0.0.0' type='version' name='version'/>
    </capability>
    <require namespace='osgi.wiring.package'
      filter='(@amp;(name=org.osgi.test.cases.util)(version&gt;=1.1.0))'/>
  </resource>
</repository>
```
5.10 Bundle Manifest Header Mapping

The following sections describe how the Bundle-Manifest sections are mapped to the generic Requirement/Extend and Capability model.

5.10.1 Bundle

Every bundle must include a capability namespace 'osgi.wiring.bundle' with the following attributes:

- version – Version, must be set, type version.
- osgi.wiring.bundle – The symbolic name of the bundle.
- manifestversion – Version of the Manifest. Must be set, type version.
- fragment-attachment – If the fragment-attachment directive on the Bundle-SymbolicName is set. One of "always", "never", "resolve-time".
- singleton – If the singleton directive is set. True or false, string type.

Example:

```xml
<capability namespace='bundle'>
  <attribute value='1'  type='version' name='manifestversion'/>
  <attribute value='aQute.eclipse.osgi' name='symbolic.name'/>
  <attribute value='1.0.1' type='version' name='version'/>
</capability>
```

5.10.2 Import and Export Package: 'package'

An Export-Package header must be split into clauses and mapped to a capability.

- The type namespace is 'osgi.wiring.package'.
- The type name must be the name of the package.
- The 'version' property is the version. This property must be set and of type version.
- Add bundle-symbolic-name and bundle-version attributes
- Remaining attributes should be added to the capability.
- Mandatory attributes must be put in a 'set' typed directive with the name mandatory. If no mandatory attributes are defined.
- Uses directives should be in a directive with the name uses.

For example:

```xml
<capability namespace='osgi.wiring.package'>
  <attribute value='0.0.0' type='version' name='version' />
  <attribute value='org.osgi.test.cases.tracker' name='name' />
</capability>
```
<directive value='vendor, var' name='mandatory' type='set'/>
<directive value='org.osgi.framework' name='uses' type='set'/>
</capability>

An Import-Package clause is mapped to a Requirement.

- The type namespace is 'osgi.wiring.package'
- The filter must assert:
  - name – the name of the package to be imported
  - version – Version or version range (the filter supports the version range syntax). E.g. (version=[1,2])
  - Any custom attributes for equality

If the clause has a directive of resolution=optional, then the Requirement is set to OPTIONAL, otherwise to UNARY. For example:

<require namespace='package'
  filter='(&(name=org.osgi.test.cases.util)(version=1.1.0))'/>

5.10.3 Require-Bundle

Require-Bundle is translated to a Requirement with the following aspects.

- namespace is 'osgi.wiring.bundle'

Assert:

- symbolic.name – The symbolic name of the bundle
- version – Version range of the required bundle. (version=[1,2])

If resolution directive is true, the requirement is UNARY, otherwise OPTIONAL.

For example:

<require namespace="osgi.wiring.bundle" name='org.eclipse.ui'
  filter="(&(symbolic.name=org.eclipse.ui)(version=0.0.0))"/>

5.10.4 Fragment-Host

The Fragment-Host is a Requirement in the 'osgi.wiring.host' with the following filter assertions:

- name – The name of the host bundle that this fragment attaches to
- version – Version range of the required bundle. (version=[1,2])

<requirement optional='false' multiple='false' extend='true'
  namespace="osgi.wiring.host" name='org.eclipse.core.resources'
5.10.5 RFC 154 Mapping

TBD.

5.10.6 Bundle-ExecutionEnvironment

The Bundle Execution Environment header is mapped to a requirement. The capabilities of this requirement must be set by the environment. Each support environment is an element of a multi-valued property called 'ee' in a 'ee' capability.

The filter must assert on 'ee' with the defined names for ee's. For example, if the bundle can run on J2SE 1.4:

```<require namespace="osgi.wiring.ee" filter="(|(ee=J2SE-1.4))"/>
```

This requirement is UNARY.

6 Open Issues

6.1 Testing of RepositoryListener

As this functionality is optional it is not possible to test for spec compliance. One solution to this would be to add a WritableRepository sub interface that defines an addResource/removeResource methods these could be used to trigger the events, however it is not clear that adding this capability is within the bounds of the original requirements for this RFC.

6.2 Extends

Mapping of fragment bundles is not yet completely specified – open issues are how to express relationship without overly complicating the model and how a resolver should treat extensions. Some extensions imply a logical merge of extendee and extender (such as the fragment bundle case) but it is not clear that this always the case. Consider a relationship where a bundle that provides bytecode weaving can modify the behaviour of another bundle – should this be modeled as an extension? Suggestion is that this does not effect the class space of the extendee?

6.3 Licensing

The value of the repository would be greatly enhanced if we would support a licensing model. Currently, certain bundles require the authentication so they can not be directly downloaded. This makes OBR like solutions impossible.
6.4 Problem Analysis

The resolver API is potentially useful during the development lifecycle and to help in runtime diagnostics when deployments go wrong. However due to the range of resolution algorithms that are possible, providing a meaningful general purpose API that can help users with diagnostic problems is very hard.

This API should reuse any API classes defined by RFC 154 for generic capabilities and work with the proposed wiring API updates.

7 Java Documentation
# Package Summary

<table>
<thead>
<tr>
<th>Package Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.osgi.framework.wiring</td>
<td>Framework Wiring Package Version 1.0.</td>
<td>31</td>
</tr>
<tr>
<td>org.osgi.service.obr</td>
<td>OSGi Bundle Repository Package Version 1.0.</td>
<td>39</td>
</tr>
</tbody>
</table>
Package org.osgi.framework.wiring

Framework Wiring Package Version 1.0.

See:  

Description

<table>
<thead>
<tr>
<th>Class Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResourceConstants</td>
<td>32</td>
</tr>
<tr>
<td>Defines standard names for the attributes, directives and name spaces for resources, capabilities and requirements.</td>
<td></td>
</tr>
</tbody>
</table>

Package org.osgi.framework.wiring Description

Framework Wiring Package Version 1.0.

Bundles wishing to use this package must list the package in the Import-Package header of the bundle's manifest. For example:

```
Import-Package: org.osgi.framework.wiring; version="[1.0,2.0)"
```
Class ResourceConstants

org.osgi.framework.wiring

java.lang.Object

final public class ResourceConstants
extends Object

Defines standard names for the attributes, directives and name spaces for resources, capabilities and requirements.

The values associated with these keys are of type String, unless otherwise indicated.

Version:
$Id: a41a70568d1a93bf270977f08cf085ff739fb7f7 $
### Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResourceConstants()</td>
<td>38</td>
</tr>
</tbody>
</table>

### Field Detail

**WIRING_PACKAGE_NAMESPACE**

public static final String **WIRING_PACKAGE_NAMESPACE** = "osgi.wiring.package"

Name space for package capabilities and requirements. For capability attributes the following applies:

1. The osgi.wiring.package attribute contains the name of the package.
2. The version attribute contains the the org.osgi.framework.Version of the package if one is specified or org.osgi.framework-Version.emptyVersion if not specified.
3. The bundle-symbolic-name attribute contains the symbolic name of the resource providing the package if one is specified.
5. All other attributes are of type String and are used as arbitrary matching attributes for the capability.

A resource provides zero or more package capabilities (this is, exported packages) and requires zero or more package requirements (that is, imported packages).
**WIRING_BUNDLE_NAMESPACE**

public static final String WIRING_BUNDLE_NAMESPACE = "osgi.wiring.bundle"

Name space for bundle capabilities and requirements. For capability attributes the following applies:

1. The osgi.wiring.bundle attribute contains the symbolic name of the bundle.
3. All other attributes are of type String and are used as arbitrary matching attributes for the capability.

A non-fragment resource with the osgi.bundle name space provides exactly one † bundle capability (that is, the bundle can be required by another bundle). A fragment resource must not declare a bundle capability. A resource requires zero or more bundle requirements (that is, required bundles).

† A resource with no symbolic name must not provide a bundle capability.

---

**WIRING_HOST_NAMESPACE**

public static final String WIRING_HOST_NAMESPACE = "osgi.wiring.host"

Name space for host capabilities and requirements. For capability attributes the following applies:

- The osgi.wiring.host attribute contains the symbolic name of the bundle.
- All other attributes are of type String and are used as arbitrary matching attributes for the capability.

A non-fragment resource with the osgi.bundle name space provides zero or one † host capabilities zero or one host capability. A fragment resource must declare exactly one host requirement.

† A resource with no bundle symbolic name must not provide a host capability.

---

**RESOURCE_BUNDLE_NAMESPACE**

public static final String RESOURCE_BUNDLE_NAMESPACE = "osgi.bundle"

The name space for OSGi bundle resources

---

**RESOURCE_CONTENT_ATTRIBUTE**

public static final String RESOURCE_CONTENT_ATTRIBUTE = "content"

A resource attribute used to specify the content of a resource. Typically this specifies a URI which can be used to locate the content of the resource.

---

**RESOURCE_SYMBOLIC_NAME_ATTRIBUTE**

public static final String RESOURCE_SYMBOLIC_NAME_ATTRIBUTE = "symbolic-name"

A resource attribute used to specify the resource symbolic name.
**RESOURCE_VERSION_ATTRIBUTE**
public static final String RESOURCE_VERSION_ATTRIBUTE = "version"

A resource attribute used to specify the resource version.

**RESOURCE_NAMESPACE_ATTRIBUTE**
public static final String RESOURCE_NAMESPACE_ATTRIBUTE = "namespace"

A resource attribute used to specify the resource name space.

**REQUIREMENT_FILTER_DIRECTIVE**
public static final String REQUIREMENT_FILTER_DIRECTIVE = "filter"

A requirement directive used to specify a capability filter. This filter is used to match against a capability's attributes.

**REQUIREMENT_RESOLUTION_DIRECTIVE**
public static final String REQUIREMENT_RESOLUTION_DIRECTIVE = "resolution"

A requirement directive used to specify the resolution type for a requirement. The default value is mandatory.

See Also: mandatory, optional

**REQUIREMENT_RESOLUTION_MANDATORY**
public static final String REQUIREMENT_RESOLUTION_MANDATORY = "mandatory"

A directive value identifying a mandatory requirement resolution type. A mandatory resolution type indicates that the requirement must be resolved when the resource is resolved. If such requirement cannot be resolved, the resource fails to resolve.

See Also: REQUIREMENT_RESOLUTION_DIRECTIVE

**REQUIREMENT_RESOLUTION_OPTIONAL**
public static final String REQUIREMENT_RESOLUTION_OPTIONAL = "optional"

A directive value identifying an optional requirement resolution type. An optional resolution type indicates that the requirement is optional and the resource may be resolved without requirement being resolved.

See Also: REQUIREMENT_RESOLUTION_DIRECTIVE
**REQUIREMENT_EFFECTIVE_DIRECTIVE**

public static final String REQUIREMENT_EFFECTIVE_DIRECTIVE = "effective"

A requirement directive used to specify the effective time for the requirement. The default value is resolve.

See Also: resolve, active

---

**EFFECTIVE_RESOLVE**

public static final String EFFECTIVE_RESOLVE = "resolve"

A directive value identifying a capability or requirement that is effective at resolve time. Capabilities and requirements with an effective time of resolve are the only capabilities which are processed while resolving a resource.

See Also: REQUIREMENT_EFFECTIVE_DIRECTIVE, CAPABILITY_EFFECTIVE_DIRECTIVE

---

**EFFECTIVE_ACTIVE**

public static final String EFFECTIVE_ACTIVE = "active"

A directive value identifying a capability or requirement that is effective at active time. Capabilities and requirements with an effective time of active are ignored while resolving a resource.

See Also: REQUIREMENT_EFFECTIVE_DIRECTIVE, CAPABILITY_EFFECTIVE_DIRECTIVE

---

**REQUIREMENT_VISIBILITY_DIRECTIVE**

public static final String REQUIREMENT_VISIBILITY_DIRECTIVE = "visibility"

A requirement directive used to specify the visibility type for a requirement. The default value is private. This directive must only be used for requirements with the require bundle name space.

See Also: private, reexport

---

**REQUIREMENT_VISIBILITY_PRIVATE**

public static final String REQUIREMENT_VISIBILITY_PRIVATE = "private"

A directive value identifying a private visibility type. A private visibility type indicates that any packages that are exported by the required bundle are not made visible on the export signature of the requiring bundle.

See Also: REQUIREMENT_VISIBILITY_DIRECTIVE
Interface CapabilityProvider

**REQUIREMENT_VISIBILITY_REEXPORT**

```java
public static final String REQUIREMENT_VISIBILITY_REEXPORT = "reexport"
```

A directive value identifying a reexport visibility type. A reexport visibility type indicates any packages that are exported by the required bundle are re-exported by the requiring bundle.

**REQUIREMENT_CARDINALITY_DIRECTIVE**

```java
public static final String REQUIREMENT_CARDINALITY_DIRECTIVE = "cardinality"
```

A requirement directive used to specify the cardinality for a requirement. The default value is singular.

See Also:
- multiple, singular

**REQUIREMENT_MULTIPLE_CARDINALITY**

```java
public static final String REQUIREMENT_MULTIPLE_CARDINALITY = "multiple"
```

A directive value identifying a multiple cardinality type.

**REQUIREMENT_SINGULAR_CARDINALITY**

```java
public static final String REQUIREMENT_SINGULAR_CARDINALITY = "singular"
```

A directive value identifying a singular cardinality type.

**CAPABILITY_USES_DIRECTIVE**

```java
public static final String CAPABILITY_USES_DIRECTIVE = "uses"
```

A capability directive used to specify the comma separated list of package names a capability uses.

**CAPABILITY_EFFECTIVE_DIRECTIVE**

```java
public static final String CAPABILITY_EFFECTIVE_DIRECTIVE = "effective"
```

A capability directive used to specify the effective time for the capability. The default value is resolve.

See Also:
- resolve, active

**CAPABILITY_MANDATORY_DIRECTIVE**

```java
public static final String CAPABILITY_MANDATORY_DIRECTIVE = "mandatory"
```
A capability directive used to specify the comma separated list of mandatory attributes which must be specified in the filter of a requirement in order for the capability to match the requirement. This directive must only be used for capabilities with the package name space.

### CAPABILITY_INCLUDE_DIRECTIVE

```java
public static final String CAPABILITY_INCLUDE_DIRECTIVE = "include"
```

A capability directive used to specify the comma separated list of classes which must be allowed to be exported. This directive must only be used for capabilities with the package name space.

### CAPABILITY_EXCLUDE_DIRECTIVE

```java
public static final String CAPABILITY_EXCLUDE_DIRECTIVE = "exclude"
```

A capability directive used to specify the comma separated list of classes which must not be allowed to be exported. This directive must only be used for capabilities with the package name space.

## Constructor Detail

### ResourceConstants

```java
public ResourceConstants()
```
Package org.osgi.service.obr

OSGi Bundle Repository Package Version 1.0.

See:  Description

### Interface Summary

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment</td>
<td>An environment provides options and constraints to the potential solution of a Resolver.resolve(Environment, Requirement...) operation.</td>
<td>40</td>
</tr>
<tr>
<td>OBRConstants</td>
<td></td>
<td>42</td>
</tr>
<tr>
<td>Repository</td>
<td>Represents a repository that contains resources.</td>
<td>44</td>
</tr>
<tr>
<td>RepositoryDelta</td>
<td>Service interface published to the OSGi registry to be notified when repositories change.</td>
<td>48</td>
</tr>
<tr>
<td>Resolver</td>
<td>A resolver is a service interface that can be used to find resolutions for specified requirements based on a supplied Environment.</td>
<td>50</td>
</tr>
</tbody>
</table>

### Class Summary

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>RepositoryChangeEvent</td>
<td>An event object that may be fired to RepositoryListener services to notify them of changes to a Repository</td>
<td>46</td>
</tr>
</tbody>
</table>

### Exception Summary

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResolutionException</td>
<td>Indicates failure to resolve a set of requirements.</td>
<td>49</td>
</tr>
</tbody>
</table>

Package org.osgi.service.obr Description

OSGi Bundle Repository Package Version 1.0.

Bundles wishing to use this package must list the package in the Import-Package header of the bundle's manifest. This package has two types of users: the consumers that use the API in this package and the providers that implement the API in this package.

Example import for consumers using the API in this package:

```
Import-Package: org.osgi.service.obr; version="[1.0,2.0)"
```

Example import for providers implementing the API in this package:

```
Import-Package: org.osgi.service.obr; version="[1.0,1.1)"
```
public interface Environment

An environment provides options and constraints to the potential solution of a Resolver.resolve(Environment, Requirement...) operation.

Environments provide capabilities options that the Resolver can use to satisfy requirements via the findProviders(Requirement...) method.

Environments also constrain solutions via the getWiring() method. A wiring consists of a map of existing resources to wires.

An environment may be used to provide capabilities via local resources and/or remote repositories.

A resolver may call the findProviders(Requirement...) and getWiring() method any number of times during a resolve using any thread. Environments may also be shared between several resolvers. As such implementors should ensure that this class is properly synchronized.

---

**Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>findProviders(org.osgi.framework.wiring.Capability)</td>
<td>Find any capabilities that can potentially provide a match to the supplied requirements.</td>
<td>40</td>
</tr>
<tr>
<td>getWiring()</td>
<td>An immutable map of wires between revisions.</td>
<td>40</td>
</tr>
</tbody>
</table>

**Method Detail**

**findProviders**

Collection<org.osgi.framework.wiring.Capability> findProviders(org.osgi.framework.wiring.Requirement... requirements)

Find any capabilities that can potentially provide a match to the supplied requirements.

A resolver should use the iteration order or the returned capability collection to infer preference in the case where multiple capabilities match a requirement. Capabilities at the start of the iteration are implied to be preferred over capabilities at the end.

**Parameters:**

- requirements - the requirements that a resolver is attempting to satisfy

**Returns:**

- an immutable collection of capabilities that match the supplied requirements

**getWiring**


---
An immutable map of wires between revisions. Multiple calls to this method for the same environment object must result in the same set of wires. TODO coordination?

**Returns:**
the wires already defined in this environment
public interface OBRConstants

Field Summary

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>String[] ATTRIBUTES</td>
<td>All attributes defined in this interface</td>
<td>43</td>
</tr>
<tr>
<td>String CHECKSUM_ALGO_ATTRIBUTE</td>
<td>The checksum algorithm used to calculate the CHECKSUM_ATTRIBUTE if not specified this is assumed to be SHA-256 - TODO need default?</td>
<td>42</td>
</tr>
<tr>
<td>String CHECKSUM_ATTRIBUTE</td>
<td>Checksum attribute of a revision</td>
<td>42</td>
</tr>
<tr>
<td>String COPYRIGHT_ATTRIBUTE</td>
<td>TODO</td>
<td>42</td>
</tr>
<tr>
<td>String DESCRIPTION_ATTRIBUTE</td>
<td>A human readable description of this revision</td>
<td>43</td>
</tr>
<tr>
<td>String DOCUMENTATION_URL_ATTRIBUTE</td>
<td>A URL where documentation for this revision can be accessed</td>
<td>43</td>
</tr>
<tr>
<td>String LICENSE_URL_ATTRIBUTE</td>
<td>TODO</td>
<td>43</td>
</tr>
<tr>
<td>String SCM_URL_ATTRIBUTE</td>
<td>TODO</td>
<td>43</td>
</tr>
<tr>
<td>String SIZE_ATTRIBUTE</td>
<td>The size of this revision in bytes.</td>
<td>43</td>
</tr>
</tbody>
</table>

Field Detail

CHECKSUM_ATTRIBUTE

public static final String CHECKSUM_ATTRIBUTE = "checksum"

Checksum attribute of a revision

CHECKSUM_ALGO_ATTRIBUTE

public static final String CHECKSUM_ALGO_ATTRIBUTE = "checksumAlgo"

The checksum algorithm used to calculate the CHECKSUM_ATTRIBUTE if not specified this is assumed to be SHA-256 - TODO need default?

COPYRIGHT_ATTRIBUTE

public static final String COPYRIGHT_ATTRIBUTE = "copyright"

TODO
Description Attribute

public static final String DESCRIPTION_ATTRIBUTE = "description"

A human readable description of this revision

Documentation Url Attribute

public static final String DOCUMENTATION_URL_ATTRIBUTE = "documentation"

A URL where documentation for this revision can be accessed

License Url Attribute

public static final String LICENSE_URL_ATTRIBUTE = "license"

TODO

Scm Url Attribute

public static final String SCM_URL_ATTRIBUTE = "scm"

TODO

Size Attribute

public static final String SIZE_ATTRIBUTE = "size"

The size of this revision in bytes. If this revision is a meta revision the corresponding attribute value must equal -1

Attributes

public static final String[] ATTRIBUTES

All attributes defined in this interface
public interface Repository

Represents a repository that contains resources.

Repositories may be registered as services and may be used as inputs to an Environment.findProviders(Requirement...) operation.

Repositories registered as services may be filtered using standard service properties.

Repositories that can be modified externally should monitor the OSGi registry for RepositoryListeners and fire RepositoryChangeEvents to inform listeners of any changes so they can take appropriate actions.

Version:
$Id: 20c788d7c43d4f5686b50c01d4f73ec72892a067 $

---

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>discoverResources(String filterExpr)</td>
<td>Discover any resources that specify attributes that match the given filter.</td>
</tr>
<tr>
<td>findProviders(org.osgi.framework.wiring.Requirement... requirements)</td>
<td>Find any capabilities from revisions contained in this repository that can potentially satisfy the supplied requirements.</td>
</tr>
<tr>
<td>getDelta(long sinceIncrement)</td>
<td>Provides a mechanism to query the changes that have occurred to this repository since the specified increment.</td>
</tr>
<tr>
<td>getIncrement()</td>
<td>A counter to indicate the state of the repository, clients can use this to check if there have been any changes to the revisions contained in this repository.</td>
</tr>
<tr>
<td>getResource(String namespace, String symbolicName, String version)</td>
<td>Lookup a revision based on a supplied namespace, name and version.</td>
</tr>
</tbody>
</table>

---

Method Detail

discoverResources

Iterator<org.osgi.framework.wiring.Resource> discoverResources(String filterExpr)

throws org.osgi.framework.InvalidSyntaxException

Discover any resources that specify attributes that match the given filter.

Parameters:
- filterExpr: A standard OSGi filter

Returns:
- List of resources matching the filters.

Throws:
- org.osgi.framework.InvalidSyntaxException - If the filter expression is invalid.
getResource

org.osgi.framework.wiring.Resource getResource(String namespace,
                         String symbolicName,
                         String version)

Lookup a revision based on a supplied namespace, name and version. TODO should we allow wild cards/ranges for version? What are semantics if version is open - highest, lowest, repository specific?

findProviders

Collection<org.osgi.framework.wiring.Capability> findProviders(org.osgi.framework.wiring.Requirement... requirements)

Find any capabilities from revisions contained in this repository that can potentially satisfy the supplied requirements.

getIncrement

long getIncrement()

A counter to indicate the state of the repository, clients can use this to check if there have been any changes to the revisions contained in this repository.

A repository implementation that supports external modifications should return a different increment.

getDelta

RepositoryDelta getDelta(long sinceIncrement)

Provides a mechanism to query the changes that have occurred to this repository since the specified increment.
public class RepositoryChangeEvent
extends EventObject

An event object that may be fired to RepositoryListener services to notify them of changes to a Repository

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>RepositoryChangeEvent(Repository repository, long increment)</td>
<td>46</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>long getIncrement()</td>
<td>46</td>
</tr>
<tr>
<td>Repository getRepository()</td>
<td>46</td>
</tr>
</tbody>
</table>

Constructor Detail

RepositoryChangeEvent

public RepositoryChangeEvent(Repository repository, long increment)

TODO

Method Detail

getRepository

public Repository getRepository()

TODO

getIncrement

public long getIncrement()

TODO
public interface RepositoryDelta

TODO

Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMPTY</td>
<td>47</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>getAddedResources()</td>
<td>47</td>
</tr>
<tr>
<td>getChangedResources()</td>
<td>47</td>
</tr>
<tr>
<td>getRemovedResources()</td>
<td>47</td>
</tr>
</tbody>
</table>

Field Detail

EMPTY

public static final RepositoryDelta EMPTY

Method Detail

getAddedResources

List<org.osgi.framework.wiring.Resource> getAddedResources()

TODO

getChangedResources

List<org.osgi.framework.wiring.Resource> getChangedResources()

TODO

getRemovedResources

List<org.osgi.framework.wiring.Resource> getRemovedResources()

TODO
public interface RepositoryListener

Service interface published to the OSGi registry to be notified when repositories change.

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>void repositoryChanged(RepositoryChangeEvent event)</td>
<td>48</td>
</tr>
</tbody>
</table>

Method Detail

repositoryChanged

void repositoryChanged(RepositoryChangeEvent event)

    TODO
Class ResolutionException

org.osgi.service.obr

java.lang.Object
  java.lang.Throwable
    java.lang.Exception
      java.lang.RuntimeException
        org.osgi.service.obr.ResolutionException

All Implemented Interfaces:
  Serializable

public class ResolutionException
extends RuntimeException

Indicates failure to resolve a set of requirements. Resolver implementations may subclass this class to provide extra state information about the reason for the resolution failure.

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResolutionException(String message)</td>
<td>49</td>
</tr>
<tr>
<td>ResolutionException(String message, Throwable cause)</td>
<td>49</td>
</tr>
<tr>
<td>ResolutionException(Throwable cause)</td>
<td>49</td>
</tr>
</tbody>
</table>

Constructor Detail

ResolutionException

public ResolutionException(String message,
                          Throwable cause)

  TODO document

ResolutionException

public ResolutionException(String message)

  TODO document

ResolutionException

public ResolutionException(Throwable cause)

  TODO document
public interface Resolver

A resolver is a service interface that can be used to find resolutions for specified requirements based on a supplied Environment.

Version:

$Id: 3ee568b95dd255adfa268410d68a59d70053d028 $

---

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Parameters</th>
</tr>
</thead>
</table>

Attempt to resolve the requirements based on the specified environment and return any new revisions or wires to the caller.

---

### Method Detail

**resolve**

```
```

Attempt to resolve the requirements based on the specified environment and return any new revisions or wires to the caller. For a given resolve call an environment should return a consistent set of capabilities and wires. The simplest mechanism of achieving this is by creating an immutable snapshot of the environment state and passing this to the resolve method.

**Parameters:**
- `environment` - the environment into which to resolve the requirements

**Returns:**
- a resolution

**Throws:**
- `ResolutionException`
- `IllegalArgumentException`
- `NullPointerException`
8 OBR Schema

```xml
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.osgi.org/xmlns/obr/v1.0.0/"
    xmlns:tns="http://www.osgi.org/xmlns/obr/v1.0.0/"
    elementFormDefault="qualified">
  <complexType name="Repository">
    <sequence>
      <choice maxOccurs="unbounded" minOccurs="0">
        <element ref="tns:revisionResource"></element>
        <element ref="tns:referral"></element>
      </choice>
    </sequence>
  </complexType>
</schema>
```
<complexType name="RevisionResource">
  <documentation>TODO Describes a general resource with properties, categories, requirements, extends, and capabilities.</documentation>
  <sequence>
    <element ref="tns:attribute" maxOccurs="unbounded" minOccurs="0">
      <element ref="tns:require" maxOccurs="unbounded" minOccurs="0">
      </element>
      <element ref="tns:capability" maxOccurs="unbounded" minOccurs="0">
      </element>
    </element>
    <attribute name="name" type="string" use="required">
      <documentation>The name of the resource. In case of a bundle, this is the Bundle Symbolic Name.</documentation>
    </attribute>
    <attribute name="namespace" type="string" use="required">
      <documentation>The namespace of the resource. In case of a bundle, this is osgi.bundle.</documentation>
    </attribute>
    <attribute name="version" type="tns:Version" use="required">
    </attribute>
  </sequence>
</complexType>
<complexType name="Referral">
    <annotation>
        <documentation>
            A referral points to another repository XML file. The purpose of this element is to create a federation of repositories that can be accessed as a single repository.
        </documentation>
    </annotation>
    <attribute name="depth" type="int" use="optional">
        <annotation>
            <documentation>
                The depth of referrals this repository acknowledges.
            </documentation>
        </annotation>
    </attribute>
    <attribute name="url" type="anyURI" use="required">
        <annotation>
            <documentation>
                The URL to the referred repository. The URL can be absolute or relative from the given repository’s URL.
            </documentation>
        </annotation>
    </attribute>
</complexType>

<element name="repository" type="tns:Repository"/>
<element name="revisionResource" type="tns:RevisionResource"/>
<element name="referral" type="tns:Referral"/>

<complexType name="Version">
    <annotation>
        <documentation>
            Version must follow the major, minor, micro, qualifier format as used the Framework’s version class. Example is “1.0.4.R128”
        </documentation>
    </annotation>
    <restriction base="string">
        <pattern value="d+(\d+(\d+)(\..+))??"></pattern>
    </restriction>
</complexType>
<element name="require" type="tns:Require"></element>

<element name="capability" type="tns:Capability"></element>

<complexType name="Capability">
  <annotation>
    <documentation>
      A named set of type properties. A capability can be used to resolve a requirement if the resource is included.
    </documentation>
  </annotation>
  <complexContent>
    <extension base="tns:RevisionResourceElement">
      <extension/>
    </complexContent>
  </complexType>

<complexType name="Require">
  <annotation>
    <documentation>
      TODO
    </documentation>
  </annotation>
  <complexContent>
    <extension base="tns:RevisionResourceElement"></extension>
  </complexContent>
</complexType>

<complexType name="RevisionResourceElement">
  <annotation>
    TODO
  </annotation>
  <sequence>
    <element ref="tns:attribute" maxOccurs="unbounded" minOccurs="0"></element>
    <element ref="tns:directive" maxOccurs="unbounded" minOccurs="0"></element>
  </sequence>
  <attribute name="namespace" type="string">
    <annotation>
      <documentation>
        Namespace of the capability. Only requirements with the same namespace must be able to match this capability.
      </documentation>
    </annotation>
    <attribute/>
  </complexType>

<element name="directive" type="tns:Directive"></element>

<element name="attribute" type="tns:Attribute"></element>
<complexType name="PropertyLike">
    <attribute name="name" type="string">
        <annotation>
            <documentation>The name of the property</documentation>
        </annotation>
    </attribute>
    <attribute name="value" type="string">
        <annotation>
            <documentation>The value of the property</documentation>
        </annotation>
    </attribute>
    <attribute name="type" type="tns:PropertyType" default="string">
        <annotation>
            <documentation>The type of the property.</documentation>
        </annotation>
    </attribute>
</complexType>

<complexType name="Attribute">
    <annotation>
        <documentation>
        </documentation>
    </annotation>
    <complexContent>
        <extension base="tns:PropertyLike"></extension>
    </complexContent>
</complexType>

<complexType name="Directive">
    <annotation>
        <documentation>
        </documentation>
    </annotation>
    <complexContent>
        <extension base="tns:PropertyLike"></extension>
    </complexContent>
</complexType>

<simpleType name="PropertyType">
    <restriction base="string">
        <enumeration value="boolean"></enumeration>
        <enumeration value="string"></enumeration>
        <enumeration value="version"></enumeration>
        <enumeration value="uri"></enumeration>
        <enumeration value="long"></enumeration>
        <enumeration value="double"></enumeration>
        <enumeration value="list"></enumeration>
    </restriction>
</simpleType>
9 Security Considerations

9.1

10 Considered Alternatives

10.1 Stateful resolver
The original OBR resolver used a stateful API design, however on discussion it has been agreed that a stateless API design is preferable as it is always possible to create a stateful wrapper around a stateless design but not to go the other way.

10.2 VersionRanges as explicit elements – separate from filters
Nimble and P2 encode the VersionRange as a separate element outside of the filter. This aids optimisation strategies but it is felt that early optimisation is a mistake for this API.

10.3 Garbage collection
Nimble supports the uninstall process which currently this specification is silent on as deployment has been removed as a goal for this specification. When a top level dependency is removed the transative set of dependencies that are no longer used is removed from the framework.

Discussion on conference calls suggested this is better handled in the Subsystem RFC work.

10.4 Separation of logical representation from physical representation
Both P2 and Nimble separate the concept of a logical unit of deployment from the physical artifact that is deployed. P2 uses the concept of an InstallableUnit with underlying artifacts. Nimble uses the concept of a Rule with underlying artifact.

Suggestion is to split resource into “part” with zero or more “resources”. A part is a logical unit of deployment that has requirements and capabilities. A resource is a physical artifact that can be accessed by URL and may have attributes such as size, checksum etc.

One use case for this separation is a service that requires some default configuration, an extreme example being a database and a database schema. In this scenario the service may explicitly require it’s configuration at deployment time before it can be meaningfully instantiated. This is related to the concept of resource builders mentioned in Error: Reference source not found.
Another use case is deployment of composite bundles – here the deployment of an “internal” bundle requires different processing than a “top level” bundle and deployment is complicated by treating sub parts of the graph as logically separate entities.

Decision taken in conference call to remove deployment from this specification so this is not a concern for the current time.

| DS: Revisit this if deployment does become part of this specification. |

10.4.1 Uses calculation

The default resolver strategy should take account of uses, however it should be an option available to the client at resolve time whether to calculate the uses constraints.

The uses problem space has been shown to be np complete [4], and so can explode into a massively time consuming task. However in a large number of situations we’ve encountered in the wild the calculations are practically unnecessary at runtime as theoretical uses constraints never occur and are an artifact of classes cross cutting package boundaries.

From a practical point of view it is useful to be able to get a quick solution to enable the user to carry on coding/testing vs wait for the end of time to prevent a class mismatch that will never occur in running code and is only an artifact of limitations in a module implementation.

10.5 Query protocol

Richard thinks the query protocol is not necessary A query protocol is not needed as resources can be simply discovered using ldap matches on attributes, more complex searches are out of scope for this specification.

| TODO expand on this if necessary or remove it completely |

10.6 Relationship to DeploymentAdmin/ResourceBuilder

Ability to apply custom deployment steps to resources on installation is key for a general purpose deployer. However there are several rabbit holes in this area that need to be explored and the decision taken was taken during conference calls to remove deployment from this specification and push this to other specifications such as Subsystems.

10.7 Querying a Web Service Based Repository

The repository can become quite large in certain cases. So large that small environments cannot handle the full repository anymore. For scalability reasons, it is therefore necessary to query the repository to only receive smaller chunks. Server based repositories are recommended to support the following query parameters after the URL:

- keywords – A space separated (before URL encoding) list of keywords. This command must return all resources that match a keyword in the description, category, copyright, etc, case insensitive.

- requirement – A structured field. The first part is the name of the requirement, followed by a legal filter expression.

- category – A category
All fields can be repeated multiple times. The server should return the subset of the resources that match all fields. That is, all fields are anded together. However, the receiver must be able to handle resources that were not selected, that is, no assumption can be made the selection worked. The purpose of the selection criteria is a potential optimization.

As a further optimization, it is allowed to specify the resources that are already received. This a comma separated lost of repository ids. The server should not send these resources again. The name of this parameter is knows.

For example

```
http://www.aqute.biz/bundles/repository.xml?requirement=package: (\n  package=org.osgi.util.measurement)&knows=1,2,3,4,9,102,89
```

This functionality has been moved out of this specification as it is possible to build this functionality on top of the existing design

11 Document Support

11.1 References


11.2 Author’s Address

<table>
<thead>
<tr>
<th>Name</th>
<th>Peter Kriens</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>aQute</td>
</tr>
<tr>
<td>Address</td>
<td>9c, Avenue St. Drezery, Beaulieu, France</td>
</tr>
<tr>
<td>Voice</td>
<td>+33 467542167</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:Peter.Kriens@aQute.biz">Peter.Kriens@aQute.biz</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Richard S. Hall</th>
</tr>
</thead>
</table>

Copyright © OSGi Alliance 2011 All Rights Reserved
## 11.3 Acronyms and Abbreviations

## 11.4 End of Document
Abstract

This specification describes how Java EE Connector Architecture resource adapters will be supported in an OSGi environment. The resource adapters requires a Java EE Connector Architecture container service. The specification describes a model for resource adapters to access and be accessed by relevant OSGi services.

NOTE: Several sections of the design are still under discussion, and the design described in the document must be considered “work in progress”.
1 Document Information

1.1 Table of Contents

1 Document Information ................................................................. 2
  1.1 Table of Contents .................................................................... 2
  1.2 Terminology and Document Conventions ................................ 3
  1.3 Revision History .................................................................... 3

2 Introduction .................................................................................. 5

3 Application Domain ................................................................. 6
  3.1.1 Structure of a Resource Adapter ........................................ 6
  3.1.2 Resource Adapter deployment descriptor ............................ 7

4 Problem Description .................................................................... 8

5 Requirements ............................................................................... 9

6 Technical Solution .................................................................... 11
  6.1 Architectural overview ......................................................... 11
  6.1.1 Resource Adapter Bundle (RAB) ....................................... 11
  6.1.2 Fragment Bundles .......................................................... 11
  6.2 Resource adapter life cycle ................................................... 12
    6.2.1 Installing a resource adapter bundle .............................. 12
    6.2.2 Starting a resource adapter bundle ............................... 12
    6.2.3 Configuring a resource adapter ..................................... 14
    6.2.4 Stopping a resource adapter bundle ............................. 14
    6.2.5 Uninstalling a resource adapter bundle ...................... 15
  6.3 OSGi Java EE Connector Architecture container .................... 15
    6.3.1 Required bundles ....................................................... 15
    6.3.2 Container requirements .............................................. 15
    6.3.3 Java SE considerations .............................................. 16
    6.3.4 Java EE considerations .............................................. 16
    6.3.5 Resource adapter class loader ..................................... 16
  6.4 Use of OSGi services ............................................................ 16
    6.4.1 EventAdmin service .................................................... 17
    6.4.2 Transaction Manager Services (RFC 98) ....................... 17
    6.4.3 JNDI Services (RFC 142) ............................................ 17

7 Security Considerations ......................................................... 18
  7.1 Java EE Connector Architecture container bundle .................. 18

8 Document Support .................................................................... 19
  8.1 References ........................................................................... 19
  8.2 Author’s Address ............................................................... 19
1.2 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULDN'T", "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.

1.3 Revision History

The last named individual in this history is currently responsible for this document.
<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial draft of RFC</td>
<td>March 14, 2009</td>
<td>Paul Parkinson, Oracle, <a href="mailto:paul.parkinson@oracle.com">paul.parkinson@oracle.com</a></td>
</tr>
<tr>
<td>2nd Draft</td>
<td>November 5, 2009</td>
<td>JJ Snyder, Oracle, <a href="mailto:jj.snyder@oracle.com">jj.snyder@oracle.com</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reformatted document to model RFC 66 (web container)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Rewrote most of the document.</td>
</tr>
<tr>
<td>0.9</td>
<td>January 13, 2010</td>
<td>Mike Keith, Oracle, <a href="mailto:michael.keith@oracle.com">michael.keith@oracle.com</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cleanup to prepare for submission to group</td>
</tr>
<tr>
<td>0.9.1</td>
<td>January 26, 2011</td>
<td>Jesper Pedersen, Red Hat, <a href="mailto:jesper.pedersen@redhat.com">jesper.pedersen@redhat.com</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Align text against the Java EE Connector Architecture 1.6 specification</td>
</tr>
<tr>
<td>0.9.2</td>
<td>January 31, 2011</td>
<td>Jesper Pedersen, Red Hat, <a href="mailto:jesper.pedersen@redhat.com">jesper.pedersen@redhat.com</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Revised 5.2.2 and 5.4 -&gt; 5.7.</td>
</tr>
<tr>
<td>0.9.3</td>
<td>February 11, 2011</td>
<td>Jesper Pedersen, Red Hat, <a href="mailto:jesper.pedersen@redhat.com">jesper.pedersen@redhat.com</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Revised 5.2, removed 5.6, 5.7.1, 5.7.2, moved 5.3 to 5.7</td>
</tr>
<tr>
<td>0.9.4</td>
<td>March 8, 2011</td>
<td>Jesper Pedersen, Red Hat, <a href="mailto:jesper.pedersen@redhat.com">jesper.pedersen@redhat.com</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added „Container requirements (5.3.2), removed „Accessing OSGi environment“ (5.3), comments from David Bosschaert (<a href="mailto:david@redhat.com">david@redhat.com</a>)</td>
</tr>
<tr>
<td>0.9.5</td>
<td>April 4, 2011</td>
<td>Jesper Pedersen, Red Hat, <a href="mailto:jesper.pedersen@redhat.com">jesper.pedersen@redhat.com</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added „Configuring a resource adapter“ to 5.2</td>
</tr>
</tbody>
</table>
2 Introduction

The OSGi for Java EE Connector Architecture container provides support for resource adapters written to the Java EE Connector Architecture Specification 1.6 [2].

The OSGi for Java EE Connector Architecture container specifies how a resource adapter packaged as a RAR can be installed into an OSGi based runtime, and how it can integrate with OSGi services. It also specifies how a resource adapter is managed by OSGi from a bundle lifecycle perspective.

Additionally, as OSGi evolves in enterprise space and new application models are emerging that leverage OSGi capabilities, it would be necessary for Java EE application components (in particular, resource adapters in the context of this RFC) to interoperate with other component models [3]. Java EE components will need to access and provide OSGi services, as well as interoperate with other components written to Declarative Services and Blueprint Services (RFC 124) [4].

While the specification provides support for OSGi programming model for resource adapters by describing how they can participate in a service base ecosystem, it is not a goal of this specification to provide support for creating resource adapters under a different component model such as Blueprint service.
3 Application Domain

Resource adapters are a critical part of applications built using Java Enterprise Edition (Java EE). The Java EE Connector Architecture model is a popular resource management and access development model that provides a well known API for applications to access various resource managers (such as databases, messaging systems, and legacy EIS and ERP systems) and SPI for resource adapters to exploit container services (such as transaction processing, security, work management, etc.). The Java EE Connector Architecture specification [2] describes how standard resource adapters function in an enterprise environment.

3.1.1 Structure of a Resource Adapter

A resource adapter is a collection of resources that is typically made up of a collection of some of the following:

1. Outbound Connection Factories and related classes
2. Inbound Activation, ActivationSpec and related classes
3. Utility classes
4. Resource files used by the Java classes
5. Descriptive meta information that ties the above elements together in a ra.xml file or provided through annotations

The following table describes the structure of a Java EE Connector Architecture resource adapter archive:

<table>
<thead>
<tr>
<th>Directory</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>/META-INF</td>
<td>Contains the MANIFEST.MF manifest file, as required for jar files. An application RAR can list its dependencies on external libraries needed in the manifest as described in JAR specification. During deployment of the resource adapter, the Connector container must make the correct versions of the extensions available to the application following the rules defined by the Optional Package Versioning mechanism (<a href="http://java.sun.com/j2se/1.4/docs/guide/extensions/">http://java.sun.com/j2se/1.4/docs/guide/extensions/</a>). This directory also contains the optional ra.xml file that ties the resource adapter classes together.</td>
</tr>
</tbody>
</table>

The Java interfaces, implementation, and utility classes required by the resource adapter must be packaged as one or more JAR files as part of the resource adapter archive. A JAR file must use the .jar file extension. These JAR files can reside anywhere inside the RAR file.

Any platform-specific libraries required by the resource adapter must be packaged within the resource adapter archive.
Resource adapters can be packaged and signed into a Resource Adapter Archive format (RAR) file using the standard Java archive tools. The Java EE Connector Architecture specification provides a detailed description of the format of a RAR archive.

3.1.2 Resource Adapter deployment descriptor

The resource adapter deployment descriptor includes the following types of configuration and deployment information:

- Initialization parameters for resource adapters, managed connection factories, admin objects, and configuration
- Inbound and outbound Connection and ConnectionFactory definitions, message listeners, and activation specifications
- Transaction support
- Security role mapping

This information is located in the ra.xml descriptor file or in annotations on the classes them self.
4 Problem Description

The OSGi compendium currently does not include support for the concept of resource adapters. A resource adapter is a system-level software driver that is used by a Java application to connect to an Enterprise Information System (EIS). The resource adapter plugs into a container or framework and provides connectivity between the EIS, the container/framework, and the enterprise application. The resource adapter serves as a protocol adapter that allows any arbitrary EIS communication protocol to be used for connectivity.

The Java EE Connector Architecture [2] defines standard contracts that allow bi-directional connectivity between enterprise applications and EISs. The Java EE Connector Architecture enables an EIS vendor to provide a standard resource adapter for its EIS.

A resource adapter can declare various application attributes declaratively in its ra.xml file or through annotations. It would be logical that resource adapters should be embraced and fully supported in the OSGi environment.

We need a specification which supports the current generation of the Java EE Connector Architecture specification and embraces the concept of deploying resource adapters to an OSGi environment.
5 Requirements

This specification addresses the following requirements:

1. There MUST be a standard programming approach that makes it possible to deploy a standard resource adapter to a Java EE Connector Architecture container programmatically. This approach may be a traditional Java programming interface registered as an object or objects in the OSGi Service Registry, it may be a programming pattern that an OSGi bundle must follow in order to ensure that it is registered as a resource adapter via reflection, or it may be something else, such as a pattern that the bundle must use to register itself on a “whiteboard.”

2. The standard MUST NOT require that a particular configuration API or system, including the OSGi Configuration Administration Service, be supported.

3. The solution MUST support deploying resource adapters as RAR files, as described in the Java Connector Specification.

4. The solution MUST not mandate that resource adapters be packaged or deployed as RAR files, as described in the Java EE Connector Architecture specification, as long as they conform to this specification.

5. The solution MUST provide the ability to access resource connection factories of arbitrary type such as database access (JDBC datasources), messaging systems (JMS connection factories), and other EIS and ERP systems (resource adapter connection factories).

6. The solution MUST allow resource adapters to specify which packages to import from the OSGi Framework.

7. The solution MUST allow resource adapters to export packages and services to the OSGi Framework, as well as allow OSGi-aware resource adapters to access other OSGi services.

8. The solution SHOULD support resource adapter components to interoperate with components written to other models such as OSGi declarative services and Blueprint services. For example, the solution could support injection of component services into Java EE components based on additional meta-data associated with the application components.

9. It SHOULD be possible for a resource adapter bundle to remain installed when its Java EE Connector Architecture container is dynamically replaced.

10. An OSGi-compliant Java EE Connector Architecture container MUST NOT be impeded from also being compliant with the Java EE Connector Architecture specification.
11. An OSGi-compliant Java EE Connector Architecture container MUST support resource adapters implemented to the Java EE Connector Architecture 1.0, 1.5 and 1.6 specifications or any later versions which are backwards compatible with these specifications.

12. The OSGi Java EE Connector Architecture container design MUST NOT require an OSGi Execution Environment greater than that which satisfies the signatures of the Java EE Connector Architecture specification.

13. An OSGi Java EE Connector Architecture container MAY provide additional aspects of the technology that are required for resource adapter support to be properly integrated in an OSGi framework but MUST NOT make any syntactic changes to the Java interfaces defined by the Java EE Connector Architecture specification.
6 Technical Solution

6.1 Architectural overview

Bundles are the deployment and management entities under OSGi. The RFC takes a design approach where a resource adapter is deployed as an OSGi bundle in the framework. There is exactly one resource adapter bundle that corresponds to each deployed resource adapter in the framework.

The specification describes the design requirements for an OSGi Java EE Connector Architecture container that supports resource adapter components written to Java EE Connector Architecture specifications. The Java EE Connector Architecture container itself is deployed as one or more OSGi bundles.

The design uses OSGi extender pattern [6], where the Java EE Connector Architecture container includes an extender that is responsible for observing the life cycle of resource adapter bundles. When a resource adapter bundle is started, the extender processes the configuration of the resource adapter and instantiates and manages lifecycle of the resource adapter. The resource adapter bundles thus become managed bundles of the extender.

6.1.1 Resource Adapter Bundle (RAB)

A Resource Adapter Bundle (RAB) is defined as a normal OSGi bundle that contains the classes, interfaces, jars, etc. necessary for accessing a resource adapter.

A RAB is defined as the follows:

- A RAB is a valid OSGi bundle and as such must fully describe its dependencies.
- A RAB follows the OSGi bundle life-cycle.
- A RAB is differentiated from a normal bundle through the existence of metadata that describes the resource adapter. This metadata is located in the META-INF/ra.xml file or in annotations.

A mandatory manifest header (META-INF/MANIFEST.MF) is required to identify a resource adapter bundle

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource-Adapter</td>
<td>&lt;empty&gt;</td>
<td>Resource adapter bundle identifier</td>
</tr>
</tbody>
</table>

The RAB follows the packaging rules of a standard Java EE Connector Architecture archive (RAR).

# TODO – File extension requirements

6.1.2 Fragment Bundles

Fragments are bundles that can be attached to one or more host bundles by the framework, which can modify the behavior of a host bundle. A fragment can add classes to the bundle class space, the bundle entry space, and
update imports and exports.  Fragments are attached by the time a bundle is resolved.  This specification limits what effects a fragment bundle can have on a RAB in the following way:

- A fragment cannot provide or replace the resource adapter metadata for a bundle.
- A fragment can provide additional vendor specific deployment descriptors.
- A fragment can contribute classes and resources to the bundle class space.
- A fragment can contribute platform-dependent native libraries to the bundle.

### 6.2 Resource adapter life cycle

# TODO DS/Blueprint

#### 6.2.1 Installing a resource adapter bundle

A RAB is a valid OSGi bundle containing the resource adapter metadata. A RAB can be installed into the framework using the standard `BundleContext.installBundle` API variants.

A RAB can be signed, but must comply with the bundle signing rules defined in the OSGi Core Specification v4.2.

Once installed, a RAB bundle’s life cycle is managed just like any other bundle in the framework.

#### 6.2.2 Starting a resource adapter bundle

A resource adapter is started by starting its corresponding resource adapter bundle. The Java EE Connector Architecture extender listens for the bundle starting and lazy activation life cycle events to initiate this process.

A resource adapter and the Java EE Connector Architecture extender may start in any order. The extender recognizes a resource adapter bundle by looking for the presence of a resource adapter bundle specific manifest header. The extender does not recognize a bundle as a resource adapter bundle unless the manifest header is present in the bundle.

After recognizing a resource adapter bundle, the extender initiates the process of deploying the resource adapter into the Java EE Connector Architecture container. It must generate a `DEPLOYING` event as described in section 5.3.

The Java EE Connector Architecture container processes deployment information by processing the resource adapter metadata and any vendor specific deployment metadata.

As resource adapters are modularized further into multiple bundles (and not deployed as RAR files only) it is possible that a resource adapter bundle can have import dependencies on other deployed bundles. The container must fully support metadata that specifies connection factories, administered objects, etc. whose classes are obtained via an `Import-Package` statement.

Any validation failures must prevent the resource adapter module from being accessed and must result in a `FAILED` event being emitted.

The Java EE Connector Architecture container performs the necessary initialization of resource adapter components in the bundles, as described in specification [1]. This involves the following:
• Obtain TransactionManager Service described in RFC98

• Create a BootstrapContext (which provides Timer, XATerminer, and WorkManager) for the resource adapter.

• Instantiate a configured javax.resource.spi.ResourceAdapter.

• Register org.osgi.jca.ResourceAdapterWrapper in the service registry. This is a wrapper of the ResourceAdapter without the lifecycle methods.
  - # TODO – What service properties should we register, if any?

• Call void start(BootstrapContext bootstrapContext) on the ResourceAdapter

The JCA container listens to bundles being installed that need to be processed. The following steps are executed for each applicable bundle:

• Instantiate configured javax.resource.spi.ManagedConnectionFactory(s).

• Call Object createConnectionFactory(ConnectionManager connectionManager) on the instantiated ManagedConnectionFactory(s).

• Register ConnectionFactory(s) in the service registry.
  - # TODO – The service is registered under the bundle context of the Resource Adapter bundle.

• Instantiate configured javax.resource.spi.AdministeredObject(s).

• Register AdministeredObject(s) in the service registry.

All instances of the ConnectionFactory(s) and AdministeredObject(s) must be registered in the service registry with an unique identifier in the supplied dictionary instance. The mandatory key is osgi.jca.service.name.

An example:

MyConnectionFactory cf = ...;

Dictionary d = new Hashtable();
d.put("osgi.jca.service.name", "MyCFInstance");
ctx.registerService(MyConnectionFactory.class, cf, d);

Note, that service registration of AdministeredObject(s) must register all specified administered object interfaces under the same key/value pair for each instance.

The Java EE Connector Architecture container is required to complete instantiation of connection factories and administration objects and register them in the service registry before the resource adapter is considered deployed.
After successful deploying the resource adapter, a **DEPLOYED** event must be generated to indicate that the resource adapter is now in service.

### 6.2.3 Configuring a resource adapter

# TODO – Describe lookup of configured resource adapter objects

An instance of a resource adapter bundle can be configured through the OSGi Configuration Admin Service in case that there isn't enough vendor specific deployment information available.

# TODO –

- We need to specify Persistent ID's for Connection Factory(ies) (really ManagedConnectionFactory) and AdminObject(s) in the bundle
- There needs to be a way for the configuration of the Connection Factory(ies) and AdminObject(s) to trigger a deployment of these objects based on the metadata stored in the container. This is currently vendor specific, but one possibility is to leave this area as an implementation detail
- We need to consider that multiple configurations of a Persistent ID can happen – in relation to the ManagedConnectionFactory and AdminObject Persistent ID "templates"
- We need to consider how the configuration should be done – API, metadata file, ... It is along the same lines what vendor do with their vendor specific deployment files
- The dynamic part of the configuration admin service needs to be handled, as f.ex. rebinding in JNDI or updating a config-property isn't always possible. Simplest solution would be to only allow the initial configuration
- There should be a standard template for ManagedServiceFactory in order to configure the standard elements of the ra.xml. This is vendor specific today

A lot of the issues around this area is that this is handled by a vendor specific implementation of the necessary services.

If multiple instances of the same resource adapter bundle is needed the ManagedServiceFactory can be used.

# TODO – RAB vs. ConfigService

### 6.2.4 Stopping a resource adapter bundle

A resource adapter is stopped by simply stopping the corresponding resource adapter bundle. In response to a bundle **STOPPING** event, the extender must initiate the process of undeploying the resource adapter from the Java EE Connector Architecture container. This will involve the following:

- An **UNDEPLOYING** event is emitted to signal that the resource adapter will be removed.
- The Java EE Connector Architecture container calls **stop** on the **ResourceAdapter**.
- Finally, an **UNDEPLOYED** event is emitted.

Once the bundle is stopped the OSGi framework will automatically unregister any services registered by the resource adapter.
6.2.5 Uninstalling a resource adapter bundle

A resource adapter can be uninstalled by uninstalling the corresponding resource adapter bundle. The resource adapter bundle will be uninstalled from the OSGi framework, and will be completely removed when the framework is refreshed.

6.3 OSGi Java EE Connector Architecture container

The specification defines an OSGi Java EE Connector Architecture container implementation as one or more OSGi bundles that collectively implements the Java EE Connector Architecture specification.

The following sections describes the requirements for an OSGi Java EE Connector Architecture container.

6.3.1 Required bundles

The container implementation must supply an OSGi bundle which exports the Java EE Connector Architecture 1.6 specification application programming interface (API).

The export packages for the bundle must be:

- javax.resource;version=1.6
- javax.resource.cci;version=1.6
- javax.resource.spi;version=1.6
- javax.resource.spi.endpoint;version=1.6
- javax.resource.spi.security;version=1.6
- javax.resource.spi.work;version=1.6

# TODO Align against other OSGi/EE specs – like OSGi/Web

6.3.2 Container requirements

An OSGi Java EE Connector Architecture container MUST use implementations of the following components that handles integration with the OSGi services

- Bootstrap context (javax.resource.spi.BootstrapContext)
- Connection manager (javax.resource.spi.ConnectionManager)

The container MUST also supply an interface for javax.resource.spi.ResourceAdapter which is used when resource adapter instance is registered in the service registry.

# TODO – Name + methods
6.3.3 Java SE considerations

The Java EE Connector Architecture 1.6 specification requires Java Platform Standard Edition 6 as the minimum execution environment. Consequently, it is the minimum execution environment for running the OSGi Java EE Connector Architecture container.

6.3.4 Java EE considerations

An OSGi Java EE Connector Architecture container implementation will need to consider additional requirements in order to be Java EE compliant.

Java EE Connector Architecture 1.6 together with the Java EE 6 specification describes a comprehensive set of requirements that a Java EE compliant Java EE Connector Architecture container must implement.

In practice, a Java EE Connector Architecture container implementation supports a subset of Java EE services and makes them available to resource adapters, even when it is not fully compliant with Java EE specification.

These include support for

- JAAS
- Environment naming context (e.g. java:comp/env access)
- Transactions

# TODO – JAAS

This specification highly recommends (but does not require) that an OSGi Java EE Connector Architecture container provide integration with:

- A Transaction manager (RFC 98)
- JNDI (RFC 142). Containers that need to support JNDI environment configuration from properties files must set the JNDI client’s bundle’s class loader to be the current Thread Context class loader prior to invoking a method on the application component. The context class loader must implement "org.osgi.framework.BundleReference" interface as described in the OSGi R4.2 core specification.

6.3.5 Resource adapter class loader

The implementation should not allow the application to override Java SE or Java EE platform classes, such as those in the java.* and javax.* namespaces, that either Java SE or Java EE do not allow to be modified.

6.4 Use of OSGi services

The Java EE Connector Architecture container may use standard OSGi services to implement some of the features of the Java EE Connector Architecture 1.6 specification.

The following OSGi services are RECOMMENDED for implementing parts of the Java EE Connector Architecture 1.6 specification.

- EventAdmin Service – Used to publish events.
• Transaction Manager Services (RFC 98) – Used as the JTA transaction manager for resource adapters that participate in XA transactions.

• JNDI Services (RFC 142) – Used to register connection factories and administered objects (optional).

6.4.1 EventAdmin service

If the EventAdmin service is registered then the Java EE Connector Architecture extender bundle must emit the following events:

• /org/osgi/service/connector/DEPLOYING – the extender has spotted a resource adapter bundle and started the process of deploying the resource adapter.

• /org/osgi/service/connector/DEPLOYED – the extender has finished deploying the resource adapter and the resource adapter is now running.

• /org/osgi/service/connector/UNDEPLOYING – the resource adapter is being undeployed.

• /org/osgi/service/connector/UNDEPLOYED – the extender has removed the resource adapter. The resource adapter is no longer in service.

• /org/osgi/service/connector/FAILED – the extender has failed to deploy the resource adapter. This will be fired after the DEPLOYING event has fired.

For each event the following properties must be published:

• "bundle.symbolicName" (String) the symbolic name of the resource adapter bundle.
• "bundle.id" (Long) the id of the resource adapter bundle.
• "bundle" (Bundle) the Bundle object of the resource adapter bundle.
• "bundle.version" (Version) the version of the resource adapter bundle.
• "timestamp" (Long) the time when the event occurred.
• "extender.bundle" (Bundle) the Bundle object of the connector extender bundle.
• "extender.bundle.id" (Long) the id of the connector extender bundle.
• "extender.bundle.symbolicName" (String) the symbolic name of the connector extender bundle.
• "extender.bundle.version" (Version) the version of the connector extender bundle.

In addition the FAILED event must also have the following property:

• "exception" (Throwable) an exception detailing the problem.

6.4.2 Transaction Manager Services (RFC 98)

Resource adapters may support JTA [7] transactions. The Java EE Connector Architecture container must support resource adapters that support JTA transactions. The Transaction services (RFC 98) provide the services that implement the JTA specification.

6.4.3 JNDI Services (RFC 142)

The Java EE Connector Architecture container can to register the connection factories and administration objects in the JNDI namespace. The JNDI services (RFC 142) provide the services that implement the JNDI specification.
7 Security Considerations

7.1 Java EE Connector Architecture container bundle

The Java EE Connector Architecture container should only be implemented by a trusted bundle. This bundle requires the following security permissions.

- ServicePermission[get] for the org.osgi.service.log.LogService interface. This allows the Java EE Connector Architecture container to log messages.
- ServicePermission[get] for the javax.xml.parsers.SAXParserFactory or javax.xml.parsers.DocumentBuilderFactory interface. The SAXParserFactory and DocumentBuilderFactory services can be used to process resource adapter descriptors.
- AdminPermission[resource,class] to call Bundle.getEntry, Bundle.getEntryPaths and Bundle.loadClass on the resource adapter bundles.
8 Document Support

8.1 References

4. RFC 124 (Blueprint services)
7. JTA Transaction specification

8.2 Author's Address

<table>
<thead>
<tr>
<th>Name</th>
<th>Jesper Pedersen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Red Hat</td>
</tr>
<tr>
<td>Address</td>
<td>314 Littleton Road, Westford, MA, 01886, USA</td>
</tr>
<tr>
<td>Voice</td>
<td>+1-978-392-1000</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:jesper.pedersen@redhat.com">jesper.pedersen@redhat.com</a></td>
</tr>
</tbody>
</table>
8.3 Acronyms and Abbreviations

RAR – Resource Adapter Archive

RAB – Resource Adapter Bundle

8.4 End of Document
Abstract

This RFC proposes a design for OSGi Subsystems, where a subsystem is a collection of bundles with sharing and isolation semantics. The requirements for this RFC were defined and agreed in RFP 121.
0 Document Information

0.1 Table of Contents

0 Document Information................................................................................................................2
  0.1 Table of Contents................................................................................................................2
  0.2 Terminology and Document Conventions...........................................................................4
  0.3 Revision History..................................................................................................................4

1 Introduction................................................................................................................................5

2 Application Domain....................................................................................................................6
  2.1 Related OSGi Specifications..............................................................................................9
    2.1.1 OBR (RFC 112)........................................................................................................9
    2.1.2 Frameworks Hooks (RFC 138).................................................................................9
    2.1.3 Initial Provisioning.....................................................................................................9
    2.1.4 Deployment Admin..................................................................................................9
    2.1.5 Application Admin...................................................................................................9
  2.2 Terminology + Abbreviations............................................................................................10

3 Problem Description................................................................................................................10
  3.1 Problem Scope....................................................................................................................11

4 Requirements...........................................................................................................................11
  4.1 Subsystem Modeling.........................................................................................................11
  4.2 Dependency Management...............................................................................................12
  4.3 Administration................................................................................................................12
  4.4 Runtime..........................................................................................................................12
  4.5 Development Support.....................................................................................................13
  4.6 RFC 138 Compatibility.....................................................................................................13

5 Technical Solution....................................................................................................................13
  5.1 Subsystems Architecture.................................................................................................13
  5.2 Subsystem Types.............................................................................................................15
    5.2.1 Application Subsystems..........................................................................................15
    5.2.2 Composite Bundle Subsystems..............................................................................16
    5.2.3 Feature Subsystems...............................................................................................17
    5.2.4 Life-cycle................................................................................................................18
    5.2.5 Listening for Internal Subsystem Events...............................................................21
    5.2.6 Class Diagrams.......................................................................................................22
    5.2.7 Installing a Subsystem............................................................................................22
    5.2.8 Resolving a Subsystem...........................................................................................23
0.2 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.

0.3 Revision History

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

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>11 19 2009</td>
<td>Created from RFP 121&lt;br&gt;Graham Charters, IBM <a href="mailto:charters@uk.ibm.com">charters@uk.ibm.com</a></td>
</tr>
<tr>
<td>0.1</td>
<td>08 01 2010</td>
<td>Added Subsystem Definition Headers&lt;br&gt;Graham Charters, IBM <a href="mailto:charters@uk.ibm.com">charters@uk.ibm.com</a></td>
</tr>
<tr>
<td>0.2</td>
<td>23 07 2010</td>
<td>Added JavaDoc. Added architecture overview/entities. Added subsystem types overview. Started adding deployment manifest details. Added archive definitions. Added lots of TODOs!&lt;br&gt;Graham Charters, IBM <a href="mailto:charters@uk.ibm.com">charters@uk.ibm.com</a></td>
</tr>
<tr>
<td>0.3</td>
<td>01 09 2010</td>
<td>Consideration of XML or Java Properties files in Alternatives.&lt;br&gt;Artefacts now in OSGI-INF instead of META-INF.&lt;br&gt;ResourceProcessor uses new Coordination Service.&lt;br&gt;SubsystemAdmin uses Futures for long-running activities.&lt;br&gt;Use of EventAdmin instead of SubsystemListeners.&lt;br&gt;Some details on Subsystem life-cycle (incomplete)&lt;br&gt;Graham Charters, IBM <a href="mailto:charters@uk.ibm.com">charters@uk.ibm.com</a></td>
</tr>
</tbody>
</table>
1 Introduction

The OSGi platform provides a very appealing deployment platform for a variety of applications. In many scenarios, and especially enterprises, an application can consist of a large number of bundles. Administrators typically think in terms of applications for tasks such as deployment, configuration and update, rather than individual bundles, and therefore there is a mismatch in concepts between those provided by the OSGi framework and those familiar to an administrator.

The problem domain was introduced and requirements agreed in RFP 121. This RFC proposes a solution to address those requirements. Input to the design in this RFC will be drawn from a number of existing sources:

- Spring dm Server - applications
- Apache Felix Karaf – features
2 Application Domain

When System Administrators, Deployers and other people think about software they typically think in terms of systems performing a certain business function. A System Administrator's job might be to make sure that these system stay up and running. The sysadmin doesn't have intimate knowledge of how these systems are constructed, but (s)he does have knowledge of the applications that compose the system. There might be an ordering application, a payment application and a stock checking application. The sysadmin knows that these applications exist and that they must be monitored to ensure that they are in a healthy state.

Other people may think of a system yet on a different level. The CIO of the company may have a higher level view. He sees the system described above as a single application unit: the company's web shop and views all of the applications that together run in his organization as 'The System'.

On the other end of the spectrum, a developer might be tasked by implementing the payment system. This system might in turn be built up of other applications. There might be a currency conversion application, a credit card charge application and an electronic bank transactions application.

Bottom line is that what one person sees as an application, might be a mere component to the next person in the chain. Application is a nestable concept and as such it is referred to by the more general word 'Subsystem'. What the people described above have in common is that a pure OSGi bundle is typically not what they view as an application. An OSGi bundle is a component used to build applications, but applications themselves, no matter from what level you look at it are always bigger than a bundle.

So from a conceptual point of view, applications are nestable. However, this does not mean that the runtime realization of these nested applications needs to exactly follow the nested structure. In OSGi systems, applications are built up from OSGi bundles, and although it would be desirable to view an OSGi system in a way that makes sense to the person looking at it, this does not mean that the bundles themselves need to be nested as well. The application view could be a virtual, organizational layer over a potentially flat bag of bundles that realize the applications at runtime. In some cases layering might be needed to enforce separation but in other systems the layering might not be used.

An application level view over an OSGi system might look like Figure 1.
At runtime, the bundles composing this application might be deployed as a flat structure where the 'Logging Bundle' and 'Financial Utils' bundle are shared. This way of deploying the application is very efficient with regard to memory footprint. See Figure 2.
Framework resolver hooks defined by RFC 138 could be used to isolate certain sub-systems while still sharing other bundles, like the logging bundle. This provides another runtime deployment option for the same system, providing more isolation. See Figure 3.

Figure 2: Runtime realization with sharing

Figure 3: Runtime realization with less sharing and more isolation
Besides the applications installed in an OSGi framework, there are also bundles installed that are considered part of the OSGi framework infrastructure, but at runtime these bundles could become part of the application. Consider a bundle providing the logging service or a bundle providing the HTTP Service. OSGi Services are a great architecture for loose coupling, fostering sharing and re-use.

## 2.1 Related OSGi Specifications

### 2.1.1 OBR (RFC 112)

RFC 112 describes the OSGi Bundle Repository. This repository could provide an excellent place to store all the bundles required to deploy an application. It would therefore be important that any new designs resulting from this requirements document fully integrate and leverage the OBR.

However, using an OBR may not be right for every use-case so it should be optional. In some contexts development might start out relatively loose, using artifact repositories such as an OBR. Over time during the development process the dependencies in the project might harden and ultimately an OBR may not be appropriate any more. In a completely hardened situation dependencies might be obtained from a local directory created by an installer process or possibly a fixed corporate web site.

### 2.1.2 Frameworks Hooks (RFC 138)

Frameworks Hooks as proposed in RFC 138 provide a way of isolating or grouping subsystems within a single OSGi Framework. Hook configuration could be used to model an application at runtime, but there may also be implementations that do not require isolation so use of Frameworks Hooks should be optional.

### 2.1.3 Initial Provisioning

The Initial Provisioning spec does not relate to an application concept at all. It rather concerns about how to deploy initial bundles on a device. This specification is unrelated to this RFP, but mentioned here to confirm that there is no potential overlap.

### 2.1.4 Deployment Admin

The Deployment Admin specification is about defining a file format to ship and deploy applications. Such an application is built up of multiple bundles and additional resources. Patching of applications, is also supported by this specification.

An additional feature provided by Deployment Admin, and which is considered useful for subsystems is the capability to roll back a deployment.

A major drawback of the Deployment Admin specification is that it explicitly prohibits sharing of bundles, which, besides preventing memory efficiency, could cause real problems when two applications use the same third-party bundle, as installing the same bundle twice is not allowed. Even different versions of the same bundle can not be installed simultaneously with Deployment Admin.

### 2.1.5 Application Admin

Application Admin provides the concept of an Application in OSGi. This specification could potentially be the basis of a runtime API into the Application Metadata and life-cycle system. It would most likely have to be extended to support all the use cases. Missing in the Application Admin spec is the Application Metadata, which should be declarative and available both inside the running OSGi framework as well as outside it to support OSGi tooling for subsystems.

Concepts missing from Application Admin:
• No file format to describe an application, purely API based
• No impact analysis for doing upgrades
• No interaction with OBR
• No application fingerprinting
• No runtime application extent analysis (what bundles beyond the core app set are used by this app)

2.2 Terminology + Abbreviations

• **Root Scope** – the scope at which the Subsystem Implementation bundle is installed. This is the level at where the subsystem implementation will look for Resource Processor services, provide events about subsystems and install transitive dependencies (e.g. bundles) in support of a subsystems managed by the subsystem implementation.

• **Unscoped Subsystem** – a subsystem that does not provide any package and/or service isolation. It effectively defines a set of bundles and or configuration that are provisioned into the OSGi framework in a flat bundle space.

• **Scoped Subsystem** – a subsystem that provides package and/or service isolation. The contents of the scoped subsystem will isolated by RFC 138 hook configuration.

• **Implicitly Scoped Subsystem** – a subsystem that provides implicit package, service and bundle isolation. Sharing is not explicitly declared. Packages, services and bundles are not shared outside the subsystem. Packages, services and bundle dependencies that are not provided inside the subsystem are share into the subsystem from outside.

• **Explicitly Scoped Subsystem** – a subsystems that provides explicit package, service and bundle sharing. Packages, service and bundles are not shared into, or out of, a subsystem unless explicit done so through hook configuration (see RFC 138).

3 Problem Description

In today's OSGi framework, what is deployed is typically just a large set of bundles. To a person not familiar with the details of the design of these bundles it is often unclear what function these bundles perform and how they are related.

Some bundles might provide shared infrastructure (e.g. a bundle providing the OSGi Log service), while other bundles might together provide an application function (e.g. a set of bundles together implementing a web-based shopping application). Today it is not possible to find out from a high level what applications are installed in the OSGi framework. The only information available is the list of bundles.
The OSGi framework needs to be enhanced with a mechanism that makes it possible to declare applications. An application has a name meaningful to the deployer. It is typically composed of a number of key bundles and services, plus their dependencies. When the deployer requests the list of installed applications, he will get a manageable result that is typically much shorter than the list of installed bundles.

The deployer also needs to be able to perform actions on the application level. Installing, uninstalling, starting, and stopping should be possible on the level of an application.

A developer may wish to think in terms of an application consisting of a set of bundles and may wish to declare those bundles as belonging to the application.

### 3.1 Problem Scope

Graphical tools are out of scope for this RFC.

In scope would be the metadata needed to define the applications plus an API that would enable actions at the application level.

### 4 Requirements

The following requirements are taken from RFP 121, “Subsystem metadata and Lifecycle”. The requirement numbers used here are identical to those in RFP 121. Some requirements were omitted from the final RFP and hence the numbering is intentionally not contiguous.

#### 4.1 Subsystem Modeling

REQ 1. The solution MUST provide a means to describe a subsystem which can be accessed both inside a running OSGi framework (e.g. through an API) as well as outside of a running framework (e.g. in a file).

REQ 2. The solution MUST define a subsystem definition format.

REQ 3. It MUST be possible to define a subsystem in terms of OSGi bundles.

REQ 4. It MUST be possible to make artifacts other than OSGi bundles part of a subsystem definition.

REQ 5. It MUST be possible to include a subsystem definition in another subsystem definition.

REQ 6. It MUST be possible to reference another subsystem in a subsystem definition, which makes the other subsystems a dependency.

REQ 7. It MUST be simple to define a scoped subsystem of the type defined in requirement 8 but with a default sharing policy that hides everything in the given subsystem from the parent of the given
subsystem and makes all bundles (for wiring purposes only), packages and services visible inside the given subsystem from its parent.

REQ 8. It MUST be possible to scope subsystems. Scoped subsystems form a hierarchy. The subsystems, bundles, packages, and services belonging to a given scoped subsystem are visible within the scope of the given subsystem. A subsystem MUST have control over a sharing policy to selectively make packages and services visible to the parent of that subsystem. A subsystem MUST have control over a sharing policy to selectively make bundles (for wiring purposes only), packages and services visible inside that subsystem from its parent. Subsystems, bundles, packages, and services belonging to unscoped subsystems are always available to all peer subsystems (scoped or unscoped) and bundles.

REQ 9. Subsystems MUST be uniquely identifiable and versioned.

REQ 10. Subsystem definitions MUST be extensible. This will allow tools to store associated data along side the subsystem definitions.

### 4.2 Dependency Management

REQ 13. It MUST be possible to use constraints to declaratively identify bundles and other artifacts in a subsystem definition.

REQ 15. It MUST be possible to define a subsystem as a number of key bundles. The transitive dependencies of the subsystem are inferred from the meta-data of the key bundles.

### 4.3 Administration

REQ 18. The solution MUST provide an API to query the subsystems available in the OSGi container, their associated state (as a snapshot) and dependencies.

REQ 19. The fidelity of the subsystem states MUST be sufficient to capture the possible states of its constituents

REQ 20. The solution MUST provide an API to drive the subsystem life-cycle.

REQ 22. The solution SHOULD provide an API to do impact analysis regarding replacing one or more bundles or subsystems. This impact analysis should list the subsystems affected and describe how these subsystems are using the affected bundle: either by importing an interface or a class from it, purely through services or through the extender pattern. It should also describe existing transitive dependencies and new transitive dependencies.

REQ 24. The APIs that modify the system SHOULD not leave the system in an inconsistent state.

### 4.4 Runtime

REQ 26. The solution MUST allow a single bundle to be part of multiple subsystems.

REQ 27. The solution MUST allow multiple versions of the same bundle.

REQ 28. Subsystems MUST have a well-defined life-cycle.
4.5 Development Support

REQ 29. The solution MUST support the development process by allowing bundles to be updated with other bundles that have the same version as the previous, similar to how Maven supports SNAPSHOT versions of artifacts.

4.6 RFC 138 Compatibility

REQ 30. The solution MUST be compatible with the scoping mechanisms possible through RFC 138.

5 Technical Solution

Subsystems enable a number of resources to be managed as a single entity, such as a set of bundles with associated configuration. Subsystems also allow selective sharing of capabilities and requirements (borrowing the terms from rfc 154), such as packages and services. For example, a subsystem might be defined in terms of a set of bundles that share a number of packages and services between themselves, but only share a subset of those capabilities outside the subsystem. This selective sharing enables the creation of assets that are more coarse-grained than individual bundles, and exhibit their own modularity.

Subsystems are designed to be easy to create for those familiar with bundle development. They therefore have many concepts and design choices in common with bundles. For example, a manifest format is used to define a subsystem, as well as describe its deployment.

The subsystem service design consists of the following elements:

1. Subsystem metadata for defining subsystems.
2. An API for the management of subsystems; install, uninstall, update, event notification.
3. An SPI for handling types of resources. This SPI is inspired by the Deployment Admin resource processor design.

5.1 Subsystems Architecture

The subsystems architecture is shown in figure 4.
The main entities in the architecture are as follows:

• **Subsystem Definition** – A Subsystem Definition is a description of a subsystem that is processed by the Subsystem Admin service. For example, it describes the contents of the subsystem and, depending on the type of subsystem, it also describes any packages and services that are shared in or out.

• **Deployment Definition** – A Deployment Definition describes the exact resources to provision for subsystem (i.e. the resolved content and any resources require to satisfy the subsystems dependencies), and any sharing policies (i.e. the packages and services that can be shared in or out of the subsystem). The deployment definition is optional.

• **Resource** – A Resource is any artifact, such as a bundle or configuration, that may be required when provisioning a subsystem.

• **Subsystem Archive** – An archive used to package a subsystem for installation into an OSGi runtime. The archive optionally contains a subsystem definition, a deployment definition, and resources.

• **Subsystem** – An implementation of this specification that provides the ability to install, uninstall, update and locate subsystems.
• Resource Processor – A Resource Processor provides the capability to process a particular type of resource, such as configuration. A number of resource processors may be required in order to process a Subsystem Definition.

• Repository – Subsystems are not required to be, and typically won’t be, transitively closed and therefore some amount of resolution will be required during installation to ensure all requirements are satisfied. Zero or more Repositories may be required in order to fully provision a Subsystem, including its transitive dependencies. The repository services used are those define in the OBR specification, RFC 112.

• Resolver – A Resolver is used by the Subsystem implementation to determine whether the subsystem resolves (i.e. is transitively closed) or requires additional resources in order to satisfy all its requirements. The Resolver is defined by the OBR specification, RFC 112.

5.2 Subsystem Types

Three types of subsystems are defined in this specification; Application, Composite Bundle and Feature. Each differs in the way in which they share requirements and capabilities and are described in more detail in the next few sections.

5.2.1 Application Subsystems

An application subsystem is a subsystem with a sharing policy associated with what people would often consider to be an application. An application subsystem consists of a number of bundles and other supporting resources. The content bundles share package and service dependencies with each other. An application does not share any package or service capabilities to bundles outside the application. Any package or service requirements that are not satisfied by the content bundles themselves are automatically imported from outside the implication.

![Diagram of Application Subsystem]

*Figure 5: Application package and service sharing example.*
An example of this point is shown in figure 5. An application consisting of 3 content bundles, A, B and C, is shown. Bundle A is exporting an API package to bundles B and C. Bundle C is providing a service which is being used by Bundle B. Bundle C also requires a package and service, neither of which are provided by bundles inside the application. These are therefore automatically shared into the application and in this example provided by bundles D and E, respectively.

### 5.2.2 Composite Bundle Subsystems

A composite bundle subsystem (hereafter referred to as, “composite”) is a subsystem with a fully explicit sharing policy that represents a composite bundle. A composite consists of a set of content bundles and other supporting resources. These content bundles share package and service dependencies inside the composite. By default a composite does not share packages or services into or out of itself. Any packages or services that are to be shared into or out of the composite, must be explicitly identified in the composite subsystem definition.

![Composite package and service sharing example.](image)

Figure 6: Composite package and service sharing example. 

Figure 6 shows an example composite subsystem that is sharing packages and service both in and out of the composite for use by bundles inside and outside the composite, respectively. In this example, bundles A, B and C have various internal package and service dependencies. However, the composite also exports a package and service provided by bundle B that are being used by bundle F. In addition to this, the composite is importing a package provided by bundle D and a service provided by bundle E. Both of these are then being used by content bundle C.

It is worth noting that a consequence of the resolution process outlined in section 5.2.8 is that a composite subsystem is not permitted to be involved in a package dependency cycle with other peer subsystems or bundles. Such cycles will result in the subsystem failing to resolve.

### 5.2.3 Feature Subsystems

A feature subsystem is a subsystem that does not impose any isolation. A feature consists of a set of bundles and associated resources. The packages and services provided by the content bundles are all automatically made available to bundles outside the feature. Packages and services required by the content bundles can automatically be satisfied by bundles outside the feature. The content bundles can be required by bundles outside the feature and the content bundles can required by bundles from outside the feature. The main purpose of a feature subsystem is to enable the life-cycle management of a set of bundles.
Subsystem

An implementation of this specification provides a subsystem service for managing the life-cycle of subsystems. The service implements the org.osgi.service.subsystem.Subsystem interface (see section 6). Each subsystem service instance manages subsystems directly visible to it in the subsystem hierarchy, as illustrated in figure 7.

![Subsystem Service instance model](image)

Figure 7: Subsystem Service instance model.

Figure 7 shows four Subsystem service instances registered, each numbered to aid description. In this example, at the outer-most level (the root scope), Subsystem service 1 is managing Subsystems A and B. A and B can be retrieved using Subsystem.getChildren(). Because this service is in the root scope a call to Subsystem.getParent() would return null as it has no parent subsystem. At the next level down, Subsystem service 2 is managing subsystem C and Subsystem service 3 is not managing any subsystems. A call to Subsystem.getChildren() on service 2 would return Subsystem C and Subsystem.getParent() would return Subsystem A. A call to Subsystem.getChildren() on service 3 would return an empty subsystem Collection as it has no child subsystems.

A subsystem implementation may choose to hide the subsystem service to prevent a subsystem's content bundles from creating their own subsystems. This is illustrated in Subsystem C, which does not contain a subsystem service.

Note, the Subsystem service view described above is logically what is expected, however a most likely implementation strategy would be to have a single service factory registration which is visible inside each subsystem and returns the appropriate subsystem instance based on the subsystem in which the requesting bundle lives.
5.2.4 Life-cycle

Subsystems have a life-cycle similar to that of Bundles. Operations on the Subsystem (e.g. install, uninstall, start and stop) cause it to move through its life-cycle. The operations also cause equivalent operations on the Subsystem content (e.g. start on a Subsystem will cause start on any content bundles). In these circumstances, the processing of the Subsystem content must also respect any life-cycle policy of the content resources. For example, starting a subsystem that contains bundle fragment resources must not require those fragments to start as this would be counter to the life-cycle policy of bundle fragments.

The use and behavior of subsystem life-cycle in the presence of the start level services is out of scope.

All subsystem states are defined by the `Subsystem.State` enum, for example, `Subsystem.State.INSTALLED`. For ease of reading these will hereafter be referred to by their short name, e.g. `INSTALLED`.

The life-cycle state of a subsystem is a reflection of the states of the contents and the last action performed through the subsystem api. For example, a Subsystem only transitions to `ACTIVE` when all its contents have successfully started, triggered by calling `start` on the subsystem. However, if a program uses the reflective api to change the state of an individual content bundle, this will not result in a change of state for of the subsystem. A subsystem's state also reflects the cases where only a subset of it's contents have reached some target state. These states are referred to as 'transitional states'. Examples include `INSTALLING` and `RESOLVING`.

A subsystem's state is persistent, however, the state of their contents is not. Restarting a framework will cause subsystems to attempt to reach their previous states, which will try to bring the state of the content resources to the same state. For example, if a subsystem was `ACTIVE` when the framework was stopped, then when that framework is started again, the subsystem will try to reach the `ACTIVE` state again, which will result in an attempt to `start` all content resource, irrespective of their actual state when the framework was stopped.

Figure 8 shows the life-cycle for a Subsystem.
5.2.4.1 Events

When the EventAdmin service is available, the subsystem runtime will produce events based on the Subsystem life-cycle. All events are delivered asynchronously. The Event Topic of these events is:
Whenever an event type is mentioned in this document, it should be assumed it belongs to the above topic, unless stated otherwise.

The following event types are generated based on the life-cycle of the Subsystem

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSTALLING</td>
<td>Indicates a subsystem has started installing</td>
</tr>
<tr>
<td>INSTALLED</td>
<td>Indicates a subsystem has completed installed</td>
</tr>
<tr>
<td>RESOLVING</td>
<td>Indicates a subsystem has started resolving</td>
</tr>
<tr>
<td>RESOLVED</td>
<td>Indicates a subsystem has been resolved</td>
</tr>
<tr>
<td>STARTING</td>
<td>Indicates a subsystem has begun starting</td>
</tr>
<tr>
<td>STARTED</td>
<td>Indicates a subsystem has become active</td>
</tr>
<tr>
<td>STOPPING</td>
<td>Indicates a subsystem has begun stopping</td>
</tr>
<tr>
<td>STOPPED</td>
<td>Indicates a subsystem has been stopped</td>
</tr>
<tr>
<td>UPDATING</td>
<td>Indicates a subsystem has started updating</td>
</tr>
<tr>
<td>UPDATED</td>
<td>Indicates a subsystem has been updated</td>
</tr>
<tr>
<td>UNINSTALLING</td>
<td>Indicates a subsystem has started uninstalling.</td>
</tr>
<tr>
<td>UNINSTALLED</td>
<td>Indicates a subsystem has been uninstalled</td>
</tr>
<tr>
<td>CANCELING</td>
<td>Indicates that a subsystem operation is being canceled.</td>
</tr>
<tr>
<td>FAILED</td>
<td>Indicates a subsystem life-cycle operation (e.g. install, update) failed</td>
</tr>
<tr>
<td>CANCELED</td>
<td>Indicates an asynchronous subsystem operation was canceled (e.g. install or update).</td>
</tr>
</tbody>
</table>

Subsystem events contain the following properties to help identify the subsystem, the time at which the event occurred and any exceptions that may have occurred.

- `SUBSYSTEM_ID` – the id of the subsystem for which the event was being generated (from SubsystemEventConstants).
- `SUBSYSTEM_LOCATION` – the location of the subsystem for which the event was generated (from SubsystemEventConstants).
- `SUBSYSTEM_SYMBOLICNAME` – the symbolic name of the subsystem from which the event was generated (from SubsystemEventConstants).
- `SUBSYSTEM_VERSION` – the symbolic name of the subsystem from which the event was generated (from SubsystemEventConstants).
- `TIMESTAMP` – the time at which the event was generated (from EventConstants).
- `SUBSYSTEM_STATE` – the current state of the subsystem. This is used when the state is not indicated by the event type (e.g. CANCELING) (from EventConstants).
5.2.5 Listening for Internal Subsystem Events

Users of subsystems may need to be aware of the internal goings on of a subsystem. One use case for this is to be able to determine when a subsystem is 'ready'. Readiness may not necessarily directly correspond to one of the subsystem states outlined in section 5.2.4. For example, a library bundle might only need to reach the RESOLVED state, whereas a blueprint bundle might only be considered ready once its blueprint container service has been registered.

The subsystems runtime enables users to snoop on the inner workings of subsystems through EventAdmin. Events that occur inside a subsystem and which are not normally propagated outside the subsystem can be observed by registering an event handler in the subsystem service’s root scope. Internal events are re-published using the following event topic pattern:

\[ \text{org/osgi/service/SubsystemInternals/}\langle \text{original-topic}\rangle \]

The use of the topic token “SubsystemInternals” keeps internal and external events in separate peer topic spaces, thus allowing wildcarding.

\langle \text{original-topic}\rangle \text{ is the original topic of the event being re-published. The event published is the event admin event that would have been generated internally. For example, a bundle STARTED event would be published as:}

\[ \text{org/osgi/service/SubsystemInternals/osg/osgi/framework/BundleEvent/STARTED} \]

Note, because event admin is not subsystem aware, it is necessary to install event admin in every level at which event admin is to be used.

The following properties are added to the event to identify the subsystem from which the event came. These properties allow handlers to filter for a specific subsystem:

- \text{SUBSYSTEM_ID} – the id of the subsystem from which the event was generated (from SubsystemEventConstants).
- \text{SUBSYSTEM_LOCATION} - the location of the subsystem from which the event was generated (from SubsystemEventConstants).
- \text{SUBSYSTEM_SYMBOLICNAME} – the symbolic name of the subsystem from which the event was generated (from SubsystemEventConstants)
- \text{SUBSYSTEM_VERSION} – the symbolic name of the subsystem from which the event was generated (from SubsystemEventConstants)

5.2.6 Class Diagrams
5.2.7 Installing a Subsystem

Subsystems are installed using one of the service's install methods. These methods carry the same arguments as BundleContext.installBundle and the same semantics, such as a null InputStream meaning the InputStream is created from the location identifier. Subsystem install operations are potentially long-running and are therefore asynchronous. Each install method returns a Subsystem with the installation process initiated, but not necessarily complete.

The following steps are required to install a subsystem:

1. If there is an existing subsystem containing the same location identifier as the Subsystem to be installed, then a SubsystemException is thrown.
2. If this is a new install, then a new Subsystem is created with its id set to the next available value (ascending order).
3. The subsystem's state is set to INSTALLING and if EventAdmin is available, an event of type INSTALLING is fired.
4. The following installation steps are then started and performed asynchronously and the new subsystem is returned to the caller.
5. The subsystem content is read from the input stream.
6. If the subsystem requires isolation (i.e. is an application or a composite), then isolation is set up while the install is in progress, such that none of the content bundles can be resolved. This isolation is not changed until the subsystem is explicitly request to resolve (i.e. as a result of a Subsystem.start() operation).
7. The resources are installed into the framework through the use of a ResourceProcessors, including for bundles. In the event of multiple resource processors being available for a particular type of resource, the processor used is the one that would be returned by BundleContext.getServiceReference(). If any resources fail to install, then failure is handled as described in 5.2.7.1.
8. The subsystem's state is set to INSTALLED and if EventAdmin is available an INSTALLED event is fired.

Note, if the subsystem requires isolation and it remains in the INSTALLING or INSTALLED state, none of it's content bundles must be permitted to resolve.

5.2.7.1 Installation Failures

If the installation of a Subsystem fails, for example, due to problems opening an InputStream, then a subsystem FAILED event is fired. The exception that caused the failure is set in the event properties exception, exception_class and exception_message. The subsystem id and location are also provided in the event to identify the failed subsystem, and finally, the timestamp is set to indicated when the failure occurred.

Failure to install one of the content resources (e.g. a content bundle) must result in a failure to install the subsystem and an attempt to undo any associated installation activities. ResourceProcesser's will be notified via the Coordination that was provided in the call to their process method and they should attempt to undo any work they have performed as part of that coordination.

When all resource processors have completed failing the install, the subsystem's state is set to UNINSTALLED and the UNINSTALLED event fired via EventAdmin, if available.
### 5.2.8 Resolving a Subsystem

Resolution of a subsystem only happens as a result of an explicit request (i.e. by calling `Subsystem.start()`). A subsystems state is only considered `RESOLVED` when all its content bundles are `RESOLVED`. The subsystem runtime issues a `RESOLVED` event when this condition is met.

To ensure a consistent and 'whole' view is presented for the composite subsystem externals, resolution happens in two steps. Resolution for Feature subsystems, which are transparent and Application subsystems, which do not export anything, can happen in a single step. Subsystem resolution occurs as follows:

1. Resolution is initiated with a call to `Subsystem.start()`.
2. If the subsystem is in the `UNINSTALLING` or `UNINSTALLED` state then a `SubsystemException` is thrown.
3. If the subsystem is already resolved, then no action is taken.
4. The subsystem’s state is set to `RESOLVING` and a `RESOLVING` event is fired via EventAdmin, if available.
5. If the subsystem is one with isolation, then isolation is configured such that its content bundles are permitted to resolve against each other and any externals that it imports. If the subsystem is a composite then it its exports are not yet made available for resolution.
6. An attempt is made to resolve each of the content bundles.
7. If the subsystem fails to resolve (perhaps due to a content bundle not resolving) then the coordination is failed and the subsystem state is set to `INSTALLED`. A subsystem `FAILED` event is fired via EventAdmin, if available.
8. If the subsystem is a composite, then once all content bundles are `RESOLVED`, the subsystem makes any packages it exports available for resolution. Any subsequent resolution is then able to wire to packages provided by the composite subsystem. The subsystem's state is set to `RESOLVED` and a `RESOLVED` is fired via EventAdmin, if available.

A consequence of this approach is that a composite subsystem is not permitted to be part of a package dependency cycle with other peer bundles or composite subsystems. If cycles exist then the composite subsystem will never resolve.

### 5.2.9 Starting a Subsystem

Subsystems are started by calling the Subsystem `start` method. Starting a subsystem performs the following steps:

1. If the Subsystem is in the `UNINSTALLING` or `UNINSTALLED` state, then a `SubsystemException` is thrown.
2. If the Subsystem is in the `ACTIVE` state, then the method returns immediately.
3. If there are existing content bundles that have not been resolved, then the runtime must attempt resolve these first, as described in section 5.2.8. If the subsystem fails to resolve then the start does not continue.
4. Once all bundles are RESOLVED (i.e. the subsystem is resolved) then the subsystem's state is set to STARTING and if EventAdmin is available, an event of type STARTING is fired.

5. From this point a framework restart must automatically start the subsystem (i.e. the subsystem's start state is persisted).

6. Process is called on the resource processors that are managing the resources for the subsystem, passing in a ResourceOperation describing the start operation to be performed. Note, the subsystem content resources must only be transiently started, meaning the persistent start of the containing subsystem will cause them to be started at the appropriate time. This avoid circumstances where content resources could be started ahead of the subsystem being set up.

7. If a resource processor is unable to start a resource then it must fail the coordination by calling the coordination fail() method, providing an exception with the reason for the failure. This will cause all participants to be notified that the coordination has failed and therefore they must attempt to undo any starts they have done under that coordination. The subsystem state must be set to RESOLVED. The FAILED event is fired with the subsystem identification information, timestamp, and any exception information regarding the reason set in the appropriate event properties.

8. If all content resources successfully start (note, for some resource types, start may be a no-op, for example, in the case of a bundle fragment) then the subsystem's state is set to ACTIVE, and the STARTED event is fired via EventAdmin, if available.

Note, as each content bundle is activated, it may register and consume services before the entire subsystem is activated. This behavior is inconsistent for composite subsystems, where the design goal is to present a “whole view”, however, to achieve this would require complex alternatives, such as replaying service events.

5.2.10 Stopping a Subsystem

Subsystems are stopped by calling the Subsystem stop method. Stopping a subsystem performs the following steps:

1. If the Subsystem is in the UNINSTALLING or UNINSTALLED state then an IllegalStateException is thrown.

2. If the subsystem's state is not STARTING or ACTIVE then this method returns immediately.

3. The Subsystem's state is set to STOPPING and the STOPPING is fired via EventAdmin, if available.

4. The stopping process is initiated asynchronously and the method returns. From this point on, the stop subsystem stop state is persisted. The stopping process performs the following actions:

5. Process is called on the resource processors for each resource to be stopped, passing in the ResourceOperation that describes the stop operation to be performed and any operation context.

6. If a resource processor process operation throws an exception then the FAILED event is fired with the subsystem identification information, timestamp, and any exception information regarding the reason set in the appropriate event properties. Stop processing continues to try to stop any remaining resources.

7. Once all resource processors have been called to stop the content resources, the Subsystem's state is set to RESOLVED and the STOPPED event is fired via EventAdmin, if available.
5.2.11 Stopping the Framework

When the framework is stopped, all subsystems must also be stopped before the start level starts taking down the bundles. This ensures subsystem stopping is managed appropriately by the subsystem runtime and not as an accidental and undesirable side-effect of the framework stopping individual bundles.

5.2.12 Unstalling a Subsystem

Subsystems are uninstalled by calling the Subsystem uninstall method. Uninstalling a subsystem results in it being put into the UNINSTALLED state and sending an UNINSTALLED event. The Framework must remove any resources related to this subsystem that it is able to remove. If this subsystem has exported any packages, the Framework must continue to make these packages available to their importing bundles or subsystems until the PackageAdmin.refreshPackages method has been called or the Framework is relaunched. The following steps are required to uninstall a subsystem:

1. If this subsystem's state is UNINSTALLED then an IllegalStateException is thrown.
2. If this subsystem's state is ACTIVE, STARTING or STOPPING, this subsystem is stopped as described in section 5.2.10.
3. The Subsystem's state is set to UNINSTALLING and the UNINSTALLING event is fired via EventAdmin, if available.
4. The uninstall process is initiated asynchronously and the method returns. The uninstall process performs the following actions:
   
   5. Process is called on the resource processors for each resource to be uninstalled, passing in a ResourceOperation describing the uninstall operation and providing the operation context.
   
   6. If a resource processor uninstall operation throws an exception then the FAILED event is fired with the subsystem identification information, timestamp, and any exception information regarding the reason set in the appropriate event properties. Uninstall processing continues to try to uninstall any remaining resources.
   
   7. Once an attempt has been made to uninstall all resources, the Subsystem state is set to UNINSTALLED and the UNINSTALLED event is fired via EventAdmin, if available.

5.2.13 Updating a Subsystem

Subsystems are updated using one of the Subsystem update methods, passing in the Subsystem to be updated, and optionally an InputStream from where to perform the update. If there is no InputStream provided then the subsystem definition's update location header is used, and if that header is not provided, then the original location specified during install is used.

5.2.14 Canceling a Subsystem operation

Asynchronous subsystem operations can be canceled through the the Subsystem cancel method. Cancellation of a subsystem performs the following steps:

1. If the subsystem is not in a transitional state initiated through the Subsystem service (i.e. not INSTALLING, UNINSTALLING, STARTING, UPDATING, STOPPING), then an IllegalStateException is thrown.

2. It is permissible for an implementation to delay canceling if it is in the final steps of completing an in-flight operation. This would result in the cancel operation blocking briefly and then an IllegalStateException being thrown.
3. A CANCELING event is fired via EventAdmin, if available.

4. The cancel process is initiated asynchronously and the method returns. The cancel process performs the following action:

5. The runtime attempts to undo all actions performed as part of the current operation. This is done by failing the coordination used to coordinate any resource processors used which should result in reverting any actions taken under the coordination.

6. If any resource process throws an exception on the failed method, then a FAILED event is fired via EventAdmin, if available and the subsystems state remains in the transitional state.

7. If the resource processors failed cleanly, then the subsystem state is set to the state it was in prior to beginning the transition, and a CANCELED event is fired via EventAdmin, if available.

5.2.15 Retrieving Subsystems

Subsystems form a tree hierarchy. The child subsystems of a subsystem can be retrieved by calling the Subsystem getChildren() method. The parent subsystem of a subsystem can be retrieved by calling its getParent() method.

5.3 Resource Processors

Resource processor design is inspired by deployment admin. A resource processor is a service that adds the ability to manage the life-cycle of resources on behalf of a subsystem. For example, a resource processor might manage subsystem configuration, or certificates.

5.3.1 Resource Processor Services

Each resource processor is registered in the service registry under the interface org.osgi.service.resource.ResourceProcessor. The types of resources a resource processor can handle are identified using the service property osgi.resource.namespace. The namespace values are the same as those used in the generic requirements and capabilities (RFC 154) and OBR (RFC 112), or should follow the best practice for defining custom namespaces. The type of this property is String+ to allow a single resource processor to provide support for multiple namespaces.

The Subsystem implementation must only use resource processor services that are visible to its implementation bundle. In other words the service must be registered in the root scope. The subsystem implementation must not look inside other scopes to find resource processors. Note, if a subsystem exports a resource processor up to the root scope then this resource processor becomes a candidate to be used by the subsystem runtime.

If there are multiple resource processors that support the same namespace, then the one returned by getServiceReference (i.e. highest ranking, lowest id (tie breaker)) must be used.

TBD: Describe how to ‘observe’ resource life-cycle without being ‘the’ resource processor (Resource Listener).

5.3.2 Resource Processor Life-cycle

TBD: Define what happens when you remove/replace a resource processor. Do we take down any dependent subsystems? Do we have a “forced” capability similar to that used by deployment admin?
5.3.3 Resource Processor Operations

Resource processors implement a process operation that is called for each of the life-cycle operations that can be performed on a subsystem itself (install, uninstall, start, stop and update). The process operation is passed a ResourceOperation that describes the following:

- The operation to be performed.
- The context of the operation (this can include a reference to the subsystem to which the resource belongs.
- The coordination under which the operation is being performed. If no coordination is provided, then the operation is performed independent of any others. If a coordination is provided then the resource processor must register as a participant and must fail the coordination if it fails to process the resource.

The subsystem runtime uses the coordinator service to coordinate the outcomes of these operations across resource processors for a single subsystem. The resource processor must process each request immediately and only return once the processing has completed. If a resource processor is notified that the coordination failed then it must cancel or attempt to undo any actions requested under that coordination. If the resource processor cannot undo all actions then it must throw an exception from the Participant.failed call.

5.4 Subsystem Definitions

The following sections describe the headers used to described subsystem definitions. As mentioned earlier, three types of subsystem are defined:

- Application – An implicitly scoped subsystem. See 5.4.2.
- Composite – An explicitly scoped subsystem. See 5.4.4.
- Feature – An unscoped subsystem. See 5.4.6.

A subsystem can exhibit some, or all, of the following features:

- Identity – a symbolic name and version (Note, this does not define uniqueness, as the same subsystem may be installed multiple times).
- Description – human readable information about the subsystems
- Content – the things that make up the subsystem content, such as bundles and configuration
- Bundle Sharing – ability to share bundles into or out of a subsystem.
- Package Sharing – the ability to share packages into or out of a subsystem. The defaults for packages sharing differ based on the type of subsystem (e.g. scoped vs unscoped).
- Service Sharing – the ability to share services into or out of a subsystem. The defaults for service sharing differ based on the type of subsystem (e.g. scoped vs unscoped).
- Nesting – the ability to define subsystems in terms of other subsystems.
• **Transitive Closure** – the ability to allow subsystems to define core content and have their remaining transitive dependencies calculated during deployment.

• **Configuration** – configuration for the subsystem and its contents.

Manifests are defined for each type of subsystem. A design using a single set of subsystem headers was considered but this resulted in a need to understand which headers are valid for which type of subsystem. So instead, the design identifies explicit types of subsystem and supporting headers for each.

### 5.4.1 Manifest Header Processing

Subsystem manifests follow the Jar manifest format, but to improve usability, there are two rules which are relaxed:

1. Line lengths can exceed the Jar manifest maximum of 72 bytes.

2. The last line is not required to be a newline.

### 5.4.2 Application Manifest Headers

**TODO:** fit-n-finish on the headers (take a look at what we have for bundles and see what equivalents we need for subsystems.

An application is an implicitly scoped subsystem and therefore has rules that govern sharing of packages and services, but makes no explicit statements about sharing. An application is not required to be transitive closed and therefore may require additional dependencies to be provisioned during deployment. Any transitive dependencies that are provisioned are done so into the “root scope” and therefore must be visible inside any ancestor subsystems in which the application resides. (see section 5.5). An Application is defined using the following manifest headers

#### 5.4.2.1 Manifest Versioning Headers

- **Manifest-Version** – the version of the scheme for manifests that this manifest conforms to. This is the standard manifest header and is not defined by this specification.

- **Application-ManifestVersion** – an application manifest must conform to a version of the application manifest headers. The version is defined by the OSGi specification and will be 1 for the first version.

#### 5.4.2.2 Identity Headers

The following headers are used to specify the identity of an application:

- **Application-SymbolicName** – the symbolic name of the application. Follows the same scheme and guidelines as Bundle-SymbolicName. This name belongs to the same namespace as all other subsystem types and therefore a subsystem with the same name and version are considered to be the same.

- **Application-Version** – the version of the application (optional). Follows the same scheme and guidelines as Bundle-Version.
5.4.2.3 Informational Headers

The following headers are informational and are intended to be human readable:

- **Application-Name** – the human readable application name. Follows the same scheme and guidelines as **Bundle-Name**.

- **Application-Description** – the description of the application. Follows the same scheme and guidelines as **Bundle-Description**.

5.4.2.4 Content Header

The following header identifies the content of an application:

- **Application-Content** – the content of the application. This is the key content for the application. At runtime, this content will be provisioned into a composite bundle. See section 5.4.2.6 for the detailed definition of application content.

5.4.2.5 Dependency Header

The following header is used to ensure a particular Bundle or Composite is used to satisfy the implicit package dependencies of an application:

- **Preferred-Provider** – a set of bundles or composite to be preferred to satisfy the application's implicit package imports. See section 5.4.2.7 for the detailed definition of use bundle.

Preferred providers are used to express a preference for a bundle or composite to satisfy implicit package dependencies. If an application has an implicit import package that is satisfied by a bundle listed in Preferred-Provider then the package import should be tied to that bundle (e.g. through bundle-symbolic-name or bundle-version matching attributes or equivalent mechanism). Any bundles listed in Preferred-Provider that satisfy implicit imports are provisioned into the subsystem root (as is the case with any other bundles satisfying implicit package dependencies).

This mechanism enables applications to share bundles or composites with other applications whilst still ensuring they use the provider of the packages they expect.

5.4.2.6 Application-Content

Application content is a list of content that can be of various types. The default type is bundle. A bundle is identified by its symbolic name. For example

```
Application-Content: com.acme.bundles.bundle1,
com.acme.bundles.bundle2
```

The following matching attribute can be applied to bundle content:

- **version** – a version range used to select the bundle version of the bundle to use. This follows the OSGi version range scheme, including the default.

The following directives can be applied to bundle content:
5.4.2.7 Preferred-Provider

Preferred provider is a list of bundles or composites that the application explicitly uses packages from. A bundle or composite is identified by its symbolic name and an optional namespace directive. For example

Use-Bundle: com.acme.bundles.bundle1,
com.acme.bundles.bundle2;namespace:=osgi.subsystem

The following matching attribute can be applied:

• version – a version range used to select the version of the bundle or composite to use. This follows the OSGi version range scheme, including the default.

The following directive can be applied to the Preferred-Provider header:

• namespace – indicates the namespace of the preferred provider. This value is used to identify uniquely the provider when resolving the subsystem. It is recommended that a reverse domain name convention is used unless those types and their processing is standardized by the OSGi Alliance (e.g. bundle). The default namespace is 'osgi.bundle'. Another subsystem is identified with a namespace of 'osgi.subsystem'.

Open Question: Is there a need for service affinity or other types of capability?

5.4.2.8 Defaulting

TODO: Each header defaulted individually. Cover defaulting of things like:

• symbolic name derived from archive name.

• Version default to 0.0.0.0

• Application name is archive name, including the extension.

• etc

5.4.3 Application Archive

An application archive is a zip file ending in the extension .eba. Unlike jar archives, there are no special rules governing the order of the archive contents. An application archive consists of the following:
1. The application manifest. This is optional and if omitted will be calculated based on the rules defined in section 5.4.2.8. The application manifest is stored in the archive in a file named, OSGI-INF/APPLICATION.MF.

2. A deployment manifest. This is optional and if omitted is calculated during provisioning of the application. It is anticipated that application archives will rarely contain deployment manifests during development, but as the application is tested prior to putting the application into production, it will be common to add the deployment manifest to 'lock down' the bundles that are provisioned. The deployment manifest is stored in the archive in a file named, OSGI-INF/DEPLOYMENT.MF.

3. Any resources that may help with the provisioning of the subsystem. Note, it is feasible for an application archive to contain no resources and have all it's contents and dependencies provisioned from a repository.

5.4.4 Composite Manifest Headers

A composite subsystem defines what is essentially a collection of bundles with explicit fully-explicit sharing and therefore looks like a bundle. By default, it does not share any packages or services in or out. Instead, any sharing must be explicitly stated in the composite manifest. The design aims to closely model that of bundles, including how package imports/exports are expressed and how composites are versioned based on update to their contents.

A composite is not required to be transitively closed and therefore may require additional dependencies to be provisioned during deployment. Any transitive dependencies that are provisioned are done so into the "root scope" and therefore must be visible inside any ancestor subsystems in which the composite resides (see section 5.5). A composite is defined using the following manifest headers:

5.4.4.1 Manifest Versioning Headers

• Manifest-Version – the version of the scheme for manifest that this manifest conforms to.

• CompositeBundle-ManifestVersion – a composite manifest must conform to a version of the composite manifest headers. The version is defined by the OSGi specification and will be 1 for the first version.

5.4.4.2 Identity Headers

The following headers are used to specify the identity of a composite:

• Bundle-SymbolicName – the symbolic name of the composite. Its syntax and usage is identical to that of bundles. This belongs to the same namespace as all other subsystem types and therefore a subsystem with the same name and version are considered to be the same.

• Bundle-Version – the version of the composite (optional). Its syntax and usage is identical to that of bundles.

5.4.4.3 Informational Headers

The following headers are informational and are intended to be human readable:

• Bundle-Name – the human readable composite name. Its syntax and usage is identical to that of bundles.
5.4.4.4 **Content Header**

The following header identifies the content of an composite:

- **CompositeBundle-Content** – the content of the composite. This is the key content for the composite. At runtime, this content will be provisioned into a composite bundle. See section 5.4.2.6 for the detailed definition of composite content.

5.4.4.5 **CompositeBundle-Content**

Composite content is a list of content that can be of various types. The default type is bundle, including composite bundle. A bundle is identified by its symbolic name. For example

```
CompositeBundle-Content: com.acme.bundles.bundle1,
com.acme.bundles.bundle2
```

The following matching attribute can be applied to bundle content:

- **version** – a version used to identify the exact version of the bundle to use.\(^1\)

The following directive can be applied to bundle content:

- **resolution** – states whether the bundle must exist in order for the composite to be considered complete. A value of `mandatory` (the default) means the bundle must exist, a value of `optional` means the composite can be complete even if the bundle does not exist.

- **namespace** – indicates the namespace of the content. This value is used to uniquely identify the content resource. It is recommended that a reverse domain name convention is used unless those types and their processing is standardized by the OSGi Alliance (e.g. bundle). The default namespace is 'osgi.bundle'. Another subsystem is identified with a namespace of 'osgi.subsystem'.

5.4.4.6 **Explicit Sharing Headers**

Composite’s can make explicit statements about service, package and bundle sharing using the following headers.

- **Import-Package** – specifies a list of package constraints to import into the composite. Uses the same syntax as bundle Import-Package. Any exported package from a bundle or composite that is a peer to this composite and which matches one of the constraints is available to satisfy any Import-Package constraints of the content bundles.

- **Export-Package** – specifies a list of package to export out of the composite. Uses the same syntax as bundle Export-Package. Any exports listed here must exactly match (i.e. they must have the same attributes and directive, but they can appear in a different order) an export from one of the content bundles.

---

\(^1\) Version ranges are problematic due to the interplay between contained bundle imports/exports and those of the composite.
• **Require-Bundle** – specifies a list of require bundle constraints to import into the composite. Uses the same syntax as Require-Bundle. Any bundle that is a peer to the composite which matches one of the constraints is available to satisfy Require-Bundle constraints of the composite's content bundles.

Note, it is not possible to for a peer of a composite to require a bundle contained in another composite. It is also meaningless to do require-bundle naming a composite because a composite does not export the packages itself. Finally, requiring a bundle into a composite does not make the required bundle's packages available to all other bundle in the composite.

• **CompositeBundle-ImportService** – specifies a list of service filters that control the services that are imported into the composite. Any services registered by peer bundles or exported by peer composites that match one of these filters are made available to the content bundles.

• **CompositeBundle-ExportService** – specifies a list of service filters that control the services that are exported out of the composite. Any services registered by content bundles that match one of these filters are made available to peer bundles and composites.

### 5.4.4.7 Defaulting

TODO: Rules will be similar to those of the application.

### 5.4.5 Composite Bundle Archive

A composite bundle archive is a zip file ending in the extension .cba. Unlike jar archives, there are no special rules governing the order of the archive contents. A composite bundle archive consists of the following:

1. The composite bundle manifest. This is optional and if omitted will be calculated based on the rules defined in section 5.4.4.7. The composite manifest is stored in the archive in a file named, OSGI-INF/COMPOSITEBUNDLE.MF.

2. A deployment manifest. This is optional and if omitted is calculated during provisioning of the composite bundle. It is anticipated that composite bundle archives will rarely contain deployment manifests during development, but as the composite is tested prior to putting the it into production, a deployment manifest may be added to 'lock down' the bundles that are provisioned. The deployment manifest is stored in the archive in a file named, OSGI-INF/DEPLOYMENT.MF.

3. Any resources that may help with the provisioning of the composite. Note, it is feasible for an composite bundle archive to contain no resources and have all it's contents and dependencies provisioned from a repository.

### 5.4.6 Feature Manifest Headers

A feature is an unscoped subsystem and therefore has no influence over runtime resolution, sharing all packages or services in and out. A feature does not provide any mechanism to restrict access. If a feature is nested inside a subsystem that does restrict visibility (e.g. a composite), then the feature will only have visibility to those things that are visible inside the subsystem in which it is nested. A feature is not required to be transitively closed and therefore may require additional dependencies to be provisioned during deployment. Any transitive dependencies that are provisioned are done so into the “root scope” and therefore must be visible inside any ancestor subsystems in which the feature resides (see section 5.5). A feature is defined using the following manifest headers
5.4.6.1 **Manifest Versioning Headers**

- **Manifest-Version** – the version of the scheme for manifest that this manifest conforms to.

- **Feature-ManifestVersion** – a feature manifest must conform to a version of the feature manifest headers. The version is defined by the OSGi specification and will be 1 for the first version.

5.4.6.2 **Identity Headers**

The following headers are used to specify the identity of a feature:

- **Feature-SymbolicName** – the symbolic name of the feature. Follows the same scheme and guidelines as Bundle-SymbolicName. This name belongs to the same namespace as all other subsystem types and therefore a subsystem with the same name and version are considered to be the same.

- **Feature-Version** – the version of the feature (optional). Follows the same scheme and guidelines as Bundle-Version.

5.4.6.3 **Informational Headers**

The following headers are informational and are intended to be human readable:

- **Feature-Name** – the human readable feature name. Follows the same scheme and guidelines as Bundle-Name.

- **Feature-Description** – the description of the feature. Follows the same scheme and guidelines as Bundle-Description.

5.4.6.4 **Content Header**

The following header identifies the content of a feature:

- **Feature-Content** – the content of the feature. This is the key content for the feature. At runtime, this content will be provisioned into an OSGi framework without applying isolation. See section 5.4.2.6 for the detailed definition of feature content.

5.4.6.5 **Feature-Content**

Feature content is a list of content that can be of various types. The default type is bundle, including composite bundle. A bundle is identified by its symbolic name. For example

```
Feature-Content: com.acme.bundles.bundle1, com.acme.bundles.bundle2
```

The following matching attribute can be applied to bundle content:

- **version** – a version range used to select the bundle version of the bundle to use. This follows the OSGi version range scheme, including the default.

The following directive can be applied to bundle content:
• **resolution** – states whether the bundle must exist in order for the feature to be considered complete. A value of **mandatory** (the default) means the bundle must exist, a value of **optional** means the feature can be complete even if the bundle does not exist.

• **namespace** – indicates the namespace of the content. This value is used to uniquely identify the content resource. It is recommended that a reverse domain name convention is used unless those types and their processing is standardized by the OSGi Alliance (e.g. bundle). The default namespace is 'osgi.bundle'. Another subsystem is identified with a namespace of 'osgi.subsystem'.

### 5.4.6.6 Defaulting

TODO: Defaulting will be similar to that of applications.

### 5.4.7 Feature Archive

A feature archive is a zip file ending in the extension .fba. Unlike jar archives, there are no special rules governing the order of the archive contents. A feature archive consists of the following:

1. The feature manifest. This is optional and if omitted will be calculated based on the rules defined in section 5.4.6.6. The feature manifest is stored in the archive in a file named, OSGI-INF/FEATURE.MF.

2. A deployment manifest. This is optional and if omitted is calculated during provisioning of the feature. It is anticipated that feature archives will rarely contain deployment manifests during development, but as the feature is tested prior to putting the it into production, a deployment manifest may be added to 'lock down' the bundles that are provisioned. The deployment manifest is stored in the archive in a file named, OSGI-INF/DEPLOYMENT.MF.

3. Any resources that may help with the provisioning of the feature. Note, it is feasible for an feature archive to contain no resources and have all it's contents and dependencies provisioned from a repository.

### 5.4.8 Manifest Localization

TODO: Describe how headers are localized. This will follow the approach used for bundle manifest headers.

### 5.4.9 Deployment Manifest

A deployment manifest describes the bundles and resources that should be provisioned for a particular subsystem definition. The deployment manifest can be packaged with the subsystem or determined during deployment. A deployment manifest could be authored by hand but this is highly unlikely as the process is complex and would be error prone. Therefore, in most cases the deployment manifest will be calculated by a resolver, either out of band in some pre-deployment step, or during the installation of the subsystem, the latter making use of the OBR specification, identified in the subsystem architecture in figure 4.

#### 5.4.9.1 Portability

A deployment manifest describes the bundles and resources that need to be provisioned to satisfy the subsystem definition. Resolution happens in the context of a target runtime, which could be a specific server instance, an empty framework, and so on. As such, a deployment manifest may not be transferable from one environment to another.
Whilst the format and location of deployment manifests is standardized it is not a requirement that a deployment manifest calculated for one runtime work in another. Installation of a subsystem can therefore determine whether the deployment manifest is valid and if not, choose to calculate a new deployment, or fail the installation.

5.4.9.2 Design

The deployment manifest describes the resources to be provisioned for a subsystem, irrespective of the subsystem type. These resources fall into two categories:

1. Deployed content: the content that is to be deployed into the subsystem.
2. Provision bundle: the bundles to be provisioned outside the subsystem in support of its external dependencies.

A deployment manifest is located in the OSGI-INF folder of the Subsystem archive file and is named DEPLOYMENT.MF.

5.4.9.3 Manifest Version Header

- **Manifest-Version** – the version of the scheme for manifest that this manifest conforms to.

5.4.9.4 Content Header

- **Deployed-Content** – the content to be deployed for the subsystem. This is the exact content that is provisioned into the Composite Bundle that implements the Subsystem. The exact details of the Deployed-Content header are described in section 5.4.9.5.

5.4.9.5 Deployed-Content

Deployed content is a list of exact content that can be of various types. The default type is bundle, including composite bundle. A bundle is identified by its symbolic name. For example

```
Deployed-Content: com.acme.bundles.bundle1,
                  com.acme.bundles.bundle2
```

Each entry must uniquely identify the resource to be provisioned into the subsystem.

The following matching attribute can be applied to bundle content:

- **deployed_version** – the exact version of the resource to be deployed. Deployed version is a specific version, not a version range, hence the use of a new attribute name.

The following directive can be applied to resource content:

- **namespace** – indicates the namespace of the content. This value is used to identify a resource processor capable of processing the content. The default value is "osgi.bundle". This specification defines "osgi.subsystem" for subsystems.

5.4.9.6 Subsystem Identification

The subsystem to which the deployment manifest applies is identified by the subsystem's symbolic name and version headers, defined earlier. The specific headers for each subsystem type are as follows:
• **Application:** Application-SymbolicName and Application-Version

• **CompositeBundle:** Bundle-SymbolicName and Bundle-Version

• **Feature:** Feature-SymbolicName and Feature-Version

It is an error if the deployment manifest is contained in a subsystem archive and the identification does not match that of the archive (either in the subsystem's definition manifest (e.g. APPLICATION.MF) or one which would be generated).

### 5.4.9.7 Provisioning Header

• **Provision-Bundle** – the bundles to be provisioned in support of the subsystem's transitive dependencies. These are the specific bundles that are provisioned outside the Composite Bundle that implements the Subsystem. The exact details of the Provision-Bundle header are described in section 5.4.9.8.

### 5.4.9.8 Provision-Bundle

Provision bundle lists the bundles to be provisioned in support of a subsystem's external dependencies. These dependencies can include package or service requirements.

Provision bundle has one required matching attribute:

• **deployed_version** – the exact version of the resource to be deployed. Deployed version is a specific version, not a version range, hence the use of a new attribute name.

### 5.4.9.9 Package Imports

### 5.4.9.10 Package Exports

### 5.4.9.11 Service Imports

### 5.4.9.12 Service Exports

### 5.4.9.13 Example Deployment Manifest

### 5.4.9.14 Use Bundle

```plaintext
Manifest-Version: 1.0
Application-SymbolicName: com.ibm.ws.eba.example.blog.app
Application-Version: 1.0
Deployed-Content: com.ibm.ws.eba.example.blog.api;deployed-version=1.0, com.ibm.ws.eba.example.blog.persistence;deployed-version=1.0.0, com.ibm.ws.eba.example.blog.web;deployed-version=1.0.0, com.ibm.ws.eba.example.blog.biz;deployed-version=1.0.0
DeployedService-Import: (&(objectClass=com.ibm.ws.eba.example.blog.api.CommentService)(objectClass=com.ibm.ws.eba.example.blog.persistence.api.CommentService))
Import-Package: com.ibm.json.java;version="1.0.0";bundle-symbolic-name="com.ibm.json.java";bundle-version="[1.0.0,1.0.0]",javax.persistence;version="1.0.0",javax.servlet.http;version="[2.5.0,3.0.0]",javax.servlet;version="[2.5.0,3.0.0]"
Deployed-Use-Bundle: com.ibm.json.java;deployed-version=1.0.0
```

*Figure 9: Example Deployment Manifest.*
5.4.10 Configuration

Notes:

- Config Admin
- Security Permissions

5.5 Transitive Closure

Notes:

- Desire expressed to be able to control 'auto-provisioning'. Some debate as to whether that is expressed in the subsystem artefact or a policy of the environment into which the subsystem is deployed. The latter seemed more appropriate. - Management Agent can choose to do this or not.
- Need to be able to handle runtime requirements
- Need to be able to handle license requirements

5.5.1 Garbage Collection

Notes:

- Transitively provisioned bundles are shared.
- Need to understand when they are no longer required and can therefore be garbage collected.

5.6 Runtime

Notes:

- Scoped subsystems map to CompositeBundles in the runtime.
- Some discussion around whether unscoped subsystems should also map to CompositeBundles.
# 6 JavaDoc

## Package Summary

<table>
<thead>
<tr>
<th>Package Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.osgi.model.resource</td>
<td>Model Resource Package Version 1.0.</td>
<td>40</td>
</tr>
<tr>
<td>org.osgi.service.resource</td>
<td>Resource Service Package Version 1.0.</td>
<td>52</td>
</tr>
<tr>
<td>org.osgi.service.subsystem</td>
<td></td>
<td>58</td>
</tr>
</tbody>
</table>
Package org.osgi.framework.wiring

Package org.osgi.model.resource

Model Resource Package Version 1.0.

See: Description

<table>
<thead>
<tr>
<th>Interface Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Capability</strong> A capability that has been declared from a Resource.</td>
<td>41</td>
</tr>
<tr>
<td><strong>Requirement</strong> A requirement that has been declared from a Resource.</td>
<td>43</td>
</tr>
<tr>
<td><strong>Resource</strong> A resource is the representation of a uniquely identified and typed data.</td>
<td>45</td>
</tr>
<tr>
<td><strong>Wire</strong> A wire connecting a Capability to a Requirement.</td>
<td>50</td>
</tr>
</tbody>
</table>

Package org.osgi.model.resource Description

Model Resource Package Version 1.0.

Bundles wishing to use this package must list the package in the Import-Package header of the bundle's manifest. For example:

```
Import-Package: org.osgi.model.resource; version="[1.0,2.0)"
```
public interface Capability

A capability that has been declared from a Resource.

Version:
$Id: a2e0206881d472363626279e238a0a14366548b7 $

ThreadSafe

<table>
<thead>
<tr>
<th>Method Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map&lt;String, Object&gt; getAttributes()</td>
<td>42</td>
</tr>
<tr>
<td>Returns the attributes of this capability.</td>
<td></td>
</tr>
<tr>
<td>Map&lt;String, String&gt; getDirectives()</td>
<td>41</td>
</tr>
<tr>
<td>Returns the directives of this capability.</td>
<td></td>
</tr>
<tr>
<td>String getNamespace()</td>
<td>41</td>
</tr>
<tr>
<td>Returns the name space of this capability.</td>
<td></td>
</tr>
<tr>
<td>Resource getResource()</td>
<td>42</td>
</tr>
<tr>
<td>The resource that declares this capability.</td>
<td></td>
</tr>
</tbody>
</table>

**Method Detail**

**getDescription**

String getDescription()

Returns the description of this capability.

Returns:
The description of this capability.

**getDirectives**

Map<String, String> getDirectives()

Returns the directives of this capability. Only the following list of directives are allowed in the returned map of directives:

9. effective
10. uses
11. mandatory - can only be present for the osgi.wiring.bundle and osgi.wiring.package name spaces.
12. exclude - can only be present for the osgi.wiring.package name space.
13. include - can only be present for the osgi.wiring.package name space.

No other directives will be present in the returned map. OSGi Alliance reserves the right to extend the set of directives.
Interface BundleCapability

Returns:
An unmodifiable map of directive names to directive values for this capability, or an empty map if this capability has no directives.

getAttributes

Map<String, Object> getAttributes()

Returns the attributes of this capability.

Returns:
An unmodifiable map of attribute names to attribute values for this capability, or an empty map if this capability has no attributes.

ggetResource

Resource getResource()

The resource that declares this capability.

Returns:
the resource
**Interface Requirement**

**org.osgi.model.resource**

```java
public interface Requirement
```

A requirement that has been declared from a `Resource`.

**Version:**

$Id: 0ca69561eb6d5f6fd0c3c6a2ab2c6d5211e22b6c2 $

**ThreadSafe**

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getDirectives()</code></td>
<td>Returns the directives of this requirement.</td>
</tr>
<tr>
<td><code>getNamespace()</code></td>
<td>Returns the name space of this requirement.</td>
</tr>
<tr>
<td><code>getResource()</code></td>
<td>Returns the resource declaring this requirement.</td>
</tr>
<tr>
<td><code>matches(Capability)</code></td>
<td>Returns whether the specified capability matches this requirement.</td>
</tr>
</tbody>
</table>

### Method Detail

#### `getNamespace`  

```java
String getNamespace()
```

Returns the name space of this requirement.

**Returns:**

The name space of this requirement.

#### `getDirectives`  

```java
Map<String, String> getDirectives()
```

Returns the directives of this requirement. Only the following list of directives are allowed in the returned map of directives:

1. `effective`
2. `filter`
3. `resolution`
4. `visibility` - can only be present for the `osgi.wiring.bundle` name space.

No other directives will be present in the returned map. OSGi Alliance reserves the right to extend the set of directives.

**Returns:**

An unmodifiable map of directive names to directive values for this requirement, or an empty map if this requirement has no directives.
**getResource**

Resource getResource()

Returns the resource declaring this requirement.

**Returns:**

The resource declaring this requirement.

**matches**

boolean matches(Capability capability)

Returns whether the specified capability matches this requirement. A capability matches this requirement when all of the following are true:

1. The specified capability has the same name space as this requirement.
2. The filter specified by the filter directive of this requirement matches the attributes of the specified capability.
3. The requirement directives and the capability directives that apply to the name space are satisfied.

**Parameters:**

capability - The capability to match to this requirement.

**Returns:**

true if the specified capability matches this this requirement. name space as this requirement and the filter for this requirement matches the attributes of the specified capability; false otherwise.
## Interface Resource

### Interface Resource

org.osgi.model.resource

#### public interface Resource

A resource is the representation of a uniquely identified and typed data. A resources can be wired together via capabilities and requirements. TODO decide on identity characteristics of a revision. Given in OSGi multiple bundles can be installed with same bsn/version this cannot be used as a key. What then is identity of a resource? Object identity? URI (needs getter method?)

#### Field Summary

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>String CAPABILITY_EFFECTIVE_DIRECTIVE</td>
<td>A capability directive used to specify the effective time for the capability.</td>
<td>47</td>
</tr>
<tr>
<td>String CAPABILITY_EXCLUDE_DIRECTIVE</td>
<td>A capability directive used to specify the comma separated list of classes which must not be allowed to be exported.</td>
<td>48</td>
</tr>
<tr>
<td>String CAPABILITY_INCLUDE_DIRECTIVE</td>
<td>A capability directive used to specify the comma separated list of classes which must be allowed to be exported.</td>
<td>48</td>
</tr>
<tr>
<td>String CAPABILITY_MANDATORY_DIRECTIVE</td>
<td>A capability directive used to specify the comma separated list of mandatory attributes which must be specified in the filter of a requirement in order for the capability to match the requirement.</td>
<td>47</td>
</tr>
<tr>
<td>String CAPABILITY_USES_DIRECTIVE</td>
<td>A capability directive used to specify the comma separated list of packages a capability uses.</td>
<td>47</td>
</tr>
<tr>
<td>String REQUIREMENT_EFFECTIVE_DIRECTIVE</td>
<td>A requirement directive used to specify the effective time for the requirement.</td>
<td>47</td>
</tr>
<tr>
<td>String REQUIREMENT_FILTER_DIRECTIVE</td>
<td>A requirement directive used to specify a capability filter.</td>
<td>46</td>
</tr>
<tr>
<td>String REQUIREMENT_RESOLUTION_DIRECTIVE</td>
<td>A requirement directive used to specify the resolution type for a requirement.</td>
<td>47</td>
</tr>
<tr>
<td>String REQUIREMENT_VISIBILITY_DIRECTIVE</td>
<td>A requirement directive used to specify the visibility type for a requirement.</td>
<td>47</td>
</tr>
<tr>
<td>String RESOURCE_BUNDLE_NAMESPACE</td>
<td>Name space for OSGi bundle resources</td>
<td>46</td>
</tr>
<tr>
<td>String RESOURCE_CONTENT_ATTRIBUTE</td>
<td>Attribute of type String used to specify the content of a resource.</td>
<td>46</td>
</tr>
<tr>
<td>String RESOURCE_NAMESPACE_ATTRIBUTE</td>
<td>Attribute of type String used to specify the resource name space.</td>
<td>46</td>
</tr>
<tr>
<td>String RESOURCE_SYMBOLIC_NAME_ATTRIBUTE</td>
<td>Attribute of type String used to specify the resource symbolic name.</td>
<td>46</td>
</tr>
<tr>
<td>String RESOURCE_VERSION_ATTRIBUTE</td>
<td>Attribute of type org.osgi.framework.Version used to specify the resource version.</td>
<td>46</td>
</tr>
</tbody>
</table>
## Method Summary

<table>
<thead>
<tr>
<th>Method Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map&lt;String, Object&gt;</td>
<td>getAttributes()&lt;br&gt;Gets the attributes associated to this resource.</td>
</tr>
<tr>
<td>List&lt;Capability&gt;</td>
<td>getCapabilities(String namespace)&lt;br&gt;Returns the capabilities declared by this resource.</td>
</tr>
<tr>
<td>List&lt;Requirement&gt;</td>
<td>getRequirements(String namespace)&lt;br&gt;Returns the requirements declared by this bundle resource.</td>
</tr>
</tbody>
</table>

## Field Detail

### RESOURCE_BUNDLE_NAMESPACE

```java
public static final String RESOURCE_BUNDLE_NAMESPACE = "osgi.bundle"
```

Name space for OSGi bundle resources

### RESOURCE_CONTENT_ATTRIBUTE

```java
public static final String RESOURCE_CONTENT_ATTRIBUTE = "content"
```

Attribute of type String used to specify the content of a resource. Typically this specifies a URI which can be used to locate the content of the resource.

### RESOURCE_SYMBOLIC_NAME_ATTRIBUTE

```java
public static final String RESOURCE_SYMBOLIC_NAME_ATTRIBUTE = "symbolic-name"
```

Attribute of type String used to specify the resource symbolic name.

### RESOURCE_VERSION_ATTRIBUTE

```java
public static final String RESOURCE_VERSION_ATTRIBUTE = "version"
```

Attribute of type org.osgi.framework.Version used to specify the resource version.

### RESOURCE_NAMESPACE_ATTRIBUTE

```java
public static final String RESOURCE_NAMESPACE_ATTRIBUTE = "namespace"
```

Attribute of type String used to specify the resource name space.

### REQUIREMENT_FILTER_DIRECTIVE

```java
public static final String REQUIREMENT_FILTER_DIRECTIVE = "filter"
```

A requirement directive used to specify a capability filter. This filter is used to match against a capability's attributes.
**REQUIREMENT_RESOLUTION_DIRECTIVE**

```java
public static final String REQUIREMENT_RESOLUTION_DIRECTIVE = "resolution"
```

A requirement directive used to specify the resolution type for a requirement. The default value is mandatory.

**See Also:**
mandatory, optional

---

**REQUIREMENT_EFFECTIVE_DIRECTIVE**

```java
public static final String REQUIREMENT_EFFECTIVE_DIRECTIVE = "effective"
```

A requirement directive used to specify the effective time for the requirement. The default value is resolve.

**See Also:**
resolve, active

---

**REQUIREMENT_VISIBILITY_DIRECTIVE**

```java
public static final String REQUIREMENT_VISIBILITY_DIRECTIVE = "visibility"
```

A requirement directive used to specify the visibility type for a requirement. The default value is private. This directive must only be used for requirements with the require bundle name space.

**See Also:**
private, reexport

---

**CAPABILITY_USES_DIRECTIVE**

```java
public static final String CAPABILITY_USES_DIRECTIVE = "uses"
```

A capability directive used to specify the comma separated list of packages a capability uses.

---

**CAPABILITY_EFFECTIVE_DIRECTIVE**

```java
public static final String CAPABILITY_EFFECTIVE_DIRECTIVE = "effective"
```

A capability directive used to specify the effective time for the capability. The default value is resolve.

**See Also:**
resolve, active

---

**CAPABILITY_MANDATORY_DIRECTIVE**

```java
public static final String CAPABILITY_MANDATORY_DIRECTIVE = "mandatory"
```
Interface Subsystem

A capability directive used to specify the comma separated list of mandatory attributes which must be specified in the filter of a requirement in order for the capability to match the requirement. This directive must only be used for capabilities with the package name space.

**CAPABILITY_INCLUDE_DIRECTIVE**

```java
public static final String CAPABILITY_INCLUDE_DIRECTIVE = "include"
```

A capability directive used to specify the comma separated list of classes which must be allowed to be exported. This directive must only be used for capabilities with the package name space.

**CAPABILITY_EXCLUDE_DIRECTIVE**

```java
public static final String CAPABILITY_EXCLUDE_DIRECTIVE = "exclude"
```

A capability directive used to specify the comma separated list of classes which must not be allowed to be exported. This directive must only be used for capabilities with the package name space.

### Method Detail

#### getCapabilities

```java
List<Capability> getCapabilities(String namespace)
```

Returns the capabilities declared by this resource.

**Parameters:**
- `namespace` - The name space of the declared capabilities to return or null to return the declared capabilities from all name spaces.

**Returns:**
- A list containing a snapshot of the declared `Capability`s, or an empty list if this resource declares no capabilities in the specified name space.

#### getRequirements

```java
List<Requirement> getRequirements(String namespace)
```

Returns the requirements declared by this bundle resource.

**Parameters:**
- `namespace` - The name space of the declared requirements to return or null to return the declared requirements from all name spaces.

**Returns:**
- A list containing a snapshot of the declared `Requirement`s, or an empty list if this resource declares no requirements in the specified name space.

#### getAttributes

```java
Map<String, Object> getAttributes()
```

Gets the attributes associated to this resource.
Returns:
The attributes associated with the resource.
public interface Wire

A wire connecting a Capability to a Requirement.

Version:
$Id: d679daeca989a5e8171f03e8d1857b5ae8850f9c $

ThreadSafe

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>getCapability()</td>
<td>50</td>
</tr>
<tr>
<td>getProvider()</td>
<td>50</td>
</tr>
<tr>
<td>getRequirement()</td>
<td>50</td>
</tr>
<tr>
<td>getRequirer()</td>
<td>51</td>
</tr>
</tbody>
</table>

Method Detail

getCapability

getCapability()

Returns the Capability for this wire.

Returns:
The Capability for this wire.

getRequirement

getRequirement()

Return the Requirement for this wire.

Returns:
The Requirement for this wire.

getProvider

getProvider()

Resource getProvider()

Return the providing resource of the capability.

Returns:
The providing resource for the capability.
Return the providing resource of the capability.

The resource returned may differ from the resource referenced by the capability.

**Returns:**
the providing resource.

getRequirer

*Resource* `getRequirer()`

Return the requiring resource of the requirement.

The resource returned may differ from the resource referenced by the requirement.

**Returns:**
the requiring resource.
Package org.osgi.service.resource

Resource Service Package Version 1.0.

See:  
Description

<table>
<thead>
<tr>
<th>Interface Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ResourceOperation</strong></td>
<td>A resource operation serves as a context for a resource processor while processing a resource. 53</td>
</tr>
<tr>
<td><strong>ResourceProcessor</strong></td>
<td>A ResourceProcessor processes resources from a specific namespace or namespaces (e.g. 57</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Enum Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ResourceOperation.Type</strong></td>
<td>The type of operation to perform on the resource. 55</td>
</tr>
</tbody>
</table>

Package org.osgi.service.resource Description

Resource Service Package Version 1.0.

Bundles wishing to use this package must list the package in the Import-Package header of the bundle's manifest. For example:

Import-Package: org.osgi.service.resource; version="[1.0,2.0)"
public interface ResourceOperation

A resource operation serves as a context for a resource processor while processing a resource.

ThreadSafe

<table>
<thead>
<tr>
<th>Nested Class Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>static enum ResourceOperation.Type</td>
<td>55</td>
</tr>
<tr>
<td>The type of operation to perform on the resource.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>void completed()</td>
<td>53</td>
</tr>
<tr>
<td>Called by the resource processor once the operation has successfully and fully completed.</td>
<td></td>
</tr>
<tr>
<td>Map&lt;String, Object&gt; getCoordination()</td>
<td>53</td>
</tr>
<tr>
<td>This resource operation is taking place within a larger context that potentially involves many other operations on other resources as part of a subsystem operation.</td>
<td></td>
</tr>
<tr>
<td>Resource getResource()</td>
<td>54</td>
</tr>
<tr>
<td>A resource operation is associated with a particular resource.</td>
<td></td>
</tr>
<tr>
<td>ResourceOperation.Type getType()</td>
<td>54</td>
</tr>
<tr>
<td>A resource operation is associated with a type that defines the processing required on the resource.</td>
<td></td>
</tr>
</tbody>
</table>

| Method Detail |

**completed**

void completed()

Called by the resource processor once the operation has successfully and fully completed. This would typically be called as the very last thing within the Participant.ended method. Calling this method will result in an appropriate event notification that the resource has been successfully processed.

**getCoordination**

org.osgi.service.coordinator.Coordination getCoordination()

This resource operation is taking place within a larger context that potentially involves many other operations on other resources as part of a subsystem operation. Resource processors are required to participate within the coordination controlling the overall process in which this resource operation plays a part.

Returns: The coordination in which the resource processor must participate.
getResource

Resource getResource()

A resource operation is associated with a particular resource.

Returns:
The resource on which this operation must be performed.

getContext

Map<String, Object> getContext()

getType

ResourceOperation.Type getType()

A resource operation is associated with a type that defines the processing required on the resource.

Returns:
The type of resource operation.
Enum ResourceOperation.Type

org.osgi.service.resource

java.lang.Object
    └ java.lang.Enum<ResourceOperation.Type>
        └ org.osgi.service.resource.ResourceOperation.Type

All Implemented Interfaces:
    Comparable<ResourceOperation.Type>, Serializable

Enclosing class:
    ResourceOperation

class ResourceOperation.

public static enum ResourceOperation.Type
    extends Enum<ResourceOperation.Type>

The type of operation to perform on the resource.

### Enum Constant Summary

<table>
<thead>
<tr>
<th>Enum Constant</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSTALL</td>
<td>Install the resource.</td>
<td>55</td>
</tr>
<tr>
<td>START</td>
<td>Start the resource.</td>
<td>55</td>
</tr>
<tr>
<td>STOP</td>
<td>Stop the resource.</td>
<td>56</td>
</tr>
<tr>
<td>UNINSTALL</td>
<td>Uninstall the resource.</td>
<td>56</td>
</tr>
<tr>
<td>UPDATE</td>
<td>Update the resource.</td>
<td>56</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>static</td>
<td>valueOf(String name)</td>
<td>56</td>
</tr>
<tr>
<td>static</td>
<td>values()</td>
<td>56</td>
</tr>
</tbody>
</table>

### Enum Constant Detail

**INSTALL**

public static final ResourceOperation.Type INSTALL

    Install the resource.

**START**

public static final ResourceOperation.Type START

    Start the resource.
STOP

```java
public static final ResourceOperation.Type STOP
```

Stop the resource.

UNINSTALL

```java
public static final ResourceOperation.Type UNINSTALL
```

Uninstall the resource.

UPDATE

```java
public static final ResourceOperation.Type UPDATE
```

Update the resource.

Method Detail

values

```java
public static ResourceOperation.Type[] values()
```

valueOf

```java
public static ResourceOperation.Type valueOf(String name)
```
A ResourceProcessor processes resources from a specific namespace or namespaces (e.g. bundle). Namespaces not defined by the OSGi Alliance should use a reverse domain name scheme to avoid collision (e.g. com.acme.config).

ResourceProcessors are registered in the OSGi Service Registry. They advertise the namespaces they support using the service property osgi.resource.namespace. The type of this property is a String+

A resource processor performs the operation corresponding to those provided by Subsystem that affect a subsystem's life-cycle (e.g. install and start). For example, Subsystem.install() would delegate to on a resource processor if there were any resources to install for the namespace that the resource processor supported.

### ThreadSafe

<table>
<thead>
<tr>
<th>Method Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>void process(ResourceOperation operation)</strong></td>
<td>57</td>
</tr>
</tbody>
</table>

Process a resource according to the provided operation. The type of processing to perform is defined by the type of ResourceOperation. The resource processor must register as a Participant with the provided Coordination and should process the resource according to the notifications given to the Participant. For example, if the resource processor participant is told the Coordination failed, then the resource processor should not perform the operation at all or should undo any processing that has already been performed. If the resource processor is unable to undo previous work, then it must throw an exception back to the Coordinator service from the Participant.failed method. Must call ResourceOperation.completed in Participant.ended.

**Throws:** Exception

```java
void process(ResourceOperation operation)
throws Exception
```

Process a resource according to the provided operation. The type of processing to perform is defined by the type of ResourceOperation. The resource processor must register as a Participant with the provided Coordination and should process the resource according to the notifications given to the Participant. For example, if the resource processor participant is told the Coordination failed, then the resource processor should not perform the operation at all or should undo any processing that has already been performed. If the resource processor is unable to undo previous work, then it must throw an exception back to the Coordinator service from the Participant.failed method. Must call ResourceOperation.completed in Participant.ended.

**Throws:** Exception

```java
void process(ResourceOperation operation)
throws Exception
```
## Package org.osgi.service.subsystem

### Interface Summary

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Subsystem</strong></td>
<td>A representation of a subsystem in the framework.</td>
<td>59</td>
</tr>
<tr>
<td><strong>SubsystemConstants</strong></td>
<td>Defines the constants used by subsystems.</td>
<td>68</td>
</tr>
</tbody>
</table>

### Enum Summary

<table>
<thead>
<tr>
<th>Enum</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Subsystem.State</strong></td>
<td>The states of a subsystem in the framework.</td>
<td>66</td>
</tr>
<tr>
<td><strong>SubsystemConstants.EventType</strong></td>
<td>The subsystem life-cycle event types that can be produced by a subsystem.</td>
<td>79</td>
</tr>
</tbody>
</table>

### Exception Summary

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SubsystemException</strong></td>
<td>Exception thrown by Subsystem when a problem occurs.</td>
<td>83</td>
</tr>
</tbody>
</table>
Interface Subsystem

`org.osgi.service.subsystem`

```java
public interface Subsystem
```

A representation of a subsystem in the framework. A subsystem is a collection of bundles and/or other resource. A subsystem has isolation semantics. Subsystem types are defined that have different default isolation semantics. For example, an Application subsystem does not export any of the packages or services provided by its content bundles, and imports any packages or services that are required to satisfy unresolved package or service dependencies of the content bundles. A subsystem is defined using a manifest format.

**ThreadSafe**

### Nested Class Summary

<table>
<thead>
<tr>
<th>Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Subsystem.State</code></td>
<td>66</td>
</tr>
</tbody>
</table>

**Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getChildren()</code></td>
<td>Gets the subsystems managed by this service.</td>
</tr>
<tr>
<td><code>getConstituents()</code></td>
<td>Gets the content bundles of this subsystem.</td>
</tr>
<tr>
<td><code>getHeaders()</code></td>
<td>Gets the headers used to define this subsystem.</td>
</tr>
<tr>
<td><code>getHeaders(String locale)</code></td>
<td>Gets the headers used to define this subsystem.</td>
</tr>
<tr>
<td><code>getLocation()</code></td>
<td>The location identifier used to install this subsystem through Subsystem.install.</td>
</tr>
<tr>
<td><code>getParent()</code></td>
<td>Gets the parent Subsystem that scopes this subsystem instance.</td>
</tr>
<tr>
<td><code>getState()</code></td>
<td>Gets the state of the subsystem.</td>
</tr>
<tr>
<td><code>getSubsystemId()</code></td>
<td>Gets the identifier of the subsystem.</td>
</tr>
<tr>
<td><code>getSymbolicName()</code></td>
<td>Gets the symbolic name of this subsystem.</td>
</tr>
<tr>
<td><code>getVersion()</code></td>
<td>Gets the version of this subsystem.</td>
</tr>
<tr>
<td><code>install(String location)</code></td>
<td>Install a new subsystem from the specified location identifier.</td>
</tr>
<tr>
<td><code>install(String location, InputStream content)</code></td>
<td>Install a new subsystem from the specified InputStream object.</td>
</tr>
<tr>
<td><code>start()</code></td>
<td>Starts the subsystem.</td>
</tr>
<tr>
<td><code>stop()</code></td>
<td>Stops the subsystem.</td>
</tr>
<tr>
<td><code>uninstall()</code></td>
<td>Uninstall the given subsystem.</td>
</tr>
</tbody>
</table>
## Method Detail

### getChildren

```java
collection<Subsystem> getChildren()
```

Gets the subsystems managed by this service. This only includes the top-level Subsystems installed in the Framework, CompositeBundle or Subsystem from which this service has been retrieved.

**Returns:**
- The Subsystems managed by this service.

### getConstituents

```java
collection<org.osgi.framework.Bundle> getConstituents()
```

Gets the content bundles of this subsystem.

**Returns:**
- The content of this subsystem.

### getHeaders

```java
map<String,String> getHeaders()
```

Gets the headers used to define this subsystem. The headers will be localized using the locale returned by java.util.Locale.getDefault. This is equivalent to calling getHeaders(null).

**Returns:**
- The headers used to define this subsystem.

**Throws:**
- `SecurityException` - If the caller does not have the appropriate AdminPermission[this,METADATA] and the runtime supports permissions.

### getHeaders

```java
map<String,String> getHeaders(String locale)
```

Gets the headers used to define this subsystem.

**Parameters:**
- `locale` - The locale name to be used to localize the headers. If the locale is null then the locale returned by java.util.Locale.getDefault is used. If the value is the empty string then the returned headers are returned unlocalized.

**Returns:**
- the headers used to define this subsystem, localized to the specified locale.
Interface Requirement

getLocation

String getLocation()

The location identifier used to install this subsystem through Subsystem.install. This identifier does not change while this subsystem remains installed, even after Subsystem.update. This location identifier is used in Subsystem.update if no other update source is specified.

Returns:
   The string representation of the subsystem's location identifier.

getParent

Subsystem getParent()

Gets the parent Subsystem that scopes this subsystem instance.

Returns:
   The Subsystem that scopes this subsystem or null if there is no parent subsystem (e.g. if the outer scope is the framework).

getState

Subsystem.State getState()

Gets the state of the subsystem.

Returns:
   The state of the subsystem.

getSubsystemId

long getSubsystemId()

Gets the identifier of the subsystem. Subsystem identifiers are assigned when the subsystem is installed and are unique within the framework.

Returns:
   The identifier of the subsystem.

getSymbolicName

String getSymbolicName()

Gets the symbolic name of this subsystem.

Returns:
   The symbolic name of this subsystem.
### Interface Requirement

**getVersion**

`org.osgi.framework.Version getVersion()`

Gets the version of this subsystem.

**Returns:**

The version of this subsystem.

---

**install**

`Subsystem install(String location)`

throws `SubsystemException`

Install a new subsystem from the specified location identifier.

This method performs the same function as calling `install(String, InputStream)` with the specified location identifier and a null InputStream.

**Parameters:**

- `location` - The location identifier of the subsystem to be installed.

**Returns:**

The installed subsystem.

**Throws:**

- `SubsystemException` - If the subsystem could not be installed for any reason.
- `SecurityException` - If the caller does not have the appropriate AdminPermission[installed subsystem,LIFECYCLE], and the Java Runtime Environment supports permissions.

---

**install**

`Subsystem install(String location, InputStream content)`

throws `SubsystemException`

Install a new subsystem from the specified InputStream object.

If the specified InputStream is null, the InputStream must be created from the specified location.

The specified location identifier will be used as the identity of the subsystem. Every installed subsystem is uniquely identified by its location identifier which is typically in the form of a URL.

TODO: Understand whether this all change when we can install the same bundle multiple times.

A subsystem and its contents must remain installed across Framework and VM restarts. The subsystem itself is installed atomically, however its contents are not.

The following steps are required to install a subsystem:

1. If there is an existing subsystem containing the same location identifier as the Subsystem to be installed, then a Future is returned that has the existing subsystem immediately available as its result.
2. If there is already an install in progress for a subsystem with the same location identifier, then the Future returned is the same as the Future returned for the first install and a new install is not started.
3. If this is a new install, then a new Future is returned with the installation process following the remaining step.
4. The subsystem content is read from the input stream.
5. Isolation is set up while the install is in progress, such that none of the content can be
resolved with bundles outside the subsystem.
6. The resources are into the framework through the use of ResourceProcessors.
7. Isolation is configured appropriate for the subsystem such that the content can be
resolved with bundles outside the subsystem.
8. The subsystem's state is set to INSTALLED.
9. The subsystem event of type INSTALLED is fired.
10. The subsystem content is started.
11. The subsystem object for the newly installed subsystem is made available from the
Future.

Parameters:
- `location` - The location identifier of the subsystem to be installed.
- `content` - The InputStream from where the subsystem is to be installed or null if the location is to
be used to create the InputStream.

Returns: The installed subsystem.

Throws:
- `SubsystemException` - If the subsystem could not be installed for any reason.
- `SecurityException` - If the caller does not have the appropriate AdminPermission[installed
subsystem,LIFECYCLE], and the Java Runtime Environment supports permissions.

---

start

```java
void start()
throws SubsystemException
```

Starts the subsystem. The subsystem is started according to the rules defined for Bundles and the content
bundles are enabled for activation.

Throws:
- `SubsystemException` - If this subsystem could not be started.
- `IllegalStateException` - If this subsystem has been uninstalled.
- `SecurityException` - If the caller does not have the appropriate AdminPermission[this,EXECUTE]
and the runtime supports permissions.

---

stop

```java
void stop()
throws SubsystemException
```

 Stops the subsystem. The subsystem is stopped according to the rules defined for Bundles and the content
bundles are disabled for activation and stopped.

Throws:
- `SubsystemException` - If an internal exception is thrown while stopping the subsystem (e.g. a
BundleException from Bundle.stop).
- `IllegalStateException` - If this subsystem has been uninstalled.
- `SecurityException` - If the caller does not have the appropriate AdminPermission[this,EXECUTE]
and the runtime supports permissions.

---

uninstall

```java
void uninstall()
throws SubsystemException
```
Uninstall the given subsystem.

This method causes the Framework to notify other bundles and subsystems that this subsystem is being uninstalled, and then puts this subsystem into the UNINSTALLED state. The Framework must remove any resources related to this subsystem that it is able to remove. It does so using the appropriate ResourceProcessor.uninstall(Subsystem, Resource, Coordination) for the resource namespace. If this subsystem has exported any packages, the Framework must continue to make these packages available to their importing bundles or subsystems until the org.osgi.service.packageadmin.PackageAdmin.refreshPackages( org.osgi.framework.Bundle[]) method has been called or the Framework is relaunched. The following steps are required to uninstall a subsystem:

1. If this subsystem's state is UNINSTALLED then an IllegalStateException is thrown.
2. If this subsystem's state is ACTIVE, STARTING or STOPPING, this subsystem is stopped as described in the Subsystem.stop() method. If Subsystem.stop() throws an exception, a Framework event of type FrameworkEvent.ERROR is fired containing the exception.
3. This subsystem's state is set to UNINSTALLED.
4. A subsystem event of type SubsystemEvent.UNINSTALLED is fired.
5. This subsystem and any persistent storage area provided for this subsystem by the Framework are removed.

Throws:

- SubsystemException - If the uninstall failed.
- IllegalStateException - If the subsystem is already in the UNINSTALLED state.
- SecurityException - If the caller does not have the appropriate AdminPermission[this,LIFECYCLE] and the Java Runtime Environment supports permissions.

---

void update()
throws SubsystemException

Update the given subsystem.

This method performs the same function as calling update(Subsystem, InputStream) with the specified subsystem and a null InputStream.

Throws:

- SubsystemException - If the subsystem could not be updated for any reason.

---

void update(InputStream content)
throws SubsystemException

Update the given subsystem from an InputStream.

If the specified InputStream is null, the InputStream must be created from the subsystem's Subsystem-UpdateLocation Manifest header if present, or this subsystem's location provided when the subsystem was originally installed.

TODO: expand on this description. For example, we need details on how update works for individual resources. We could follow the deploymentadmin approach and uninstall bundles that are removed and install new ones. This would happen if we had a different (updated) deployment calculated for the same version of the application.
Interface Requirement

Parameters:

- content: The InputStream from which to update the subsystem or null if the Subsystem-UpdateLocation or original location are to be used.

Throws:

- SubsystemException
- IllegalStateException
- SecurityException - If the subsystem is in the UNINSTALLED state.
  If the caller does not have the appropriate AdminPermission[this,LIFECYCLE] for both the current subsystem and the updated subsystem and the Java Runtime Environment supports permissions.
interface Resource

public static enum Subsystem.State
extends Enum<Subsystem.State>

The states of a subsystem in the framework. These states match those of a Bundle and are derived using the same rules as CompositeBundles. As such, they are more a reflection of what content bundles are permitted to do rather than an aggregation of the content bundle states.

Enum Constant Summary

<table>
<thead>
<tr>
<th>Enum</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTIVE</td>
<td>66</td>
</tr>
<tr>
<td>INSTALLED</td>
<td>67</td>
</tr>
<tr>
<td>RESOLVED</td>
<td>67</td>
</tr>
<tr>
<td>STARTING</td>
<td>67</td>
</tr>
<tr>
<td>STOPPING</td>
<td>67</td>
</tr>
<tr>
<td>UNINSTALLABLE</td>
<td>67</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>static Subsystem.State.valueOf(String name)</td>
<td>67</td>
</tr>
<tr>
<td>static Subsystem.State.values()</td>
<td>67</td>
</tr>
</tbody>
</table>

Enum Constant Detail

ACTIVE

public static final Subsystem.State ACTIVE
A subsystem is in the ACTIVE state when it has reached the beginning start-level (for starting it's contents), and all its persistently started content bundles that are resolved and have had their start-levels met have completed, or failed, their activator start method.

**INSTALLED**

public static final Subsystem.State INSTALLED

A subsystem is in the INSTALLED state when it is initially created.

**RESOLVED**

public static final Subsystem.State RESOLVED

A subsystem in the RESOLVED is allowed to have its content bundles resolved.

**STARTING**

public static final Subsystem.State STARTING

A subsystem is in the STARTING state when all its content bundles are enabled for activation.

**STOPPING**

public static final Subsystem.State STOPPING

A subsystem in the STOPPING state is in the process of taking its its active start level to zero, stopping all the content bundles.

**UNINSTALLED**

public static final Subsystem.State UNINSTALLED

A subsystem is in the UNINSTALLED state when all its content bundles and uninstalled and its system bundle context is invalidated.

**Method Detail**

**values**

public static Subsystem.State[] values()

**valueOf**

public static Subsystem.State valueOf(String name)
public interface SubsystemConstants

Defines the constants used by subsystems. These fall into four categories:

1. Headers for the applications subsystem type.
2. Headers for the composite bundle subsystem type. Where appropriate, these header values are identical to those for a bundle.
3. Headers for the library subsystem type.
4. Attributes used in the above headers

### Nested Class Summary

<table>
<thead>
<tr>
<th>Enum Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SubsystemConstants.EventType</td>
<td>The subsystem lifecycle event types that can be produced by a subsystem.</td>
<td>79</td>
</tr>
</tbody>
</table>

### Field Summary

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPLICATION_CONTENT</td>
<td>The list of application contents identified by a symbolic name and version.</td>
<td>70</td>
</tr>
<tr>
<td>APPLICATION_DESCRIPTION</td>
<td>Human readable description.</td>
<td>71</td>
</tr>
<tr>
<td>APPLICATION_LOCALIZATION</td>
<td>Name of the resource to use for localized headers.</td>
<td>71</td>
</tr>
<tr>
<td>APPLICATION_MANIFESTVERSION</td>
<td>The subsystem manifest version header must be present and equals to 1.0 for this version of applications.</td>
<td>71</td>
</tr>
<tr>
<td>APPLICATION_NAME</td>
<td>Human readable application name.</td>
<td>71</td>
</tr>
<tr>
<td>APPLICATION_RESOURCES</td>
<td>Descriptions of resources contained within the application archive which cannot be determine reflectively.</td>
<td>71</td>
</tr>
<tr>
<td>APPLICATION_SYMBOLICNAME</td>
<td>Symbolic name for the application.</td>
<td>71</td>
</tr>
<tr>
<td>APPLICATION_UPDATELOCATION</td>
<td>Location to use when updating the application.</td>
<td>71</td>
</tr>
<tr>
<td>APPLICATION_VERSION</td>
<td>Version of the application.</td>
<td>71</td>
</tr>
<tr>
<td>BUNDLE_DESCRIPTION</td>
<td>The human readable composite bundle description (re-using the bundle header).</td>
<td>72</td>
</tr>
<tr>
<td>BUNDLE_EXPORTPACKAGE</td>
<td>The packages to be exported by the composite bundle for use outside the composite bundle (re-using the bundle header).</td>
<td>72</td>
</tr>
<tr>
<td>BUNDLE_IMPORTPACKAGE</td>
<td>The packages to be imported into the composite bundle for use by the composite bundle contents (re-using the bundle header).</td>
<td>72</td>
</tr>
<tr>
<td>BUNDLE_LOCALIZATION</td>
<td>Name of the resource to use for localized headers (re-using the bundle header).</td>
<td>72</td>
</tr>
<tr>
<td>String</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td><strong>BUNDLE_NAME</strong></td>
<td>The human readable composite bundle name (re-using the bundle header).</td>
<td></td>
</tr>
<tr>
<td><strong>BUNDLE_REQUIREBUNDLE</strong></td>
<td>A list of Bundles the composite bundle requires (re-using the bundle header).</td>
<td></td>
</tr>
<tr>
<td><strong>BUNDLE_SYMBOLICNAME</strong></td>
<td>Symbolic name for the composite bundle (re-using the bundle header).</td>
<td></td>
</tr>
<tr>
<td><strong>BUNDLE_UPDATELOCATION</strong></td>
<td>Location to use when updating the composite bundle (re-using the bundle header).</td>
<td></td>
</tr>
<tr>
<td><strong>BUNDLE_VERSION</strong></td>
<td>Version of the composite bundle (re-using the bundle header).</td>
<td></td>
</tr>
<tr>
<td><strong>COMPOSITEBUNDLECONTENT</strong></td>
<td>The list of composite bundle contents identified by a symbolic name and version.</td>
<td></td>
</tr>
<tr>
<td><strong>COMPOSITEBUNDLEEXPORTSERVICE</strong></td>
<td>A list of service filters used to identify the services provided by bundles inside the composite bundle that can be exported outside the composite bundle.</td>
<td></td>
</tr>
<tr>
<td><strong>COMPOSITEBUNDLEIMPORTSERVICE</strong></td>
<td>A list of service filters used to identify services that this composite bundle requires.</td>
<td></td>
</tr>
<tr>
<td><strong>COMPOSITEBUNDLEMANIFESTVERSION</strong></td>
<td>The composite bundle manifest version header must be present and equal to 1.0 for this version of composite bundles.</td>
<td></td>
</tr>
<tr>
<td><strong>COMPOSITEBUNDLERESOURCES</strong></td>
<td>Descriptions of resources contained within the composite bundle archive which cannot be determined reflectively.</td>
<td></td>
</tr>
<tr>
<td><strong>FEATURE_CONTENT</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>FEATURE_DESCRIPTION</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>FEATURE_MANIFESTVERSION</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>FEATURE_NAME</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>FEATURE_SYMBOLICNAME</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>FEATURE_VERSION</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>LIBRARYCONTENT</strong></td>
<td>The list of library contents identified by a symbolic name and version.</td>
<td></td>
</tr>
<tr>
<td><strong>LIBRARYDESCRIPTION</strong></td>
<td>Human readable library description.</td>
<td></td>
</tr>
<tr>
<td><strong>LIBRARYLOCALIZATION</strong></td>
<td>Name of the resource to use for localized headers.</td>
<td></td>
</tr>
<tr>
<td><strong>LIBRARYMANIFESTVERSION</strong></td>
<td>The subsystem manifest version header must be present and equals to 1.0 for this version of applications.</td>
<td></td>
</tr>
<tr>
<td><strong>LIBRARYNAME</strong></td>
<td>Human readable library name.</td>
<td></td>
</tr>
<tr>
<td><strong>LIBRARYRESOURCES</strong></td>
<td>Descriptions of resources contained within the library archive which cannot be determined reflectively.</td>
<td></td>
</tr>
<tr>
<td><strong>LIBRARYSYMBOLICNAME</strong></td>
<td>Symbolic name for the application.</td>
<td></td>
</tr>
<tr>
<td><strong>LIBRARYUPDATELOCATION</strong></td>
<td>Location to use when updating the library.</td>
<td></td>
</tr>
<tr>
<td><strong>LIBRARYVERSION</strong></td>
<td>Version of the application.</td>
<td></td>
</tr>
<tr>
<td><strong>OSGIRESOURCENAMESPACE</strong></td>
<td>Service property to be set on ResourceProcessor services to indicate which namespaces of resource it can process.</td>
<td></td>
</tr>
</tbody>
</table>
## Interface Resource

<table>
<thead>
<tr>
<th>String</th>
<th>RESOURCE_LOCATION_ATTRIBUTE</th>
<th>Attribute to indicate the location of a resource.</th>
<th>75</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>RESOURCE_NAMESPACE_ATTRIBUTE</td>
<td>Attribute to indicate the namespace of a resource.</td>
<td>75</td>
</tr>
<tr>
<td>String</td>
<td>RESOURCE_NAMESPACE_BUNDLE</td>
<td>Identify resources that are bundles (this is the default type).</td>
<td>76</td>
</tr>
<tr>
<td>String</td>
<td>RESOURCE_NAMESPACE_SUBSYSTEM</td>
<td>Identify resources that are subsystems.</td>
<td>76</td>
</tr>
<tr>
<td>String</td>
<td>RESOURCE_START_ATTRIBUTE</td>
<td>Attribute to indicate a bundle needs to be started, defaults to true.</td>
<td>76</td>
</tr>
<tr>
<td>String</td>
<td>RESOURCE_START_LEVEL_ATTRIBUTE</td>
<td>Attribute to indicate the start level that must be associated to a constituent bundle or subsystem.</td>
<td>76</td>
</tr>
<tr>
<td>String</td>
<td>RESOURCE_TYPE_ATTRIBUTE</td>
<td>Attribute to indicate the type on a resource.</td>
<td>76</td>
</tr>
<tr>
<td>String</td>
<td>RESOURCE_TYPE_BUNDLE</td>
<td>Identify resources that are bundles (this is the default type).</td>
<td>76</td>
</tr>
<tr>
<td>String</td>
<td>RESOURCE_TYPE_SUBSYSTEM</td>
<td>Identify resources that are subsystems.</td>
<td>76</td>
</tr>
<tr>
<td>String</td>
<td>RESOURCE_UPDATE_ATTRIBUTE</td>
<td>Attribute to indicate a bundle needs to be forced updated, even if the version is the same, defaults to false</td>
<td>77</td>
</tr>
<tr>
<td>String</td>
<td>SERVICE_RESOURCE_TYPE</td>
<td>Service property to be set on ResourceProcessor services to indicate which types of resource it can process.</td>
<td>77</td>
</tr>
<tr>
<td>String</td>
<td>SUBSYSTEM_DIRECTIVE</td>
<td>Directive on the manifest header of a composite indicating the composite is managed as a subsystem.</td>
<td>77</td>
</tr>
<tr>
<td>String</td>
<td>SUBSYSTEM_ID</td>
<td>Key for the event property that holds the subsystem id.</td>
<td>77</td>
</tr>
<tr>
<td>String</td>
<td>SUBSYSTEM_LOCATION</td>
<td>Key for the event property that holds the subsystem location.</td>
<td>77</td>
</tr>
<tr>
<td>String</td>
<td>SUBSYSTEM_SYMBOLICNAME</td>
<td>Key for the event property that holds the subsystem symbolic name.</td>
<td>77</td>
</tr>
<tr>
<td>String</td>
<td>SUBSYSTEM_VERSION</td>
<td>Key for the event property that holds the subsystem version.</td>
<td>77</td>
</tr>
<tr>
<td>String</td>
<td>TOPIC</td>
<td>The topic for subsystem event admin events.</td>
<td>77</td>
</tr>
<tr>
<td>String</td>
<td>TOPIC_INTERNALS</td>
<td>The topic for subsystem internal event admin events.</td>
<td>78</td>
</tr>
</tbody>
</table>

### Field Detail

**APPLICATION_CONTENT**

```java
public static final String APPLICATION_CONTENT = ""
```

The list of application contents identified by a symbolic name and version.
APPLICATION_DESCRIPTION
public static final String APPLICATION_DESCRIPTION = ""

Human readable description.

APPLICATION_LOCALIZATION
public static final String APPLICATION_LOCALIZATION = ""

Name of the resource to use for localized headers.

APPLICATION_MANIFESTVERSION
public static final String APPLICATION_MANIFESTVERSION = ""

The subsystem manifest version header must be present and equals to 1.0 for this version of applications.

APPLICATION_NAME
public static final String APPLICATION_NAME = ""

Human readable application name.

APPLICATION_RESOURCES
public static final String APPLICATION_RESOURCES = ""

Descriptions of resources contained within the application archive which cannot be determine reflectively.

APPLICATION_SYMBOLICNAME
public static final String APPLICATION_SYMBOLICNAME = ""

Symbolic name for the application. Must be present.

APPLICATION_UPDATELOCATION
public static final String APPLICATION_UPDATELOCATION = ""

Location to use when updating the application.

APPLICATION_VERSION
public static final String APPLICATION_VERSION = ""

Version of the application. If not present, the default value is 0.0.0.
BUNDLE_DESCRIPTION

public static final String BUNDLE_DESCRIPTION = ""

The human readable composite bundle description (re-using the bundle header).

BUNDLE_EXPORTPACKAGE

public static final String BUNDLE_EXPORTPACKAGE = ""

The packages to be exported by the composite bundle for use outside the composite bundle (re-using the bundle header). The packages declarations must match a package provide by a bundle listed in the composite bundle content.

BUNDLE_IMPORTPACKAGE

public static final String BUNDLE_IMPORTPACKAGE = ""

The packages to be imported into the composite bundle for use by the composite bundle contents (re-using the bundle header).

BUNDLE_LOCALIZATION

public static final String BUNDLE_LOCALIZATION = ""

Name of the resource to use for localized headers (re-using the bundle header).

BUNDLE_NAME

public static final String BUNDLE_NAME = ""

The human readable composite bundle name (re-using the bundle header).

BUNDLE_REQUIREBUNDLE

public static final String BUNDLE_REQUIREBUNDLE = ""

A list of Bundles the composite bundle requires (re-using the bundle header). These bundles are made available to satisfy Require-Bundle statements for the composite bundle content bundles.

BUNDLE_SYMBOLICNAME

public static final String BUNDLE_SYMBOLICNAME = ""

Symbolic name for the composite bundle (re-using the bundle header). Must be present.
**BUNDLE_UPDATELOCATION**

public static final String **BUNDLE_UPDATELOCATION** = ""

Location to use when updating the composite bundle (re-using the bundle header).

**BUNDLE_VERSION**

public static final String **BUNDLE_VERSION** = ""

Version of the composite bundle (re-using the bundle header). If not present, the default value is 0.0.0.

**COMPOSITEBUNDLE_CONTENT**

public static final String **COMPOSITEBUNDLE_CONTENT** = ""

The list of composite bundle contents identified by a symbolic name and version.

**COMPOSITEBUNDLE_EXPORTSERVICE**

public static final String **COMPOSITEBUNDLE_EXPORTSERVICE** = ""

A list of service filters used to identify the services provided by bundles inside the composite bundle that can be exported outside the composite bundle.

**COMPOSITEBUNDLE_IMPORTSERVICE**

public static final String **COMPOSITEBUNDLE_IMPORTSERVICE** = ""

A list of service filters used to identify services that this composite bundle requires.

**COMPOSITEBUNDLE_MANIFESTVERSION**

public static final String **COMPOSITEBUNDLE_MANIFESTVERSION** = ""

The composite bundle manifest version header must be present and equal to 1.0 for this version of composite bundles.

**COMPOSITEBUNDLE_RESOURCES**

public static final String **COMPOSITEBUNDLE_RESOURCES** = ""

Descriptions of resources contained within the composite bundle archive which cannot be determine reflectively.

**FEATURE_CONTENT**

public static final String **FEATURE_CONTENT** = "Feature-Content"
FEATURE_DESCRIPTION
public static final String FEATURE_DESCRIPTION = "Feature-Description"

FEATURE_MANIFESTVERSION
public static final String FEATURE_MANIFESTVERSION = "Feature-ManifestVersion"

FEATURE_NAME
public static final String FEATURE_NAME = "Feature-Name"

FEATURE_SYMBOLICNAME
public static final String FEATURE_SYMBOLICNAME = "Feature-SymbolicName"

FEATURE_VERSION
public static final String FEATURE_VERSION = "Feature-Version"

LIBRARY_CONTENT
public static final String LIBRARY_CONTENT = ""

   The list of library contents identified by a symbolic name and version.

LIBRARY_DESCRIPTION
public static final String LIBRARY_DESCRIPTION = ""

   Human readable library description.

LIBRARY_LOCALIZATION
public static final String LIBRARY_LOCALIZATION = ""

   Name of the resource to use for localized headers.

LIBRARY_MANIFESTVERSION
public static final String LIBRARY_MANIFESTVERSION = ""

   The subsystem manifest version header must be present and equals to 1.0 for this version of applications.
**LIBRARY_NAME**

```java
public static final String LIBRARY_NAME = ""
```

Human readable library name.

**LIBRARY_RESOURCES**

```java
public static final String LIBRARY_RESOURCES = ""
```

Descriptions of resources contained within the library archive which cannot be determined reflectively.

**LIBRARY_SYMBOLICNAME**

```java
public static final String LIBRARY_SYMBOLICNAME = ""
```

Symbolic name for the application. Must be present.

**LIBRARY_UPDATELOCATION**

```java
public static final String LIBRARY_UPDATELOCATION = ""
```

Location to use when updating the library.

**LIBRARY_VERSION**

```java
public static final String LIBRARY_VERSION = ""
```

Version of the application. If not present, the default value is 0.0.0.

**OSGi RESOURCE_NAMESPACE**

```java
public static final String OSGI_RESOURCE_NAMESPACE = ""
```

Service property to be set on ResourceProcessor services to indicate which namespaces of resource it can process.

**RESOURCE_LOCATION_ATTRIBUTE**

```java
public static final String RESOURCE_LOCATION_ATTRIBUTE = ""
```

Attribute to indicate the location of a resource.

**RESOURCE_NAMESPACE_ATTRIBUTE**

```java
public static final String RESOURCE_NAMESPACE_ATTRIBUTE = "osgi.resource.namespace"
```

Interface Resource

Attribute to indicate the namespace of a resource.

RESOURCE_NAMESPACE_BUNDLE

public static final String RESOURCE_NAMESPACE_BUNDLE = ""

Identify resources that are bundles (this is the default type).

RESOURCE_NAMESPACE_SUBSYSTEM

public static final String RESOURCE_NAMESPACE_SUBSYSTEM = ""

Identify resources that are subsystems.

RESOURCE_START_ATTRIBUTE

public static final String RESOURCE_START_ATTRIBUTE = ""

Attribute to indicate a bundle needs to be started, defaults to true.

RESOURCE_START_LEVEL_ATTRIBUTE

public static final String RESOURCE_START_LEVEL_ATTRIBUTE = ""

Attribute to indicate the start level that must be associated to a constituent bundle or subsystem.

RESOURCE_TYPE_ATTRIBUTE

public static final String RESOURCE_TYPE_ATTRIBUTE = ""

Attribute to indicate the type on a resource.

RESOURCE_TYPE_BUNDLE

public static final String RESOURCE_TYPE_BUNDLE = ""

Identify resources that are bundles (this is the default type).

RESOURCE_TYPE_SUBSYSTEM

public static final String RESOURCE_TYPE_SUBSYSTEM = ""

Identify resources that are subsystems.
**RESOURCE_UPDATE_ATTRIBUTE**

```java
public static final String RESOURCE_UPDATE_ATTRIBUTE = ""
```

Attribute to indicate a bundle needs to be forced updated, even if the version is the same, defaults to false.

**SERVICE_RESOURCE_TYPE**

```java
public static final String SERVICE_RESOURCE_TYPE = ""
```

Service property to be set on ResourceProcessor services to indicate which types of resource it can process.

**SUBSYSTEM_DIRECTIVE**

```java
public static final String SUBSYSTEM_DIRECTIVE = ""
```

Directive on the manifest header of a composite indicating the composite is managed as a subsystem.

**SUBSYSTEM_ID**

```java
public static final String SUBSYSTEM_ID = "subsystem.id"
```

Key for the event property that holds the subsystem id.

**SUBSYSTEM_LOCATION**

```java
public static final String SUBSYSTEM_LOCATION = "subsystem.location"
```

Key for the event property that holds the subsystem location.

**SUBSYSTEM_SYMBOLICNAME**

```java
public static final String SUBSYSTEM_SYMBOLICNAME = "subsystem.symbolicname"
```

Key for the event property that holds the subsystem symbolic name.

**SUBSYSTEM_VERSION**

```java
public static final String SUBSYSTEM_VERSION = "subsystem.version"
```

Key for the event property that holds the subsystem version.

**TOPIC**

```java
public static final String TOPIC = "org/osgi/service/Subsystem"
```
The topic for subsystem event admin events.

---

**TOPIC_INTERNALS**

```java
public static final String TOPIC_INTERNALS = "org/osgi/service/SubsystemInternals"
```

The topic for subsystem internal event admin events.
The subsystem lifecycle event types that can be produced by a subsystem. See ? and Subsystem for details on the circumstances under which these events are fired.

<table>
<thead>
<tr>
<th>Enum Constant Summary</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANCELED</td>
<td>80</td>
</tr>
<tr>
<td>Event type used to indicate that the operations was cancelled (e.g.</td>
<td></td>
</tr>
<tr>
<td>CANCELING</td>
<td>80</td>
</tr>
<tr>
<td>Event type used to indicate that a subsystem operation is being cancelled.</td>
<td></td>
</tr>
<tr>
<td>FAILED</td>
<td>80</td>
</tr>
<tr>
<td>Event type used to indicate that the operation failed (e.g.</td>
<td></td>
</tr>
<tr>
<td>INSTALLED</td>
<td>80</td>
</tr>
<tr>
<td>Event type used to indicate a subsystem has been installed.</td>
<td></td>
</tr>
<tr>
<td>INSTALLING</td>
<td>80</td>
</tr>
<tr>
<td>Event type used to indicate a subsystem is installing.</td>
<td></td>
</tr>
<tr>
<td>RESOLVED</td>
<td>80</td>
</tr>
<tr>
<td>Event type used to indicate a subsystem has been resolved.</td>
<td></td>
</tr>
<tr>
<td>RESOLVING</td>
<td>81</td>
</tr>
<tr>
<td>Event type used to indicate a subsystem is resolving.</td>
<td></td>
</tr>
<tr>
<td>STARTED</td>
<td>81</td>
</tr>
<tr>
<td>Event type used to indicate a subsystem has been started.</td>
<td></td>
</tr>
<tr>
<td>STARTING</td>
<td>81</td>
</tr>
<tr>
<td>Event type used to indicate a subsystem is starting.</td>
<td></td>
</tr>
<tr>
<td>STOPPED</td>
<td>81</td>
</tr>
<tr>
<td>Event type used to indicate a subsystem has been stopped.</td>
<td></td>
</tr>
<tr>
<td>STOPPING</td>
<td>81</td>
</tr>
<tr>
<td>Event type used to indicate a subsystem is stopping.</td>
<td></td>
</tr>
<tr>
<td>UNINSTALLED</td>
<td>81</td>
</tr>
<tr>
<td>Event type used to indicate a subsystem has been uninstalled.</td>
<td></td>
</tr>
<tr>
<td>UPDATED</td>
<td>81</td>
</tr>
<tr>
<td>Event type used to indicate a subsystem has been updated.</td>
<td></td>
</tr>
<tr>
<td>UPDATING</td>
<td>81</td>
</tr>
<tr>
<td>Event type used to indicate a subsystem is updating.</td>
<td></td>
</tr>
</tbody>
</table>
## SubsystemConstants.EventType

### Enum Constant Detail

#### CANCELED

public static final SubsystemConstants.EventType CANCELED

Event type used to indicate that the operations was cancelled (e.g. an install was cancelled).

#### CANCELING

public static final SubsystemConstants.EventType CANCELING

Event type used to indicate that a subsystem operation is being cancelled.

#### FAILED

public static final SubsystemConstants.EventType FAILED

Event type used to indicate that the operation failed (e.g. an exception was thrown during installation).

#### INSTALLED

public static final SubsystemConstants.EventType INSTALLED

Event type used to indicate a subsystem has been installed.

#### INSTALLING

public static final SubsystemConstants.EventType INSTALLING

Event type used to indicate a subsystem is installing.

#### RESOLVED

public static final SubsystemConstants.EventType RESOLVED

Event type used to indicate a subsystem has been resolved.
package org.osgi.service.obr

RESOLVING

public static final SubsystemConstants.EventType RESOLVING

Event type used to indicate a subsystem is resolving.

STARTED

public static final SubsystemConstants.EventType STARTED

Event type used to indicate a subsystem has been started.

STARTING

public static final SubsystemConstants.EventType STARTING

Event type used to indicate a subsystem is starting.

STOPPED

public static final SubsystemConstants.EventType STOPPED

Event type used to indicate a subsystem has been stopped.

STOPPING

public static final SubsystemConstants.EventType STOPPING

Event type used to indicate a subsystem is stopping.

UNINSTALLED

public static final SubsystemConstants.EventType UNINSTALLED

Event type used to indicate a subsystem has been uninstalled.

UPDATED

public static final SubsystemConstants.EventType UPDATED

Event type used to indicate a subsystem has been updated.

UPDATING

public static final SubsystemConstants.EventType UPDATING

Event type used to indicate a subsystem is updating.
Method Detail

values

public static SubsystemConstants.EventType[] values()

valueOf

public static SubsystemConstants.EventType valueOf(String name)
Class SubsystemException

org.osgi.service.subsystem

declares:
(java.lang.Object)
java.lang.Throwable
java.lang.Exception
java.lang.RuntimeException
org.osgi.service.subsystem.SubsystemException

All Implemented Interfaces:
Serializable

public class SubsystemException
extends RuntimeException

Exception thrown by Subsystem when a problem occurs.

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>SubsystemException()</td>
</tr>
<tr>
<td>Construct a subsystem exception with no message.</td>
</tr>
<tr>
<td>\hline SubsystemException(String message)</td>
</tr>
<tr>
<td>Construct a subsystem exception specifying a message.</td>
</tr>
<tr>
<td>\hline SubsystemException(String message, Throwable cause)</td>
</tr>
<tr>
<td>Construct a subsystem exception specifying a message and wrapping an existing exception.</td>
</tr>
<tr>
<td>\hline SubsystemException(Throwable cause)</td>
</tr>
<tr>
<td>Construct a subsystem exception wrapping an existing exception.</td>
</tr>
</tbody>
</table>

Constructor Detail

SubsystemException

public SubsystemException()

Construct a subsystem exception with no message.

SubsystemException

public SubsystemException(String message)

Construct a subsystem exception specifying a message.

Parameters:
message - The message to include in the exception.

SubsystemException

public SubsystemException(Throwable cause)

Construct a subsystem exception wrapping an existing exception.
Parameters:
  cause - The cause of the exception.

SubsystemException

public SubsystemException(String message,
                           Throwable cause)

Construct a subsystem exception specifying a message and wrapping an existing exception.

Parameters:
  message - The message to include in the exception.
  cause - The cause of the exception.

7 State Tables

7.1 State lock with cancelable operations
The following approach treats operations as uninterruptable with the exception of the explicit use of cancel:

1. Transitional states lock out other operations - causes them to be queued up.
2. If a queued operation cannot acquire the lock within a 'reasonable time period' then an exception is thrown for that operation.
3. Transitional states can be canceled through the subsystem cancel() operation which uses the coordinator and resource processors to try to return the subsystem to its previous state.
4. Errors are typically IllegalStateExceptions.
5. No-ops are shown by blank cells.
<table>
<thead>
<tr>
<th>State</th>
<th>cancel</th>
<th>install</th>
<th>uninstall</th>
<th>start</th>
<th>stop</th>
<th>update</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATED?</td>
<td></td>
<td>Install</td>
<td>Error</td>
<td>Install, Resolve and Start</td>
<td>Error</td>
<td>Error</td>
</tr>
<tr>
<td>INSTALLING</td>
<td>Fail the Coordination (end up in CREATED? - bundles in UNINSTALLED?)</td>
<td></td>
<td>Wait for Install to complete, then start Uninstall.</td>
<td>Wait for Install to complete then Start.</td>
<td></td>
<td>Wait for Install to complete then Update.</td>
</tr>
<tr>
<td>INSTALLED</td>
<td></td>
<td>Uninstall</td>
<td>Resolve and Start.</td>
<td></td>
<td></td>
<td>Update</td>
</tr>
<tr>
<td>RESOLVING</td>
<td></td>
<td>Uninstall</td>
<td>Wait for resolve to complete then Start.</td>
<td></td>
<td></td>
<td>Wait for Resolve to complete then Update.</td>
</tr>
<tr>
<td>RESOLVED</td>
<td></td>
<td>Uninstall</td>
<td>Start</td>
<td></td>
<td></td>
<td>Update</td>
</tr>
<tr>
<td>STARTING</td>
<td>Fail the Coordination (end up in RESOLVED)</td>
<td></td>
<td>Wait for Start to complete then uninstall.</td>
<td>Wait for Start to complete then Stop.</td>
<td></td>
<td>Wait for Start to complete then Update.</td>
</tr>
<tr>
<td>ACTIVE</td>
<td></td>
<td>Stop and Uninstall</td>
<td>Stop</td>
<td></td>
<td></td>
<td>Update</td>
</tr>
<tr>
<td>STOPPING</td>
<td>Fail the Coordination (end up in ACTIVE)</td>
<td></td>
<td>Wait for stop to complete then Uninstall.</td>
<td>Wait for Stop to complete then Start.</td>
<td></td>
<td>Wait for Stop to complete then Update.</td>
</tr>
<tr>
<td>UPDATING</td>
<td>Fail the Coordination (go back to previous state)</td>
<td></td>
<td>Wait for Update to complete then Uninstall</td>
<td>Wait for Update to complete then Start.</td>
<td></td>
<td>Wait for Update to complete then Update.</td>
</tr>
<tr>
<td>UNINSTALLING</td>
<td>Fail the Coordination (end up in INSTALLED)</td>
<td>Error</td>
<td></td>
<td>Error</td>
<td>Error</td>
<td>Error</td>
</tr>
<tr>
<td>UNINSTALLED</td>
<td>Error</td>
<td></td>
<td>Error</td>
<td>Error</td>
<td>Error</td>
<td>Error</td>
</tr>
</tbody>
</table>
8 Considered Alternatives

For posterity, record the design alternatives that were considered but rejected along with the reason for rejection. This is especially important for external/earlier solutions that were deemed not applicable.

8.1 State Tables: Interruptable operations

Note: This approach was considered and rejected in favor of the simpler option outlined in section 7.1.

The following approach treats interrupts to transitional states as the norm. These interrupts are performed through the use of the Coordinator:

6. Transitional states are interrupted via Coordination. This can be done in one of two ways:
   1. A resource processor detects a problem and fails the Coordination.
   2. An admin agent chooses to cancel an in-flight operation (e.g. cancelling a start operation whilst in the STARTING state).

7. If the subsystem is in a state beyond (needs defining) that which would be achieved by processing the operation, then it is considered a no-op (e.g. install on an ACTIVE subsystem does not cause it to go to INSTALLED).

8. If an operation is called during a transitional state, and the transitional state is before the state that would be reached then it runs to completion and then performs the requested operation (e.g. start on an INSTALLING subsystem completes the install and then tries to reach ACTIVE).

9. Errors are typically IllegalStateExceptions.

10. No-ops are shown with blank cells.
<table>
<thead>
<tr>
<th>State</th>
<th>cancel</th>
<th>install</th>
<th>uninstall</th>
<th>start</th>
<th>stop</th>
<th>update</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATED?</td>
<td></td>
<td>Install</td>
<td>Error</td>
<td>Error</td>
<td>Error</td>
<td></td>
</tr>
<tr>
<td>INSTALLING</td>
<td>Fail the Coordination (end up in CREATED state? - bundles in UNINSTALL ED?)</td>
<td>Fail Coordination (end up in CREATED state?)</td>
<td>Complete Install, then do Resolve and Start</td>
<td></td>
<td>Error</td>
<td></td>
</tr>
<tr>
<td>INSTALLED</td>
<td></td>
<td>Uninstall</td>
<td>Resolve and Start resources</td>
<td>Update</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESOLVING</td>
<td></td>
<td>Uninstall</td>
<td>Complete Resolve then Start</td>
<td>Complete Resolve then do Update</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESOLVED</td>
<td></td>
<td>Uninstall</td>
<td>Start</td>
<td>Update</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STARTING</td>
<td>Fail the Coordination (end up in RESOLVED )</td>
<td>Fail the Start Coordination and Uninstall</td>
<td>Fail the Start (end up in RESOLVED)</td>
<td>Fail the Start then do Update</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACTIVE</td>
<td></td>
<td>Stop then Uninstall</td>
<td>Stop</td>
<td>Update</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STOPPING</td>
<td>Fail the Coordination (end up in ACTIVE)</td>
<td>Complete the Stop then Uninstall</td>
<td>Fail the Stop Coordination (end up in ACTIVE)</td>
<td>Complete the Stop then do Update</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UPDATING</td>
<td>Fail the Coordination (end up in state prior to update).</td>
<td>Fail the Update then Uninstall</td>
<td>Complete Update then do Start</td>
<td>Complete the Update then do next Update</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNINSTALLING</td>
<td>Fail the Coordination (end up in INSTALLED)</td>
<td>Error</td>
<td>Error</td>
<td>Error</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNINSTALLED</td>
<td>Error</td>
<td>Error</td>
<td>Error</td>
<td>Error</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 8.2 Metadata Formats

XML was considered for the format of the Subsystem and Deployment definitions but rejected because the power and associated complexity/verbosity of XML was not required to express the subsystem information.
Java properties files were also considered for the format of the Subsystem and Deployment definitions but these suffered from the opposite problem from XML. Whilst Java properties files can use the same colon separator as Java manifests for name-value pairs and would therefore be familiar to OSGi developers, they do not have the concept of attributes and are therefore too simplistic to easily represent a Subsystem or Deployment definition.

### 8.3 Zip versus Jar

This design proposes the use of a zip file for the subsystem archive format. A design which used a jar format was considered for the following reason:

1. to enable subsystems to be installed as bundles using the extender pattern.
2. To enable the artefact to be signed.
3. To enable JarInputStream to be used to load them without needing to test first whether or not it was a zip.

After investigation the following conclusions were reached. Regarding 1, whilst it seems attractive to manage a subsystem as a bundle, this would lead to strange lifecycle bundles living in the runtime. Also, these bundles would not be able to replace the need for a SubsystemAdmin services and therefore their value is limited. Regarding 2, the current belief is it is sufficient to sign the bundles contained within the archive, not the archive itself. If this is found not to be sufficient, then this discussion can be revisited. Regarding 3, it is simple to load a zip file using a JarInputStream and therefore this is a non-issue.

One concern that stems from the use of the jar format is the need to have a META-INF/MANIFEST.MF that serves no purpose other than to potentially confuse the developer.

### 9 Security Considerations

_Description of all known vulnerabilities this may either introduce or address as well as scenarios of how the weaknesses could be circumvented._

### 10 Document Support

#### 10.1 References

Add references simply by adding new items. You can then cross-refer to them by choosing <Insert><Cross Reference><Numbered Item> and then selecting the paragraph. STATIC REFERENCES (I.E. BODGED) ARE NOT ACCEPTABLE, SOMEONE WILL HAVE TO UPDATE THEM LATER, SO DO IT PROPERLY NOW.

### 10.2 Author’s Address

<table>
<thead>
<tr>
<th>Name</th>
<th>Guillaume Nodet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Progress</td>
</tr>
<tr>
<td>Address</td>
<td></td>
</tr>
<tr>
<td>Voice</td>
<td></td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:gnodet@progress.com">gnodet@progress.com</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Graham Charters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>IBM</td>
</tr>
<tr>
<td>Address</td>
<td></td>
</tr>
<tr>
<td>Voice</td>
<td></td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:charters@uk.ibm.com">charters@uk.ibm.com</a></td>
</tr>
</tbody>
</table>

### 10.3 Acronyms and Abbreviations

### 10.4 End of Document
Abstract

Blueprint declarative transaction provides container managed transaction for blueprint beans which enables application developers to focus on the application logic and provides better separation of application and infrastructure code.
# 0 Document Information

## 0.1 Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 Document Information</td>
<td>2</td>
</tr>
<tr>
<td>0.1 Table of Contents</td>
<td>2</td>
</tr>
<tr>
<td>0.2 Terminology and Document Conventions</td>
<td>3</td>
</tr>
<tr>
<td>0.3 Revision History</td>
<td>3</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>3</td>
</tr>
<tr>
<td>2 Application Domain</td>
<td>4</td>
</tr>
<tr>
<td>2.1 Overview</td>
<td>4</td>
</tr>
<tr>
<td>2.1 Sample User Scenario</td>
<td>4</td>
</tr>
<tr>
<td>3 Problem Description</td>
<td>5</td>
</tr>
<tr>
<td>4 Requirements</td>
<td>6</td>
</tr>
<tr>
<td>5 Technical Solution</td>
<td>7</td>
</tr>
<tr>
<td>5.1 Related RFCs</td>
<td>7</td>
</tr>
<tr>
<td>5.2 Blueprint Transaction XML Schema</td>
<td>7</td>
</tr>
<tr>
<td>5.2.1 The Transaction Element</td>
<td>9</td>
</tr>
<tr>
<td>5.2.2 The Method Attribute</td>
<td>10</td>
</tr>
<tr>
<td>5.2.3 The Transaction Attribute</td>
<td>10</td>
</tr>
<tr>
<td>5.2.4 The Bean Attribute</td>
<td>11</td>
</tr>
<tr>
<td>5.2.5 Transaction Examples At Bean Level</td>
<td>11</td>
</tr>
<tr>
<td>5.2.6 Transaction Examples At Bundle Wide Level</td>
<td>12</td>
</tr>
<tr>
<td>5.2.7 Transaction Examples At Bean Level And Bundle Wide Level</td>
<td>13</td>
</tr>
<tr>
<td>5.3 Sample User Scenario</td>
<td>14</td>
</tr>
<tr>
<td>6 Command Line API</td>
<td>15</td>
</tr>
<tr>
<td>7 JMX API</td>
<td>15</td>
</tr>
<tr>
<td>8 Initial Spec Chapter</td>
<td>15</td>
</tr>
<tr>
<td>9 Considered Alternatives</td>
<td>16</td>
</tr>
<tr>
<td>10 Security Considerations</td>
<td>16</td>
</tr>
<tr>
<td>11 Document Support</td>
<td>16</td>
</tr>
</tbody>
</table>
0.2 Terminology and Document Conventions

Namespace Handler – the component of the blueprint container runtime responsible for handling the blueprint transactions namespace. The mechanism by which this is ‘plugged in’ to the blueprint container is not covered in this specification. This could be blueprint container implementation specific or a standard mechanism may be defined in another specification.

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 11.1.

Source code is shown in this typeface.

0.3 Revision History

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

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>07/12/2010</td>
<td>Initial version</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lin Sun, IBM <a href="mailto:linsun@us.ibm.com">linsun@us.ibm.com</a></td>
</tr>
<tr>
<td>0.1</td>
<td>07/26/2010</td>
<td>Address additional comments -</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1. Disallow two *s next to each other</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Ability to specify bundle wide transaction.</td>
</tr>
<tr>
<td>0.2</td>
<td>09/17/2010</td>
<td>Simplified the rules to determine the transaction attribute</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Simplified the value for the method and bean attributes.</td>
</tr>
<tr>
<td>0.3</td>
<td>11/16/2010</td>
<td>Add: statement for synthetic beans.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarify namespace handler requirement and a few minor errors here and there.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mark transaction attribute as optional and default to Required.</td>
</tr>
<tr>
<td>0.31</td>
<td>05/04/11</td>
<td>Graham Charters</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tidy up for draft publication.</td>
</tr>
</tbody>
</table>
1 Introduction

RFP 138 was approved which defined requirements for blueprint declarative transaction. This RFC describes the means to use blueprint interceptors and customized blueprint namespace to support blueprint declarative transaction.

2 Application Domain

2.1 Overview

The Blueprint Container specification version 1.0 in OSGi Compendium Release 4.2 defines a dependency injection framework for OSGi bundles that understands the unique dynamic nature of services. Bundles in this programming model contain a number of XML definition resources which are used by the Blueprint Container to wire the application together and start it when the bundle is active.

The Java Transaction API (JTA) Transaction Services specification in OSGi Enterprise Release 4.2 is based on JTA 1.1 specification and defines the use of JTA through 3 transaction services: User Transaction, Transaction Manager, and Transaction Synchronization.

With these two specifications, a blueprint application developer can obtain the transaction services from the OSGi service registry, set the transaction boundaries and manage the transaction in the application code. However, this is no defined method to enable the developer to declare transactional demarcation for blueprint components and no defined contract that a blueprint container could offer when processing such transactional declarations.

2.1 Sample User Scenario

The blueprint container manages various components such as bean, service, reference-list and reference elements. Blueprint beans are declared using the bean element. The following
defines a single bean called TestBeanImpl implemented by the POJO org.apache.aries.simple.TestBeanImpl.

**blueprint.xml**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
  <bean id="TestBeanImpl" class="org.apache.aries.simple.TestBeanImpl" />
</blueprint>
```

If a transaction is required in the POJO org.apache.aries.simple.TestBeanImpl, the application developer would currently have to obtain the User Transaction service from OSGi service registry, set the boundaries of transaction and manage the transaction in the application code, for example the insertRow method in TestBeanImpl.java:

**TestBeanImpl.java**

```java
public class TestBeanImpl {

  public TestBeanImpl() {
    ...
  }

  public void insertRow(String name, int value) throws SQLException {
    // get the user transaction service
    UserTransaction ut = getUserTransactionService();

    // begin the user transaction
    ut.begin();

    try {
      // start the operation to insert name, value into the db
      ...
    } finally {
      // commit the user transaction
      ut.commit();
    }

  }
}
```

This form of transaction management can be referred to as “application managed transactions” or “bean managed transactions”, whereby the bean code explicitly manages the boundaries of the transaction. Given that the primary purpose of a container is to manage the lifecycle and qualities of service of the entities it contains, it is desirable to define a standard mechanism for
container to manage transactions on behalf of those entities. This enables the bean developer to focus on the application logic and provides better separation of application and infrastructure code.

A useful model to refer to that is extremely familiar to Java EE developers is the Java EE enterprise bean with container managed transactions, where the Enterprise Java Bean (EJB) container manages the boundaries of the transactions. The vast majority of enterprise beans use container-managed rather than bean-managed transactions.

3 Problem Description

It is possible that a transaction is required in one or more methods of the blueprint managed components defined in blueprint XML definition. The blueprint container specification does not describe a means to allow application developers to define transaction attributes (such as Required, RequiresNew, NotSupported, Supports, Mandatory and Never, as provided by the EJB container) in the blueprint XML definition. Instead, if an application developer has chosen to use the blueprint container to manage his bundles, the application developer would have to set the transaction boundary and manage the transaction in the application code, whenever a transaction is required. There is no container managed transaction defined for blueprint components.

This proposal is to provide container managed transaction for blueprint managed components, where application developers can use blueprint XML definition to declare transactions and their associated transaction attributes within one or more methods of the blueprint container managed components, so that the transaction can be managed by the container automatically based on the specified transaction attributes and the bean methods to which they apply.
4 Requirements

BDT01 - Blueprint Declarative Transaction must support these types of Transaction attributes: (Required, RequiresNew, NotSupported, Mandatory, Supports and Never).

BDT02 - Blueprint Declarative Transaction must support configuration at the blueprint bean level.

BDT03 - Blueprint Declarative Transaction must support configuration at the bean method level.

BDT04 - Blueprint Declarative Transaction must support configuration of a default transaction attribute that can be overridden by more specific declarations.

BDT05 - Blueprint Declarative Transaction must define the container’s behavior for completing transactions in the presence of a normal successful method execution or exception.

BDT06 – Blueprint declarative transaction must be configurable in blueprint XML definition file.

BDT07 - It MUST be possible to implement Blueprint Declarative Transactions using Blueprint Interceptors.

BDT08 - Blueprint Declarative Transactions MUST define a custom namespace.

5 Technical Solution

5.1 Related RFCs

RFC 155 Blueprint namespace enables user defined namespace using an XML schema file.
In this RFC, a transaction custom namespace is introduced to allow users to easily declare
transaction attribute within one or more methods of the blueprint bean component. RFC 166
blueprint bean interceptor can be used to allow blueprint container to intercept method
invocation to add additional transactional demarcation operations before and after the method
invocation.

5.2 Blueprint Transaction XML Schema

By leveraging RFC 155, the following blueprint transaction XML schema is proposed for
blueprint transaction:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns="http://aries.apache.org/xmlns/transactions/v1.0.0"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://aries.apache.org/xmlns/transactions/v1.0.0"
    elementFormDefault="qualified" attributeFormDefault="unqualified"
    version="1.0.0">

    <xsd:annotation>
        <xsd:documentation><![CDATA[
This is the XML Schema for the OSGi Blueprint declarative
transaction 1.0.0 development descriptor. Blueprint declarative transaction
is a custom namespace for OSGi Blueprint service 1.0.0 development
descriptor. It is designed to decorate transaction attribute of the bean
components, which can be done at the bean level or at the bundle wide level.
In other words, the transaction element can reside in the bean element or in
the root of the blueprint element as a top level transaction element.
Blueprint configuration files using this schema must indicate the schema
using the transactions/v1.0.0 namespace. For example,
<transaction
    xmlns="http://aries.apache.org/xmlns/transactions/v1.0.0">
    if used as a qualified namespace, "tx" is the recommended namespace
prefix.
]]></xsd:documentation>
</xsd:annotation>

    <xsd:simpleType name="TtransactionAttribute">
        <xsd:annotation>
            <xsd:documentation><![CDATA[
The TtransactionAttribute type defines the transaction attribute
for blueprint declarative transaction.
]]></xsd:documentation>
    </xsd:annotation>
</xsd:simpleType>
</xsd:schema>
```
When the tx element is not specified in the blueprint XML definition file, the default behavior is equivalent to:

```xml
<tx:transaction value="NotSupported" />
```
5.2.1 The Transaction Element

The transaction element can be used as a child of the bean element to specify the bean level transaction configuration for the bean. The transaction element can also be used at the top level (e.g., at the root of the blueprint element) to specify the bundle wide transaction for the blueprint managed bundle. The following must be used from high priority to low priority to determine which transaction attributes apply to a particular method in a bean component:

1. bean level transaction element with the method attribute.
2. bean level transaction element without the method attribute.
3. top level transaction element with both the bean attribute and method attribute
4. top level transaction element with only the bean attribute
5. top level transaction element with only the method attribute
6. top level transaction elements with no bean or method attribute specified

When using the above selection rules, if the bean attribute or method attribute is set to ‘*’, it must be treated as the attribute is not specified. The top level transaction elements should not apply to synthetic beans.

If there are potential multiple matches for bean level transaction, the transaction namespace handler must throw an Exception back to the Blueprint Container to fail the creation of the Blueprint Container. If there are multiple matches for bundle wide transaction, the transaction namespace handler must throw an Exception back to the Blueprint Container to fail the creation of the Blueprint Container. The Blueprint FAILURE event must be emitted as the result of the failure, which is specified in the Blueprint Container Specification.

5.2.2 The Method Attribute

The method attribute is used to specify one or more method names and method names can be wild-carded with one ‘*’. ‘*’ can appear anywhere in the name and can only appear once. Methods can be listed (whitespace or comma separated) and be a mixture of fixed string and one wild-card. If there are potential multiple matches, the transaction namespace handler must throw an Exception back to the Blueprint Container to fail the creation of the Blueprint Container.

For example, when the transaction namespace handler parses the 2nd transaction element inside the bean, it detects multiple matching for potential methods like count1Row thus the namespace handler must throw a Throwable back to the Blueprint Container to fail the creation of the Blueprint Container for the bundle.
<bean ...
   <tx:transaction method="count*" value="Required" />
   <tx:transaction method="count*Row" value="RequiresNew" />
</bean>

The method attribute is optional. When the method attribute is not being specified, it is equivalent to method attribute set to '*'. For example:

<bean ...
   <tx:transaction method="*" value="Required" />
</bean>

5.2.3 The Transaction Attribute
The transaction attributes (Required, Mandatory, RequiresNew, Supports, NotSupported, and Never) are defined in Enterprise Java Bean3.0 specification section 136.6.2. The Blueprint container will manage transaction boundaries in the same manner defined for the EJB container, which is not repeated here. After parsing the blueprint XML definition file, the blueprint container must demarcate transactions based on the transaction attribute specified in the blueprint XML definition file.

5.2.4 The Bean Attribute
The bean attribute can only be specified in top level transaction element. It is used to specify one or more bean component ids and beans can be wild-carded with one '*'. '*' can appear anywhere in the name and can only appear once. Beans can be listed (whitespace or comma separated) and be a mixture of fixed string and wild-card.

The bean attribute is optional. When the bean attribute is not specified, the default value is an asterisk character (*). For example:

<blueprint>
   <tx:transaction method="Insert*" value="Required" />
</blueprint>

The blueprint namespace handler design must provide a way to allow the transaction namespace handler to perform post blueprint namespace processing. This ensures all blueprint components are registered in the Component Definition Registry when the transaction namespace handler parses the bundle wide transaction elements and uses the Component Definition Registry to validate the bean attribute.

5.2.5 Transaction Examples At Bean Level
Transactions can be configured at the bean level. You must specify the value attribute in the transaction element. Valid values are those that are defined by Java EE, that is, Required, RequiresNew, NotSupported, Mandatory, Supports or Never. For example:

```xml
<bean ...>
   <tx:transaction method="updateOrder" value="Required" />
</bean>
```

You can also use the wildcard character anywhere in a method name and but you can only use it once. For example:

```xml
<bean ...>
   <tx:transaction method="update*Ord" value="Required" />
</bean>
```

Methods can be listed (whitespace or comma separated) and be a mixture of fixed string and wild-cards, for example,

```xml
<bean ...>
   <tx:transaction method="update* makeItSo" value="Required" />
</bean>
```

Multiple method configurations can appear in the same component, for example

```xml
<bean ...>
   <tx:transaction method="update* makeItSo" value="Required" />
   <tx:transaction method="recordStatus" value="RequiresNew" />
</bean>
```

Method configurations can appear alongside one component configuration, where the methods config takes precedence.

```xml
<bean ...>
   <tx:transaction value="Required" />
   <tx:transaction method="recordStatus" value="RequiresNew" />
</bean>
```

Wildcard matching and selection behavior is determined by the rules specified in section 5.1.2.

### 5.2.6 Transaction Examples At Bundle Wide Level
Transactions can be configured at the bundle wide level. Same as bean level configuration, you must specify the value attribute in the transaction element, for example:

```xml
<blueprint>
   <tx:transaction value="Required" />
   ...
</blueprint>
```
You can also use the wildcard character anywhere in a method name or a bean name but you can use the wildcard character only once. For example:

```xml
<blueprint>
  <tx:transaction bean="bean1*" method="update*Ord" value="Required" />
  ...
</blueprint>
```

Methods or Beans can be listed (whitespace or comma separated) and be a mixture of fixed string and wild-cards, for example,

```xml
<blueprint>
  <tx:transaction bean="bean1* bean2*" method="update* makeItSo" value="Required" />
  ...
</blueprint>
```

Multiple configurations can appear in the same bundle, for example

```xml
<blueprint>
  <tx:transaction method="update* makeItSo" value="Required" />
  <tx:transaction bean="bean1" bean="update*" value="RequiresNew" />
</blueprint>
```

When multiple configurations exist, the selection behavior is determined by the rules in section 5.2.1.

### 5.2.7 Transaction Examples At Bean Level And Bundle Wide Level

Transaction can be configured at bean level, bundle wide level or both.

```xml
<blueprint>
  <tx:transaction value="Required" />
  <tx:transaction method="get*" value="Supports" />
  <tx:transaction bean="noTx" value="Never" />
  <tx:transaction bean="someTx" method="get*" value="Mandatory" />
</blueprint>
```

```xml
<bean id="requiresNew" class="foo">
  <tx:transaction value="RequiresNew"/>
</bean>
<bean id="noTx" class="foo"/>
```
<bean id="someTx" class="foo"/>

<bean id="anotherBean" class="foo"/>

<blueprint/>

In the above example the following would apply:

- For the bean requiresNew, all methods have REQUIRESNEW behaviour.
- For the bean noTx, all methods have NEVER behaviour.
- For the bean someTx, methods starting with "get" are MANDATORY, and all other methods are REQUIRED.
- For the bean anotherBean, all methods that start with "get" are SUPPORTS, and all other methods are REQUIRED.

## 5.3 Sample User Scenario

Continuing to use the example in the section 2, the following defines the single bean called TestBeanImpl, using the declarative transaction on the method insertRow with transaction attribute value required.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
    xmlns:tx="http://aries.apache.org/xmlns/transactions/v1.0.0">
    <bean id="TestBeanImpl" class="org.apache.aries.simple.TestBeanImpl">
        <tx:transaction method="insertRow" value="required"/>
    </bean>
</blueprint>
```

The TestBeanImpl.java example used in section 2 can be simplified as below:

```java
public class TestBeanImpl {

    public TestBeanImpl() {
        ...
    }

    public void insertRow(String name, int value) throws SQLException {
        // start the operation to insert name, value into the db
        ...
    }
}
```
Based on the application’s requirement, application developers can set different transaction attributes on specific methods, such as Required, RequiresNew, Mandatory, NotSupported, Supports, and Never.

This simplifies greatly the transaction management in application code TestBeanImpl.java.

### 6 Command Line API

None

### 7 JMX API

RFC 139 describes the JMX API to the OSGi Framework and a number of standard OSGi Services. The JMX specification is also present as chapter 124 in the OSGi spec documents.

For all new functionality added to the OSGi Framework the question should be asked: would this feature benefit from a JMX API? The expectation is that in most cases it would.

The JMX API for the design in this RFC should be described here and if there is no JMX API an explanation should be given explaining why this is not applicable in this case.

None
8 Initial Spec Chapter

Provide a link to where the Initial Spec Chapter can be found. The Initial Spec Chapter is typically written by someone other than the author(s) of this RFC and represents a rewrite of this document as close as possible to what will ultimately appear in the OSGi Specifications. It will be used by the Specification Editor as the basis for the ultimate specification chapter.

The spec template and writing guidelines can be found here:


9 Considered Alternatives

For posterity, record the design alternatives that were considered but rejected along with the reason for rejection. This is especially important for external/earlier solutions that were deemed not applicable.

10 Security Considerations

Description of all known vulnerabilities this may either introduce or address as well as scenarios of how the weaknesses could be circumvented.
11 Document Support

11.1 References


Add references simply by adding new items. You can then cross-refer to them by choosing <Insert><Cross Reference><Numbered Item> and then selecting the paragraph. STATIC REFERENCES (I.E. BODGED) ARE NOT ACCEPTABLE, SOMEONE WILL HAVE TO UPDATE THEM LATER, SO DO IT PROPERLY NOW.

11.1 Author’s Address

<table>
<thead>
<tr>
<th>Name</th>
<th>Lin Sun</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>IBM</td>
</tr>
<tr>
<td>Address</td>
<td>3039 Cornwallis Road, RTP, NC, 27560</td>
</tr>
<tr>
<td>Voice</td>
<td>919-254-6388</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:linsun@us.ibm.com">linsun@us.ibm.com</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Graham Charters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>IBM</td>
</tr>
<tr>
<td>Address</td>
<td>IBM UK Ltd, MP211, Hursley Park Road, Winchester, SO21 2JN</td>
</tr>
<tr>
<td>Voice</td>
<td>+44 1962 816527</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:charters@uk.ibm.com">charters@uk.ibm.com</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Ian Robinson</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>IBM</td>
</tr>
<tr>
<td>Address</td>
<td>IBM UK Ltd, MP211, Hursley Park Road, Winchester, SO21 2JN</td>
</tr>
<tr>
<td>Voice</td>
<td>+44 1962 818626</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:ian_robinson@uk.ibm.com">ian_robinson@uk.ibm.com</a></td>
</tr>
</tbody>
</table>

Copyright © IBM Corporation 2011 All Rights Reserved
11.2 Acronyms and Abbreviations

11.3 End of Document
RFC 166 - Blueprint Bean Interceptors

Draft

16 Pages

Abstract

This RFC describes the requirements and solution for Blueprint Bean Interceptors as an extension of the Blueprint container specification.
0.2 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 11.1.

Namespace Handler – the component of the blueprint container runtime responsible for handling the blueprint transactions namespace and other Blueprint namespaces. The mechanism by which this is ‘plugged in’ to the blueprint container is not covered in this specification. This could be blueprint container implementation specific or a standard mechanism may be defined in another specification.

Source code is shown in this typeface.

0.3 Revision History

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

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>07/09/10</td>
<td>Initial version</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Joe Bohn, IBM, <a href="mailto:jbohn@us.ibm.com">jbohn@us.ibm.com</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>with input from:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Graham Charters, IBM, <a href="mailto:charters@uk.ibm.com">charters@uk.ibm.com</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Andrew James Osborne, IBM, <a href="mailto:ozzy@us.ibm.com">ozzy@us.ibm.com</a></td>
</tr>
<tr>
<td>0.1</td>
<td>08/09/10</td>
<td>Removed references to Interceptors being services in Command Line API and JMX API sections</td>
</tr>
<tr>
<td>0.2</td>
<td>12/11/10</td>
<td>Updates based upon discussions from 9/8/10 EEG meeting and 9/17/10 F2F meetings:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Register Interceptors are Services per bean/bundle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Add service references (optional or mandatory) for interceptor services</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• use Service Ranking to order interceptors</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Require that all bean managers provide a unique bean-id, even for inline or anonymous beans.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Lifecycle of Interceptor tied to lifecycle of Namespace Handler</td>
</tr>
</tbody>
</table>
1 Introduction

RFP 137 [8]. was approved defining requirements for Blueprint Bean Interceptors. This RFC describes a proposal to meet those requirements. A slightly modified version of the introduction from the RFP is included next in this introduction for convenience.

The Blueprint Container specification version 1.0 in OSGi Compendium Release 4.2 defines a dependency injection framework for OSGi bundles that supports the unique dynamic nature of beans and services. Bundles in this programming model contain a number of XML definition resources which are used by the Blueprint Container to wire the application together and start it when the bundle becomes active.

A proposed extension to the Blueprint Container Specification is the Blueprint Declarative Transaction feature [4]. One of the necessary capabilities of the Blueprint Declarative Transaction feature is a mechanism to intercept a Blueprint Bean method invocation to add logic for transaction management. The same capability to intercept a blueprint bean method invocation and perform some function can also be useful for other features on Blueprint bean definitions that leverage Blueprint Namespace [3]. and require some runtime capability when processing blueprint bean methods.

This RFC refines the requirements and describes a solution for Blueprint Bean Interceptors as an extension of the Blueprint Container Specification.

Note that this RFC will be developed in conjunction with the Blueprint Declarative Transactions [4], and Blueprint Namespaces [3]. RFCs. This RFC may be leveraged to fulfill the capabilities defined in the Blueprint Declarative Transactions design and may propose additional requirements for the Blueprint Namespaces design.
2 Application Domain

2.1 Overview

The Blueprint Container manages the creation and life cycle of blueprint beans. The Blueprint Namespace extension allows for the introduction of custom components or additional features on standard blueprint components.

Blueprint Namespace extensions can be applied to bean definitions as well as other components. When applied to bean definitions, a namespace handler can be used to enhance a blueprint bean definition or inject properties into the bean. As such, the Blueprint specification and namespace extensions are primarily concerned with the creation of blueprint beans. However, with the current capabilities they could not be involved in subsequent method invocation against the bean.

When introducing custom features for a blueprint bean there are times when it may be necessary to enhance the functionality of the bean during execution and perhaps on a per-method invocation basis. For example, to support capabilities such as those required by Blueprint Declarative Transactions it is necessary to gain access into the flow of control before and after a bean method invocation. Blueprint Declarative Transactions requires this access to facilitate transaction management.

Therefore, a solution for Blueprint Bean Interceptors requires some mechanism to extend a bean by permitting logic to be injected prior to and immediately after a bean method invocation.

It seems reasonable that such an extension would always be associated with a comparable extension in the blueprint bean component definition which is currently accomplished using the Blueprint Namespace extension. Therefore, it seems reasonable that a Blueprint Bean Interceptor would be associated with a Blueprint Namespace and should be introduced into the system by the namespace handler. It should therefore be possible to support the implementation of Blueprint Bean Interceptors using Namespace Handlers. However, this should not preclude possible future extension mechanisms such as Annotations which may also provide similar capabilities to extend a blueprint bean component definition and therefore could also introduce and support blueprint bean interceptors.

2.2 Example using Declarative Transaction Support

As mentioned earlier the Declarative Transaction Support that is currently being proposed (see [4],) has a need to intercept bean methods. One possible implementation for this capability would be to use Blueprint Bean Interceptors as defined in this RFC. In addition to the Transaction Namespace Handler required to parse the custom namespace a blueprint bean interceptor could also be created. The interceptor includes logic that is processed before a bean method invocation (pre-call), and after a method invocation (post-call). In such a scenario, a declarative transaction namespace entry is applied to the blueprint bean definition in the XML configuration file. The namespace handler is invoked when parsing the XML configuration file. At this point it is important to be able to associate the interceptor with the bean definition. This interceptor can then be retrieved when the bean definition is being interpreted by the Bean Manager. When the blueprint bean is later instantiated, the associated interceptor(s) is bound to the bean (possibly using a proxy) such that the appropriate interceptor logic can be invoked before and/or after the bean method invocation. For Blueprint Declarative Transaction support, the logic creates any necessary transactions prior to the bean method call and commits or rolls-back those transactions following the method call as necessary for the method result.

A unique method correlator is also required to match processing before the method invocation (pre-call) with logic
after the method invocation (post-call). In the case of Blueprint Declarative Transactions this correlator is used to match the transaction created in the pre-call with its response for commit or rollback in the post-call.

Given that there could be more than one interceptor associated with a particular bean, and that the processing of one interceptor could have some impact on the processing of another interceptor, some mechanism must be provided to ensure a deterministic order in invocation of interceptors for a given bean method pre-call and post-call.

It is intended that the bean interceptor capability is used in conjunction with Blueprint Namespaces as an optional feature when blueprint bean method level enhancement is also a requirement.

### 3 Problem Description

A Blueprint Namespace extension to a bean definition may require some logic that must be invoked in conjunction with a blueprint bean method invocation. As already discussed, this is the case with Blueprint Declarative Transactions to support the creation of a transaction prior to invoking methods on a bean and close the transaction following the bean method invocation. Declarative Transactions uses a Blueprint Namespace extension to declare the level of transaction support required for the blueprint bean methods (see [4], for details). Without a feature such as Blueprint Bean Interceptors there is no solution to intercept the bean method invocations to begin, commit, or rollback a transaction.

This proposal is to provide a mechanism to define, register, and invoke interceptors before and after bean method invocations when appropriately specified in the Blueprint XML for the associated namespace or similar extension mechanism. The extension mechanism could then register an interceptor for the bean that would perform necessary functionality for the namespace.

### 4 Requirements

1. Blueprint Bean Interceptors MUST support interceptor invocation prior to a bean method call (pre-call) with the access to component metadata and all facets of the method necessary for successful pre-call processing (such as method name, parameters, etc...)

2. Blueprint Bean Interceptors MUST support a unique method invocation correlator that can be used to match a method pre-call with subsequent post-call results.
3. Blueprint Bean Interceptors MUST support interceptor invocation following a successful bean method call with access to component metadata and all facets of the method necessary for successful post-call processing (such as method name, parameters, correlator, result, etc.).

4. Blueprint Bean Interceptors MUST support interceptor invocation following an exception result to a method call with access to component metadata and all facets of the method necessary for successful post-call exception processing (such as method name, parameters, correlator, exception, etc...).

5. Blueprint Bean Interceptors MUST provide a mechanism to support deterministic ordering (relative rank) of interceptor invocation pre-call and post-call in the event that more than one interceptor is registered for a given bean method.

6. It MUST be possible to implement RFP 138 Declarative Transactions in terms of a Blueprint Bean Interceptor but it is not a requirement that Declarative Transactions leverage this capability.

7. Blueprint Bean Interceptors MUST NOT interfere with the normal flow of control for a bean which is not defined to utilize this capability.

8. It MUST be possible to implement Blueprint Bean Interceptors using RFC 155 Namespace Handlers[3].

## 5 Technical Solution

To implement a Bean Interceptor there are several primary tasks that must be considered. First, one must define the structure and behavior of a Blueprint Bean Interceptor itself. Second, one must define some mechanism to associate one or more Bean Interceptors with particular Blueprint Beans. Finally, one must define a mechanism for invoking the appropriate Interceptor methods in conjunction with calls to the Blueprint Bean methods.

A slightly more detailed list of the items that must be supported in a Blueprint Bean Interceptor solution follows:

1. Create a mechanism to associate Interceptors with Blueprint Bean definitions (we will use custom Namespaces).

2. Define a standard interface that Interceptors can implement (the `Interceptor` interface)

3. Define a mechanism for the Blueprint Bean Manager to locate and invoke Interceptors for Blueprint Bean method invocations.

4. Define a mechanism to order the processing of multiple Blueprint Bean Interceptor invocations.

5. Define a mechanism to deal with potential errors produced by a Blueprint Bean Interceptor implementation.

6. Define a mechanism to correlate pre-method invocations with post-method or error results.
5.1 Namespace Handler – Associating Interceptors with Beans

As mentioned above, the first primary task is to associate the Interceptors with the designated Beans. At this point in time Namespace Handlers are the primary mechanism for customization and so this design will be in terms of customization using Namespace Handlers. However, additional mechanisms of customization may be introduced in the future such as custom annotations. When such mechanisms are introduced they may also provide an appropriate mechanism to associate bean interceptors with specific bean definitions.

Namespace handlers are a natural mechanism to use for introducing Interceptors. In fact, it is difficult to conceive of a scenario where an interceptor would be required without some corresponding metadata extensions. For example, the capabilities required for Declarative Transactions require not only the introduction of a custom Namespace to associate the feature with a bean but also the runtime capability provided by Blueprint Bean interceptors to act on the method level. In the case of Declarative Transaction support the Namespace Handler has a need to allocate a transaction at the start of certain methods and commit or rollback that transaction based upon the results from the bean method. At the same time the Namespace is necessary to define the criteria applied the transaction.

5.1.1 Registering Interceptors

A Namespace Handler must first obtain a reference to the Interceptor. The Interceptor itself, it is simply a POJO that implements the Interceptor interface. The interface specifies the various methods that must be supported to participate as a Blueprint Bean Interceptor. The interceptor implementation can be created within the Namespace Handler itself or it can be created by the Blueprint Container and injected into the Namespace handler similar to any other Java object.

Next, the Interceptor must be associated with the Blueprint Bean metadata. This RFC proposes that this is accomplished by registering a service in the Service Registry for the interceptor using the bundle-Id of the bundle being parsed, the bean-id for the associated bean, and the URI for the specific Namespace as properties on the service. The Interceptor service should also be registered as implementing the the Interceptor interface (see section 5.2).

The set of service properties that should be advertised on the Interceptor when registered are as follows:

- `osgi.service.blueprint.bean-id` = the bean-id as specified in the blueprint plan or generated by the blueprint container.
- `osgi.service.blueprint.bundle-id` = the bundle-id for the blueprint bundle that is being parsed
- `osgi.service.blueprint.namespace` = the URI for the Namespace handler registering this Interceptor in the Service Registry. In the event that the Namespace handler supports multiple URIs the namespace handler should chose just one for registration with the interceptor service. The URI only serves to differentiate this interceptor service from other interceptor services that apply to the same bean instance.

Finally, there needs to be some association between the bundle being parsed that requires this interceptor and the service registered for the interceptor. This is accomplished by the Namespace handler creating a Service Reference and adding it to the component definitions for the bundle being parse. The Service Reference can be either mandatory or optional depending on the required availability of this interceptor.

Note, at this time interceptors are only being proposed for Blueprint Beans and hence it is expected that interceptors will only be registered against Blueprint Bean components. However, this should not be presumed in an implementation. Future enhancements or extensions may find it useful to also registered Interceptors against
other component types and therefore any implementation specifically created for a Blueprint Bean component should gracefully ignore any registration or invocation for a different component type.

5.1.2 Using bean-id for identification in Interceptor Service Registration

The use of the bean-id along with bundle-id and namespace when registering an interceptor service has a significant implication. It implies that bean-id must be available for any bean that a namespace handler may desire to attach an interceptor, including anonymous and inner beans. Therefore, it is proposed by this RFC that the blueprint specification is changed so that it no longer requires ComponentMetadata.getId() to return null for inlined and anonymous bean managers and in fact requires that all bean managers return a unique bean id within the Blueprint Container.

5.1.3 Locating and Ordering Interceptors

The purpose of registering an interceptor with a bean is to enable it to be later retrieved and invoked. Therefore it is necessary to have some means to retrieve the interceptor reference. Also, because multiple interceptors can be registered for the same bean definition, it must be possible that multiple interceptors are applicable for a given bean and they must be invoked in some deterministic order. To accomplish this interceptors are ranked using the rank of the service registered for the interceptor. Interceptors are ordered by descending rank in a manner consistent with Services. In the event of a tie by rank the order of registration is used. In other words, priority is given to the highest ranked interceptors and the earliest registered interceptors.

The retrieval of interceptors is performed by Blueprint Bean Manager when instantiating a bean. The Bean Manager must leverage the service references to locate applicable interceptors for the specified bean, honoring the service reference availability of mandatory or optional and honoring the service rank to process interceptors in the appropriate order. If an interceptor service reference is mandatory then an exception should be thrown if the required interceptor service is unavailable for some reason. If the interceptor service reference is optional an unavailable service reference should be ignored by the Blueprint Bean Manager.

5.1.4 Summary of Namespace Handler requirements

- Register an interceptor service for each bean instance in the service registry using the bundle context of the namespace handler for each bean instance that requires runtime enhancement.

- Create mandatory or optional service references added to the component definitions for the bundle being parsed for each interceptor services registered for a bean by this namespace handler.

5.2 The Interceptor Interface

All interceptors must implement the common interface to provide for pre, post, and error Interceptor processing on a bean method invocation. The proposed API for Interceptor is as follows:

```java
/**
 * An Interceptor interface provides support for custom interceptor implementation.
 */
public interface Interceptor {

/**
 * This is called just before the method m is invocation.
 * @param cm : the component's metadata
 * @param m: the method to be invoked
 */
```
5.3 Processing Interceptors

It is the responsibility of the Blueprint Bean Manager to locate and manage the interceptors as associated with Blueprint Beans. The Blueprint Bean Manager is also responsible to construct Blueprint Bean instances. The Blueprint Bean Manager must therefore ensure that it constructs Bean instances that will honor the Interceptors that have been defined. This would involve the following steps:

1. Interrogating component definitions for service references that registered with the Interceptor interface for this bean.

2. Construct an instance of the Bean in such a way that future invocations of methods on the bean will invoke the Interceptors that have been defined.

3. Invoking the interceptors on each method invocation. The mechanisms used to instantiate Blueprint Beans are not prescribed in the Blueprint Container RFC [5]. using specific objects and are therefore left as an internal implementation detail of a Blueprint Bean Manager. Therefore, this RFC will not call out explicit constructs or mechanisms (such as a BeanRecipe) that should be involved in the interrogation of the interceptors or construction of a bean that will honor the intention of the interceptors. This RFC will simply specify the expected behavior and assume that this is implemented by the Bean Manager as appropriate. However, one possible mechanism that can be employed would be to allocate a proxy
object for the bean that would defer to the interceptors before and after each method call against the Bean.

4. If multiple interceptors are associated with a bean they must be invoked in the order specified by the rank specified in the Service for the interceptor. The processing should invoke the highest ranked interceptor preCall method and add the knowledge of that interceptor and returned correlator to a logical “stack” for later use with the Bean method returns. Processing should then continue with the next highest interceptor until all interceptor preCall methods have been invoked. The bean’s method should then be invoked. When processing the result of the bean method (either the return value or exception) interceptors would be removed from the logical “stack” to process the appropriate postCall* method. This will effectively invoke the postCall* methods in reverse order from the preCall methods as interceptors are removed from the stack and provide symmetry on the calls.

5. Care must be taken to preserve the correlator returned from the preCall Interceptor method invocations so that they can be passed into the appropriate postCall* method invocation after the Bean method completes.

6. Any exceptions returned from an Interceptor preCall or postCall* method should be logged and interrupt program flow. (see 5.5).

7. Interceptors should not modify the method, parameters, results or exceptions provided to or returned by a Blueprint Bean method invocation. These parameters are only provided to the interceptor for reference. Introducing a solution to protect the bean data is recommended but not required. The only case where it is appropriate to make any change is if the Interceptor itself has a need to thrown an exception during processing in which case it may override a bean method exception (see 5.5).

5.4 Ordering Interceptors (Rank)

Order of Interceptors is determined by the self defined rank provided by the Namespace Handler when the Interceptor Service is registered. The rank is defined as a simple integer and is processed from highest to lowest. In the event of multiple Interceptors with the same rank the order is based upon the order of registration which is evident using the service id. The service with the lower service id was registered earlier (see section 5.2.5 of OSGi core specification R4V42).

Multiple interceptors associated with a single bean must be processed in order by rank and placed in a logical stack for processing. This is to provide symmetry for interceptor processing. Therefore, if there are 2 interceptors prioritized as A and B the processing order would be the following:

- A.preCall
- B.preCall
- bean method invocation
- B.postCall*
- A.postCall*

5.5 Error processing for Interceptors

There are cases where an interceptor may need thrown an exception. This is an indication that an error has occurred in the interceptor and processing either cannot or should not continue. One potential use of this function
by an Interceptor would be to block the bean itself and additional interceptors from being invoked. For example, a validation interceptor may have a need to block subsequent access to additional interceptors and the bean method if the appropriate credential was not present. Any exception thrown from within an Interceptor in the **preCall** or **postCall** method processing should be treated as a termination condition. The exception should be logged, and re-thrown.

An exception can be thrown from an Interceptor in the **preCall**, **postCallWithReturn**, or **postCallWithException**.

1. **preCall** - In the event an exception is thrown by an Interceptor in a **preCall** the bean method should not be invoked and all Interceptors that have not yet been processed should be ignored. The **postCallWithException** methods of each interceptor that has already processed the **preCall** should be executed in the correct order before the exception is propagated back to the caller.

2. **postCallWithReturn** - In the event an exception is thrown by an Interceptor in a **postCallWithReturn** method the exception should be propagated and all remaining Interceptors should be invoked in the appropriate order with the **postCallWithException** method to indicate the error processing.

3. **postCallWithException** – In the event an exception is thrown by an Interceptor in the **postCallWithException** method the exception should be propagated and all remaining Interceptors should be invoked in the appropriate order using the **postCallWithException** method to indicate error processing.

4. If multiple exceptions are thrown from multiple Interceptors in multiple methods (**preCall** or **postCall**) the first exception thrown should be the one propagated back to the caller.

The blueprint bean itself may also throw an exception. If this is the case, the exception should be propagated back to the caller if it is not a **RuntimeException**. In the case of a **RuntimeException** it should be returned to the caller if and only if an exception was not also thrown from an Interceptor while processing the **postCallWithException**. Basically, an exception thrown by an Interceptor is considered of greater importance than a **RuntimeException** from the bean.

### 5.6 Correlating pre/post/exception Interceptor calls

The **preCall** method returns a correlator Object that can be used by the Interceptor to match **preCall** and **postCall** method invocations. The Object can be anything that is useful for the Interceptor and will simply be preserved and returned on matching methods. The Bean manager will ensure that a bean definition that includes interceptors will invoke the **preCall** method on the associated interceptor(s) and save the correlator for subsequent processing with the same interceptor **postCall** method.

### 5.7 Interceptor Lifecycle

There is a high cohesion between an interceptor and the namespace handler that requires the interceptor capability to fulfill it's purpose. It was considered if the Interceptor should be managed independently of the namespace handler. This seems to be an approach that would introduce a lot of complexity for very little benefit. Any changes in a namespace handler that leverages the interceptor support would most likely result in changes to the corresponding interceptor to react to schema or other changes in the namespace. Furthermore, it is unlikely that there would be changes in an underlying interceptor without corresponding changes in the namespace handler. Therefore, for the purposes of this design it is assumed that the lifecycle of an interceptor is consistent with the lifecycle of a namespace handler and most likely both will be delivered in the same bundle.
Because the namespace handler is the primary element that registers and manages any associated interceptors it seems appropriate that lifecycle scenarios are primarily (and perhaps exclusively) driven from the namespace handler. The namespace specification should address the issue of managing blueprint containers that have been extended by the namespace handler such that the blueprint bundle can be updated (perhaps re-parsed) if the namespace handler is updated. The creation, removal, or updating of associated interceptors via advertised interceptor services in the service registry and corresponding service references in the blueprint components would be managed by the namespace handler implementation following guidelines from the blueprint namespace handler specification.

6 Command Line API

If this specification would benefit from a command line interface, describe it here. Commands should be realized as described in RFC 147.

At this point in time we do not envision a command line API that corresponds to interceptors.

7 JMX API

RFC 139 describes the JMX API to the OSGi Framework and a number of standard OSGi Services. The JMX specification is also present as chapter 124 in the OSGi spec documents.

For all new functionality added to the OSGi Framework the question should be asked: would this feature benefit from a JMX API? The expectation is that in most cases it would.

The JMX API for the design in this RFC should be described here and if there is no JMX API an explanation should be given explaining why this is not applicable in this case.

At this point in time we do not envision an specific JMX APIs for Interceptors.
8 Initial Spec Chapter

Provide a link to where the Initial Spec Chapter can be found. The Initial Spec Chapter is typically written by someone other than the author(s) of this RFC and represents a rewrite of this document as close as possible to what will ultimately appear in the OSGi Specifications. It will be used by the Specification Editor as the basis for the ultimate specification chapter.

The spec template and writing guidelines can be found here:


9 Considered Alternatives

For posterity, record the design alternatives that were considered but rejected along with the reason for rejection. This is especially important for external/earlier solutions that were deemed not applicable.

9.1 Bytecode Weaving required implementation

Bytecode Weaving is another alternative that could be considered to implement Bean Interceptor capability. For details on Bytecode Weaving reference [6] and [7].

There is nothing in this design that prohibits a Bytecode Weaving solution. A Bytecode weaving solution could be triggered by a Namespace Handler which could weave the interceptor logic into the specified bean. Mechanisms similar to those defined in this design would still be necessary to prioritize and coordinate the activities of multiple interceptors and maintain knowledge of them in the Component Definition Registry. At this point in time it appears that requiring a Bytecode weaving solution would not result in a simplified or vastly superior solution and would appear to bring more complexity into the design for little benefit. Therefore, a Bytecode Weaving approach was not pursued as the primary solution for this design.
10 Security Considerations

Description of all known vulnerabilities this may either introduce or address as well as scenarios of how the weaknesses could be circumvented.

11 Document Support

11.1 References


[3]. RFC 155 – Blueprint Namespaces is available in the OSGi Alliance subversion repository in rfcs/rfc0155/rfc-155-blueprint-namespaces.pdf.

[4]. RFC 164 – Blueprint Declarative Transaction is available in the OSGi Alliance subversion repository in rfcs/rfc0164/rfc-0164-blueprint_transaction.pdf.

[5]. RFC 124 – Blueprint Service is available in the OSGi Alliance subversion repository in rfcs/rfc0124/rfc-124.pdf.

[6]. RFP 139 – Bytecode Weaving Requirements is available in the OSGi Alliance subversion repository in rfps/rfp-0139-bytecode-weaving.pdf

[7]. RFC 159 – Bytecode Weaving Design is available in the OSGi Alliance subversion repository in rfcs/rfc0159/rfc-159-BytecodeWeaving.pdf.

[8]. RFP 137 – Blueprint Bean Interceptors Requirement is available in the OSGi Alliance subversion repository in rfps/rfp-0137-Blueprint_Bean_Interceptors.pdf

11.2 Author’s Address
11.3 Acronyms and Abbreviations

11.4 End of Document
Abstract

The Sun SPI model is a plug-in model widely used in the Java platform. However, there are issues with this model when used in an OSGi Framework.

This RFC proposes a solution to this problem in order to make the SPI model usable from within OSGi.
# 0 Document Information

## 0.1 Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 Document Information</td>
<td>2</td>
</tr>
<tr>
<td>0.1 Table of Contents</td>
<td>2</td>
</tr>
<tr>
<td>0.2 Terminology and Document Conventions</td>
<td>3</td>
</tr>
<tr>
<td>0.3 Revision History</td>
<td>3</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>4</td>
</tr>
<tr>
<td>2 Application Domain</td>
<td>4</td>
</tr>
<tr>
<td>2.1 SPI Providers</td>
<td>4</td>
</tr>
<tr>
<td>2.2 SPI Consumers</td>
<td>4</td>
</tr>
<tr>
<td>3 Problem Description</td>
<td>4</td>
</tr>
<tr>
<td>3.1 FactoryFinder</td>
<td>5</td>
</tr>
<tr>
<td>3.2 ServiceLoader</td>
<td>6</td>
</tr>
<tr>
<td>3.3 JavaMail and non-empty constructors</td>
<td>6</td>
</tr>
<tr>
<td>4 Requirements</td>
<td>7</td>
</tr>
<tr>
<td>4.1 Provider side Requirements</td>
<td>7</td>
</tr>
<tr>
<td>4.2 Client side Requirements</td>
<td>7</td>
</tr>
<tr>
<td>5 Technical Solution</td>
<td>7</td>
</tr>
<tr>
<td>5.1 Scope</td>
<td>7</td>
</tr>
<tr>
<td>5.2 Providers</td>
<td>8</td>
</tr>
<tr>
<td>5.2.1 OSGi Service Registrations</td>
<td>8</td>
</tr>
<tr>
<td>5.3 OSGi-aware consumers</td>
<td>9</td>
</tr>
<tr>
<td>5.4 Non-OSGi-aware Consumers</td>
<td>9</td>
</tr>
<tr>
<td>5.4.1 ServiceLoader.load(Class c)</td>
<td>9</td>
</tr>
<tr>
<td>5.4.2 ServiceLoader.load(Class c, ClassLoader ci)</td>
<td>10</td>
</tr>
<tr>
<td>5.4.3 SPI-Consumer header</td>
<td>10</td>
</tr>
<tr>
<td>5.5 Handling JRE-provided implementations</td>
<td>11</td>
</tr>
<tr>
<td>5.6 RFC 153 Generic Requirement</td>
<td>11</td>
</tr>
<tr>
<td>5.7 Examples</td>
<td>11</td>
</tr>
<tr>
<td>5.7.1 Bundle directly using ServiceLoader.load()</td>
<td>11</td>
</tr>
<tr>
<td>5.7.2 Bundle using a library that collects several service implementations using ServiceLoader.load()</td>
<td>12</td>
</tr>
<tr>
<td>5.8 MultiDelegationClassLoader</td>
<td>13</td>
</tr>
<tr>
<td>6 Command Line API</td>
<td>14</td>
</tr>
<tr>
<td>7 JMX API</td>
<td>15</td>
</tr>
</tbody>
</table>
8 Initial Spec Chapter.................................................................................................................... 15

9 Considered Alternatives............................................................................................................. 15
  9.1 Technical Solution.................................................................................................................. 15
     9.1.1 'Legacy' consumers........................................................................................................ 15
  9.2 Examples.............................................................................................................................. 16
     9.2.1 Bundle using JAXP DocumentBuilderFactory.newInstance()....................................... 16
     9.2.2 Bundle using library that uses ServiceLoader.load() and DocumentBuilderFactory.newInstance().................................................................................................................. 17
     9.2.3 Bundle using JAXP from an Apache bundle, not from the System bundle nor any other provider.................................................................................................................................... 17
     9.2.4 Bundle using JAXP from an from the System bundle not from any other provider bundle 18
     9.2.5 JAXB anomaly................................................................................................................ 18
     9.2.6 Obtaining the SPI service from the OSGi Service Registry........................................... 18

10 Security Considerations............................................................................................................ 18

11 Document Support.................................................................................................................... 19
  11.1 References.......................................................................................................................... 19
  11.2 Author's Address.................................................................................................................. 19
  11.3 Acronyms and Abbreviations.............................................................................................. 19
  11.4 End of Document.................................................................................................................. 20

0.2 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 11.1.

Source code is shown in this typeface.

0.3 Revision History

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

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>October</td>
<td>David Bosschaert, initial version</td>
</tr>
<tr>
<td>0.2</td>
<td>January</td>
<td>David Bosschaert, enhanced technical solution section</td>
</tr>
<tr>
<td>0.3</td>
<td>March</td>
<td>David Bosschaert, feedback from EEG concall review</td>
</tr>
<tr>
<td>0.4</td>
<td>April</td>
<td>David Bosschaert, restricted solution to java.util.ServiceLoader, moved legacy support to Considered Alternatives. Also made the diagram in 5.7.1 more explicit.</td>
</tr>
</tbody>
</table>
1 Introduction

The Sun SPI model is a plug-in model widely used in the Java platform. In fact, many implementations of JSR standards are plugged into the JRE using the Sun SPI model. However, there are issues with this model when used in an OSGi Framework.

This RFC proposes a solution to this problem in order to make the SPI model usable from within OSGi and enable this model for both existing applications that may have been developed outside of OSGi as well as for new applications that haven been developed as OSGi bundles.

2 Application Domain

The application domain is any OSGi-based Java application that uses a third-party library which relies on the Sun SPI model.

There are two sides to the problem, the SPI provider side and the SPI consumer side.

2.1 SPI Providers

SPI providers are typically Jar files that contain an implementation of a standardized interface. An SPI provider could be OSGi-aware or non-OSGi-aware.

2.2 SPI Consumers

SPI consumers are clients of a standardized API that use it to obtain an implementation which is provided through the Sun SPI mechanism. These clients could be OSGi-aware or non-OSGi-aware.

3 Problem Description

In Java the most widely used pluggability model is the Sun SPI model. This model allows plugging in an implementation of typically a standardized interface such as a JAXP compliant XML parser, a JSR 311 compliant REST reader or writer or a SOAP implementation.
### 3.1 FactoryFinder

The SPI model is generally based on the following mechanics, although the implementations vary slightly across the board.

It generally uses a 'FactoryFinder' class which tries to obtain a Factory class which binds it to a certain implementation.

The Factory Finder typically does two things:

1. First it tries to work out the name of the factory class implementation for factory with id x.y.Z, typically using the following algorithm:
   
a) It checks for the existence of a system property x.y.Z that might hold the class name.

   b) It looks for a well known properties file in $java.home/lib which might hold a value for the x.y.Z key.

   c) It tries to load the resource META-INF/services/x.y.Z using the Thread Context Classloader, the System Classloader or the classloader that loaded the Factory Finder class. If found it reads the implementation class name from the resource.

   d) If all of the above fails it uses a fall-back class name which is typically hard coded in the JRE implementation.

2. Once a class name has been obtained, the Factory Finder will try to load the class.

   a) In some cases an actual classloader is passed in to the Factory Finder in which case that classloader is used.

   b) Otherwise the Thread Context Classloader is tried or if that isn't set the System Classloader is used.

   c) If that fails the Factory Finder calls Class.forName() with its own classloader as the classloader argument.

This pattern has a number of disadvantages when used in an OSGi framework:

- A typical 3rd party library implementation that uses the SPI model relies on step 1c, where a resource is loaded from the META-INF/services directory through a classloader. In OSGi frameworks this resource is normally not visible outside of the library as only exported packages are made visible to classloaders outside of the bundle. Exporting the META-INF/services directory as a package is not an option as many libraries might have this directory but the OSGi framework can only resolve a package to a single bundle.

- Even if the class name can be obtained somehow in the OSGi framework, the loading of the implementation class will end up being challenging as it does not take the classloader of the bundle into account.

  This can often be worked around by setting the Thread Context Classloader to the bundle's classloader, but this requires modifications to the user's bundle code.

- To be loadable by another bundle the implementation class needs to be in a exported package of the providing bundle. This is generally undesirable as it exposes packages internal to an implementation outside its bundle, something highly discouraged in OSGi modularity.
Besides the above issues, there is also a lifecycle issue in general with the SPI pattern as the Factory Finder often uses static variables and hence the Factory can only be set once in the life time of the VM in certain scenarios.

Since the Sun SPI model is highly prevalent in the Java library ecosystem and in fact the standard mechanism used within the JDK itself, an OSGi developer should not have to worry about getting around the problems with this mechanism. It should just work.

Other OSGi specifications have addressed this issue on a case-by-case basis, but a general solution to this problem is not available.

The OSGi Alliance should create a generic mechanism to deal with the Sun SPI model so that libraries utilizing this model can be used in an OSGi framework.

## 3.2 ServiceLoader

The JRE also contains a class called java.util.ServiceLoader which provides a general algorithm for this finding SPI implementors. The algorithm is different than the one described above in that it only visits the META-INF/services. It doesn't involve system properties, nor files in the java home directory. It also doesn't have a default class name built in.

Additionally, the ServiceLoader class allows the client to obtain all the providers, where the FactoryFinder only selects a single one using the rules above.

## 3.3 JavaMail and non-empty constructors

The JavaMail API uses a SPI mechanism as well, but in a slightly different way. Classes are specified in a META-INF/javamail.properties file with a number of attributes, like this:

```properties
# JavaMail IMAP provider Sun Microsystems, Inc
protocol=imap; type=store; class=com.sun.mail.imap.IMAPStore; vendor=Sun;
protocol=imaps; type=store; class=com.sun.mail.imap.IMAPSSLStore; vendor=Sun;
```

The JavaMail framework expects the class specified to have a Constructor that has the following signature:

```java
IMAPStore(Session session, URLName url)
```

When a consumer calls one of the Session.getStore() APIs this will result in the JavaMail framework to instantiate the correct class with the constructor as specified above.

During the RFP stage it has been decided that an anomaly like this would not be supported by this standard.
4 Requirements

4.1 Provider side Requirements

• SP01 – SPI service providers should be automatically registered in the service registry.

• SP02 – The solution should work with providers that are not OSGi-aware. These providers will need to be bundelized, however.

• SP03 – Providers need to opt-in using a Manifest Header.

• SP04 – Bundelized providers should support the full bundle lifecycle and should therefore be able to be uninstalled.

4.2 Client side Requirements

• SP10 – Non-OSGi-aware clients should work without modifications to the source code. It is not expected that these clients will get OSGi lifecycle benefits.

• SP11 – OSGi-aware Clients should be able to use OSGi services to obtain SPI implementations. These clients must get OSGi lifecycle benefits wrt to the SPI implementations.

• SP12 – The semantics for non-OSGi-aware clients must not change. For example clients must receive new SPI instances for every `ServiceLoader.load()` invocation.

5 Technical Solution

The technical solution aims to address both OSGi-aware use-cases as well as non-OSGi-aware use-cases.

Providers and consumers are described separately as they need to be handled in different ways.

5.1 Scope

The technical solution focuses on the how the bundle developer interacts with the system to get the desired behavior. There is a multitude of options available to achieve the specified behavior including RFC 159-based load-time weaving, static weaving, install-level interception, framework-level classloader control and possibly others. This RFC leaves the implementation choice to the implementor.
A pilot implementation is available in Apache Aries which uses an RFC 159 WeavingHook and ASM for the consumer side. On the provider side a BundleTracker is used.

5.2 Providers

SPI Service Providers advertise their services through resources that are present in the META-INF/services directory of their jar file. This mechanism is supported by the JavaSE 6 java.util.ServiceLoader class and is also used in other implementations that predate Java 6. The following pattern is typically used (and this is the only one supported by this RFC): for every resource in the META-INF/services directory its name represents a Java class name or interface name. The contents of the resource is in text file format and lists implementation classes, one class per line. The implementation classes extend or implement the class or interface with the same name as the resource name.

For example, take a resource:

- META-INF/services/org.acme.MyService
  with contents:
  org.acme.impl1.MyService1
  org.acme.impl2.MyService2

This means that the jar file contains the classes MyService1 and MyService2 which implement or extend the MyService class. The MyService class does not need to be present in the jar file, but needs to be visible to the classloader that loads the jar file.

Note that it's quite common that the actual resource in the META-INF/services directory is actually located in a Jar file which is embedded in the bundle. The discovery process needs to take this into account.

This RFC describes that Service Providers are handled in two ways by the system: firstly they are registered in the Service Registry to support OSGi-aware consumers, secondly the providing bundle or its classloader is recorded in an internal data structure which is used to support non-OSGi-aware consumers.

To enable OSGi support, Service Provider bundles need to opt in to the process by specifying the SPI-Provider header in the bundle manifest, using one of the following:

1. SPI-Provider: *

   When specified as such any services found in the META-INF/services directory will be used.

2. SPI-Provider: org.acme.Service1, org.acme.Service2

   This allows a bundle to specify a subset of the services available in META-INF/services which should be enabled in OSGi.

5.2.1 OSGi Service Registrations

Service Providers are registered in the OSGi Service Registry as a ServiceFactory under the class name of the META-INF/services resource that declares them. Every consuming bundle will receive a separate instance of the service, which is similar to the semantics followed by java.util.ServiceLoader.load(). The objects will be instantiated using a no-arg constructor. And the Service Factories are registered with the bundle context of the bundle that provides the implementation.

Service Properties
<table>
<thead>
<tr>
<th>Property</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spi.provider.url</td>
<td>String</td>
<td>A URL in String form that resolves to the resource that contained the service declaration. Besides being able to trace back where the service came from, the property also indicates that the service was registered through OSGi SPI support.</td>
</tr>
</tbody>
</table>

5.3 OSGi-aware consumers

OSGi-aware consumers can obtain the services form the OSGi Service Registry. This is the preferred mechanism of interaction as it provides proper lifecycle support in case a service provider bundle is removed and/or a new service provider bundle is added to the system.

5.4 Non-OSGi-aware Consumers

Non-OSGi-aware consumers obtain service instances by calling `java.util.ServiceLoader.load()`.

Non-OSGi-aware Consumers can be made to work in an OSGi environment, but they do suffer from the disadvantage that they won't get lifecycle support, i.e. they will not get notified when a service provider is uninstalled. This is a consequence of the lack of lifecycle support in the design of `java.util.ServiceLoader`.

5.4.1 `ServiceLoader.load(Class c)`

As of Java 6 `java.util.ServiceLoader.load(Class c)` is the suggested API that SPI Service Consumers use. This is a static API of which returns an instance of the ServiceLoader class which in itself is Iterable; iterating through the result will produce the actual service instances found. `ServiceLoader` is a final class and has a private constructor so it's impossible to return an alternative implementation of `ServiceLoader`. The only option we have here is to influence the way `ServiceLoader` works within its current design. `ServiceLoader` internally uses the `ThreadContextClassLoader` to find service implementations. To make the API work in an OSGi environment we must set the TCCL to a classloader with visibility of the appropriate bundles for the duration of the `ServiceLoader.load()` call.

A consumer bundle that wishes to use this behavior for the duration of `ServiceLoader.load(Class c)` can simply opt-in by specifying the following header:

```
SPI-Consumer: *
```

The above includes any suitable provider bundle in the `ServiceLoader.load()` process.

5.4.1.1 Targeting specific bundles

To target a specific bundle or bundles to take part in `ServiceLoader.load()` requests, the provider bundle can be specified in the SPI-Consumer header. The bundle can be specified by Symbolic Name and Version or Bundle ID. Optionally a version can be specified.

```
SPI-Consumer: *;bundle=myBundle1
```

load SPI providers from myBundle1, as no version is specified, the highest available version is selected.

```
SPI-Consumer: *;bundle=myBundle1:version=1.2|myBundle2:version=2
```
load SPI providers from myBundle1 with version 1.2 or myBundle2 with version 2

SPI-Consumer: *;bundleId=0

load SPI providers from the bundle with ID 0, i.e. the system bundle.

5.4.1.2 Targeting specific bundles per service type

It is possible to distinguish which bundle or set of bundles should take part in ServiceLoader.load() requests based on the actual service type requested. The following header shows this:

SPI-Consumer: org.acme.MyService1;bundle=myBundle1,org.acme.MyService2;bundle =myBundle2|myBundle3

Calls to java.util.ServiceLoader.load(org.acme.MyService1.class) are only handled in the context of myBundle1, while calls to java.util.ServiceLoader.load(org.acme.MyService2.class) are handled in the context of myBundle2 and myBundle3.

5.4.2 ServiceLoader.load(Class c, ClassLoader cl)

Consumers using this API don't need special treatment as they already provide a classloader to be used when looking up the service implementation.

5.4.3 SPI-Consumer header

In general, the SPI-Consumer header has the following syntax:

SPI-Consumer: (**|fqcn){;attributes}{,(**|fqcn){;attributes}}*

The header enumerates all the SPI consumer API calls that need to be treated to obtain the correct service instance(s).

<table>
<thead>
<tr>
<th>Element</th>
<th>Example(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fqcn</td>
<td>*</td>
<td>When '*' is specified all ServiceLoader.load() calls are treated, when a class name is specified, only ServiceLoader.load() calls for that specific SPI are treated.</td>
</tr>
<tr>
<td></td>
<td>org.acme.MyService</td>
<td></td>
</tr>
</tbody>
</table>

Supported attributes are:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Example(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bundle</td>
<td>bundle=mySPIImp1</td>
<td>mySpilmp3</td>
</tr>
<tr>
<td></td>
<td>bundle=mySPIBundle:version=1.2.3</td>
<td>A list of bundle symbolic names of bundles that are eligible to provide the service for this consumer. If this attribute is not specified then any bundle qualifies. The listed bundles can be marked with a version suffix to target a specific version.</td>
</tr>
<tr>
<td>BundleId</td>
<td>bundleId=0</td>
<td>The bundle ID or IDs of bundles that are eligible to provide the service.</td>
</tr>
<tr>
<td></td>
<td>bundleId=2</td>
<td>3</td>
</tr>
</tbody>
</table>
5.5 Handling JRE-provided implementations

These can be selected by requesting that they be provided from bundle with ID 0.

TODO what is the behavior if no bundle restriction is specified? Should the JRE be included?

TODO add a more detailed example

5.6 RFC 153 Generic Requirement

TODO generic requirement namespace for a bundle to declare that it requires a framework which supports RFC 167

*** There is an issue with the fact that generic capabilities/requirements are currently resolve-time and this would require active-time capabilities.

5.7 Examples

5.7.1 Bundle directly using ServiceLoader.load()

A consumer bundle uses the ServiceLoader.load() API directly.
5.7.2 Bundle using a library that collects several service implementations using ServiceLoader.load()

The ServiceLoader.load() API can return multiple instances of the service. In turn these services should be allowed to come from various bundles.

* This diagram only depicts the logical flow. The actual loading of the resource and instantiation happens lazily on the ServiceLoader iterator callback.
ServiceLoader internally only calls ClassLoader.getResources() once, however the desired result can be achieved by using a delegation classloader that delegates to a number of other classloaders, like the MultiDelegationClassLoader from section 5.8 as the Thread Context Classloader for the duration of the ServiceLoader.load() call.

5.8 MultiDelegationClassLoader

This section contains a simple implementation of a classloader that delegates to multiple other classloaders. Since the SPI mechanisms can only take 1 classloader as the thread context classloader a classloader like this one can be useful to allow the system to find implementations of an SPI divided over multiple bundles. In this case you can create a MultiDelegationClassLoader that has the classloader of all the bundles involved embedded. Then you set this MultiDelegationClassLoader as the TCCL for the duration of the SPI lookup call.

TODO do we need privileged blocks here?

TODO is it possible at all to handle the case where multiple delegates can load the same class (same classname)?

```java
import java.io.IOException;
import java.io.InputStream;
import java.net.URL;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Enumeration;

public class MultiDelegationClassLoader extends ClassLoader {
    private final ClassLoader[] delegates;

    public MultiDelegationClassLoader(ClassLoader ... classLoaders) {
        if (classLoaders == null)
            throw new NullPointerException();

        delegates = classLoaders.clone();
    }

    @Override
    public URL getResource(String name) {
        for (ClassLoader cl : delegates) {
            URL res = cl.getResource(name);
            if (res != null)
                return res;
        }
        return null;
    }

    @Override
    public InputStream getResourceAsStream(String name) {
        for (ClassLoader cl : delegates) {
            InputStream is = cl.getResourceAsStream(name);
            if (is != null)
                return is;
        }
        return null;
    }
}
```

Copyright © Red Hat 2011 All Rights Reserved
6 Command Line API

If this specification would benefit from a command line interface, describe it here. Commands should be realized as described in RFC 147.
7  JMX API

RFC 139 describes the JMX API to the OSGi Framework and a number of standard OSGi Services. The JMX specification is also present as chapter 124 in the OSGi spec documents.

For all new functionality added to the OSGi Framework the question should be asked: would this feature benefit from a JMX API? The expectation is that in most cases it would.

The JMX API for the design in this RFC should be described here and if there is no JMX API an explanation should be given explaining why this is not applicable in this case.

8  Initial Spec Chapter

Provide a link to where the Initial Spec Chapter can be found. The Initial Spec Chapter is typically written by someone other than the author(s) of this RFC and represents a rewrite of this document as close as possible to what will ultimately appear in the OSGi Specifications. It will be used by the Specification Editor as the basis for the ultimate specification chapter.

The spec template and writing guidelines can be found here:


9  Considered Alternatives

9.1  Technical Solution

9.1.1  'Legacy' consumers

Before the ServiceLoader class became part of the JRE in Java 6, a number of technologies existed that used their own proprietary classes to handle providers that advertise themselves through resources in the META-
INF/services directory. Examples of such technologies include JAXP, JAXB, JAX-WS, media codecs and others. Although these technologies have their own implementation of a ‘ServiceLoader’ class, (often called ‘FactoryFinder’), the mechanism that they use is roughly similar and is generally also influenced by the Thread Context Classloader. So a treatment similar to that of the `ServiceLoader.load(Class)` method is also helpful here.

Legacy consumers are also configured through the SPI-Consumer header, e.g to select the JAXP from a bundle with BSN apache-xerces, specify the header as follows:

```
SPI-Consumer: javax.xml.parsers.DocumentBuilderFactory#newInstance();bundle=apache-xerces
```

While JAXP can be selected from the JRE by obtaining it through the System Bundle:

```
SPI-Consumer: javax.xml.parsers.DocumentBuilderFactory#newInstance();bundleId =0
```

There is a difference in how `ServiceLoader.load()` is treated because `ServiceLoader.load()` is a generic mechanism: the service API is provided as the argument to `ServiceLoader.load()` while in the other cases the class on which the invocation is made is itself the API. To clarify, a simple header of

```
SPI-Consumer: javax.xml.parsers.DocumentBuilderFactory#newInstance()
```

Will enable the SPI consumer mechanism for any provider of the DocumentBuilderFactory service.

While

```
SPI-Consumer: java.util.ServiceLoader#load(java.lang.Class[org.acme.MySvc])
```

Will enable the mechanism for the MySvc service when obtained via `ServiceLoader.load()`.

### 9.2 Examples

#### 9.2.1 Bundle using JAXP DocumentBuilderFactory.newInstance()

In this case the consumer bundle directly calls a JAXP API:

```
javax.xml.parsers.DocumentBuilderFactory.newInstance()
```

As this API deviates from the standard `ServiceLoader.load()` the SPI-Consumer header in the consumer bundle needs to specify the actual API.
9.2.2 Bundle using library that uses ServiceLoader.load() and DocumentBuilderFactory.newInstance()

An ordinary bundle uses an API which in turn uses both JAXP as well as another SPI provider bundle.

My Bundle
Import-Package: org.acme.foo

Library Bundle
Export-Package: org.acme.foo
Import-Package javax.xml.parsers
SPI-Consumer: java.util.ServiceLoader#load(),
javax.xml.parsers.DocumentBuilderFactory#newInstance()

JAXP Impl Bundle
SPI-Provider: *

SPI Impl Bundle
SPI-Provider: *

The SPI-Consumer header needs to be in the Library bundle. The consumer bundle that indirectly consumes the SPI services does not need any specific headers.

9.2.3 Bundle using JAXP from an Apache bundle, not from the System bundle nor any other provider

JAXP Consumer Bundle
Import-Package javax.xml.parsers
SPI-Consumer:
javax.xml.parsers.DocumentBuilderFactory#newInstance();
bundle=org.apache.xerces;bundle-version=1.8.10

SPI Impl 1
BSN: System
SPI-Provider: *

SPI Impl Bundle 2
BSN: o.a.xerces
B-Version: 1.8.8
SPI-Provider: *

SPI Impl Bundle 3
BSN: o.a.xerces
B-Version: 1.8.10
SPI-Provider: *
9.2.4 Bundle using JAXP from an System bundle not from any other provider bundle

JAXP Consumer Bundle

Import-Package: javax.xml.parsers
SPI-Consumer:
  javax.xml.parsers.DocumentBuilderFactory#newInstance();
  bundle-id=0

<table>
<thead>
<tr>
<th>SPI Impl 1</th>
<th>SPI Impl Bundle 2</th>
<th>SPI Impl Bundle 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSN: System</td>
<td>BSN: o.a.xerces</td>
<td>BSN: o.a.xerces</td>
</tr>
<tr>
<td></td>
<td>B-Version: 1.8.8</td>
<td>B-Version: 1.8.10</td>
</tr>
<tr>
<td></td>
<td>SPI-Provider: *</td>
<td>SPI-Provider: *</td>
</tr>
</tbody>
</table>

9.2.5 JAXB anomaly

JAXB is initialized by calling JAXBContext.newInstance(). The files in the META-INF/services are indeed called javax.xml.bind.JAXBContext but the class name in there refers to a class called com.sun.xml.bind.v2.ContextFactory which is not a subclass of JAXBContext. On that class a method called createContext(String, Classloader, Map) is called which returns a JAXBContext object.

Open question: what will we do with JAXB? Will we attempt to support this?

9.2.6 Obtaining the SPI service from the OSGi Service Registry

10 Security Considerations

Description of all known vulnerabilities this may either introduce or address as well as scenarios of how the weaknesses could be circumvented.
11 Document Support

11.1 References


Add references simply by adding new items. You can then cross-refer to them by choosing <Insert><Cross Reference><Numbered Item> and then selecting the paragraph. STATIC REFERENCES (I.E. BODGED) ARE NOT ACCEPTABLE, SOMEONE WILL HAVE TO UPDATE THEM LATER, SO DO IT PROPERLY NOW.

11.2 Author’s Address

<table>
<thead>
<tr>
<th>Name</th>
<th>David Bosschaert</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Red Hat</td>
</tr>
<tr>
<td>Address</td>
<td>6700 Cork Airport Business Park</td>
</tr>
<tr>
<td></td>
<td>Kinsale Road</td>
</tr>
<tr>
<td></td>
<td>Cork</td>
</tr>
<tr>
<td></td>
<td>Ireland</td>
</tr>
<tr>
<td>Voice</td>
<td>+353 21 230 3400</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:david@redhat.com">david@redhat.com</a></td>
</tr>
</tbody>
</table>

11.3 Acronyms and Abbreviations
11.4 End of Document
RFC 169 - JMX Update

Draft

16 Pages

Abstract

Chapter 124 of the OSGi Enterprise Specification describes the how the OSGi Framework can be managed through the JMX Management Model. This RFC updates the OSGi-JMX specification to follow the OSGi 4.3 Core framework updates. Additionally it addresses a number of bugs filed in relation to the OSGi-JMX specification.
1 Document Information

1.1 Table of Contents

1 Document Information ............................................................................................................................ 2
  1.1 Table of Contents ............................................................................................................................ 2
  1.2 Terminology and Document Conventions ....................................................................................... 3
  1.3 Revision History .............................................................................................................................. 3

2 Introduction ............................................................................................................................................. 3

3 Application Domain ................................................................................................................................ 4

4 Problem Description ................................................................................................................................ 4
  4.1 OSGi 4.3 Core Framework Update (Bug 1762) .............................................................................. 4
  4.2 JBoss OSGi-JMX extensions .......................................................................................................... 4
    4.2.1 BundleStateMBean ................................................................................................................. 4
    4.2.2 ServiceStateMBean ............................................................................................................... 5
    4.2.3 FrameworkMBean .................................................................................................................. 5
  4.3 Bug 1846 OSGi MBeans do not target a particular OSGi framework instance ............................... 5
  4.4 Email from Jürgen Kissner / SAP .................................................................................................. 5
  4.5 Bug 1592 UserAdminMBean - differences between spec and RI ................................................. 6
  4.6 Bug 1616 Add properties support ................................................................................................ 6
  4.7 Bug 1645 Invalid RuntimeException handling in CT ..................................................................... 6
  4.8 Bug 1646 UserAdminMBean issues ............................................................................................... 6
  4.9 Bug 1647 BundleStateMBean.getRequiredBundles() clarification ............................................ 7
  4.10 Bug 1649 ConfigurationAdminMBean clarification .................................................................... 7

5 Requirements ........................................................................................................................................... 7

6 Technical Solution .................................................................................................................................. 7
  6.1 OSGi MBean Object Names ............................................................................................................. 7
  6.2 BundleStateMBean .......................................................................................................................... 8
    6.2.1 Headers ..................................................................................................................................... 9
    6.2.2 Requiring Bundles .................................................................................................................. 9
  6.3 ServiceStateMBean .......................................................................................................................... 9
  6.4 FrameworkMBean .......................................................................................................................... 10
  6.5 BundleWiringMBean ...................................................................................................................... 11

7 Command Line API ................................................................................................................................ 13

8 JMX API .................................................................................................................................................. 14

9 Initial Spec Chapter ................................................................................................................................. 14
1.2 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 11.1.

Source code is shown in this typeface.

1.3 Revision History

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

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>December 2010</td>
<td>David Bosschaert, Initial Version</td>
</tr>
<tr>
<td>0.1</td>
<td>January 2011</td>
<td>David Bosschaert, added JMX Notification to problem description.</td>
</tr>
<tr>
<td>0.2</td>
<td>January 2011</td>
<td>Alexandre Alves, First draft of technical solution</td>
</tr>
<tr>
<td>0.3</td>
<td>January 2011</td>
<td>Alexandre Alves, Incorporating David's feedback.</td>
</tr>
<tr>
<td>0.4</td>
<td>February 2011</td>
<td>Alexandre Alves, Incorporating feedback from Berlin's face-to-face meeting.</td>
</tr>
<tr>
<td>0.5</td>
<td>March, 2011</td>
<td>Alexandre Alves, Supporting Bundle Wiring changes</td>
</tr>
</tbody>
</table>

2 Introduction

A number of bugs and emailed suggestions have surfaced with regard to the OSGi-JMX specification. Additionally, the core framework has been significantly updated for version 4.3 with new APIs. This RFC aims to update the OSGi JMX specification to address the concerns and suggestions made, as well as support the updated Core Framework API where applicable.
3 Application Domain

This RFC applies to access to the OSGi Framework through the Java Management Extensions (JMX) Technology.

4 Problem Description

This RFC is based on a number of bugs and otherwise communicated enhancements and fixes to the existing OSGi-JMX specification. There is no RFP that this relates to besides the original OSGi-JMX RFP 105.

4.1 OSGi 4.3 Core Framework Update (Bug 1762)

The OSGi 4.3 Core framework has been enhanced with a number of APIs such as BundleRevision, BundleWiring, BundleWirings, FrameworkWiring all of which are obtained through a the Bundle.adapt() method. The JMX API needs to updated accordingly, where applicable.

4.2 JBoss OSGi-JMX extensions

While working with the OSGi JMX support the JBoss OSGi framework enhanced the JMX support with the following:

4.2.1 BundleStateMBean


4.2.1.1 Bug 1598 - [JMX] BundleStateMBean does not allow to introspect single bundle

Suggested API:

    CompositeData getBundle(long bundleId) throws IOException;

4.2.1.2 Bug 1599 - [JMX] BundleStateMBean does not allow localized header access

Suggested API:

    TabularData getHeaders(long bundleId, String locale) throws IOException;

4.2.1.3 Bug 1600 - [JMX] BundleStateMBean does not allow access to bundle properties

Suggested API:

    CompositeData getProperty(long bundleId, String key) throws IOException;
4.2.1.4 Bug 1601 - [JMX] BundleStateMBean possible additions

This bug contains suggested enhancements which are useful for debugging/troubleshooting purposes. The suggested APIs include:

```java
long loadClass(long bundleId, String name);
String getEntry(long bundleId, String path);
String getResource(long bundleId, String name);
String getDataFile(long bundleId, String filename);
```

The APIs are not intended for the client to actually use the content, but rather to inform the user where the content is resolved to.

4.2.2 ServiceStateMBean

4.2.2.1 Bug 1602 - [JMX] ServiceStateMBean does not allow to introspect single service

Suggested API:

```java
CompositeData getService(long serviceId)
```

4.2.2.2 Bug 1603 - [JMX] ServiceStateMBean does not allow to filter services

Suggested API:

```java
TabularData getServices(String clazz, String filter) throws IOException;
```

4.2.3 FrameworkMBean

4.2.3.1 Bug 1606 - [JMX] FrameworkMBean.resolveBundles does not define behaviour for invalid bundle ids

4.3 Bug 1846 OSGi MBeans do not target a particular OSGi framework instance

The ObjectNames for the OSGi MBeans do not seem to consider the fact that there may be more than one running OSGi framework instances at a time in the same JVM and thus likely the same MBean server.

For example, the ObjectName for the FrameworkMBean is "osgi.core:type=framework,version=1.5" which has no indication of which OSGi framework instance it is target to.

4.4 Email from Jürgen Kissner / SAP

The OSGi Service Platform Enterprise Specification, Release 4, Version 4.2 specifies the BundleStateMBean interface (Section 124.9.2). The listBundles() method (124.9.2.79) is the only way I can find to get information about all bundles available in an OSGi runtime environment.

The problem that we have is that that method is rather expensive to use as it returns a collection of BUNDLE_TYPE structures - basically all header information for all bundles plus some state information. (See section 124.5.2 for the type definition).

We found that invoking that method takes several seconds on standard hardware with an OSGi runtime that contains around 100 bundles.
An important use case is to determine the status of certain bundles. Usually the bundle names are known. With the current interface we have to retrieve all information and then filter what we need. It would be preferable to be able to specify some kind of filter as an input argument or to restrict the output in some way (e.g., it would be sufficient to just return the Bundle-ID, name, version and state). Alternatively it would be helpful to retrieve a list of bundle-Ids, as that could be used to retrieve the additional information.

4.5 Bug 1592 UserAdminMBean - differences between spec and RI

There are differences between the spec description and RI for UserAdminMBean; here is the list with problems I found:

• Spec 124.13.1.14 defines that ROLE_TYPE contains NAME, TYPE and PROPERTIES; within the RI PROPERTIES field is missing

• Spec 124.13.1.19 defines that USER_TYPE contains CREDENTIALS and NAME, TYPE, PROPERTIES coming from ROLE_TYPE; within the RI CREDENTIALS and PROPERTIES fields are missing

• Spec 124.13.1.4 defines that GROUP_TYPE contains MEMBERS, REQUIRED MEMBERS and CREDENTIALS, NAME, TYPE, PROPERTIES coming from USER_TYPE; within the RI CREDENTIALS and PROPERTIES fields are missing

• Spec 124.13.1.1 defines that AUTORIZATION_TYPE contains NAME and TYPE; within the RI TYPE field is missing

Comment from Hal: Add property support to the next release of the spec.

4.6 Bug 1616 Add properties support

When I'm calling the method listServices of ServiceStateMBean, it returns object from TabularData type (SERVICES_TYPE). When I get row composite data type from it (SERVICE_TYPE), the attribute "Properties" is missing in it (such attribute is specified in the spec).

---A large amount of discussion including code snippets can be found in the bug

4.7 Bug 1645 Invalid RuntimeException handling in CT

This is a CT bug, it's mentioned here to ensure that it is dealt with along with the other JMX work.

4.8 Bug 1646 UserAdminMBean issues

> Two issues:
> 1) The description of UserAdminMBean does not define what should the GROUP_TYPE CompositeData contain if there are no member or required-members. That is, if the lookup on member of required-members should return null or an empty array. The CT assumes empty array but probably should handle both.

Hal: I believe it should only be an empty array. An empty array never throws a null pointer exception and always does the right thing.
> 2) UserAdminMBean.addMember() or .addRequiredMember() does not allow
> IllegalArgumentException to be raised when invalid group name is passed. This
> is inconsistent with other such methods in this mbean, for example,
> addCredential() or getGroup()

Hal: That's because, as other bugs have forced, there are no runtime exceptions to be thrown by the beans.

Patch available in the bug.

### 4.9 Bug 1647 BundleStateMBean.getRequiredBundles() clarification

Clarify the behaviour of BundleStateMBean.getRequiredBundles() in the spec.

### 4.10 Bug 1649 ConfigurationAdminMBean clarification

The specification does not say what should happen in the TabularType passed into the ConfigurationAdminMBean.update() and updateForLocation() functions is not of JmxConstants.PROPERTIES_TYPE.

Our implementation expects JmxConstants.PROPERTIES_TYPE type but the CT passes a slightly different type (i.e. the type name is "Properties" instead of "PROPERTIES").

Hal: Yes, this should be consistent.

## 5 Requirements

The requirements of this RFC are described in RFP 105.

## 6 Technical Solution

### 6.1 OSGi MBean Object Names

A Java process may launch more than one OSGi framework instance at a time; therefore it must be possible for each OSGi framework instance to register their own instances of the OSGi MBeans.

The distinction of OSGi MBeans across OSGi framework instances is accomplished through the addition of the key property *uuid* to the OSGi MBean Object Names defined originally by the JMX Management Model specification, version 1.0. The value of this property must be equal to the OSGi framework UUID of the launched framework instance.
For example, the following Object Name identifies the FrameworkMBean for the first launched instance of the Apache Felix implementation of the OSGi framework:

```
osgi.core:type=framework,version=1.5,uuid=f81d4fae-7dec-11d0-a765-00a0c91e6bf6,...
```

However, UUIDs are not user-friendly and thus hard to manage in JMX. Therefore, the key property `framework` is also added to the OSGi MBean Object Names. The `framework` property must be equal to the Bundle-SymbolicName of the OSGi framework instance’s System Bundle.

Following, we revisit the previous example to include the `framework` property:

```
osgi.core:type=framework,version=1.5,uuid=f81d4fae-7dec-11d0-a765-00a0c91e6bf6,
        framework=org.apache.felix.framework
```

Likewise, a possible Object Name for the BundleStateMBean is:

```
osgi.core:type=bundleState,version=1.5,uuid=f81d4fae-7dec-11d0-a765-00a0c91e6bf6
        framework=org.apache.felix.framework
```

The advantage of the `framework` property is that it can be used to simplify the querying for the MBeans using Object Name patterns (e.g. names with asterisks). For instance, the following query allows a client to find all FrameworkMBeans for a Felix implementation without having to rely on knowing the UUID:

```
ObjectName frameworkObjName =
    new ObjectName("osgi.core:type=framework,version=1.5,
        framework=org.apache.felix.framework,*");
mbeanServerConnection.queryMBeans(frameworkObjectName, null);
```

Furthermore, in many cases, a JMX client may appropriately assume that only a single instance of the OSGi framework exists in the managed system, as in the following example:

```
ObjectName frameworkObjName =
    new ObjectName("osgi.core:type=framework,version=1.5,*");
mbeanServerConnection.queryMBeans(frameworkObjectName, null);
```

The `uuid` and `framework` key properties are only applicable to OSGi JMX Package Version 1.? and above. JMX clients using Version 1.0 should refrain from including these properties.

To maintain backward compatibility, a OSGi JMX Package Version 1.? may register the first instantiation of an OSGi framework using both the Version 1.0 Object Names as well as the Object Names outlined in this specification. In other words, a JMX client may not specify the `uuid` and/or `framework` properties, and still retrieve the MBeans for a OSGi framework instance.

### 6.2 BundleStateMBean

The state of a bundle consists of several items, such as its exported packages, its imported packages, its headers, etc. Although each one of these items can be individually retrieved for a bundle, there is no single API to retrieve all of these collectively. Although this may not be a concern for local call invocations, it is typically a goal of JMX APIs to minimize the amount of information exchanged due to network constraints. Considering this, the following method is added to the BundleStateMBean class:

```
CompositeData getBundle(long bundleId) throws IOException;
```

The Composite Data returned is of type `BUNDLE_TYPE`.

It is often necessary to find the state of all bundles installed in an OSGi framework, however generally not all the state is needed, but rather a sub-set of it, which is application specific. For example, a management tool may be interested only on the symbolic name. The method `listBundles()` could be used to retrieve all the state, including the symbolic name, however `listBundles()` is expensive. Therefore, a variation of `listBundles()` is added to allow the selection of which items should be returned as part of the `BUNDLE_TYPE` composite data:
TabularData listBundles(String [] bundleTypeItems) throws IOException;

For example, to find out the symbolic name and the location of all installed bundles, one can issue the following operation:
listBundles(new String[]{BundleStateMBean.SYMBOLIC_NAME,
BundleStateMBean.LOCATION});

Specification of an unsupported item causes an IllegalArgumentException to be thrown.

Even though the method listBundles() allows for the retrieval of all bundle identifiers, it is sometimes preferable to reference this information as a MBean attribute, therefore a BundleIds JMX attribute is added.

The following operation is added:
boolean isActivationPolicyUsed()

This operation maps to the equivalent method of the BundleStateLevel interface.

6.2.1 Headers

All the headers of a bundle can be retrieved with the method getHeaders(), but likewise there is no single method to retrieve a a single header. The following method is added to allow for this:
String getHeader(long bundleId, String key) throws IOException;

Bundle headers may be localized. In the absence of an argument indicating the locale to be used, such as in the case of the methods getHeaders(long bundleId) and getHeader(long bundleId, String key), the default locale is used. The following methods are added to the BundleStateMBean for dealing with locales:
TabularData getHeaders(long bundleId, String locale) throws IOException;
CompositeData getHeader(long bundleId, String key, String locale)
throws IOException;

The interpretation of the locale argument is similar to that defined by the method Bundle.getHeaders(String locale).

6.2.2 Requiring Bundles

The methods getRequiredBundles(long bundleId) and getRequiringBundles(long bundleId) can be used to discover the relationship between bundles established by the use of the Require-Bundle manifest header.

For example, consider a bundle A with the following header:
Require-Bundle: B, C

Furthermore, bundle D defines the following header:
Require-Bundle: A

A call to getRequiredBundles() for bundle A would return the bundle identifiers for bundles B and C. Conversely, a call to getRequiringBundles() for bundle A would yield the bundle identifier for bundle D.

6.3 ServiceStateMBean

Tailing the pattern established for the BundleStateMBean, the following new methods are added to the ServiceStateMBean:
CompositeData getService(long serviceId) throws IOException;

The Composite Data returned is of type SERVICE_TYPE.
CompositeData getProperty(long serviceId, String key) throws IOException;

The Composite Data returned is of type PROPERTY_TYPE. Note that we could not simply return the value, as the value is encoded as a String and the item TYPE_ITEM is needed to understand how to decode it.

TabularData listServices(String clazz, String filter) throws IOException;

The Tabular Data returned is of type SERVICES_TYPE. The arguments clazz and filter have the semantic as of the method BundleContext.getAllServiceReferences().

Furthermore, the Tabular Data JmxConstants.PROPERTIES_TYPE is added to the SERVICE_TYPE composite data. By doing so, we now include all of the state of a service in the calls to listServices() and getService().

Finally, the following methods are used to allow the selection of which items are to be returned by the method listServices():

TabularData listBundles(String [] bundleTypeItems) throws IOException;
TabularData listServices(String clazz, String filter, String [] bundleTypeItems) throws IOException;

The Tabular Data returned is of type SERVICES_TYPE, however only contains the selected items relating to SERVICE_TYPE.

Likewise, for ease of use and performance, a ServiceIds JMX attribute is added.

### 6.4 FrameworkMBean

The FrameworkMBean supports the batching of operations, such as installBundles(), and uninstallBundles(). These return a Composite Data of type BATCH_ACTION_RESULT_TYPE indicating the individual results of the batched operations.

The method resolveBundles() should likewise be changed to return a Composite Data of type BATCH_ACTION_RESULT_TYPE, however whereas the other batch operations need not return any additional information in case of success, in the case of the resolveBundles() a bundle may not resolve and still be considered a successful operation. In other words, a successful resolve operation results into a boolean true or false. An unsuccessful operation, such as in the case of an invalid bundle id, throws an exception and provides no result.

To accommodate this behavior, the semantic of the COMPLETED item in the BATCH_ACTION_RESULT_TYPE Composite Data for the resolveBundles() method is changed to return an array of Boolean objects (e.g. Boolean []) and to be always present, even in the case of a successful operation.

Furthermore, a “refreshBundles() : CompositeData” method is added. This method is synchronous, and shall likewise make use of the BATCH_ACTION_RESULT_TYPE to return its results.

The existing “refreshBundles(): void” method is kept asynchronous, however it will post a JMX notification with its results mimicking the same format as of the synchronous refreshBundles() method.

The following operations are added to the FrameworkMBean to support the FrameworkWiring class:

long [] getDependencyClosure(long [] bundles) throws IOException
long [] getRemovalPendingBundles() throws IOException

Both methods return an array of bundle Ids, or an empty array.
6.5 BundleWiringMBean

The following MXBean is added to support the org.osgi.framework.wiring package:

```java
package org.osgi.jmx.framework;

import java.util.List;
import java.util.Map;

/**
 * This MBean represents the bundle wiring state.
 * Note that not all information from the BundleWiring Java API is provided.
 * Particularly, the limitations are:
 * - Cannot retrieve references to resources (e.g. class) of a particular bundle
 * wiring.
 */
public interface BundleWiringMBean {

    /**
     * This MXBean structure represents a bundle requirement.
     */
    interface BundleRequirementComposite {
        Map<String, Object> getAttributes(); /* REVIEW: How do you work with
         Objects in MXBeans? */
        Map<String, String> getDirectives();
        String getNamespace(); /* TODO: Define constants */
    }

    /**
     * This MXBean structure represents a bundle capability.
     */
    interface BundleCapabilityComposite {
        Map<String, Object> getAttributes();
        Map<String, String> getDirectives();
        String getNamespace();
    }

    /**
     * This MXBean structure represents the live association between a provider of
     * a capability and a requirer of the corresponding requirement.
     */

    // Additional methods...
}
```

Copyright © Red Hat, Oracle, SAP, IBM and others 2011 All Rights Reserved
interface BundleWireComposite {
    BundleRequirementComposite getWiredRequirement();
    BundleCapabilityComposite getWiredCapability();
    long getProvider();
    long getRequirer();
}

/**
 * @param bundleId
 * @param namespace
 * @return the declared requirements for the current revision of <code>bundleId</code>
 */
    List<BundleRequirementComposite> getCurrentRevisionDeclaredRequirements(long bundleId, String namespace);

/**
 * @param bundleId
 * @param namespace
 * @return the declared capabilities for the current revision of <code>bundleId</code>
 */
    List<BundleCapabilityComposite> getCurrentRevisionDeclaredCapabilities(long bundleId, String namespace);

/**
 * @param bundleId
 * @param namespace
 * @param inUse
 * @return the wires for the current revision of <code>bundleId</code>
 */
    List<BundleWireComposite> getCurrentWires(long bundleId, String namespace);

/**
 * @param bundleId
 * @param namespace
 * @param inUse
 * @return the declared requirements for all revisions of <code>bundleId</code>
 */
    List<List<BundleRequirementComposite>> getRevisionsDeclaredRequirements(long bundleId, String namespace, boolean inUse);

Copyright © Red Hat, Oracle, SAP, IBM and others 2011 All Rights Reserved
/**
 * @param bundleId
 * @param namespace
 * @param inUse
 * @return the declared capabilities for all revisions of <code>bundleId</code>
 */
List<List<BundleCapabilityComposite>> getRevisionsDeclaredCapabilities(long bundleId,
                        String namespace, boolean inUse);

/**
 * @param bundleId
 * @param namespace
 * @return the wires for all revisions of <code>bundleId</code>
 */
* REVIEW No need to have a 'inUse' param as wires are by definition 'in use'.
* /
List<List<BundleWireComposite>> getRevisionsWires(long bundleId, String namespace);

/**
 * @param capability
 * @param requirement
 * @return true if capability matches with requirement.
 */
boolean matches(BundleCapabilityComposite capability, BundleRequirementComposite requirement);
}

Although BundleWiringMBean provides an alternative approach to retrieving the same state as supported by the
PackageAdminMBean, the latter is kept as it provides a more user-friendly API for dealing exclusively with
packages.

### 7 Command Line API

*If this specification would benefit from a command line interface, describe it here. Commands should be realized
as described in RFC 147.*

*This section is optional and could also be provided in a separate RFC.*
8  JMX API

RFC 139 describes the JMX API to the OSGi Framework and a number of standard OSGi Services. The JMX specification is also present as chapter 124 in the OSGi spec documents.

For all new functionality added to the OSGi Framework the question should be asked: would this feature benefit from a JMX API? The expectation is that in most cases it would.

The JMX API for the design in this RFC should be described here and if there is no JMX API an explanation should be given explaining why this is not applicable in this case.

This section is optional and could also be provided in a separate RFC.

This RFC is specifically about the JMX API.

9  Initial Spec Chapter

Provide a link to where the Initial Spec Chapter can be found. The Initial Spec Chapter is typically written by someone other than the author(s) of this RFC and represents a rewrite of this document as close as possible to what will ultimately appear in the OSGi Specifications. It will be used by the Specification Editor as the basis for the ultimate specification chapter.

The spec template and writing guidelines can be found here:


10  Considered Alternatives

For posterity, record the design alternatives that were considered but rejected along with the reason for rejection. This is especially important for external/earlier solutions that were deemed not applicable.
11 Security Considerations

Description of all known vulnerabilities this may either introduce or address as well as scenarios of how the weaknesses could be circumvented.

12 Document Support

12.1 References


Add references simply by adding new items. You can then cross-refer to them by choosing <Insert><Cross Reference><Numbered Item> and then selecting the paragraph. STATIC REFERENCES (I.E. BODGED) ARE NOT ACCEPTABLE, SOMEONE WILL HAVE TO UPDATE THEM LATER, SO DO IT PROPERLY NOW.

12.2 Author’s Address

<table>
<thead>
<tr>
<th>Name</th>
<th>David Bosschaert</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Red Hat</td>
</tr>
<tr>
<td>Address</td>
<td>6700 Cork Airport Business Park</td>
</tr>
<tr>
<td></td>
<td>Kinsale Road</td>
</tr>
<tr>
<td></td>
<td>Cork</td>
</tr>
<tr>
<td></td>
<td>Ireland</td>
</tr>
<tr>
<td>Voice</td>
<td>+353 21 230 3400</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:david@redhat.com">david@redhat.com</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Alexandre Alves</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company</td>
<td>Oracle Corporation</td>
</tr>
<tr>
<td>Address</td>
<td></td>
</tr>
<tr>
<td>Voice</td>
<td>+1 650-607-0878</td>
</tr>
<tr>
<td>e-mail</td>
<td><a href="mailto:alex.alves@oracle.com">alex.alves@oracle.com</a></td>
</tr>
</tbody>
</table>
12.3 Acronyms and Abbreviations

12.4 End of Document
The OSGi Alliance and its members specify, create, advance, and promote wide industry adoption of an open delivery and management platform for application services in home, commercial buildings, automotive and industrial environments. The OSGi Alliance serves as the focal point for a collaborative ecosystem of service providers, developers, manufacturers, and consumers. The OSGi specifications define a standardized, component oriented, computing environment for networked services. OSGi technology is currently being delivered in products and services shipping from several Fortune 100 companies. The OSGi Alliance’s horizontal software integration platform is ideal for both vertical and cross-industry business models within home, vehicle, mobile and industrial environments. As an independent non-profit corporation, the OSGi Alliance also provides for the fair and uniform creation and distribution of relevant intellectual property – including specifications, reference implementations, and test suites – to all its members.

HOW TO REACH US:

OSGi Alliance
Bishop Ranch 6
2400 Camino Ramon, Suite 375
San Ramon, CA 94583 USA

Phone: +1.925.275.6625
E-mail: marketinginfo@osgi.org
Web: http://www.osgi.org

OSGi is a trademark of the OSGi Alliance in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

All other marks are trademarks of their respective companies.